МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ ГРИ З ВИКОРИСТАННЯМ PIXIJS ТА VITE»**

Виконала: студентка 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
А.І. Сачковська
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Сачковській Аліні Ігорівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Проектування та розробка веб гри з використанням PixiJs та Vite

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.03.2024	
2.	Збір вихідних даних.	10.04.2024	
3.	Обробка методичних та теоретичних	15.05.2024	
	джерел.		
4.	Розробка першого та другого розділу.	24.06.2024	
5.	Розробка третього розділу.	09.09.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ А.І. Сачковська
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Проектування та розробка веб гри з використанням PixiJs та Vite»: 104 с., 43 рис., 10 джерел, 3 додатки.

ВЕБ ГРА, РОЗРОБКА ІГОР, РОЛЬОВА ГРА, 2D-ГРАФІКА, JAVASCRIPT, PIXIJS, VITE.

Об’єкт дослідження – розробка двовимірної рольової веб гри з використанням PixiJs.

Мета роботи: розробка веб гри засобами PixiJs.

Метод дослідження – методи програмної інженерії: об’єктно-орієнтоване програмування, UML-методологія, тестування.

У роботі було досліджено процес розробки та створено веб гру з використанням PixiJs та Vite. Наведено аргументацію вибору інструментів, визначено вимоги до застосунку, побудовано відповідні діаграми та реалізовано веб гру з усім необхідним функціоналом. Розроблена гра має двовимірну графіку, рольову систему класів, покрокову механіку бою, систему ігрового прогресу, графічний інтерфейс користувача та можливість реєстрації для збереження прогресу.

Результати роботи можуть бути використані для дослідження розробки веб ігор з двох-вимірною графікою, а також у сфері розробки ігрових елементів для вебзастосунків.

SUMMARY

Master’s qualifying paper “Design and Development of a Web Game Using PixiJs and Vite”: 104 pages, 43 figures, 10 references, 3 supplements.

WEB GAME, GAME DEVELOPMENT, ROLE-PLAYING GAME, 2D-GRAPHICS, JAVASCRIPT, PIXIJS, VITE.

The object of the study is development of two-dimensional role-playing Web Game using PixiJS.

The aim of the study is to develop a Web Game using PixiJs.

The methods of research are methods of software engineering: object-oriented programming, UML-methodology, testing.

Work contains research of development and creation of a Web Game using PixiJs and Vite. Given argumentation of instrument choice, defined requirements for application, built corresponding diagrams and implementation of Web Game with all required functionality. Developed game has two-dimensional graphics, role-playing system of classes, turn-based combat system, progression system, a graphical user interface and user registration for progress saving.

Results of work can be used for research of development of Web Games with two-dimensional graphics, as well as in the field of developing game elements for web applications.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області..... 8
1.1 Актуальність дослідження 8
1.2 Аналіз засобів розробки веб ігор..... 14
1.3 Аналіз функціональних вимог..... 17
1.4 Висновки до розділу 1 19
2 Проєктування застосунку..... 20
2.1 Проєктування структури застосунку 20
2.2 Висновки до розділу 2 23
3 Програмна реалізація..... 24
3.1 Програмна реалізація гри..... 24
3.2 Висновки до розділу 3 45
Висновки 47
Перелік посилань..... 48
Додаток А Код ігрової логіки 49
Додаток Б Код інтерфейсу користувача 81
Додаток В Код серверної частини..... 99

ВСТУП

Розвиток технологій зараз призводить до величезного розповсюдження використання різноманітної техніки, в особливості комп'ютерів та смартфонів, що в свою чергу сприяє збільшенню популярності відеоігор серед їх користувачів, тому майже кожен зараз використовує ігрові додатки.

Отже, актуальною задачею є розробка веб ігор з використанням PixiJs.

Метою кваліфікаційної роботи є розробка двовимірної рольової веб гри з використанням PixiJs та Vite.

Задачі, які необхідно розв'язати для досягнення поставленої мети:

- визначити методи дослідження та набір технологій для розробки;
- провести порівняльний аналіз існуючих аналогів;
- визначити основні функції розроблюваного додатку;
- створити діаграму прецедентів;
- визначити список вимог до розроблюваного додатку;
- розробити інші необхідні діаграми;
- протестувати розроблений додаток та виправити помилки.

Об'єкт дослідження – розробка двовимірної рольової веб гри з використанням PixiJs.

Предмет дослідження – процес розробки веб ігор з використанням PixiJs.

Методи дослідження – методи програмної інженерії: об'єктно-орієнтоване програмування, UML-методологія, тестування.

Структурно дипломна робота складається з таких компонентів: вступ, 3 розділи, висновки, 3 додатки, перелік посилань з 10 найменувань.

В першому розділі проведено порівняльний аналіз існуючих аналогів, а також засобів розробки веб ігор. В другому розділі визначено необхідний функціонал розроблюваного додатку, наведено процес проєктування застосунку. В третьому розділі показано процес реалізації веб гри, наведено детальні пояснення механізму роботи застосунку, а також приклади роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність дослідження

В останні роки, коли відеоігри отримали величезну популярність та розповсюдженість по всьому світу та серед всіх категорій населення, а прибутки ігрової індустрії складають мільярди доларів щороку, немає сумніву що розробка веб ігор є актуальною задачею. В особливості зараз, через загальне зниження якості ігрових продуктів з причини відсутності новизни ідей та цікавості ігрового процесу, дуже популярними стають відеоігри в старому стилі, що викликають відчуття ностальгії в більшості користувачів – з піксельною двовимірною графікою, рольовою системою та покроковою системою бою. Окрім цього, браузерні ігри завжди були актуальними через свою доступність, оскільки далеко не всі можуть дозволити собі придбання спеціалізованих ігрових консолей, а веб ігри зараз можна запустити з майже будь-якого комп’ютера або смартфону, якими користується більшість людей.

Для визначення актуальності та новизни дослідження «Розробка вебігор засобами PixiJs та Vite», а саме унікальності двовимірної рольової веб гри з покроковою системою бою, було проведено аналіз літератури та існуючих аналогових продуктів на предмет наявності актуальних проблем та недоліків в даній області.

Наявна наукова література в галузі розробки веб ігор є не дуже багаточисельною, а також показує процес розробки веб ігор значно простіших за механіками та ігровим процесом, аніж будь-яка рольова покрокова гра. Також важливим є вибір технології, оскільки дослідження вивчає процес розробки саме на PixiJS. З використанням цієї бібліотеки не розроблено жодної офіційної веб гри, а єдине наукове дослідження (книга «Learn Pixi.js» [1]) розробляє дуже просту веб гру.

Серед існуючих аналогів продуктів можна виділити кілька представників. Одними з найвідоміших аналогів, що досі мають своїх прихильників, є «Runescape» та «Drakensang Online» (рис. 1.1 та 1.2 відповідно), що є класичними представниками рольових браузерних ігор.



Рисунок 1.1 – «Runescape»



Рисунок 1.2 – «Drakensang Online»

Однак вони мають певні відмінності відносно розроблюваного застосунку, однакові для обох додатків, такі як:

– тривимірна графіка;

– мультиплеєр;

– бойова система в стилі Action-RPG (Action Role Playing Game).

Таким чином, ці додатки, будучи найпопулярнішими серед аналогів, є найменш схожими на створюваний в процесі дослідження програмний застосунок, а також вже є застарілими, оскільки вище зазначені додатки були випущені більше 10 років назад на момент проведення дослідження, а технології розробки ігор розвиваються дуже швидко.

Також дуже популярною є веб гра «Tibia» (рис. 1.3), що має всі необхідні якості та найбільш схожа розроблюваний застосунок, включаючи такі особливості:

– піксельна двовимірна графіка,

– рольова система,

– покрокова система бою.



Рисунок 1.3 – «Tibia»

Однак, як і попередні приклади, вона є багатокористувацькою, що означає відсутність повноцінної одиночної сюжетної кампанії, а також багатьох механік та особливостей, притаманних одиночним іграм, що також є дуже важливим аспектом. Багатокористувацькі ігри були предметом суперечок гравців протягом багатьох років, оскільки вони не надають користувачу такий самий досвід, як ігри призначені для одного гравця, оскільки найчастіше вони розраховані на спілкування та взаємодію гравців, нескінченне виконання однакових за своєю суттю завдань та повторення одноманітних дій для підвищення рівня або отримання різних предметів, в той час як однокористувацькі ігри більше приваблюють гравця своїми оригінальними механіками, цікавим сюжетом та завданнями.

Також важливо зазначити, що «Tibia» була випущена в 1997 році, та вже давно не є актуальною та цікавою для багатьох користувачів.

Поміж одиночних рольових ігор є багато схожих представників з усіма необхідними параметрами, таких як старі ігри серії «Pokemon», ігри «Dragon Quest» та «Sea of Stars». Вони є представниками двовимірних рольових ігор, однак ці ігри не є браузерними, а також не дають можливості гравцю повністю створити свого персонажа, як це можна зробити в будь-якій мультиплеєрній грі, а дозволяють лише грати за головного героя, що створений завчасно розробниками гри. Далі буде детальніше розглянуто вище зазначені додатки.

Ігри серії «Pokemon», від «Pokemon Red and Blue» до «Pokemon FireRed & LeafGreen» (рис. 1.4) є двовимірними піксельними покроковими рольовими іграми. Їх відмінністю від майбутнього додатку є аспект створення власної команди для бою, а саме збору покемонів що будуть битися на стороні головного героя. Як і в інших рольових іграх, гравець може підвищувати рівень покемонів, давати їм спорядження та вивчати різні навички. Проте замість одного героя гравець повинен слідкувати за великою їх кількістю, що не завжди є позитивною якістю. До того ж, на даний час є дуже важким завданням спробувати ігри цієї серії, оскільки спеціальні ігрові консолі є старими та їх важко дістати, а сучасна техніка вимагає використання емуляторів для відтворення подібних старих ігор.



Рисунок 1.4 – «Pokemon FireRed & LeafGreen»

Одним з класичних представників рольових ігор є ігри «Dragon Quest» (рис. 1.5), які випускаються і зараз, та деякі з них навіть мають двовимірний ігровий режим, що виглядає так само як і старі представники серії, що лише підтверджує актуальність розробки подібної двовимірної рольової гри в старому стилі. Однак ігри цієї серії мають схожі особливості з іграми «Pokemon», оскільки гравцю необхідно керувати цілою командою персонажів замість одного, а також відтворення старих ігор серії є не простою задачею.

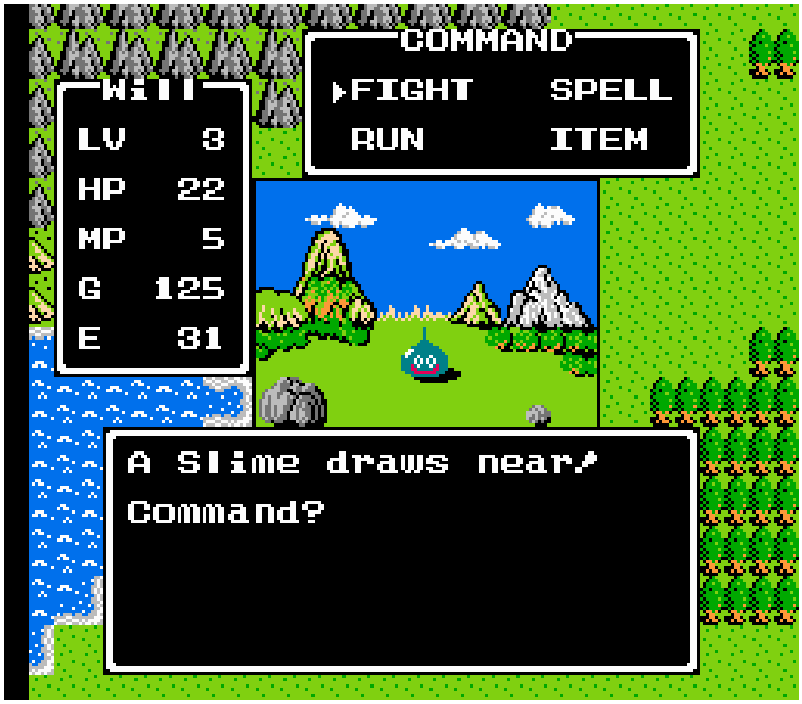


Рисунок 1.5 – «Dragon Quest 1»

Найсучаснішим представником двовимірних рольових ігор є «Sea of Stars», показана на рисунку 1.6, що була розроблена в 2023 році спеціально в стилі класичних JRPG (японських рольових ігор), які раніше були неймовірно популярними. Ця гра є сучасною, має піксельну двовимірну графіку, є рольовою, має покрокову бойову систему та є доступною на комп'ютерах та всіх сучасних ігрових консолях. «Sea of Stars» також вимагає від гравця керування своєю командою окрім головного персонажа, як і попередні ігри. Ще однією особливістю цієї гри є її система бою, що є покроковою, однак для виконання певних команд вимагає від гравця певних дій, таких як вчасне натискання клавіш для посилення виконуваної дії, як атака чи захист.

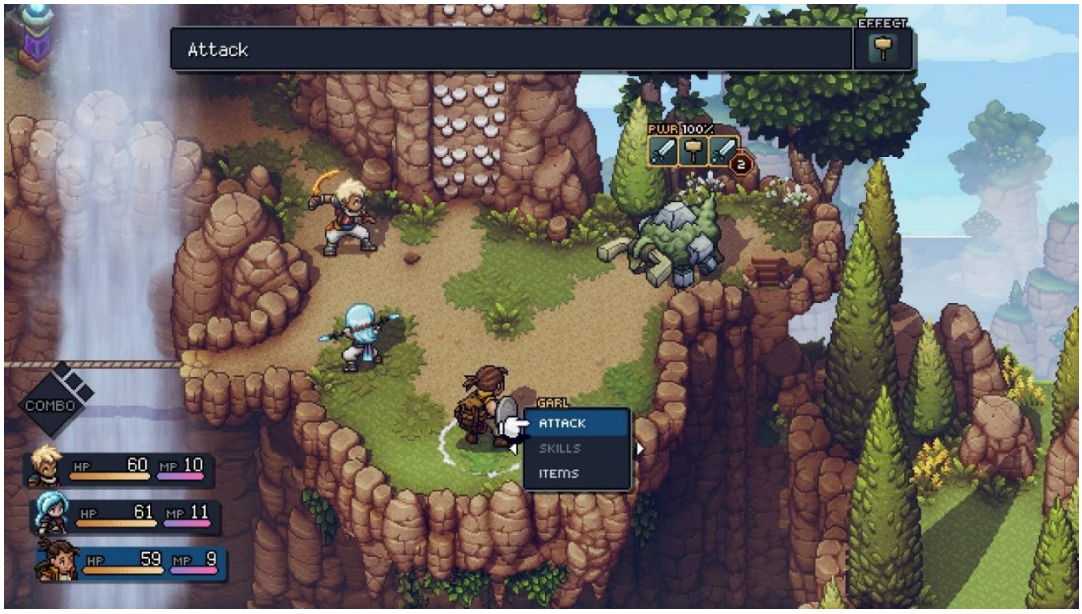


Рисунок 1.6 – «Sea of Stars»

Отже, можна зробити висновок, що відомі ігрові аналоги найбільш схожі за механіками не являються веб іграми, а наявні веб ігри найбільш близькі до теми дослідження є багатокористувацькими, тому можна побачити прогалину в цій сфері – відсутність повноцінної двовимірної браузерної рольової гри для одного гравця.

Окрім всього вищезазначеного, існують різноманітні розробки у вільному доступі в галузі вебігор, що не є офіційними продуктами, що лише підтверджує актуальність розробки браузерних ігор.

Таким чином чітко видно, що дослідження, в ході якого буде розроблено двовимірну піксельну рольову покрокову веб гру, є актуальним та необхідним.

1.2 Аналіз засобів розробки веб ігор

Для розробки вебігор існує декілька найбільш відомих засобів, які мають певні відмінності, свої переваги та недоліки.

Одним із найпопулярніших засобів розробки та відображення графіки у браузері є WebGL, що підтримується більшістю сучасних браузерів. Він дозволяє створювати двовимірну та трьох вимірну графіку в браузері, а також є основою кількох інших бібліотек для розробки вебдодатків з сучасною графікою. Однак, WebGL є дуже низькорівневим API, тому робота з ним може викликати труднощі, також цей засіб не дуже добре оптимізований та відображення великої кількісті графічних об’єктів без помилок вимагає наявність потужної техніки.

Також серед відомих засобів розробки можна виділити ThreeJS – графічну бібліотеку для роботи з трьох вимірною графікою засновану на WebGL. Ця бібліотека набагато краще оптимізована, є високорівневою, а тому простою для використання, а також надає багато інструментів для анімацій та взаємодії з об’єктами. Проте ThreeJS не має прямої можливості для роботи з двовимірною графікою, лише обхідні шляхи, які не є коректним способом двовимірної розробки, тому не є підходящою для даного дослідження.

Зараз дуже популярною є бібліотека PixiJs – це дуже потужний обробник графіки, що дозволяє створювати сучасні графічні елементи як для вебсайтів, так і для веб ігор, є дуже добре оптимізованою для відображення якісного зображення та великої кількості об’єктів одночасно, має багато різноманітних графічних фільтрів, потужний текстовий обробник, а також великий функціонал для інтерактивності, та є простою у використанні. Як висловлюється автор книги “Learn Pixi.js” [1, с. 19], Рекс ван дер Спай, – «Pixi

це неймовірно швидкий рушій для рендерингу 2D-спрайтів, що допомагає вам відображати, анімувати, та керувати інтерактивною графікою, що робить легким для вас створення будь-якого візуально багатого інтерактивного програмного забезпечення, яке ви можете уявити, з використання JavaScript та інших технологій HTML5» [2]. Окрім цього, PixiJs має багато супутніх бібліотек з додатковим функціоналом, таких як pixi/tilemap для створення ігрової мапи, pixi/sound для використання аудіо матеріалів, а також pixi/ui для спрощення розробки інтерфейсу користувача.

Також існує фреймворк для розробки двовимірних веб ігор Phaser, заснований на PixiJS. Він є більш високорівневим, а тому простішим у використанні для більшості користувачів, та має багато вбудованих функцій для розробки ігор, таких як анімації та фізичний рушій. Однак, через велику кількість інших функцій цей фреймворк є повільнішим, та має менші графічні можливості ніж PixiJs, оскільки він використовує лише частину функцій цієї бібліотеки, тому не має її повної потужності.

Хоча Phaser має багато функції для швидкої розробки, для досягнення поставленої мети, а саме розробки двовимірної рольової гри з покроковою системою бою, фізичний рушій не потрібен, тому функції PixiJs буде повністю достатньо. До того ж, ця бібліотека надає більше можливостей для детальної роботи з графікою, а також можливість використовувати велику кількість об’єктів та створити добре оптимізовану гру для браузеру.

Окрім інструменту для розробки графіки та ігрових механік для роботи над проєктом необхідним є пакувальник для поєднання модулів у цільну програму.

Найсучаснішим та найзручнішим пакувальником наразі є Vite, що підтримується більшістю сучасних браузерів, а також є дуже швидким та зручним за рахунок підключення модулів до проєкту без необхідності його перезапуску.

Аналогом цього пакувальника є Webpack, що багато років був популярним засобом для збірки проєктів. Однак зараз він є застарілим та

повільним, незручним в роботі та потребує постійного перезапуску проєкту при найменших змінах в збірці, тому цей продукт більше не є актуальним.

Також для розробки додатку потрібна СУБД (система управління базами даних), оскільки програма міститиме реєстрацію користувачів та збереження ігрового прогресу.

Одним з найвідоміших представників є MySQL – це реляційна СУБД, що використовується в усіх сферах розробки. Через свою реляційну структуру вона вимагає чіткої структури даних, що не завжди є зручним для розробників. Хоча MySQL є швидкою, вона має великі можливості для складних запитів для пошуку та фільтрації даних, та саме через це вона може ставати повільнішою за інші СУБД.

Через незручність реляційної схеми даних, враховуючи тип розроблюваного додатку, доцільним буде обрати СУБД нереляційного типу. Найпопулярнішими представниками цього типу є MongoDB та Redis, однак вони мають свої переваги та недоліки.

MongoDB зберігає дані у вигляді документів, а Redis – в парах «ключ-значення», через що MongoDB зручніше використовувати для більших за обсягом елементів даних, доступ до яких потрібен рідше, таких як дані користувача, в той час як Redis підходить для зберігання малих обсягів даних, запити до яких робляться набагато частіше, наприклад логи чатів з іменем та повідомленням. Redis вважається швидшим через невеликі обсяги збережених даних, однак MongoDB є більш гнучким та має багато функцій для зручної роботи з документами та доступу до даних.

Отже можна зробити висновок, що для даного проєкту, в якому база даних необхідна для збереження даних користувачів, таких як ім’я, пароль, а також ігрові дані, найбільш підходящим варіантом є використання MongoDB, оскільки необхідно зберегти не малий обсяг даних, доступ до яких потрібен одночасно та не часто, лише при реєстрації, авторизації та збереженні гри.

Таким чином з проведеного аналізу наявних засобів розробки вебігор видно, що дослідження розробки вебігор з використанням саме PixiJs є вартим

уваги, а також доцільним є використання пакувальника Vite для збірки проєкту та СУБД MongoDB для збереження даних користувача.

1.3 Аналіз функціональних вимог

Після вибору стеку технологій для розробки необхідно визначити список функціональних вимог до майбутнього додатку, а саме на що повинен бути спроможний додаток та які можливості надавати користувачам.

В першу чергу було складено перелік кроків, необхідних для успішної та послідовної реалізації додатку, а також для полегшення майбутнього проєктування та розробки, таких як:

– підібрати необхідні графічні та аудіо матеріали для розробки;

– створити ігрову мапу з усіма необхідними об’єктами;

– створити персонажа, додати йому можливість переміщення та відповідні анімації;

– розробити графічний інтерфейс користувача всередині гри (шкала здоров’я, інвентар, панель персонажа, і т. д.);

– додати кілька різних локацій та можливість переміщення між ними;

– створити ворогів, їх переміщення й відповідні анімації;

– розробити покрокову систему бою;

– додати алгоритм поведінки ворогів у бою;

– розробити систему класів персонажа та можливість підвищення рівня;

– додати інших неігрових персонажів з заданими репліками та поведінкою;

– створити систему ігрових завдань із нагородами;

– додати ігрові магазини та різноманітне спорядження для героя;

– додати звуковий супровід;

– додати на сайт з грою сторінку реєстрації користувача;

– реалізувати авторизацію користувачів з використанням бази даних;

– зробити можливим збереження ігрового прогресу для зареєстрованих гравців.

Після розробки додатку з виконанням всіх вище зазначених кроків, користувач повинен мати такі можливості:

– реєструватися на сайті;

– обирати клас свого персонажа;

– переміщуватися по ігровій мапі;

– користуватися ігровим інтерфейсом;

– переміщуватися з однієї локації на іншу;

– взаємодіяти з іншими неігровими персонажами;

– користуватися магазинами (продавати та купувати спорядження);

– екіпірувати спорядження;

– вступати в бій з ігровими ворогами;

– користуватися покроковою системою бою за допомогою відповідного інтерфейсу;

– приймати ігрові завдання, виконувати їх, отримувати за них нагороди;

– підвищувати рівень свого персонажа;

– зберігати ігровий прогрес;

– завантажувати ігровий прогрес при авторизації на сайті.

При виконанні всіх описаних кроків для реалізації, а також реалізації всіх зазначених можливостей користувача, далі буде розроблено повноцінний додаток, а саме двовимірну рольову веб гру з покроковою системою бою.

1.4 Висновки до розділу 1

В розділі 1 було проведено порівняльний аналіз аналогових продуктів, в результаті чого обґрунтовано актуальність та новизну дослідження, а саме розробки двовимірної рольової веб гри з покроковою системою бою. Також було наведено аналіз існуючих засобів розробки браузерної графіки та веб ігор, та наведено аргументацію вибраних технологій для розробки. Окрім цього було визначено кроки до реалізації майбутнього додатку та описано функціональні вимоги та можливості користувача.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Проєктування структури застосунку

Після вибору технологій для реалізації застосунку а також визначення всіх функціональних вимог можна переходи до етапу проєктування додатку. Для цього буде використано UML-методологію, щоб розробити відповідні діаграми для подальшого полегшення та прискорення розробки, користуючись правилами та рекомендаціями до розробки UML-діаграм з книги “ UML 2.0 in Action” [3].

В першу чергу для візуалізації всіх можливостей користувача та для зручності необхідно створити діаграму прецедентів, використовуючи список з розділу 1. Діаграма прецедентів, зображена на рисунку 2.1, чітко показує, як саме користувач зможе взаємодіяти з майбутнім додатком, що зможе робити при його використанні.

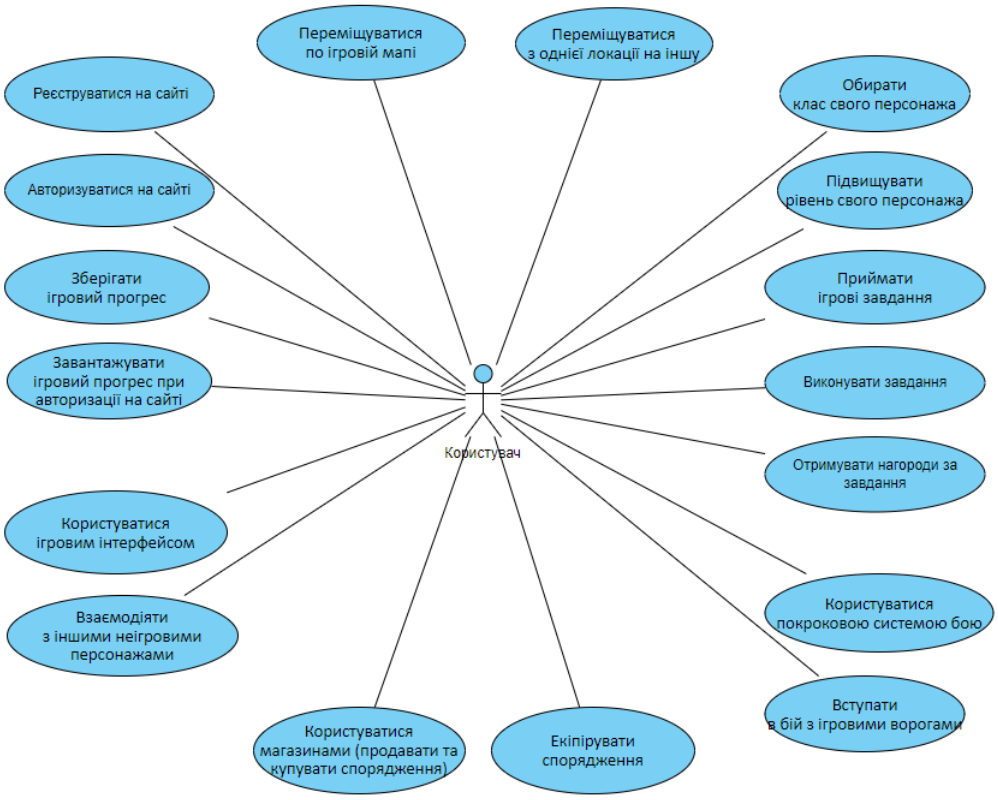


Рисунок 2.1 – Діаграма прецедентів

Після розробки діаграми прецедентів, доцільним буде визначити структуру додатку за допомогою діаграми компонентів. На рисунку 2.2 показано, які компоненти містить додаток та як вони взаємодіють між собою. Одним із головних елементів є клієнт, що використовує Vite для збірки проєкту, а також містить в собі вебсторінки, розроблені з використанням JavaScript, HTML та CSS. Оскільки основою додатку є вебгра, вона розміщується на одній зі сторінок. Гра розробляється з використанням PixiJs, Pixi/UI, Pixi/Sound та Pixi/Tilemap, а також потребує використання додаткових матеріалів, таких як зображення, аудіо, мапи та ігрові дані. Також додаток повинен мати реєстрацію, для чого додається серверна частина на NodeJS та Express, що використовує MongoDB для зберігання даних користувача.

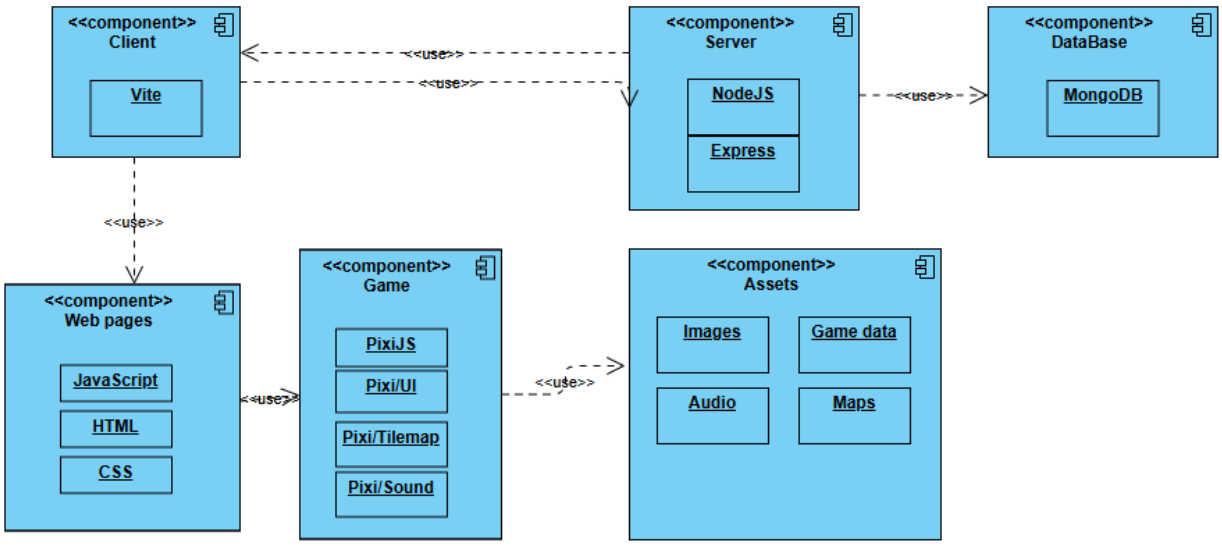


Рисунок 2.2 – Діаграма компонентів

Коли структуру проєкту було визначено, можна перейти до проєктування самої програми використовуючи UML-діаграму класів. На рисунку 2.3 зображено діаграму класів вебгри, що показує необхідні для функціонування додатку класи та зв’язки між ними, їх поля та методи. Серед них є такі важливі класи як гравець, мапа, ігрова логіка, вороги, неігрові персонажі, предмети, класи героя, ігровий бій та навіть менеджер квестів гравця.

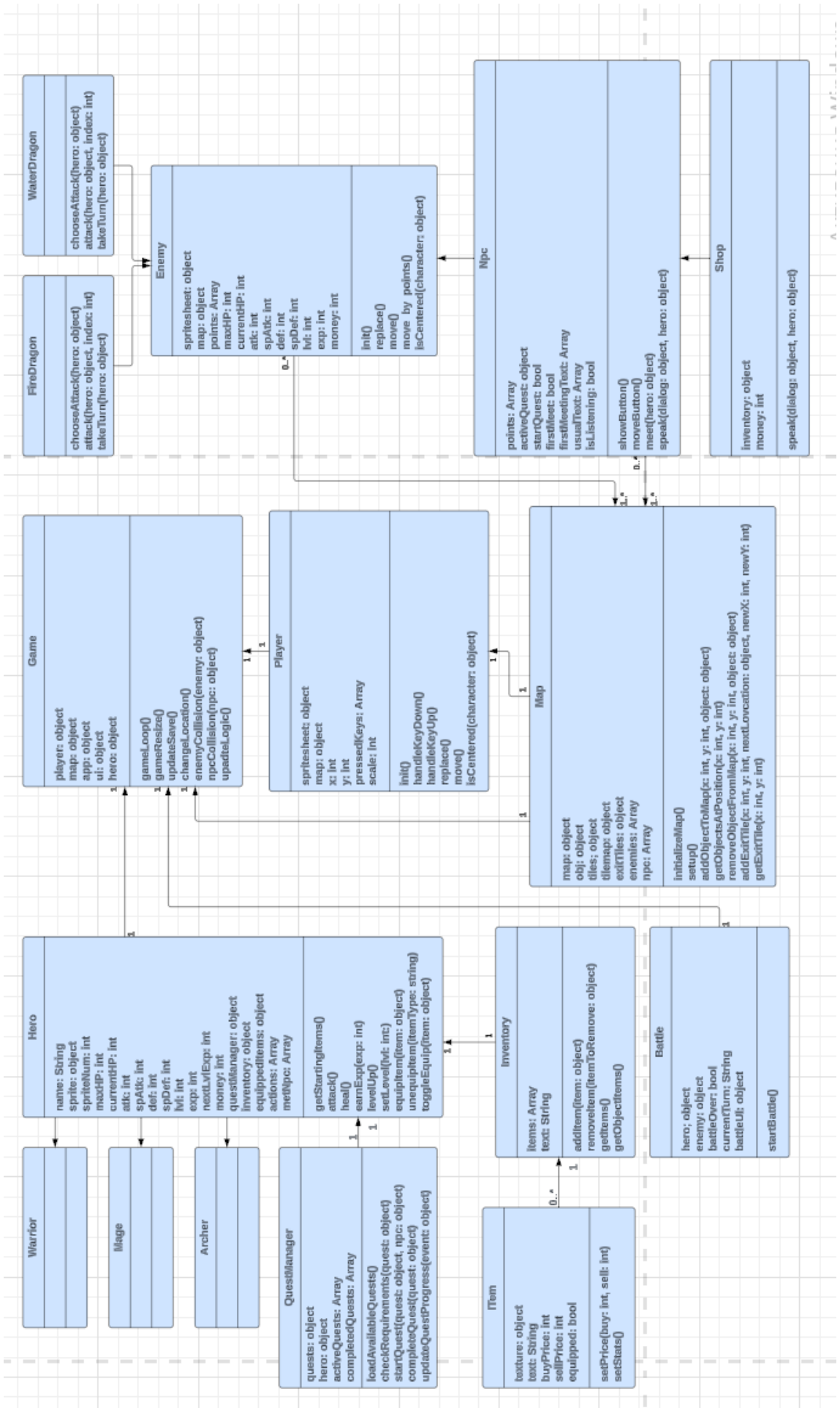


Рисунок 2.3 – Діаграма класів веб гри

Також, оскільки додаток використовує базу даних, потрібно визначити її схему, а саме які вона містить сутності та їх параметри. Схема бази даних показана на рисунку 2.4. Можна побачити, що база даних буде містити дві сутності – користувача та дані гри. Користувач має такі атрибути, як ім’я, електронна пошта та пароль. Дані гри в свою чергу містять ідентифікатор користувача, якому належить збережена гра, а також об’єкт з параметрами гри. Окрім цього на діаграмі видно, що сутності мають відношення «один до одного», оскільки один користувач може мати лише одну збережену гру, в той час як збережена гра може належати лише одному користувачу.

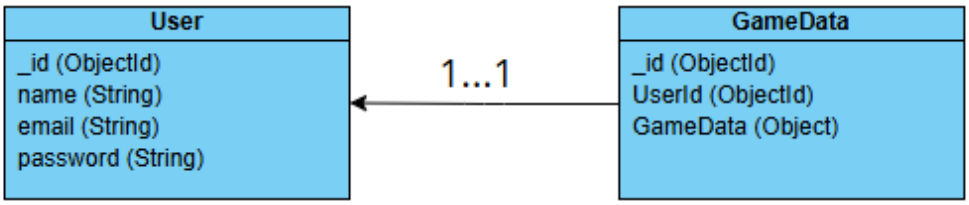


Рисунок 2.4 – Схема бази даних

Таким чином було розроблено діаграму прецедентів з врахуванням всіх можливостей користувача, діаграму компонентів, що демонструє взаємодію компонентів у додатку, діаграму класів, а також схему даних, що показує як дані зберігаються в базі даних.

2.2 Висновки до розділу 2

В розділі 2 було спроєктовано майбутній застосунок, для чого було розроблено необхідні UML-діаграми, такі як діаграма прецедентів, діаграма компонентів, діаграма класів, а також розроблено схему бази даних. До кожної з розроблених діаграм надано коротке зрозуміле пояснення щодо їх змісту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація гри

Після аналізу предметної області, обґрунтування актуальності розроблюваного додатку, вибору технологій та засобів розробки, створення списків вимог та можливостей користувача, проєктування застосунку та розробки необхідних діаграм, можна переходити до етапу програмної реалізації продукту.

В першу чергу проводиться налаштування проєкту, а саме створюється новий проєкт з використанням Vite, після чого встановлюються необхідні бібліотеки, такі як PixiJs, Pixi/UI, Pixi/Tilemap, Pixi/Sound. Під час дослідження використовується версія PixiJs 7.4.2, що є останньою оптимізованою та працюючою версією.

Спочатку, як показано в документації до PixiJs [6], необхідно ініціалізувати додаток та приєднати то HTML-елементу canvas, рисунок 3.1.

```
import * as PIXI from 'pixi.js';
const app = new PIXI.Application({ width: 640, height: 360});
document.body.appendChild(app.view);
```

Рисунок 3.1 – Ініціалізація додатку PixiJs

Далі для функціонування гри створюється клас GameLogic, що відповідає за оновлення зображення через певні проміжки часу та виклик необхідних функцій, які вимагають постійного оновлення, оскільки реагують на дії користувача. На рисунку 3.2 показано дві функції цього класу, одна з яких оновлює зображення та викликає функцію, що в свою чергу пізніше викликатиме інші. Друга функція змінює розмір додатку відповідно до розміру екрану таким чином, щоб розмір гри відповідав масштабу 16:9, що є класичним масштабом екранів та додатків. Також ця функція розраховує

значення для масштабування всіх ігрових об’єктів, оскільки якщо збільшується площина відображення гри, то й об’єкти повинні збільшуватися.

```
gameLoop() {
    this.updateLogic();
    this.app.renderer.render(this.app.stage);
    requestAnimationFrame(this.gameLoop.bind(this));
  }

  gameResize() {
    if (window.innerWidth / window.innerHeight > 16 / 9) {
       this.app.renderer.resize(16 * window.innerHeight / 9, window.innerHeight);
    } else {
      this.app.renderer.resize(window.innerWidth, 9 * window.innerWidth / 16);
    }
    const scaleX = Math.round(this.app.renderer.width / 360);
    Utils.setScale(scaleX, this.app.renderer.width, this.app.renderer.height);
  }
```

Рисунок 3.2 – Функції оновлення та масштабування гри

Також важливо зазначити, що основою всієї гри є ресурси, в особливості графічні та аудіо матеріали. Для їх використання спочатку необхідно їх завантажити, що відбувається в окремому файлі, приклад показано на рисунку 3.3 за інструкціями з відповідних документацій [6, 9].

```
import * as PIXI from 'pixi.js';
import { sound } from '@pixi/sound';

export const tiles = await PIXI.Assets.load('./assets/img/roguelikeSheet_transparent.png');
export const start = await PIXI.Assets.load('./assets/maps/start.json');
export const sea = await PIXI.Assets.load('./assets/maps/sea.json');
sound.add('ambient','./assets/sounds/ambient.mp3');
sound.add('openMenu','./assets/sounds/openMenu.mp3');
```

Рисунок 3.3 – Завантаження ресурсів

Одним з перших завдань є розробка ігрової мапи, по якій буде переміщуватися гравець. Для створення графічної частини мапи з набору «тайлів» (клітин для мапи) використовується редактор «Tiled», основи роботи з яким описано в книзі «The Advanced Game Developer’s Toolkit» [4]. В

результаті можна отримати графічне зображення готової мапи в бажаному форматі, а також JSON-файл, в якому будуть описані номери використаних тайлів, шари мапи, а також її розміри та інші параметри, що є необхідним для подальшого завантаження та обробки мапи. Приклад роботи редактора показано на рисунку 3.4.

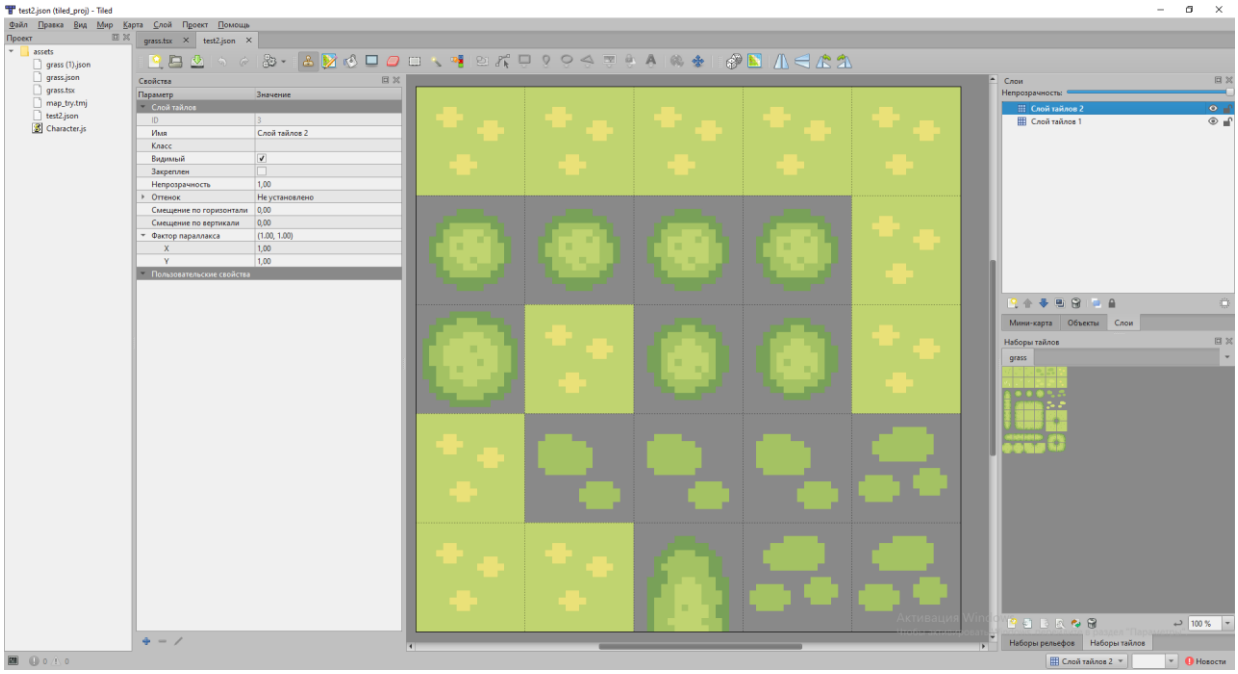


Рисунок 3.4 – Робота з редактором «Tiled»

Для роботи з мапою розробляється клас мапи з усіма необхідними функціями [5]. Для використання отриманого JSON-файлу потрібно створити відповідну функцію, що буде розбирати отримані дані для подальшого використання. На рисунку 3.5 наведено функцію створення мапи на основі даних з редактору мап. За допомогою циклу функція проходить по всіх шарах мапи, для кожного з яких створює новий шар в об’єкті, де зберігається готова мапа (для цього використовується бібліотека Pixi/Tilemap [7]). Після цього для кожного шару функція зчитує по черзі всі номери тайлів, які необхідно завантажити з зображення з набором тайлів. Використовуючи номер, інша функція розраховує його позицію на зображенні, завдяки якій створюється нова текстура та додається в мапу як нова клітина. Для другого шару мапи створюються об’єкти, оскільки надалі вони будуть фізичними об’єктами крізь

які неможливо пройти, такі як стіни або інші перешкоди. Також важливим є момент масштабування кожної нової текстури, як це показано на рисунку 3.6, для того щоб при будь якому розмірі екрану зображення не було розпливчастим. Для цього використовується метод масштабування «найближчий сусід», що є швидким та зручним.

```
        for (let layerIndex = 0; layerIndex < tilemap.layers.length; layerIndex++) {
            const layerData = tilemap.layers[layerIndex].data;
            const layer = new CompositeRectTileLayer(0, []);
            this.map.addChild(layer);

            for (let row = 0; row < tilemap.height; row++) {
                for (let col = 0; col < tilemap.width; col++) {
                    const tileId = layerData[row * tilemap.width + col];

                    if (tileId !== 0) {
                        const coordinates = Utils.getTileCoordinates(tileId, 16, 968);
                        const tileTexture = new PIXI.Texture(tiles,
Utils.calculateTileRectangle(coordinates, 16, 16, 1));
                        Utils.scaleTexture(tileTexture);
                        const x = col * this.map.tileWidth;
                        const y = row * this.map.tileHeight;

                        layer.tile(tileTexture, x, y);
                        layer.scale.set(Utils.scale);
                        //second layer is objects which are obstacles
                        //player cant walk over objects
                        if (layerIndex === 1) {
                            const object = new PIXI.Sprite(tileTexture);
                            object.x = x;
                            object.y = y;
                            object.scale.set(Utils.scale);
                            object.zIndex = 1;
                          this.addObjectToMap(object.x * Utils.scale, object.y * Utils.scale, object);
                        }
                    }
                }
            }
        }
```

Рисунок 3.5 – Створення мапи

```
        function scaleTexture(texture){
            texture.baseTexture.scaleMode = PIXI.SCALE_MODES.NEAREST;
        }
```

Рисунок 3.6 – Масштабування

Також одним з найважливіших елементів гри є персонаж, яким гравець буде керувати. Для його створення в першу чергу необхідно створити файл з об’єктом, де будуть зазначені майбутні анімації персонажа, тобто зазначено позицію та розмір кожного зображення для анімації – спрайтів, а також їх порядок. Приклад опису анімації руху вправо та вліво персонажа показано на рисунку 3.7. Також на рисунку 3.8 наведено приклад графічного зображення всіх спрайтів персонажа, що використовуються в розробці.

```
const character = {
  frames: {
     walkleft1: { frame: { x: 48, y: 1, w: 16, h: 16 } },
     walkleft2: { frame: { x: 48, y: 18, w: 16, h: 16 } },
     walkleft3: { frame: { x: 48, y: 35, w: 16, h: 16 } },

     walkright1: { frame: { x: 16, y: 1, w: 16, h: 16 } },
     walkright2: { frame: { x: 16, y: 18, w: 16, h: 16 } },
     walkright3: { frame: { x: 16, y: 35, w: 16, h: 16 } }
   },
  animations: {
   walk_left: ['walkleft1', 'walkleft2', 'walkleft1', 'walkleft3'],
   walk_right: ['walkright1', 'walkright2', 'walkright1', 'walkright3']
  },
  meta: {
   image: './assets/F_01.png',
   size: { w: 64, h: 51 },
   scale: 1
  }
};
```

Рисунок 3.7 – Об’єкт з анімаціями персонажа



Рисунок 3.8 – Спрайти персонажа

Далі можна перейти до створення класу персонажа, що буде відповідати за його анімації та переміщення на мапі, а також контролювати сутички з

іншими об’єктами. На рисунку 3.9 наведено частину функції переміщення персонажа. При натисканні клавіші зображення персонажа повинно переміщуватися у відповідну сторону, а також програється відповідна анімація.

```
        move() {

        if(this.scale != Utils.scale) this.replace();
            if(this.pressedKeys.length !== 0 && this.isCentered(this.map.map)) this.lastKeyPressed = this.pressedKeys[this.pressedKeys.length - 1];
            if(this.pressedKeys.length!==0 || !this.isCentered(this.map.map)){
              if (this.lastKeyPressed === 'ArrowUp') {
                if(this.player.textures !== this.spritesheet.animations.walk_up) this.player.textures = this.spritesheet.animations.walk_up;
                if(this.map.getObjectsAtPosition(this.x, this.y-16*Utils.scale).length===0 || !this.isCentered(this.map.map)) {this.map.map.y += Utils.scale/2; this.y -= Utils.scale/2;}
              } else if (this.lastKeyPressed === 'ArrowDown') {
                if(this.player.textures !== this.spritesheet.animations.walk_down) this.player.textures = this.spritesheet.animations.walk_down;
                if(this.map.getObjectsAtPosition(this.x, this.y+16*Utils.scale).length===0 || !this.isCentered(this.map.map)) {this.map.map.y -= Utils.scale/2; this.y +=Utils.scale/2;}
              }

            this.player.play();
            }

            else if(this.pressedKeys.length===0){
              if (this.lastKeyPressed === 'ArrowUp') {
                if(this.player.textures !== this.spritesheet.animations.stand_up) this.player.textures = this.spritesheet.animations.stand_up;
              } else if (this.lastKeyPressed === 'ArrowDown') {
                if(this.player.textures !== this.spritesheet.animations.stand_down) this.player.textures = this.spritesheet.animations.stand_down;
              }

              this.player.play();
              }
            }
```

Рисунок 3.9 – Функція переміщення персонажа

Однак варто зазначити, що, по перше, переміщення відбувається по клітинках, тобто на 16 пікселів за раз, і якщо гравець відпускає клавішу раніше, ніж персонаж досягне середини наступної клітинки, програма автоматично пересуне персонажа на потрібне місце, а після зупинки буде

програватися анімація персонажа, що стоїть на місці та дивиться в потрібний бік. По друге, для кращого візуального ефекту, як і прийнято в більшості ігор, переміщується не сам персонаж, а мапа поза ним – таким чином персонаж залишається в центрі екрану та створюється ефект переміщення. Також програма перевіряє, чи немає на шляху інших об’єктів, та якщо вони є – персонаж стоїть на місці, однак програється анімація руху, щоб гравець розумів що не може рухатися в цей бік. Ця функція є однією з тих, що вимагають постійного виклику разом з оновленням зображення, оскільки вона залежить від дій користувача.

Також, враховуючи раніше описану функцію зміни масштабу всіх ігрових обєктів, необхідно зробити це для персонажа. Ця функція змінює його розмір відповідно до розрахованого раніше масштабу, а також змінює позицію персонажа, враховуючи масштаб мапи, рисунок 3.10.

```
replace(){
    this.player.scale.set(Utils.scale);
    //this.player.animationSpeed = 0.02 * Utils.scale;
    this.player.x = (this.player.x/this.scale) * Utils.scale;
    this.player.y = (this.player.y/this.scale) * Utils.scale;
    this.map.map.x = (this.map.map.x/this.scale) * Utils.scale;
    this.map.map.y = (this.map.map.y/this.scale) * Utils.scale;
    this.x = (this.x/this.scale) * Utils.scale;
    this.y = (this.y/this.scale) * Utils.scale;
    this.scale = Utils.scale;
  }
```

Рисунок 3.10 – Масштабування персонажа

Оскільки тепер програма має ігрові мапи, а персонажем можна керувати й переміщувати його, необхідно надати гравцю можливість переходити між різними локаціями. Для цього для кожної мапи створено певні клітини виходу з локації, в файлі з даними про всі мапи позначено координати таких клітин, на яку мапу вони ведуть та на яку клітину нової мапи вийде гравець (приклад даних показано на рис. 3.11). В класі GameLogic описаному вище регулярно викликається функція для зміни локації, що перевіряє позицію гравця та за необхідності переміщує його та замінює поточну мапу на нову, рисунок 3.12.

```
"exitTiles": [
        {
          "x": 0,
          "y": 96,
          "toMap": "castle",
          "targetX": 448,
          "targetY": 144
        },
        {
          "x": 464,
          "y": 128,
          "toMap": "dragonLand",
          "targetX": 16,
          "targetY": 144
        }]
```

Рисунок 3.11 – Збереження клітин виходу з локації

```
//when step on exit tile switch to another location
   changeLocation(){
     var exit = this.map.getExitTile(this.player.x/Utils.scale, this.player.y/Utils.scale);
     if(exit.length != 0){
       //remove current map from screen
       this.map.map.x = 0;
       this.map.map.y = 0;
       this.map.map.visible = false;
       this.map.enemies.forEach(enemy => {
         enemy.enemy.visible = false;
       });

       //set new current map, update positions for everything
       this.map = exit[0];
       this.player.map = exit[0];

       this.player.map.map.x -= (exit[1]-this.player.player.x/Utils.scale)*Utils.scale;
       this.player.map.map.y -= (exit[2]-this.player.player.y/Utils.scale)*Utils.scale;
       this.player.x = exit[1]*Utils.scale;
       this.player.y = exit[2]*Utils.scale;
       this.map.map.visible = true;
       this.map.enemies.forEach(enemy => {
         enemy.enemy.visible = true;
         enemy.init();
       });
     }
   }
```

Рисунок 3.12 – Функція зміни поточної мапи

Для ігрової системи важливо створити для гравця класи персонажа з різними характеристиками та особливостями. В цьому додатку створюється основний батьківський клас героя та похідні від нього класи – воїн, маг та

лучник. Батьківський клас героя має базові характеристики, а також функції для підвищення рівня – коли гравець отримує ігровий досвід, гра перевіряє чи достатньо його для наступного рівня (рис. 3.13). Якщо це так, то рівень збільшується, надлишковий досвід залишається, характеристики підвищуються разом із кількістю досвіду необхідного для наступного рівня.

```
    earnExp(exp){
        this.exp += exp;
        this.levelUp();
    }

    levelUp(){
        //if enough exp increase lvl, stats and heal
        while(this.exp >= this.nextLvlExp){
            this.lvl += 1;
            this.exp -= this.nextLvlExp;

            this.maxHP += 10;
            this.currentHP = this.maxHP;
            this.atk += 2;
            this.spAtk += 3;
            this.def += 1;
            this.spDef += 1;
        }
        //set new exp requirement
        this.nextLvlExp = this.lvl * 100;
    }
```

Рисунок 3.13 – Функції підвищення рівня героя

Також для героя створюється клас інвентарю, що може зберігати набори предметів, для яких також створено окремий клас. Загальний клас предмету містить свою текстуру, ціну покупки й продажу. Для предметів також створено кілька похідних класів, таких як броня, зброя або лікувальний предмет. Ці класи мають власні текстури, а також ефекти, такі як підвищення здоров’я, атаки або захисту героя. В батьківському класі героя створюється інвентар гравця, а також функціонал для екіпірування різного спорядження, рисунок 3.14. Важливим аспектом є те, що гравець може одночасно екіпірувати лише один предмет одного типу, тобто наприклад герой не може одночасно використовувати дві зброї, тощо.

```
//change equip state of item
  //one item of same category can be equipped in same time
  toggleEquip(item) {
    const currentItemType = item.constructor.name.toLowerCase();
    const previouslyEquippedItem = this.equippedItems[currentItemType];

    if (previouslyEquippedItem === item) {
      this.unequipItem(currentItemType);
      return previouslyEquippedItem;
    } else {
      if (previouslyEquippedItem) {
        this.unequipItem(currentItemType);
      }
      this.equipItem(item);
      return previouslyEquippedItem;
    }
  }
```

Рисунок 3.14 – Екіпірування предметів

Наступник кроком в переліку функціональних вимог є розробка графічного інтерфейсу користувача, для чого буде використано Pixi/UI [8]. На цьому етапу буде створено такі елементи інтерфейсу як шкала здоров’я та досвіду на головному екрані, панель персонажа з показниками героя та його інвентарем, журнал з активними та завершеними квестами (завданнями), а також інтерфейс ігрового магазину.

Для інтерфейсу головного екрану створюється елемент шкали здоров’я, що складається з простої графіки в PixiJS та малює заповнення шкали базуючись на максимальному та поточному рівні здоров’я героя. Приклад створення графічного контейнеру показано на рисунку 3.15.

```
this.container = new PIXI.Container();
    this.container.position.set(x, y);

    //grey bg
    this.background = new PIXI.Graphics();
    this.background.beginFill(0x999999);
    this.background.drawRoundedRect(0, 0, width, height, 10);
    this.background.endFill();
    this.container.addChild(this.background);
```

Рисунок 3.15 – Приклад створення графічного контейнеру

Клас інтерфейсу головного екрану використовує елемент шкали здоров’я для створення відповідної шкали здоров’я героя, а також шкали заповнення досвіду до наступного рівня. Окрім цього на екрані гри простим текстом відображається кількість грошей гравця.

Для розробки більшості елементів інтерфейсу користувача використовуються кілька загальних функцій, серед яких створення кнопки, створення тексту, додавання тексту або іконки на кнопку (рис. 3.16), створення фону для контейнера (рис. 3.17), створення набору кнопок та інше.

```
export function createItemButton(text, itemTexture, x, y, width, height) {
    const itemButton = createButton();
    itemButton.scale_button(width/3, height/8);
    itemButton.button.view.x = x;
    itemButton.button.view.y = y;
    itemButton.addText(text);
    itemButton.addIcon(itemTexture);

    return itemButton;
}
```

Рисунок 3.16 – Створення кнопки з текстом та іконкою

```
export function createBackground(width = Utils.width-200, height = Utils.height - 200)
{
    const bg_texture = new PIXI.Texture(Assets.menu);
    Utils.scaleTexture(bg_texture);
    const bg_sprite = new PIXI.Sprite(bg_texture);
    bg_sprite.width = width;
    bg_sprite.height = height;
    return bg_sprite;
}
```

Рисунок 3.17 – Створення заднього фону для контейнера

З використанням подібних елементів розробляється панель персонажа з його інвентарем, журнал гравця з квестами, інтерфейс магазину, тощо.

Окрім всього, дуже важливим елементом інтерфейсу є панель діалогу або ж текстове вікно з повідомленнями, що буде відображатися під час розмови з іншими персонажами та під час бою відтворюватиме необхідну інформацію. Особливістю такого діалогового вікна є поступове виведення

тексту, літера за літерою, оскільки саме такий підхід є притаманним багатьом рольовим іграм, включаючи старі. На рисунку 3.18 показано алгоритм розбиття тексту на слова та окремі літери для поступового відображення через невеликі проміжки часу, а також підрахунку та перенесення слів на новий рядок, якщо текст не вміщується. Важливо зазначити що при діалогах такі вікна також слугують кнопками, що при натисканні відтворюють наступну репліку персонажа або ж завершують діалог, якщо реплік більше немає.

```
        //show letters one by one
            const addLetter = (word, wordIndex) => {
                return new Promise((resolve) => {
                    const addLetterRecursive = () => {
                        if (index < word.length) {
                            ButtonText.text += word[index];
                            index++;
                            setTimeout(addLetterRecursive, 50);
                        } else {
                            ButtonText.text += ' ';
                            resolve(); //end when full word is shown
                        }
                    };
                    addLetterRecursive();
                });
            };

            //add word by word
            const addWord = async (word, wordIndex) => {
                //if line cant fit into the box
                if ((currentLine.length + word.length) * 10 > style.wordWrapWidth) {
                    ButtonText.text += '\n'; //new line
                    currentLine = '';
                }
                currentLine += word + ' ';
                index = 0;
                  await addLetter(word, wordIndex); //wait for the word to be printed letter by
letter
            };

            const words = text.split(' '); //split the text into words

            for (let i = 0; i < words.length; i++) {
                await addWord(words[i], i); //wait for each word to be printed before continuing
            }
```

Рисунок 3.18 – Алгоритм поступового виводу тексту

Далі необхідним кроком є створення ігрових ворогів, з якими гравець може зіткнутися та почати бій. Клас ворогів є схожим до класів героя та гравця разом, оскільки містить одночасно функції для анімації та переміщення, а також параметри ворога, такі як здоров’я, атака, захист та інше. Вороги рухаються по заздалегідь заданим точкам на мапі, переміщуючись від однієї до іншої по порядку, після чого повертаються на першу точку. Функція переміщення за точками наведена на рисунку 3.19.

```
move_by_points(){
    //move enemy along predefined points
                if((this.points[this.nextPoint].x*Utils.scale !== this.x || this.points[this.nextPoint].y*Utils.scale !== this.y)){
                    if(this.points[this.nextPoint].x*Utils.scale > this.x) {this.enemy.x += Utils.scale/2; this.x += Utils.scale/2;}
                    if(this.points[this.nextPoint].x*Utils.scale < this.x) {this.enemy.x -= Utils.scale/2; this.x -= Utils.scale/2;}
                    if(this.points[this.nextPoint].y*Utils.scale > this.y) {this.enemy.y += Utils.scale/2; this.y += Utils.scale/2;}
                    if(this.points[this.nextPoint].y*Utils.scale < this.y) {this.enemy.y -= Utils.scale/2; this.y -= Utils.scale/2;}
                }
                else {
                    if(this.nextPoint<this.points.length-1) this.nextPoint++;
                    else this.nextPoint = 0;
                }
            }
```

Рисунок 3.19 – Переміщення ворогів по точкам

Ігрові вороги також мають кілька похідних класів, що відрізняють їх за типами. Різні типи ворогів мають різні характеристики, а також різні атаки, які вони використовують під час бою.

Перед тим як перейти до розробки систему бою, необхідним кроком є впровадження функції зіткнення гравця з ворогами. Принцип роботи такої функції є досить простим – програма підраховує відстань між положенням гравця та ворога, та якщо вони знаходяться на відстані однієї клітини починається ігровий бій, під час якого решта гри призупиняється для збереження ресурсів комп’ютера та запобігання несподіваним ситуаціям.

Після того як гравець зіткнувся з ворогом починається бій та відкривається відповідний елемент інтерфейсу, що повністю закриває ігровий

екран. Інтерфейс складається зі списку кнопок дій гравця (різні види атак та лікування), діалогового вікна для повідомлень про атаки та інші дії, а також спрайтів гравця й ворога з двома шкалами здоров’я.

Система бою є відносно простою та представляє собою цикл, в якому гравець й ворог по черзі атакують один одного. Гра чекає на дію гравця, підраховує нові значення для здоров’я після атаки, після чого передає хід ворогу. Дії ворога визначає функція атаки у відповідному класі, у якій обирається випадкова атака або ж найсильніша, якщо в гравця мало здоров’я. Після кожної атаки виводиться відповідне повідомлення про тип атаки та завдану шкоду, а також програється звук удару та невелика анімація переміщення спрайту вперед та назад (рис. 3.20).

```
animateSprite(sprite, distance = 10, duration = 400) {
    const startX = sprite.x;
    const halfDuration = duration / 2;
    let elapsed = 0;
    let direction = 1;
    function update(delta) {
      elapsed += delta * (1000 / 60);
      if (elapsed <= halfDuration) {
        sprite.x = startX + direction * distance * (elapsed / halfDuration);
      } else {
        direction *= -1;
        elapsed = 0;
      }

      if (direction === -1 && elapsed >= halfDuration) {
        sprite.x = startX;
        container.ticker.remove(update);
      }
    }

    const container = this.container;
    container.ticker = container.ticker || new PIXI.Ticker();
    container.ticker.add(update);
    container.ticker.start();
  }
```

Рисунок 3.20 – Анімація з використанням PixiJs

Після кожної атаки в циклі перевіряється визначення переможця – якщо в героя чи ворога здоров’я дорівнює нулю бій завершується, а на екран

виводиться повідомлення про перемогу чи поразку та програється відповідний звуковий ефект. Якщо гравець перемагає, він отримує нагороду зазначену в класі ворога – певну кількість грошей та досвіду, якщо гравець програє то завантажується остання збережена гра (ці функції будуть пояснені пізніше).

Також необхідною механікою будь-якої рольової гри є система квестів (ігрових завдань). Для впровадження такої системи спершу створюється JSON-файл з даними про всі квести – їх назви, завдання, нагороди, персонажа що видає квест, а також умови розблокування, наприклад які завдання потрібно завершити перед цим та необхідний рівень персонажа.

В самій програмі для кожного героя створюється менеджер квестів, що зберігає дані про всі доступні, активні та виконані квести, перевіряє виконання вимог завдання, змінює статус квестів. На рисунку 3.21 показано функцію, що перевіряє виконання вимог для розблокування квесту.

```
checkRequirements(quest){
        const unlockConditions = quest.unlockConditions;

        const isRequiredQuestCompleted = unlockConditions.requiredQuest
            ? this.completedQuests.some(completedQuest => completedQuest.id === unlockConditions.requiredQuest)
            : true;
        if (this.hero.lvl >= unlockConditions.playerLevel && isRequiredQuestCompleted) {
            return true;
        } else {
            return false;
        }
        ;
    }
```

Рисунок 3.21 – Перевірка вимог для розблокування квесту

Наступним кроком є додавання неігрових персонажів до гри. Як і зазначено на діаграмі класів, клас Npc наслідує клас Enemy, тобто за своєю суттю є різновидом ворогів. Такий підхід використовується для зручності розміщення неігрових персонажів на мапі. Вони мають схожу функцію зчитування сутичок із гравцем, однак при цьому не починається бій, а

відображається позначка з літерою «E» над головою персонажа, що означає кнопку, при натисканні якої розпочнеться діалог. Також при сутичці з неігровим персонажем спочатку перевіряється список доступних квестів, які цей персонаж може видати гравцю. На рисунку 3.22 показано функцію, що визначає репліки неігрового персонажа враховуючи наявні квести та чи вперше гравець зустрічає цього персонажа.

```
return new Promise((resolve) => {
    let text = [];
    if(this.activeQuest){
        if(this.activeQuest.status == "inProgress" && this.startQuest){
            if (this.firstMeet && !hero.metNpc.includes(this.name)) {
                text = [...this.firstMeetingText, ...this.activeQuest.text.start];
                this.meet(hero);
            } else {
                text = this.activeQuest.text.start;
            }
            this.startQuest = false;
        }
        else if(this.activeQuest.status == "inProgress")
            text = this.activeQuest.text.progress;
        else if(this.activeQuest.status == "completed"){
            text = this.activeQuest.text.complete;
            this.activeQuest = null;
        }
    }
    else {
        if (this.firstMeet && !hero.metNpc.includes(this.name)) {
            text = this.firstMeetingText;
            this.meet(hero);
        } else{
            text = this.usualText;
        }
    }
});
```

Рисунок 3.22 – Визначення реплік неігрового персонажа

Таким самим чином клас Shop наслідує клас Npc, тобто деякі неігрові персонажі представляють собою торговців (магазини) та окрім іншого виконують необхідні функції. Після завершення діалогу з торговцем як зі звичайним неігровим персонажем відкривається вікно торгівлі, де гравець може придбати або продати спорядження. Важливим аспектом також є те, що

гроші в торговця не є нескінченними як в багатьох іграх, а тому гравець не зможе продати більше, аніж торговець зможе в нього купити. Якщо в гравця або в торговця недостатньо грошей для проведення операції їхній грошовий баланс підсвічується червоним, вказуючи гравцю на помилку. Після покупки предмет переміщується з інвентаря торговця до гравця та навпаки. Функція покупки предмета наведена на рисунку 3.23.

```
async  buyItem(item) {
        //remove item from shop
        if(this.hero.money>=item.buyPrice){
            Assets.sound.play('money');
            this.shop.inventory.removeItem(item);
            this.hero.inventory.addItem(item);

            this.hero.money -= item.buyPrice;
            this.shop.money += item.buyPrice;
        }
        else{
            await this.flashMoneyText(this.heroMoneyText);
        }
        this.updateUI();
    }
```

Рисунок 3.23 – Функція покупки предмета

Коли основна частина гри з усіма головними робочими механіками розроблена, потрібно створити панель створення ігрового персонажа для самого початку нової гри. Така панель дозволить новим гравцям створити свого героя – ввести ім’я, обрати зовнішній вигляд з наявних варіантів, а також обрати один із трьох наявних класів героя, після чого почати нову гру.

Сама панель створюється аналогічно до попередніх елементів інтерфейсу та складається з текстового поля вводу, кнопок, текстових елементів й спрайтів персонажів для вибору. До панелі прив’язані функції, що після кожної дії гравця – вибору персонажа, класу чи написання імені, – зберігають отримані дані та після підтвердження форми надсилають їх до основної функції початку нової гри. На рисунку 3.24 показана частина функції, що створює нового персонажа, базуючись на даних переданих з форми створення героя.

```
const create = new CreateCharacter();
        app.stage.addChild(create.container);
        await create.create();
        app.stage.removeChild(create.container);
        if(create.currentClass === 0){
            hero = new Warrior(create.name.value, create.characterSprite);
        }
        else if(create.currentClass === 1){
            hero = new Mage(create.name.value, create.characterSprite);
        }
        else if(create.currentClass === 2){
            hero = new Archer(create.name.value, create.characterSprite);
        }
        hero.spriteNum = create.currentCharacter;
        hero.getStartingItems();
```

Рисунок 3.24 – Створення нового персонажа

Після розробки всіх вищезазначених функцій можна перейти до створення серверної частини, що відповідатиме за реєстрацію та авторизацію користувачів, а також збереження та завантаження гри з бази даних.

В першу чергу створюється додаток з використанням Express для створення роутів та відбувається підключення до MongoDB з використанням mongoose [10]. Після цього створюються схеми користувача та ігрових даних для збереження в БД (рис. 3.25).

```
const UserSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    date: { type: Date, default: Date.now }
});
const GameSaveSchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    gameData: { type: Object, required: true },
    date: { type: Date, default: Date.now }
});
```

Рисунок 3.25 – Схеми користувача та ігрових даних

Далі необхідно розробити відповідні роути для всіх вищезазначених функцій. Функції реєстрації та авторизації є простими та їх реалізація наведена в додатку А. Для реєстрації при підтвердженні форми програма перевіряє, чи

існує користувач з такою електронною адресою, якщо ні – створює нового користувача в базі даних та шифрує пароль за допомогою bcrypt. Логін в свою чергу також перевіряє існування користувача за електронною адресою, після чого порівнює введений пароль з паролем з БД. В разі успіху дані авторизованого користувача записуються в local storage та гравець перенаправляється на сторінку початку гри або завантаження збережених даних.

Щоб зберегти ігрові дані в першу чергу їх необхідно створити. Для цього клас GameLogic описаний раніше регулярно зберігає дані гравця всередині програми, а при натисканні кнопки збереження гри посилається відповідний запит на сервер. Оскільки гравець може мати лише одну збережену гру програма перевіряє, чи існують ігрові дані для поточного користувача за його ідентифікатором, і залежно від результату замінює або створює нові ігрові дані, як показано на рисунку 3.26.

```
app.post('/save-game', async (req, res) => {
    if (!req.session.user) {
        return res.status(401).json({ message: 'Unauthorized' });
    }
    const { gameData } = req.body;
    try {
        const existingSave = await GameSave.findOne({ userId: req.session.user.id });
        if (existingSave) {
            existingSave.gameData = gameData;
            await existingSave.save();
        } else {
            const newSave = new GameSave({
                userId: req.session.user.id,
                gameData
            });
            await newSave.save();}
        res.json({ message: 'Game saved successfully' });
    } catch (err) {
        res.status(500).json({ message: 'Failed to save game', error: err.message });
    }
});
```

Рисунок 3.26 – Збереження ігрових даних

Функція завантаження гри працює аналогічним чином, однак замість запису нових даних до БД вона знаходить їх та повертає до програми, після

чого передає у функцію створення нової гри. Ця функція залежно від наявності переданих даних гри або відтворює збережену гру використовуючи ці дані, або за їх відсутності відкриває панель створення персонажа та починає нову гру.

На рисунках 3.27–3.33 наведено приклади роботи додатку з відповідними пояснюючими підписами.

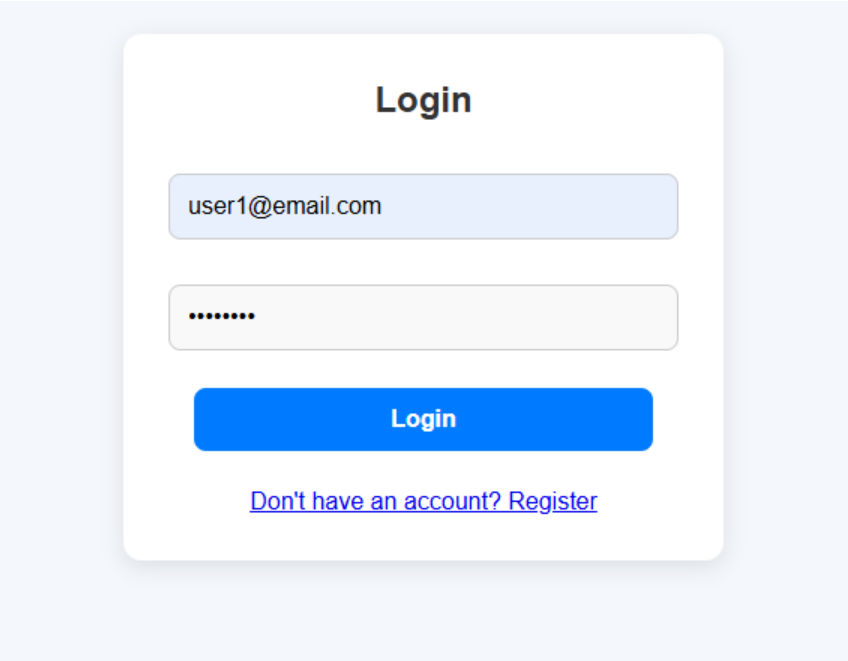


Рисунок 3.27 – Сторінка авторизації користувача

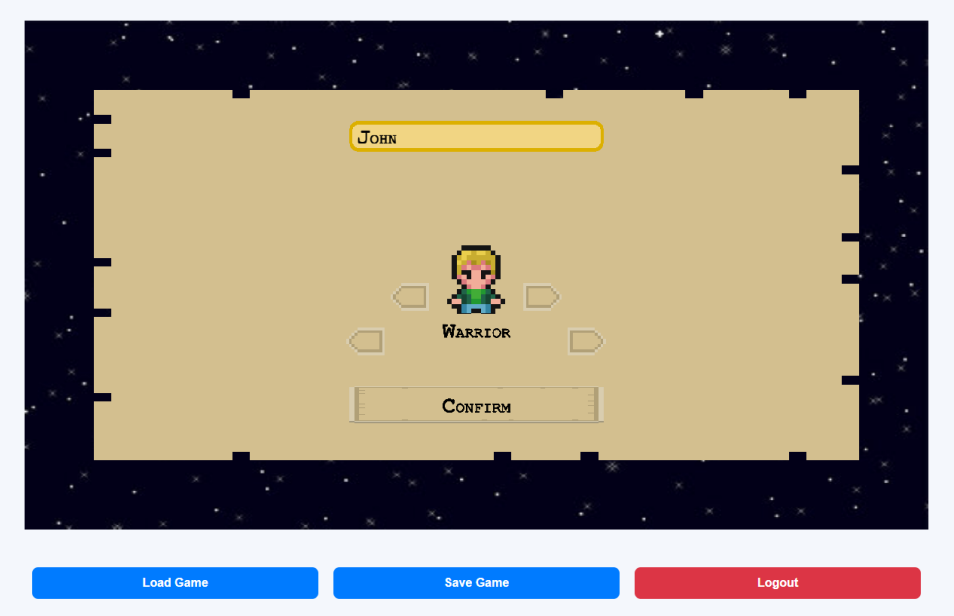


Рисунок 3.28 – Панель створення нового персонажа та ігрова сторінка



Рисунок 3.29 – Приклад діалогу з неігровим персонажем



Рисунок 3.30 – Приклад іншої локації та наявності ворогів



Рисунок 3.31 – Інтерфейс ігрового бою

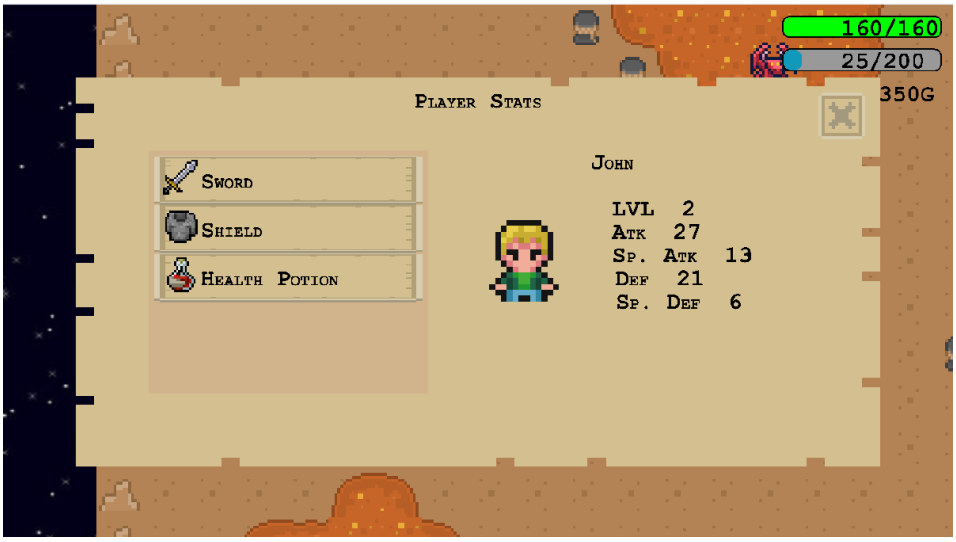


Рисунок 3.32 – Характеристики персонажа та інвентар



Рисунок 3.33 – Інтерфейс ігрового магазину

Таким чином в програмі було реалізовано всі функціональні вимоги до веб гри та створено всі необхідні функції для повноцінної роботи додатку, а також показано приклади роботи програми.

3.2 Висновки до розділу 3

В розділі 3 було детально описано програмну реалізацію рольової вебгри з використанням PixiJs та Vite. Були пояснені основні функції гри, їх

принципи роботи, показано реалізацію ігрової логіки, графічного інтерфейсу користувача, а також серверної частини для роботи з базою даних. В кінці було проведено тестування роботи додатку та наведено зображення з демонстрацією зовнішнього вигляду веб гри.

ВИСНОВКИ

В процесі роботи було спроєктовано та розроблено двовимірну рольову покрокову веб гру з використанням PixiJs та Vite, наведено відповідні діаграми, описано процес розробки та принципи роботи основних функцій.

Спочатку було проведено порівняльний аналіз існуючих аналогових продуктів та обґрунтовано актуальність розробки двовимірної рольової вебгри з покроковою системою бою для одного гравця. Після цього було проведено огляд та порівняння найбільш розповсюджених засобів розробки веб ігор та браузерної графіки, а також обґрунтовано вибір засобів для розробки серверної частини гри. Окрім цього було описано функціональні вимоги та можливості користувача, які необхідно реалізувати в процесі дослідження.

Після цього було спроєктовано майбутній застосунок, для чого було розроблено необхідні UML-діаграми, такі як діаграма прецедентів, діаграма компонентів, діаграма класів, а також розроблено схему бази даних, коротке пояснення до яких наведено в розділі 2.

Останнім кроком була програмна реалізація всіх складових програми з використанням PixiJs та Vite та її детальне описання. В розділі 3 було реалізовано всі функціональні вимоги та створено застосунок відповідно до попередньо розроблених діаграм. Також було описано основні функції та принципи їх роботи, розказано про логіку роботи веб гри та її алгоритмів. В кінці розроблений застосунок було протестовано та показано приклади роботи.

Таким чином в результаті дослідження було розроблено двовимірну рольову веб гру з покроковою системою бою та піксельною графікою з використанням PixiJs та Vite. Було спроєктовано застосунок, наведено відповідні діаграми та описано програмну реалізацію основних функцій вебдодатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Van Der Spuy R. Learn Pixi.js Create Great Interactive Graphics for Games and the Web. New York : Apress, 2015. 210 p.

2. Тлумачний словник з інформатики / Півняк Г. Г., Бусигін Б. С., Дівізінюк М. М. та ін. Дніпро : Нац. гірнич. ун-т, 2010. 600 с.

3. Patrick G. Henriette B. Philippe B. UML 2.0 in Action. Birmingham : Packt Publishing, 2005. 229 p.

4. Van Der Spuy R. The Advanced Game Developer's Toolkit: Create Amazing Web-based Games with JavaScript and HTML5. Toronto, 2017. 185 p.

5. JavaScript documentation. DevDocs API Documentation. URL : https://devdocs.io/javascript/ (дата звернення: 15.05.2024).

6. PixiJS documentation. PixiJS API Documentation. URL : https://pixijs.download/dev/docs/index.html (дата звернення: 10.05.2024).

7. PixiJS Tilemap documentation. PixiJS Tilemap API Documentation. URL : https://pixijs.io/tilemap/docs/ (дата звернення: 20.05.2024).

8. PixiJS UI documentation. PixiJS UI. URL : https://pixijs.io/ui/ (дата звернення: 10.08.2024).

9. PixiJS Sound documentation. PixiJS Sound API Documentation. URL : https://pixijs.io/sound/docs/index.html (дата звернення: 10.10.2024).

10. Vasan S. Pro MERN Stack. Bangalore : Apress, 2017. 328 p.

ДОДАТОК А

Код ігрової логіки

Main.js

```
import './style.css';
import * as PIXI from 'pixi.js';
import { Player } from './src/game/Player.js';
import { GameLogic } from './src/game/Game.js';
import * as Asset from './src/Assets.js';
import * as Utils from './src/utils.js';
import {Stats} from './src/ui/Stats.js';
import { CreateCharacter } from './src/ui/CreateCharacter.js';
import * as Locations from './src/game/Locations.js';
import { Warrior } from './src/game/heroes/Warrior.js';
import { Mage } from './src/game/heroes/Mage.js';
import { Archer } from './src/game/heroes/Archer.js';
import { ScreenUI } from './src/ui/ScreenUI.js';
import { Journal } from './src/ui/Journal.js';

export async function setup(gameData={}) {

    const app = new PIXI.Application({ width: 640, height: 360});
    document.getElementById('game-container').appendChild(app.view);

    const scaleFactor = 0.8;

      let newWidth, newHeight;

      if (window.innerWidth / window.innerHeight > 16 / 9) {
          newHeight = window.innerHeight * scaleFactor;
          newWidth = newHeight * (16 / 9);
      } else {
          newWidth = window.innerWidth * scaleFactor;
          newHeight = newWidth / (16 / 9);
      }

      //resize the app for screen
      app.renderer.resize(newWidth, newHeight);

      const scaleX = Math.round(app.renderer.width / 360);
      Utils.setScale(scaleX, app.renderer.width, app.renderer.height);

      //set bg image
        const background = new PIXI.Sprite(Asset.cloud);
        background.width = app.screen.width;
        background.height = app.screen.height;
        app.stage.addChild(background);


    //if no loaded data then start new game and create new character
    var hero;
    if(JSON.stringify(gameData) === '{}'){
        const create = new CreateCharacter();
        app.stage.addChild(create.container);
        await create.create();
        app.stage.removeChild(create.container);
```

```
    if(create.currentClass === 0){
        hero = new Warrior(create.name.value, create.characterSprite);
    }
    else if(create.currentClass === 1){
        hero = new Mage(create.name.value, create.characterSprite);
    }
    else if(create.currentClass === 2){
        hero = new Archer(create.name.value, create.characterSprite);
    }
    hero.spriteNum = create.currentCharacter;
    hero.getStartingItems();
}
else{
    //else load data into game
    const newTexture = new PIXI.Texture(
        Asset.characters[gameData.sprite],
        Utils.calculateTileRectangle({ x: 0, y: 0 }, 16, 16, 1)
    );
    Utils.scaleTexture(newTexture);
    const newSprite = new PIXI.Sprite(newTexture);
    if(gameData.class === 'warrior'){
        hero = new Warrior(gameData.name, newSprite);
    }
    else if(gameData.class === 'mage'){
        hero = new Mage(gameData.name, newSprite);
    }
    else if(gameData.class === 'archer'){
        hero = new Archer(gameData.name, newSprite);
    }
    hero.spriteNum = gameData.sprite;
    hero.exp = gameData.exp;
    hero.money = gameData.money;
    hero.setLevel(gameData.lvl);
    hero.questManager.activeQuests = gameData.activeQuests;
    hero.questManager.completedQuests = gameData.completedQuests;
    hero.questManager.loadAvailableQuests();
    hero.metNpc = gameData.metNpc;
    for (const [itemType, itemDetails] of Object.entries(gameData.inventory)) {
        itemDetails.forEach(itemDetail => {
            const { name, equipped } = itemDetail;

            //load hero items
            const item = Locations.loadItemByName(name, itemType);

            hero.inventory.addItem(item);

            //equip items that vere equipped
            if (equipped) {
                hero.equipItem(item);
            }
        });
    }

}

//load maps to game
var maps = await Locations.initializeGame();
var currentMap;
console.log(maps)

maps.forEach(map => {
    console.log(map)
    app.stage.addChild(map.map);
```

```
    });

    currentMap = maps[0];
    currentMap.map.visible = true;

    //create player object, load spritesheet with animations info
    const spritesheet = new PIXI.Spritesheet(Asset.characters[hero.spriteNum], Asset.character);
    await spritesheet.parse();
    const player = new Player(16*12, 16*7, Asset.characters[hero.spriteNum], spritesheet);
    app.stage.addChild(player.player);

    //create ui elements
    const ui = new ScreenUI(hero);
    app.stage.addChild(ui.container);

    const stats = new Stats(hero);
    app.stage.addChild(stats.container);


    const journal = new Journal(hero);
    app.stage.addChild(journal.container);

    //create game logic
    const Game = new GameLogic(app, hero, player, currentMap, ui);

    //add event listeners to open and close inventory and journal ui
    window.addEventListener('keydown', (event) => {
        if (event.key === 'i' || event.key === 'I') {
            Asset.sound.play('openMenu');
            stats.container.visible = !stats.container.visible;
            stats.addStats();
        }
    });

    window.addEventListener('keydown', (event) => {
        if (event.key === 'j' || event.key === 'J') {
            Asset.sound.play('openMenu');
            journal.createQuestList();
            journal.container.visible = !journal.container.visible;

        }
    });

}
```

Assets.js

```
import * as PIXI from 'pixi.js';
import { sound } from '@pixi/sound';
import { character } from "../assets/data/Character.js";
import { dragon } from '../assets/data/Dragon.js';

//game data files
const response = await fetch('../assets/data/gameData.json');
const gameData = await response.json();
const response2 = await fetch('../assets/data/questData.json');
const questData = await response2.json();
const response3 = await fetch('../assets/data/Items.json');
const itemsData = await response3.json();

//maps
export const tiles = await PIXI.Assets.load('./assets/img/roguelikeSheet_transparent.png');
export const start = await PIXI.Assets.load('./assets/maps/start.json');
```

```
export const sea = await PIXI.Assets.load('./assets/maps/sea.json');
export const dragonLand = await PIXI.Assets.load('./assets/maps/dragonLand.json');
export const castle = await PIXI.Assets.load('./assets/maps/castle.json');

//UI elements
const UIset = await PIXI.Assets.load('./assets/img/UIpackSheet_transparent.png');
const menu = await PIXI.Assets.load('./assets/Ancient/tan_inlay.png');
const button_default = await PIXI.Assets.load('./assets/Ancient/tan.png');
const button_pressed = await PIXI.Assets.load('./assets/Ancient/brown_pressed.png');
const Items = await PIXI.Assets.load('./assets/img/roguelikeitems.png');
const battleBackground = await PIXI.Assets.load('./assets/img/battle.png');
const cloud = await PIXI.Assets.load('./assets/img/cloud.jpg');
//sounds
   sound.add('ambient','./assets/sounds/ambient.mp3');
   sound.add('openMenu','./assets/sounds/openMenu.mp3');
   sound.add('quest','./assets/sounds/quest.mp3');
   sound.add('steps','./assets/sounds/steps.mp3');
   sound.add('sword','./assets/sounds/sword.mp3');
   sound.add('battle','./assets/sounds/battle.mp3');
   sound.add('money','./assets/sounds/money.mp3');
   sound.add('fail','./assets/sounds/fail.mp3');

   //characters
const imagePaths = [
   './assets/characters/F_01.png',
   './assets/characters/F_02.png',
   './assets/characters/F_03.png',
   './assets/characters/F_04.png',
   './assets/characters/F_05.png',
   './assets/characters/M_01.png',
   './assets/characters/M_02.png',
   './assets/characters/M_03.png',
   './assets/characters/M_04.png',
   './assets/characters/M_05.png',

];
const characters = [];
for (const path of imagePaths) {
   const texture = await PIXI.Assets.load(path);
   characters.push(texture);
}

//enemies
export const AquaDrake = await PIXI.Assets.load('./assets/img/AquaDrakeIdleSide.png');
export const AdultRed = await PIXI.Assets.load('./assets/img/AdultRedDragonIdleSide.png');

export {sound, gameData, questData, itemsData, character, dragon, characters, UIset, menu, button_default,
button_pressed, Items, battleBackground, cloud}
```

utils.js

```
import * as PIXI from 'pixi.js';

var scale = 3; //number for scaling all objects
var width = 1600;
var height = 900;
var save = {}; //save data

function setup(renderer){
   render = renderer;
}

//find tile location on spritesheet
```

```
function getTileCoordinates(tilesetId, tileWidth, tilesheetWidth){
    const tilesPerRow = 57;
    const row = Math.floor(tilesetId / tilesPerRow);
    const col = tilesetId % tilesPerRow-1;

    return {x: col, y: row};
}

//calculate tile pixel location on spritesheet
function calculateTileRectangle(coord, tileWidth, tileHeight, margin) {

    const x = coord.x * tileWidth + coord.x * margin;
    const y = coord.y * tileHeight + coord.y * margin;

    return new PIXI.Rectangle(x, y, tileWidth, tileHeight);
  }

function scaleTexture(texture){
    texture.baseTexture.scaleMode = PIXI.SCALE_MODES.NEAREST;
}

function setScale(setScale, x, y){
    scale = setScale;
    width = x;
    height = y;
}

function setSave(hero){
    save = {
        name: hero.name,
        sprite: hero.spriteNum,
        class: hero.constructor.name.toLowerCase(),
        lvl: hero.lvl,
        exp: hero.exp,
        money: hero.money,
        inventory: hero.inventory.getObjectItems(),
        activeQuests: hero.questManager.activeQuests,
        completedQuests: hero.questManager.completedQuests,
        metNpc: hero.metNpc,
      }
}

  export {setup, calculateTileRectangle, getTileCoordinates, scaleTexture, setScale, setSave, scale, width, height,
save}
```

Game.js

```
import * as Utils from '../utils.js';
import { Battle } from './Battle.js';
import { createButton, createDialog } from '../ui/uiUtils.js';
import { Shop } from './Shop.js';
import * as Assets from '../Assets';

//class for all game logic and game updates
class GameLogic {
    constructor(app, hero, player, map, ui) {
      this.player = player;
      this.player.map = map;
      this.map = map;
      this.app = app;
      this.ui = ui;
      this.hero = hero;
```

```
  //basic music
  Assets.sound.play('ambient', {loop: true});

  this.gameLoop();
 }

 //game loop for updating
 async gameLoop() {
  //update required logic
  await this.updateLogic();
  this.app.renderer.render(this.app.stage);
  requestAnimationFrame(this.gameLoop.bind(this));
 }

 gameResize() {
  const scaleFactor = 0.8;

  let newWidth, newHeight;

  //calculate game size to be 16:9
  if (window.innerWidth / window.innerHeight > 16 / 9) {
     newHeight = window.innerHeight * scaleFactor;
     newWidth = newHeight * (16 / 9);
  } else {
     newWidth = window.innerWidth * scaleFactor;
     newHeight = newWidth / (16 / 9);
  }

  // resize app
  this.app.renderer.resize(newWidth, newHeight);

  const scaleX = Math.round(this.app.renderer.width / 360);
  Utils.setScale(scaleX, this.app.renderer.width, this.app.renderer.height);
 }

 //update stored save data
 updateSave(){
  Utils.setSave(this.hero);
 }

 //when step on exit tile switch to another location
 changeLocation(){
  var exit = this.map.getExitTile(this.player.x/Utils.scale, this.player.y/Utils.scale);
  if(exit.length != 0){
    //remove current map from screen
    this.map.map.x = 0;
    this.map.map.y = 0;
    this.map.map.visible = false;
    this.map.enemies.forEach(enemy => {
      enemy.enemy.visible = false;
    });

    //set new current map, update positions for everything
    this.map = exit[0];
    this.player.map = exit[0];

    this.player.map.map.x -= (exit[1]-this.player.player.x/Utils.scale)*Utils.scale;
    this.player.map.map.y -= (exit[2]-this.player.player.y/Utils.scale)*Utils.scale;
    this.player.x = exit[1]*Utils.scale;
    this.player.y = exit[2]*Utils.scale;
    this.map.map.visible = true;
    this.map.enemies.forEach(enemy => {
      enemy.enemy.visible = true;
```

```
        enemy.init();
      });
    }
  }

  //if player too close to enemy
  async enemyCollision(enemy) {
    const distanceX = Math.abs(this.player.x - enemy.enemy.x);
    const distanceY = Math.abs(this.player.y - enemy.enemy.y);
    const collisionThreshold = 1 * 16 * Utils.scale;

    //check for collission
    if (distanceX <= collisionThreshold && distanceY <= collisionThreshold) {
        this.app.renderer.background.color = 0x1099bb;

        //start battle
        this.battle = new Battle(this.hero, enemy);
        this.app.stage.addChild(this.battle.battleUI.container);
        this.battle.battleUI.container.visible = true;
        const winner = await this.battle.startBattle();

        //end battle and hide UI
        this.battle.battleUI.container.visible = false;

        if (winner === "hero") {
            //if defeated enemy update quests that required it
            const event = { type: "defeat", target: enemy.name };
            const completedQuestsCount = this.hero.questManager.completedQuests.length;
            this.hero.questManager.updateQuestProgress(event);

            //if quest completed show notification
            if (completedQuestsCount < this.hero.questManager.completedQuests.length) {
                this.app.stage.addChild(this.hero.questManager.complete.button.view);
                await new Promise(resolve => setTimeout(resolve, 1000));
                this.app.stage.removeChild(this.hero.questManager.complete.button.view);
            }

            //if enemy defeated remove it from map
            const enemyIndex = this.map.enemies.findIndex(e => e === enemy);
            if (enemyIndex > -1) {
                this.map.map.removeChild(enemy.enemy);
                this.map.enemies.splice(enemyIndex, 1);
            }
        }
    } else {
        this.app.renderer.background.color = 0x000000;
    }
  }

  //collission with npc check
   async npcCollision(npc) {

    if (!npc.handleKeyPress) {
        npc.handleKeyPress = async (event) => {
            if (event.key === 'E' || event.key === 'e') {

              //start dialog if E pressed
                const event = { type: "talk", target: npc.name };
              //update quests if it required talk to npc
                var l = this.hero.questManager.completedQuests.length;
                this.hero.questManager.updateQuestProgress(event);

                //if quest was completed show notification
```

```
            if(l<this.hero.questManager.completedQuests.length){
              this.app.stage.addChild(this.hero.questManager.complete.button.view);
              await new Promise(resolve => setTimeout(resolve, 1000));
              this.app.stage.removeChild(this.hero.questManager.complete.button.view)
            }

            //start dialog with npc
            var dialog = createDialog();
            this.app.stage.addChild(dialog.button.view);

            //check if this npc has available quests for player
              var availableQuest = this.hero.questManager.quests.find(q => q.giver === npc.name && q.status ===
"notStarted");
               var currentQuest = this.hero.questManager.quests.find(q => q.giver === npc.name && (q.status ===
"inProgress"));
            if(currentQuest) npc.activeQuest = currentQuest;
            var newQuest = undefined;
            if(availableQuest !== undefined && this.hero.questManager.checkRequirements(availableQuest)) {
               this.hero.questManager.startQuest(availableQuest, npc);
              //start quest and show notification
               newQuest = createButton();
               newQuest.addText("New Quest!");
               newQuest.button.view.setTransform(Utils.width/3, 4*Utils.scale);
               newQuest.scale_button(Utils.width/3, Utils.height/15);
            }

            //after dialog show shop if npc is seller
            if(npc instanceof Shop){
              let shopUI = await npc.speak(dialog, this.hero);
              this.app.stage.removeChild(dialog.button.view);
              this.app.stage.addChild(shopUI.container);
              //wait for shop close to continue game
              await shopUI.waitForClose();
              this.app.stage.removeChild(shopUI.container);
            }
            else{
              await npc.speak(dialog, this.hero);
              this.app.stage.removeChild(dialog.button.view);
            }

            if(newQuest!==undefined) {
              this.app.stage.addChild(newQuest.button.view);
              await new Promise(resolve => setTimeout(resolve, 1000));
              this.app.stage.removeChild(newQuest.button.view)
            }

            document.removeEventListener('keydown', npc.handleKeyPress);
            npc.isListening = false; // Reset the flag
          }
        };
    }

    if (Math.abs(this.player.x - npc.x) <= 1 * 16 * Utils.scale &&
        Math.abs(this.player.y - npc.y) <= 1 * 16 * Utils.scale) {

        this.app.renderer.background.color = 0x1099bb;

        // dasplay npc talk button
        if (!this.app.stage.children.includes(npc.talkButton.button.view)) {
            this.app.stage.addChild(npc.talkButton.button.view);
        }

        npc.moveButton();
```

```
            //event listener for pressing E button
            if (!npc.isListening) {
                npc.isListening = true;
                document.addEventListener('keydown', npc.handleKeyPress);
            }

        } else {
            this.app.renderer.background.color = 0x000000;
            this.app.stage.removeChild(npc.talkButton.button.view);

            // remove the keypress listener if the player moves away
            if (npc.isListening) {
                document.removeEventListener('keydown', npc.handleKeyPress);
                npc.isListening = false;
            }
        }
    }

    //updating game logic function (called frequently)
    async updateLogic() {

      this.ui.update();
      this.updateSave();
      document.addEventListener('keydown', (event) => this.player.handleKeyDown(event.key));
      document.addEventListener('keyup', (event) => this.player.handleKeyUp(event.key));
      this.player.move();
      this.map.setup();
      this.changeLocation();

      //check for collissions with enemies or npcs
      for (const enemy of this.map.enemies) {
        enemy.move();
        await this.enemyCollision(enemy);
      }

      for (const npc of this.map.npc) {
        npc.move();
        npc.moveButton();
        await this.npcCollision(npc);
      }

    }
  }

  export {GameLogic}
```

Player.js

```
import * as PIXI from 'pixi.js';
import * as Utils from '../utils.js';

class Player {
    constructor(x, y, texture, spritesheet) {
      this.spritesheet = spritesheet;

      this.map;
      this.player = new PIXI.AnimatedSprite(this.spritesheet.animations.stand_down);
      this.player.x = x; //actual player sprite position
      this.player.y = y;
      this.x = x; //position related to map, affects calculations but not actual player sprite position
      this.y = y;
      this.pressedKeys = [];
      this.scale = 1;
```

```
    this.init();
  }

  init() {
    this.player.animationSpeed = 0.07;
    this.player.anchor.set(0, 0.25);
    Utils.scaleTexture(this.player.texture);
    this.player.scale.set(Utils.scale);
    this.player.zIndex = 1;
    this.player.play();
    this.lastKeyPressed = 'ArrowDown';

  }

  handleKeyDown(key) {
    if (!this.pressedKeys.includes(key)) {
      this.pressedKeys.push(key);
    }
  }

  handleKeyUp(key) {
    const index = this.pressedKeys.indexOf(key);
    if (index !== -1) {
      this.pressedKeys.splice(index, 1);
    }
  }

  //if app scale is changed replace correctly
  replace(){
    this.player.scale.set(Utils.scale);
    this.player.x = (this.player.x/this.scale) * Utils.scale;
    this.player.y = (this.player.y/this.scale) * Utils.scale;
    this.map.map.x = (this.map.map.x/this.scale) * Utils.scale;
    this.map.map.y = (this.map.map.y/this.scale) * Utils.scale;
    this.x = (this.x/this.scale) * Utils.scale;
    this.y = (this.y/this.scale) * Utils.scale;
    this.scale = Utils.scale;
  }

  move() {
    //set direction to go based on pressed keys
    if(this.scale != Utils.scale) this.replace();
            if(this.pressedKeys.length !== 0 && this.isCentered(this.map.map)) this.lastKeyPressed =
this.pressedKeys[this.pressedKeys.length - 1];
    if(this.pressedKeys.length!==0 || !this.isCentered(this.map.map)){
      if (this.lastKeyPressed === 'ArrowUp') {
                    if(this.player.textures !== this.spritesheet.animations.walk_up) this.player.textures =
this.spritesheet.animations.walk_up;
        if(this.map.getObjectsAtPosition(this.x, this.y-16*Utils.scale).length===0 || !this.isCentered(this.map.map))
{this.map.map.y += Utils.scale/2; this.y -= Utils.scale/2;}
      } else if (this.lastKeyPressed === 'ArrowDown') {
                    if(this.player.textures !== this.spritesheet.animations.walk_down) this.player.textures =
this.spritesheet.animations.walk_down;
        if(this.map.getObjectsAtPosition(this.x, this.y+16*Utils.scale).length===0 || !this.isCentered(this.map.map))
{this.map.map.y -= Utils.scale/2; this.y +=Utils.scale/2;}
      } else if (this.lastKeyPressed === 'ArrowLeft') {
                    if(this.player.textures !== this.spritesheet.animations.walk_left) this.player.textures =
this.spritesheet.animations.walk_left;
        if(this.map.getObjectsAtPosition(this.x-16*Utils.scale, this.y).length===0 || !this.isCentered(this.map.map))
{this.map.map.x += Utils.scale/2; this.x -=Utils.scale/2;}
      } else if (this.lastKeyPressed === 'ArrowRight') {
                    if(this.player.textures !== this.spritesheet.animations.walk_right) this.player.textures =
this.spritesheet.animations.walk_right;
```

```
        if(this.map.getObjectsAtPosition(this.x+16*Utils.scale, this.y).length===0 || !this.isCentered(this.map))
{this.map.map.x -= Utils.scale/2; this.x += Utils.scale/2;}
      }
   this.player.play();
   }

   else if(this.pressedKeys.length===0){

     //after all buttons released player turned to correct side
     if (this.lastKeyPressed === 'ArrowUp') {
                    if(this.player.textures      !==      this.spritesheet.animations.stand_up)      this.player.textures      =
this.spritesheet.animations.stand_up;
     } else if (this.lastKeyPressed === 'ArrowDown') {
                   if(this.player.textures      !==      this.spritesheet.animations.stand_down)      this.player.textures      =
this.spritesheet.animations.stand_down;
     } else if (this.lastKeyPressed === 'ArrowLeft') {
                    if(this.player.textures      !==      this.spritesheet.animations.stand_left)      this.player.textures      =
this.spritesheet.animations.stand_left;
     } else if (this.lastKeyPressed === 'ArrowRight') {
                    if(this.player.textures      !==      this.spritesheet.animations.stand_right)      this.player.textures      =
this.spritesheet.animations.stand_right;
     }

     this.player.play();
     }
   }

   isCentered(character){
     return (character.x%(16*Utils.scale)===0 && character.y%(16*Utils.scale)===0);
   }
  }

  export {Player};
```

Map.js

```
import * as PIXI from 'pixi.js';
import { CompositeRectTileLayer } from '@pixi/tilemap';
import * as Utils from '../utils';
class Map {
  constructor(tiles, tilemap) {
     this.map = new PIXI.Container();
     this.map.visible = false;
     this.obj = {};
     this.tiles = tiles;
     this.tilemap = tilemap;

     this.exitTiles = {};
     this.enemies = [];
     this.npc = [];

     this.initializeMap();

  }

  initializeMap() {
    const tiles = this.tiles;
    const tilemap = this.tilemap;
    this.map.tileWidth = 16;
    this.map.tileHeight = 16;
    this.map.removeChildren();

    //create tilemap based on json data
```

```
    for (let layerIndex = 0; layerIndex < tilemap.layers.length; layerIndex++) {
        const layerData = tilemap.layers[layerIndex].data;
        const layer = new CompositeRectTileLayer(0, []);
        this.map.addChild(layer);

        for (let row = 0; row < tilemap.height; row++) {
            for (let col = 0; col < tilemap.width; col++) {
                const tileId = layerData[row * tilemap.width + col];

                if (tileId !== 0) {
                    const coordinates = Utils.getTileCoordinates(tileId, 16, 968);
                    const tileTexture = new PIXI.Texture(tiles, Utils.calculateTileRectangle(coordinates, 16, 16, 1));
                    Utils.scaleTexture(tileTexture);

                    const x = col * this.map.tileWidth;
                    const y = row * this.map.tileHeight;

                    layer.tile(tileTexture, x, y);
                    layer.scale.set(Utils.scale);

                    //second layer is objects which are obstacles
                    //player cant walk over objects
                    if (layerIndex === 1) {
                        const object = new PIXI.Sprite(tileTexture);
                        object.x = x;
                        object.y = y;
                        object.scale.set(Utils.scale);
                        object.zIndex = 1;
                        this.addObjectToMap(object.x * Utils.scale, object.y * Utils.scale, object);
                    }
                }
            }
        }
    }

    //add enemies
    this.enemies.forEach(enemy => {
      this.map.addChild(enemy.enemy);
      enemy.enemy.visible = true;
  });

  //add npcs
  this.npc.forEach(npc => {
      this.map.addChild(npc.enemy);
      npc.enemy.visible = true;
  });
}

//updating npc and enemies staates
  setup() {

    this.enemies.forEach(enemy => {
      this.map.addChild(enemy.enemy);
      enemy.enemy.visible = true;
  });

  this.npc.forEach(npc => {
    this.map.addChild(npc.enemy);
    npc.enemy.visible = true;
});
}

  addObjectToMap(x, y, object) {
```

```
    if (this.obj[x] && this.obj[x][y]) {
      this.obj[x][y].push(object);
    } else {
        if (!this.obj[x]) this.obj[x] = {};
        this.obj[x][y] = [object];
    }
  }

  getObjectsAtPosition(x, y) {
    return ((this.obj[x] && this.obj[x][y]) ? this.obj[x][y] : []);
  }

  removeObjectFromMap(x, y, object) {
    if (this.obj[x] && this.obj[x][y]) {
        const index = this.obj[x][y].indexOf(object);
        if (index !== -1) {
          this.obj[x][y].splice(index, 1);
        }
    }
  }

  //tiles for changing location
  addExitTile(x, y, nextLocation, newX, newY){
     this.exitTiles[x] = {};
     this.exitTiles[x][y] = [nextLocation, newX, newY];

  }

  getExitTile(x, y) {
    return ((this.exitTiles[x] && this.exitTiles[x][y]) ? this.exitTiles[x][y] : []);
  }

}

export {Map}
```

Locations.js

```
import * as PIXI from 'pixi.js';
import * as Asset from '../Assets.js';
import { FireDragon } from './enemies/FireDragon.js';
import { WaterDragon } from './enemies/WaterDragon.js';
import { Npc } from './Npc.js';
import { Shop } from './Shop.js';
import { Map } from '../game/Map.js';
import { Item, Accessory, Armor, Weapon, HealingItem } from './Item.js';

//load all game data from json files
async function loadGameFromData() {
    const maps = [];
    const mapNameToInstance = {};

    //npc's and enemies
    const npcData = Asset.gameData.npcs;
    const enemyData = Asset.gameData.enemies;

    for (const mapData of Asset.gameData.maps) {
        //create map objects
        const map = new Map( Asset[mapData.tiles], Asset[mapData.tilemap]);
        map.setup();

        //for each map add npcs
        for (const npcInstance of mapData.npcs) {
```

```
        const npcInfo = npcData[npcInstance.type];
        const npcSpritesheet = new PIXI.Spritesheet(Asset.characters[npcInfo.spritesheet], Asset.character);
        await npcSpritesheet.parse();

        let npc;
        //if npc also a shop (sells sth)
        if (npcInfo.job === "Shop") {
            npc = new Shop(npcInstance.type, npcSpritesheet, map, npcInstance.positions);

            //items for shop
            npcInfo.items.forEach(item => {
                const loadedItem = loadItemByName(item.name, item.type);
                if (loadedItem) {
                    npc.inventory.addItem(loadedItem);
                }
            });
        } else {
            npc = new Npc(npcInstance.type, npcSpritesheet, map, npcInstance.positions);
        }

        //set text messages for npcs
        npc.firstMeetingText = npcInfo.firstMeetingText;
        npc.usualText = npcInfo.usualText;

        map.npc.push(npc);
    }

    //add enemies for each map
    for (const enemyInstance of mapData.enemies) {
        const enemyInfo = enemyData[enemyInstance.type];
        const enemySpritesheet = new PIXI.Spritesheet(Asset[enemyInfo.spritesheet], Asset.dragon);
        await enemySpritesheet.parse();

        let enemy;
        if (enemyInstance.type === "FireDragon") {
            enemy = new FireDragon(enemyInstance.type, enemySpritesheet, map, enemyInstance.positions);
        } else if (enemyInstance.type === "WaterDragon") {
            enemy = new WaterDragon(enemyInstance.type, enemySpritesheet, map, enemyInstance.positions);
        }

        enemy.maxHP = enemyInfo.stats.maxHP;
        enemy.currentHP = enemyInfo.stats.currentHP;
        enemy.atk = enemyInfo.stats.atk;
        enemy.spAtk = enemyInfo.stats.spAtk;
        enemy.def = enemyInfo.stats.def;
        enemy.spDef = enemyInfo.stats.spDef;
        enemy.lvl = enemyInfo.stats.lvl;
        enemy.exp = enemyInfo.stats.exp;
        enemy.money = enemyInfo.stats.money;

        map.enemies.push(enemy);
    }

    mapNameToInstance[mapData.name] = map;
    maps.push(map);
}

//add change location tiles for maps
for (const mapData of Asset.gameData.maps) {
    const map = mapNameToInstance[mapData.name];
    for (const exitTile of mapData.exitTiles) {
        const targetMap = mapNameToInstance[exitTile.toMap];
        if (targetMap) {
```

```
                map.addExitTile(exitTile.x, exitTile.y, targetMap, exitTile.targetX, exitTile.targetY);
            }
        }
    }

    //return map list
    return maps;
}

export async function initializeGame() {
    const gameMaps = await loadGameFromData();
    console.log("Game loaded successfully", gameMaps);

    return gameMaps;
}

//load items data and create corresponding objects
export function loadItemByName(name, itemType) {
    const itemsData = Asset.itemsData;

    let itemData;
    console.log(itemType)
    switch (itemType) {
        case 'weapon':
            itemData = itemsData.weapons[name];
            if (!itemData) throw new Error("Weapon not found.");
            const weapon = new Weapon(name, itemData.position.x, itemData.position.y);
            weapon.setStats(itemData.atk, itemData.spAtk);
            weapon.setPrice(itemData.buyPrice, itemData.sellPrice);
            return weapon;

        case 'armor':
            itemData = itemsData.armor[name];
            if (!itemData) throw new Error("Armor not found.");
            const armor = new Armor(name, itemData.position.x, itemData.position.y);
            armor.setStats(itemData.def, itemData.spDef);
            armor.setPrice(itemData.buyPrice, itemData.sellPrice);
            return armor;

        case 'healingitem':
            itemData = itemsData.healingItems[name];
            if (!itemData) throw new Error("Healing item not found.");
            const healingItem = new HealingItem(name, itemData.position.x, itemData.position.y);
            healingItem.setStats(itemData.healingAmount);
            healingItem.setPrice(itemData.buyPrice, itemData.sellPrice);
            return healingItem;

        case 'accessory':
            itemData = itemsData.accessories[name];
            if (!itemData) throw new Error("Accessory not found.");
            const accessory = new Accessory(name, itemData.specialEffect, itemData.position.x, itemData.position.y);
            accessory.setStats(itemData.atk, itemData.spAtk, itemData.def, itemData.spDef);
            accessory.setPrice(itemData.buyPrice, itemData.sellPrice);
            return accessory;

        default:
            throw new Error("Invalid item type.");
    }
}
```

Enemy.js

```
import * as PIXI from 'pixi.js';
import * as Utils from '../utils.js';

//general enemy class
class Enemy{
   constructor(spritesheet, map, points) {
      this.spritesheet = spritesheet;
      this.map = map;

      this.enemy = new PIXI.AnimatedSprite(this.spritesheet.animations.stand_down);
      this.enemy.visible = false;

      this.startPoints = points;
      this.points = [];
      this.nextPoint = 0;

      this.scale = 1;

      this.maxHP = 100;
      this.currentHP = 100;
      this.atk = 10;
      this.spAtk = 15;
      this.def = 5;
      this.spDef = 10;
      this.lvl = 1;
      this.exp = 0;
      this.money = 0;

      this.init();
    }

    init() {
      //points for enemy to move between (loop)
      this.points = [];
      this.startPoints.forEach(point => {
        this.points.push(JSON.parse(JSON.stringify(point)));
       });

      //position info
      this.mapX = this.map.map.x;
      this.mapY = this.map.map.y;
      this.enemy.x = this.points[0].x;
      this.enemy.y = this.points[0].y;
      this.x = this.points[0].x;
      this.y = this.points[0].y;

      //animation and sprite info
      this.enemy.animationSpeed = 0.07;
      this.enemy.anchor.set(0, 0.25);
      Utils.scaleTexture(this.enemy.texture);
      this.enemy.scale.set(Utils.scale);
      this.enemy.zIndex = 1;
      this.enemy.play();
    }

    //replace correctly if scale is changed
    replace(){
      this.enemy.scale.set(Utils.scale);
      this.enemy.x = (this.enemy.x/this.scale) * Utils.scale;
      this.enemy.y = (this.enemy.y/this.scale) * Utils.scale;
      this.x = (this.x/this.scale) * Utils.scale;
      this.y = (this.y/this.scale) * Utils.scale;
      this.scale = Utils.scale;
```

```
      this.mapX=this.map.map.x;
      this.mapY=this.map.map.y;
    }

    //move on map
    move() {
      if(this.scale != Utils.scale) this.replace();
      this.mapX=this.map.map.x;
      this.mapY=this.map.map.y;

      if(this.points.length > 0) this.move_by_points();
    }

    move_by_points(){
      //move enemy along predefined points
      if((this.points[this.nextPoint].x*Utils.scale !== this.x || this.points[this.nextPoint].y*Utils.scale !== this.y)){
         if(this.points[this.nextPoint].x*Utils.scale > this.x) {this.enemy.x += Utils.scale/2; this.x += Utils.scale/2;}
         if(this.points[this.nextPoint].x*Utils.scale < this.x) {this.enemy.x -= Utils.scale/2; this.x -= Utils.scale/2;}
         if(this.points[this.nextPoint].y*Utils.scale > this.y) {this.enemy.y += Utils.scale/2; this.y += Utils.scale/2;}
         if(this.points[this.nextPoint].y*Utils.scale < this.y) {this.enemy.y -= Utils.scale/2; this.y -= Utils.scale/2;}
      }
      else {
         if(this.nextPoint<this.points.length-1) this.nextPoint++;
         else this.nextPoint = 0;
      }
    }

    isCentered(character){
      return (character.x%(16*Utils.scale)===0 && character.y%(16*Utils.scale)===0);
    }

}

export {Enemy}
```

Npc.js

```
import * as Utils from '../utils.js';
import { Enemy } from './Enemy.js';
import { createButton } from '../ui/uiUtils.js';

class Npc extends Enemy{
   constructor(name, spritesheet, map, points){
      super(spritesheet, map, points);

      this.name = name;
      this.showButton();

      this.activeQuest = null;
      this.startQuest = false;
      this.firstMeet = true;
      this.firstMeetingText = [];
      this.usualText = [];

      this.isListening = false;
      this.handleKeyPress;
   }

   //E button to show that you can talk
   showButton(){
      this.talkButton = createButton();
      this.talkButton.addText("E");
   }
```

```
    moveButton() {
        //change talk button position
        this.talkButton.button.view.x = this.x + this.mapX;
        this.talkButton.button.view.y = this.y + this.mapY - 16 * Utils.scale;
    }

    //change state if you already met npc
    meet(hero){
        this.firstMeet = false;
        hero.metNpc.push(this.name);
    }

    speak(dialog, hero) {
        //select messages depending on meeting state, quest progress etc
        return new Promise((resolve) => {
            let text = [];
            if(this.activeQuest){
                if(this.activeQuest.status == "inProgress" && this.startQuest){
                    if (this.firstMeet && !hero.metNpc.includes(this.name)) {
                        text = [...this.firstMeetingText, ...this.activeQuest.text.start];
                        this.meet(hero);
                    } else {
                        text = this.activeQuest.text.start;
                    }
                    this.startQuest = false;
                }
                else if(this.activeQuest.status == "inProgress")
                    text = this.activeQuest.text.progress;
                else if(this.activeQuest.status == "completed"){
                    text = this.activeQuest.text.complete;
                    this.activeQuest = null;
                }

            }
            else {
                if (this.firstMeet && !hero.metNpc.includes(this.name)) {
                    text = this.firstMeetingText;
                    this.meet(hero);
                } else{
                    text = this.usualText;
                }
            }

            let i = 0;

            dialog.slowText(text[i]);

            //on dialog click switch to next message
            dialog.button.onUp.connect(() => {
                i++;
                if (i < text.length) {
                    dialog.slowText(text[i]);
                } else {
                    resolve();
                }
            });
        });
    }

}
export {Npc}
```

Shop.js

```
import { Npc } from './Npc.js';
import { ShopUI } from '../ui/ShopUI.js';
import { Inventory } from './Inventory.js';

//same as npc but has shop instance to trade
class Shop extends Npc{
    constructor(name, spritesheet, map, points){
        super(name, spritesheet, map, points);

        this.inventory = new Inventory();
        this.money = 100;
    }

    speak(dialog, hero) {
        return new Promise((resolve) => {
            console.log(this.startQuest)
            let text = [];
            if(this.activeQuest){
                if(this.activeQuest.status == "inProgress" && this.startQuest){
                    if (this.firstMeet && !hero.metNpc.includes(this.name)) {
                        text = [...this.firstMeetingText, ...this.activeQuest.text.start];
                        this.meet(hero);
                    } else {
                        text = this.activeQuest.text.start;
                    }
                    this.startQuest = false;
                }
                else if(this.activeQuest.status == "inProgress")
                    text = this.activeQuest.text.progress;
                else if(this.activeQuest.status == "completed"){
                    text = this.activeQuest.text.complete;
                    this.activeQuest = null;
                }
            }
            else {
                if (this.firstMeet&& !hero.metNpc.includes(this.name)) {
                    text = this.firstMeetingText;
                    this.meet(hero);
                } else{
                    text = this.usualText;
                }
            }
            let i = 0;
            dialog.slowText(text[i]);

            dialog.button.onUp.connect(() => {
                i++;
                if (i < text.length) {
                    dialog.slowText(text[i]);

                } else {
                    //after dialog show shop instance so you can trade
                    const shop = new ShopUI(this, hero);
                    resolve(shop);
                }
            });
        });
    }
}

export {Shop}
```

QuestManager.js

```
import * as Assets from '../Assets';
import * as Utils from '../utils';
import { createButton } from '../ui/uiUtils';

//class for managing players quest progress
class QuestManager {
    constructor(hero) {
        this.quests = Assets.questData.quests;

        this.hero = hero;

        this.activeQuests = [];
        this.completedQuests = [];

        this.complete = createButton();
        this.complete.addText("Quest Completed!");
        this.complete.button.view.setTransform(Utils.width/3, 4*Utils.scale);
        this.complete.scale_button(Utils.width/3, Utils.height/15);
        this.complete.button.view.visible = true;
    }

    //update initial quests states based on palyer save file
    loadAvailableQuests() {
        console.log(this.activeQuests);

        this.activeQuests.forEach(activeQuest => {
            const questInList = this.quests.find(quest => quest.id === activeQuest.id);

            if (questInList) {
                questInList.status = "inProgress";
            }
        });

        this.completedQuests.forEach(completeQuest => {
            const questInList = this.quests.find(quest => quest.id === completeQuest.id);
            if (questInList) {
                questInList.status = "completed";
            }
        });
    }


    //check if player can start quest
    checkRequirements(quest){
        const unlockConditions = quest.unlockConditions;

        const isRequiredQuestCompleted = unlockConditions.requiredQuest
            ? this.completedQuests.some(completedQuest => completedQuest.id === unlockConditions.requiredQuest)
            : true;

        if (this.hero.lvl >= unlockConditions.playerLevel && isRequiredQuestCompleted) {
            return true;
        } else {
            return false;
        }
        ;
    }

    startQuest(quest, npc){
        quest.status = "inProgress";
        this.activeQuests.push(quest);
```

```
        npc.activeQuest = quest;
        npc.startQuest = true;
    }

    completeQuest(quest) {
        quest.status = "completed";
        Assets.sound.play('quest');
        this.completedQuests.push(quest);
        this.activeQuests = this.activeQuests.filter(q => q.id !== quest.id);

        if(quest.rewards.gold) this.hero.money += quest.rewards.gold;
        if(quest.rewards.exp) this.hero.earnExp(quest.rewards.exp);
    }

    updateQuestProgress(event) {
        this.activeQuests.forEach(quest => {
            quest.objectives.forEach(obj => {
                if (event.type === obj.type && event.target === obj.target) {
                    obj.currentAmount++;
                    if (obj.currentAmount >= obj.requiredAmount) {
                        this.completeQuest(quest);
                    }
                }
            });
        });
    }
}

export { QuestManager };
```

Item.js

```
import * as PIXI from 'pixi.js';
import * as Assets from '../Assets';
import * as Utils from '../utils';

//parent general item class
class Item{
    constructor(text, x=5, y=5){
        this.Texture = new PIXI.Texture(Assets.Items, Utils.calculateTileRectangle({x: x, y: y}, 16, 16, 0));
        this.text = text;

        this.buyPrice = 100;
        this.sellPrice = 50;

        this.equipped = false;
    }

    setPrice(buy, sell){
        this.buyPrice = buy;
        this.sellPrice = sell;
    }

    setStats(){}
}
//child classes of items of different types
class Weapon extends Item {
    constructor(text, x = 1, y = 7) {
        super(text, x, y);
    }

    setStats(atk, spAtk) {
        this.atk = atk;
```

```
        this.spAtk = spAtk;
    }
}

class Armor extends Item {
    constructor(text, x = 9, y = 9) {
        super(text, x, y);
    }

    setStats(def, spDef) {
        this.def = def;
        this.spDef = spDef;
    }
}

class Accessory extends Item {
    constructor(text, specialEffect, x = 0, y = 2) {
        super(text, x, y);
        this.specialEffect = specialEffect;
    }

    setStats(atk, spAtk, def, spDef) {
        this.atk = atk;
        this.spAtk = spAtk;
        this.def = def;
        this.spDef = spDef;
    }
}

class HealingItem extends Item {
    constructor(text, x = 11, y = 4) {
        super(text, x, y);
    }

    setStats(healingAmount) {
        this.healingAmount = healingAmount;
    }
}

export { Weapon, Armor, Accessory, HealingItem };
export{Item}
```

Inventory.js

```
import * as PIXI from 'pixi.js';
import * as Assets from '../Assets';
import * as Utils from '../utils';

class Inventory {
    constructor() {
        this.items = [];
        this.Texture = new PIXI.Texture(Assets.Items, Utils.calculateTileRectangle({x: 5, y: 5}, 16, 16, 0));
        this.text = "Inventory";
    }

    addItem(item) {
        this.items.push(item);
    }

    removeItem(itemToRemove) {
        const index = this.items.findIndex(item => item.text === itemToRemove.text);
        if (index !== -1) {
            this.items.splice(index, 1);
```

```
        }
    }

    getItems() {
        return this.items;
    }

    //get object with all items stored with names and item types
    getObjectItems() {
        var items = {};

        this.items.forEach(item => {
            const currentItemType = item.constructor.name.toLowerCase();
            const name = item.text;

            if (!items[currentItemType]) {
                items[currentItemType] = [];
            }

            items[currentItemType].push({ name, equipped: item.equipped });
        });

        return items;
    }

}

export { Inventory };
```

Battle.js

```
import {BattleUI} from '../ui/BattleUI.js';
import * as Assets from '../Assets.js'
import axios from 'axios'

class Battle{
    constructor(hero, enemy){
        this.hero = hero;
        this.enemy = enemy;

        this.battleOver = false;
        this.currentTurn = "player";

        this.battleUI = new BattleUI(this.hero, this.enemy);
    }

    //battle loop
    async startBattle() {
        //music
        Assets.sound.stop('ambient');
        Assets.sound.play('battle', {singleInstance: true, loop: true});

        //loop for battle until someone wins
        while (!this.battleOver) {

            //player turn
            if (this.currentTurn === "player") {
                await this.battleUI.playerTurn();
                this.currentTurn = "enemy";
            } else {
                //enemy turn logic
                let text = this.enemy.takeTurn(this.hero);
                Assets.sound.play('sword');
```

```
            this.battleUI.animateSprite(this.battleUI.spriteEnemy);
            await this.battleUI.createActionDialog(text);
            this.currentTurn = "player";
        }

        //check if battle is over
        if (this.hero.currentHP <= 0 || this.enemy.currentHP <= 0) {

            this.battleOver = true;
            console.log('Battle ended');
            this.winner = this.hero.currentHP > 0 ? "hero" : "enemy";
            await this.battleUI.battleOverScreen(this.winner);

            if(this.winner === "hero"){
                this.hero.earnExp(this.enemy.exp);
                this.hero.money += this.enemy.money;
            }
            else{
                //if player lost - load last saved data
                try {
                    const response = await axios.get('/load-game', { withCredentials: true });

                    if (response.data.gameData) {
                        localStorage.setItem('gameData', JSON.stringify(response.data.gameData));
                        window.location.href = 'game.html';
                    } else {
                        document.getElementById('game-message').textContent = 'No saved game found';
                    }
                } catch (error) {
                    console.log(error);
                    document.getElementById('game-message').textContent = 'Error loading game';
                }
            }
        }
        await new Promise(resolve => setTimeout(resolve, 100));
    }
    Assets.sound.stop('battle');
    Assets.sound.play('ambient', {singleInstance: true, loop: true});
    return this.winner;
  }
}

export {Battle}
```

Hero.js

```
import { Inventory } from '../Inventory';
import { Item, Weapon, Accessory, Armor, HealingItem } from '../Item';
import { QuestManager } from '../QuestManager';

//common parent hero class
class Hero{
  constructor(name, sprite){
    this.name = name;
    this.sprite = sprite;
    this.spriteNum = 0;
    this.maxHP = 100;
    this.currentHP = 100;
    this.atk = 10;
    this.spAtk = 15;
    this.def = 5;
    this.spDef = 10;
    this.lvl = 1;
```

```
    this.exp = 0;
    this.nextLvlExp = 100;
    this.money = 100;

    this.questManager = new QuestManager(this);

    this.inventory = new Inventory();
    this.equippedItems = {
        weapon: null,
        armor: null,
        accessory: null,
        healingitem: null
    };

    this.actions = ['Attack'];

    this.metNpc = [];
}

getStartingItems(){}
attack(){}
specialAttack(){}
heal() {
    //healing method
    //heal with equipped healing item
    if (this.equippedItems.healingitem) {
        const healAmount = this.equippedItems.healingitem.healingAmount;
        this.currentHP += healAmount;

        //cant heal more than max hp
        if (this.currentHP > this.maxHP) {
            this.currentHP = this.maxHP;
        }

        //return action message
       return `${this.name} healed for ${healAmount}. Current HP: ${this.currentHP}`;
    } else {
       return `${this.name} has no healing item equipped.`;
    }
}

// guard(){}
// escape(){}

//add exp, check for lvlup
earnExp(exp){
    this.exp += exp;
    this.levelUp();
}

levelUp(){
    //if enough exp increase lvl, stats and heal
    while(this.exp >= this.nextLvlExp){
        this.lvl += 1;
        this.exp -= this.nextLvlExp;

        this.maxHP += 10;
        this.currentHP = this.maxHP;
        this.atk += 2;
        this.spAtk += 3;
        this.def += 1;
        this.spDef += 1;
```

```
    }
    //set new exp requirement
    this.nextLvlExp = this.lvl * 100;

}

//set lvl stats when loading hero
setLevel(lvl){
    //increase stats till required lvl reached
    for(let i=2; i<=lvl; i++){
        this.maxHP += 10;
        this.currentHP = this.maxHP;
        this.atk += 2;
        this.spAtk += 3;
        this.def += 1;
        this.spDef += 1;

    }
    this.lvl = lvl;
}

equipItem(item) {
    //increase stats depending on item
    if (item instanceof Weapon) {
        this.equippedItems.weapon = item;
        this.atk += item.atk;
        this.spAtk += item.spAtk;
    } else if (item instanceof Armor) {
        this.equippedItems.armor = item;
        this.def += item.def;
        this.spDef += item.spDef;
    } else if (item instanceof Accessory) {
        this.equippedItems.accessory = item;
        this.atk += item.atk;
        this.spAtk += item.spAtk;
        this.def += item.def;
        this.spDef += item.spDef;
    } else if (item instanceof HealingItem) {
        this.equippedItems.healingitem = item;
    }
    item.equipped = true;
}

unequipItem(itemType) {
    const item = this.equippedItems[itemType];

    if (!item) return;
    //decrease stats when unequipp item
    if (item instanceof Weapon) {
        this.atk -= item.atk;
        this.spAtk -= item.spAtk;
    } else if (item instanceof Armor) {
        this.def -= item.def;
        this.spDef -= item.spDef;
    } else if (item instanceof Accessory) {
        this.atk -= item.atk;
        this.spAtk -= item.spAtk;
        this.def -= item.def;
        this.spDef -= item.spDef;
    }

    this.equippedItems[itemType] = null;
    item.equipped = false;
```

```
    }


    //change equip state of item
    //one item of same category can be equipped in same time
    toggleEquip(item) {
        const currentItemType = item.constructor.name.toLowerCase();
        const previouslyEquippedItem = this.equippedItems[currentItemType];

        if (previouslyEquippedItem === item) {
            this.unequipItem(currentItemType);
            return previouslyEquippedItem;
        } else {
            if (previouslyEquippedItem) {
                this.unequipItem(currentItemType);
            }
            this.equipItem(item);
            return previouslyEquippedItem;
        }
    }

}

export {Hero}
```

Archer.js

```
import { Hero } from './Hero.js';
import { loadItemByName } from '../Locations.js';

class Archer extends Hero {
    constructor(name, sprite) {
        super(name, sprite);
        this.maxHP = 120;
        this.currentHP = 120;
        this.atk = 20;
        this.spAtk = 5;
        this.def = 15;
        this.spDef = 10;
        this.actions = ['Arrow Shot', 'Special Arrow', 'Quick Strike', 'Heal'];
    }

    attack(enemy, index) {
        let damage = 0;
        let attackType = "";

        switch (index) {
            case 0:
                damage = this.atk - enemy.def;
                attackType = `${this.name} used Arrow Shot!`;
                break;
            case 1:
                damage = (this.spAtk * 2) - enemy.spDef;
                attackType = `${this.name} used Special Arrow!`;
                break;
            case 2:
                damage = this.atk - enemy.def;
                attackType = `${this.name} used Quick Strike!`;
                break;
            case 3:
                attackType = this.heal();
                break;
            default:
```

```
            return "Invalid attack";
        }

        damage = Math.round(Math.max(1, damage));
        enemy.currentHP -= damage;

        return `${attackType} Enemy took ${damage} damage!`;
    }

    getStartingItems() {

        this.inventory.addItem(loadItemByName("Bow", 'weapon'));
        this.inventory.addItem(loadItemByName('Leather Armor', 'armor'));
        this.inventory.addItem(loadItemByName('Arrow Quiver', 'healingitem'));
    }
}

export {Archer}
```

Mage.js

```
import { Hero } from './Hero.js';
import { loadItemByName } from '../Locations.js';

class Mage extends Hero {
    constructor(name, sprite) {
        super(name, sprite);
        this.maxHP = 100;
        this.currentHP = 100;
        this.atk = 10;
        this.spAtk = 30;
        this.def = 10;
        this.spDef = 15;
        this.actions = ['Magic Blast', 'Staff Strike', 'Fireball', 'Heal'];
    }

    attack(enemy, index) {
        let damage = 0;
        let attackType = "";

        switch (index) {
            case 0:
                damage = this.spAtk - enemy.spDef;
                attackType = `${this.name} used Magic Blast!`;
                break;
            case 1:
                damage = this.atk - enemy.def;
                attackType = `${this.name} used Staff Strike!`;
                break;
            case 2:
                damage = (this.spAtk * 1.5) - enemy.spDef;
                attackType = `${this.name} used Fireball!`;
                break;
            case 3:
                attackType = this.heal();
                break;
            default:
                return "Invalid attack";
        }

        damage = Math.round(Math.max(1, damage));
        enemy.currentHP -= damage;
```

```
        return `${attackType} Enemy took ${damage} damage!`;
    }

    getStartingItems() {

        this.inventory.addItem(loadItemByName("Staff", 'weapon'));
        this.inventory.addItem(loadItemByName('Gloves', 'armor'));
        this.inventory.addItem(loadItemByName('Mana Potion', 'healingitem'));
    }
}

export {Mage}
```

Warrior.js

```
import { Hero } from './Hero.js';
import { loadItemByName } from '../Locations.js';

class Warrior extends Hero{
    constructor(name, sprite){
        super(name, sprite);
        this.maxHP = 150;
        this.currentHP = 150;
        this.atk = 25;
        this.spAtk = 10;
        this.def = 20;
        this.spDef = 5;

        this.actions = ['Light attack', 'Heavy attack', 'Magical attack', 'Heal'];
    }

     attack(enemy, index) {
        let damage = 0;
        let attackType = "";

        switch (index) {
            case 0:
                damage = this.atk - enemy.def;
                attackType = `${this.name} used Light attack!`;
                break;
            case 1:
                damage = (this.atk * 1.5) - enemy.def;
                attackType = `${this.name} used Heavy attack!`;
                break;
            case 2:
                damage = this.spAtk - enemy.spDef;
                attackType = `${this.name} used Magical attack!`;
                break;
            case 3:
                attackType = this.heal();
                break;
            default:
                return "Invalid attack";
        }

        damage = Math.round(Math.max(0, damage));
        enemy.currentHP -= damage;

        return `${attackType} Enemy took ${damage} damage!`;
    }

    getStartingItems() {
```

```
        this.inventory.addItem(loadItemByName("Sword", 'weapon'));
        this.inventory.addItem(loadItemByName('Shield', 'armor'));
        this.inventory.addItem(loadItemByName('Health Potion', 'healingitem'));
    }
}

export {Warrior}
```

FireDragon.js

```
import { Enemy } from "../Enemy";

class FireDragon extends Enemy {
    constructor(name, spritesheet, map, points) {
        super(spritesheet, map, points);
        this.name = name;
        this.maxHP = 120;
        this.currentHP = 120;
        this.atk = 20;
        this.spAtk = 15;
        this.def = 30;
        this.spDef = 10;
        this.lvl = 1;
        this.exp = 50;

        //attack names
        this.actions = ['Bite', 'Claw Swipe', 'Fireball'];
    }

    //enemy select how to attack
    chooseAttack(hero) {
        let chosenAttack;

        //if hero hp low then do powerfull attack
        if (hero.currentHP < hero.maxHP * 0.3) {
            chosenAttack = 2;
        } else {
            //random attaxk select
            chosenAttack = Math.floor(Math.random() * this.actions.length);
        }

        return chosenAttack;
    }

    //calculate damage and write info message about attack
    attack(hero, index) {
        let damage = 0;
        let attackType = "";

        switch (index) {
            case 0:
                damage = this.atk - hero.def;
                attackType = `${this.name} used Bite!`;
                break;
            case 1:
                damage = (this.atk * 1.2) - hero.def;
                attackType = `${this.name} used Claw Swipe!`;
                break;
            case 2:
                damage = this.spAtk - hero.spDef;
                attackType = `${this.name} used Fireball!`;
                break;
            default:
```

```
            return "Invalid attack";
        }

        damage = Math.max(1, damage);

        //update players hp
        hero.currentHP -= damage;

        //return message
        return `${attackType} Hero took ${damage} damage!`;
    }

    //enemy turn
    takeTurn(hero) {
        //select attack
        const attackIndex = this.chooseAttack(hero);

        //attack player
        return this.attack(hero, attackIndex);
    }

}

export {FireDragon}
```

WaterDragon.js

```
import { Enemy } from "../Enemy";

class WaterDragon extends Enemy {
    constructor(name, spritesheet, map, points) {
        super(spritesheet, map, points);
        this.name = name;
        this.maxHP = 150;
        this.currentHP = 150;
        this.atk = 20;
        this.spAtk = 35;
        this.def = 38;
        this.spDef = 50;
        this.lvl = 1;
        this.exp = 50;

        this.actions = ['Bite', 'Water strike', 'Waterfall'];
    }

    chooseAttack(hero) {
        let chosenAttack;

        if (hero.currentHP < hero.maxHP * 0.3) {
            chosenAttack = 2;
        } else {
            chosenAttack = Math.floor(Math.random() * this.actions.length);
        }

        return chosenAttack;
    }

    attack(hero, index) {
        let damage = 0;
        let attackType = "";

        switch (index) {
            case 0:
```

```
            damage = this.atk - hero.def;
            attackType = `${this.name} used Bite!`;
            break;
        case 1:
            damage = (this.atk * 1.2) - hero.def;
            attackType = `${this.name} used Water Strike!`;
            break;
        case 2:
            damage = this.spAtk - hero.spDef;
            attackType = `${this.name} used Waterfall!`;
            break;
        default:
            return "Invalid attack";
    }

    damage = Math.max(1, damage);
    hero.currentHP -= damage;

    return `${attackType} Hero took ${damage} damage!`;
  }

  takeTurn(hero) {
    const attackIndex = this.chooseAttack(hero);
    return this.attack(hero, attackIndex);
  }

}

export {WaterDragon}
```

ДОДАТОК Б

Код інтерфейсу користувача

uiUtils.js

```
import * as PIXI from 'pixi.js';
import { ScrollBox } from '@pixi/ui';
import * as Assets from '../Assets';
import * as Utils from '../utils';
import { CustomButton } from './Button';
import { Dialog} from './Dialog';

//utility functions for creating UI elements
export function createBackground(width = Utils.width-200, height = Utils.height - 200) {
    const bg_texture = new PIXI.Texture(Assets.menu);
    Utils.scaleTexture(bg_texture);
    const bg_sprite = new PIXI.Sprite(bg_texture);
    bg_sprite.width = width;
    bg_sprite.height = height;
    return bg_sprite;
}

export function createItemButton(text, itemTexture, x, y, width, height) {
    const itemButton = createButton();
    itemButton.scale_button(width/3, height/8);
    itemButton.button.view.x = x;
    itemButton.button.view.y = y;
    itemButton.addText(text);
    itemButton.addIcon(itemTexture);

    return itemButton;
}

export function createCloseButton(width) {
    const textureDefault = new PIXI.Texture(Assets.UIset, Utils.calculateTileRectangle({x: 9, y: 23}, 16, 16, 2));
    const texturePressed = new PIXI.Texture(Assets.UIset, Utils.calculateTileRectangle({x: 6, y: 23}, 16, 16, 2));
    Utils.scaleTexture(textureDefault);
    Utils.scaleTexture(texturePressed);

    const closeButton = new CustomButton(textureDefault, texturePressed);
    closeButton.scale_button(width/15, width/15);
    closeButton.button.view.x = width - (width/15+4*Utils.scale);
    closeButton.button.view.y = 4*Utils.scale;
    closeButton.button.view.zIndex = 2;

    return closeButton;
}

export function createItemList(items, width, height) {
    var i = 0;
    var buttons = [];
    items.forEach(item => {
        var x = 8*Utils.scale;
        var y = 8*Utils.scale + (4*Utils.scale + height/8)*i;
       buttons.push(createItemButton(item.text, item.Texture, x, y, width, height));
       i++;
    });
```

```
    return buttons;
}

export function createButton() {
    const texture1 = new PIXI.Texture(Assets.button_default);
    const texture2 = new PIXI.Texture(Assets.button_pressed);

    const itemButton = new CustomButton(texture1, texture2);
    return itemButton;
}

export function createCustomButton(tileRectangle, tileRectanglePressed) {
    const texture1 = new PIXI.Texture(Assets.UIset, tileRectangle);
    const texture2 = new PIXI.Texture(Assets.UIset, tileRectanglePressed);

    const itemButton = new CustomButton(texture1, texture2);
    return itemButton;
}

export function createDialog() {
    const texture1 = new PIXI.Texture(Assets.menu);
    const texture2 = new PIXI.Texture(Assets.button_pressed);

    const dialog = new Dialog(texture1, texture2);
    return dialog;
}

export function createScrollBox(buttons, width, height){
    return new ScrollBox({
        background: 0xD2B48C,
        padding: 8,
        width: width,
        height: height,
        items: buttons,
    });
}

export function createText(desiredText, x, y, anchor = true){
    const style = new PIXI.TextStyle({
        fontFamily: "Courier New",
        fontSize: 8*Utils.scale,
        fontVariant: "small-caps",
        fontWeight: "800"
    });
    const text = new PIXI.Text(desiredText, style);
    text.x = x;
    text.y = y;
    text.resolution = 10;
    if(anchor) text.anchor.set(0.5, 0);
    return text;
}
```

Button.js

```
import * as PIXI from 'pixi.js';
import { Button } from '@pixi/ui';
import { UIset } from '../Assets';
import * as Utils from '../utils'

class CustomButton{
    constructor(textureDefault, texturePressed){
        this.textureDefault = textureDefault;
        this.texturePressed = texturePressed;
```

```
    Utils.scaleTexture(this.textureDefault);
    Utils.scaleTexture(this.texturePressed);
    this.content = new PIXI.Container();
    this.sprite = new PIXI.Sprite(textureDefault);

    this.content.addChild(this.sprite);
    this.button = new Button(this.content);
    //this.button.view.addChild(this.content);
    this.button.eventMode = "dynamic";
    this.button.buttonMode = true;
    this.setup();
}

setup(){
    this.button.onDown.connect( () => {
        this.sprite.texture = this.texturePressed;
    });

    this.button.onUp.connect(() => {
        this.sprite.texture = this.textureDefault;
    });

}

scale_button(x, y){
    this.sprite.width = x;
    this.sprite.height = y;
}

addIcon(textrure){
    Utils.scaleTexture(textrure);
    const icon = new PIXI.Sprite(textrure);
    icon.anchor.set(0.25, 0.25);
    icon.setTransform(icon.width*2-8, (this.button.view.height-icon.height) / 3, Utils.scale-1, Utils.scale-1);

    this.text.x = 16+icon.width;
    this.content.addChild(icon);
}

addEndText(text){
    const style = new PIXI.TextStyle({
        fontFamily: "Courier New",
        fontSize: 8*Utils.scale,
        fontVariant: "small-caps",
        fontWeight: "800"
    });
    const ButtonText = new PIXI.Text(text, style);
    ButtonText.x = this.button.view.width-ButtonText.width-4*Utils.scale;
    ButtonText.y = (this.button.view.height-ButtonText.height) / 2;
    ButtonText.resolution = 10;
    this.endText = ButtonText;
    //ButtonText.anchor.set(0.25, 0);
    this.content.addChild(this.endText);
}

removeEndText(){
    this.content.removeChild(this.endText);
    this.endText = null;
}

addText(text){
    const style = new PIXI.TextStyle({
        fontFamily: "Courier New",
```

```
            fontSize: 8*Utils.scale,
            fontVariant: "small-caps",
            fontWeight: "800"
        });
        const ButtonText = new PIXI.Text(text, style);
        ButtonText.x = 4*Utils.scale;
        ButtonText.y = (this.button.view.height-ButtonText.height) / 2;
        ButtonText.resolution = 10;
        this.text = ButtonText;
        //ButtonText.anchor.set(0.25, 0);
        this.content.addChild(this.text);
    }
}

export {CustomButton}
```

Dialog.js

```
import * as PIXI from 'pixi.js';
import { Button } from '@pixi/ui';
import * as Utils from '../utils'
import { CustomButton } from './Button';

//dialog panel that shows text messages
class Dialog extends CustomButton{
    constructor(textureDefault, texturePressed){
        super(textureDefault, texturePressed);
        this.width = Utils.width-100;
        this.height = Utils.height/3;
        this.scale_button(this.width, this.height);
        this.button.view.setTransform(50, Utils.height-this.height - 50);

        this.text = "";
    }

    //add text stat shows letter by letter, not all together
    async slowText(text) {

        this.content.removeChild(this.text);
        const style = new PIXI.TextStyle({
            fontFamily: "Courier New",
            fontSize: 10 * Utils.scale,
            resolution: window.devicePixelRatio || 1,
            wordWrap: true,
            wordWrapWidth: this.width - 12 * Utils.scale
        });
        const ButtonText = new PIXI.Text('', style);

        ButtonText.x = 16 * Utils.scale;
        ButtonText.y = 6 * Utils.scale;
        this.content.addChild(ButtonText);
        this.text = ButtonText;

        let currentLine = '';
        let index = 0;

        //show letters one by one
        const addLetter = (word, wordIndex) => {
            return new Promise((resolve) => {
                const addLetterRecursive = () => {
                    if (index < word.length) {
                        ButtonText.text += word[index];
```

```
                index++;
                setTimeout(addLetterRecursive, 50);
            } else {
                ButtonText.text += ' ';
                resolve(); //end when full word is shown
            }
        };
        addLetterRecursive();
    });
};

//add word by word
const addWord = async (word, wordIndex) => {
    //if line cant fit into the box
    if ((currentLine.length + word.length) * 10 > style.wordWrapWidth) {
        ButtonText.text += '\n'; //new line
        currentLine = '';
    }
    currentLine += word + ' ';
    index = 0;
    await addLetter(word, wordIndex); //wait for the word to be printed letter by letter
};

const words = text.split(' '); //split the text into words

for (let i = 0; i < words.length; i++) {
    await addWord(words[i], i); //wait for each word to be printed before continuing
}
}

}
export {Dialog}

BattleUI.js
import * as PIXI from 'pixi.js';
import * as Assets from '../Assets';
import * as Utils from '../utils';
import { HealthBar } from './HealthBar';
import { Item } from '../game/Item';
import { Inventory } from '../game/Inventory';
import { createItemList, createText, createDialog } from './uiUtils';

//UI for battle screen
class BattleUI {
    constructor(hero, enemy) {
        this.container = new PIXI.Container();
        this.container.visible = false;

        this.sprite = new PIXI.Sprite(Assets.battleBackground);
        this.sprite.width = Utils.width;
        this.sprite.height = Utils.height;
        this.container.addChild(this.sprite);

        this.hero = hero;
        this.enemy = enemy;
        this.init();
    }

    async init(){
        //player and enemy health bar and info
            this.container.addChild(createText(this.hero.name + " Lvl. " + this.hero.lvl, 16*10*Utils.scale,
16*1*Utils.scale));
```

```
        this.heroHealth = new HealthBar(this.hero.maxHP, this.hero.currentHP, 8*16*Utils.scale, 16*2*Utils.scale, Utils.width/4, Utils.height/25);
        this.container.addChild(this.heroHealth.container);
            this.container.addChild(createText(this.enemy.name + " Lvl. " + this.enemy.lvl, 18*16*Utils.scale, 16*1*Utils.scale));
             this.enemyHealth = new HealthBar(this.enemy.maxHP, this.enemy.currentHP, 15*16*Utils.scale, 16*2*Utils.scale, Utils.width/4, Utils.height/25);
        this.container.addChild(this.enemyHealth.container);

        //list for player actions
        var spells = new Inventory();
        this.hero.actions.forEach(action => {
            var spell1 = new Item(action);
            spells.addItem(spell1);
        });

        [this.attacksButtons, this.attacks] = this.createMenuList(spells);
        this.container.addChild(this.attacks);

        //place hero and enemy sprites on screen
        this.setCharacters();
        await this.createActionDialog(this.enemy.name + " attacks!");
    }

    async playerTurn() {

        this.attacksButtons.forEach((attackButton) => {
            attackButton.button.enabled = true;
            attackButton.button.buttonMode = true;
        });
        var text = "";
        const playerAction = () => {
            //update health after attack
            this.enemyHealth.setCurrentHealth(this.enemy.currentHP, this.enemy.maxHP);
            this.heroHealth.setCurrentHealth(this.hero.currentHP, this.hero.maxHP);

            return new Promise((resolve) => {
                this.attacksButtons.forEach((attackButton, index) => {
                    const handler = () => {
                        attackButton.sprite.texturePressed = attackButton.texture;
                        text = this.hero.attack(this.enemy, index);

                        this.enemyHealth.setCurrentHealth(this.enemy.currentHP, this.enemy.maxHP);
                        this.heroHealth.setCurrentHealth(this.hero.currentHP, this.hero.maxHP);

                        Assets.sound.play('sword');
                        this.animateSprite(this.player)
                        resolve(index);

                    };
                    attackButton.button.onUp.disconnectAll();
                    attackButton.button.onUp.connect(handler);
                    attackButton.setup();
                });
            });
        };

        //wait for player to attack
        await playerAction();
        //display action message
        await this.createActionDialog(text);

        //disable buttons for enemy turn
```

```
    this.attacksButtons.forEach((attackButton) => {
        attackButton.button.enabled = false;
        attackButton.button.buttonMode = false;
    });
}

createMenuList(buttons){
    var actions = new PIXI.Container();

    actions.setTransform(0, 16);
    actions.width = Utils.width/4;
    actions.height = Utils.height/3;

    var act = createItemList(buttons.getItems(), this.container.width, this.container.height);

    act.forEach((action) => {
        actions.addChild(action.button.view);
    })
    actions.visible = true;

    return [act, actions];
}

async setCharacters(){

    this.player = this.hero.sprite;
    this.player.anchor.set(0, 0.25);
    this.container.addChild(this.player);
    this.player.setTransform(Utils.width/2, Utils.height/2, 3+Utils.scale, 3+Utils.scale);

    this.spriteEnemy = new PIXI.AnimatedSprite(this.enemy.spritesheet.animations.stand_down);
    this.spriteEnemy.anchor.set(0, 0.25);
    this.container.addChild(this.spriteEnemy);
    this.spriteEnemy.setTransform(Utils.width/3*2, Utils.height/2, 3+Utils.scale, 3+Utils.scale);
    this.spriteEnemy.play();
    this.spriteEnemy.animationSpeed = 0.08;
}

//animation for shuffling sprites when action is performed
animateSprite(sprite, distance = 10, duration = 400) {
    const startX = sprite.x;
    const halfDuration = duration / 2;
    let elapsed = 0;
    let direction = 1;

    function update(delta) {
        elapsed += delta * (1000 / 60);

        if (elapsed <= halfDuration) {
            sprite.x = startX + direction * distance * (elapsed / halfDuration);
        } else {
            direction *= -1;
            elapsed = 0;
        }

        if (direction === -1 && elapsed >= halfDuration) {
            sprite.x = startX;
            container.ticker.remove(update);
        }
    }

    const container = this.container;
    container.ticker = container.ticker || new PIXI.Ticker();
```

```
        container.ticker.add(update);
        container.ticker.start();
    }


    //add dialog with action message info
    async createActionDialog(text){

        this.dialog = createDialog();
        this.dialog.button.view.setTransform(50, Utils.height/3*2-16);
        this.container.addChild(this.dialog.button.view);

        await this.dialog.slowText(text);
    }

    //screen for battle end (you win or you lost)
    async battleOverScreen(winner) {
        const endScreen = new PIXI.Container();

        const background = new PIXI.Graphics();
        background.beginFill(0x000000, 0.8);
        background.drawRect(0, 0, Utils.width, Utils.height);
        background.endFill();
        endScreen.addChild(background);

        //create title text
        const titleText = new PIXI.Text(winner === 'hero' ? 'You Win!' : 'Defeat', {
            fontFamily: 'Arial',
            fontSize: 64,
            fill: '#ffffff',
            align: 'center'
        });
        titleText.anchor.set(0.5);
        titleText.x = Utils.width / 2;
        titleText.y = Utils.height / 2;
        endScreen.addChild(titleText);

        this.container.addChild(endScreen);
        if(winner === 'hero') Assets.sound.play('quest');
        else Assets.sound.play('fail');
        //show the screen for 3 seconds
        await new Promise(resolve => setTimeout(resolve, 3000));

        this.container.removeChild(endScreen);
    }
}

export{BattleUI}
```

CreateCharacter.js

```
import * as PIXI from 'pixi.js';
import { Input } from '@pixi/ui';
import * as Assets from '../Assets';
import * as Utils from '../utils';
import { createButton, createCustomButton, createItemButton, createBackground, createText } from './uiUtils';

//ui for creating character at new game
class CreateCharacter {
    constructor() {
        this.container = new PIXI.Container();
        this.container.visible = true;
        this.container.setTransform(100, 100);
```

```
        this.container.width = Utils.width-200;
        this.container.height = Utils.height-200;
        this.container.addChild(createBackground());

        this.name = this.createInput();
        this.container.addChild(this.name);
        this.classes = ['Warrior', 'Mage', 'Archer'];

        this.currentCharacter = 0;
        this.characterSprite = this.characterSelect();
        this.container.addChild(this.characterSprite);

        this.currentClass = 0;
        this.ClassText = this.classSelect();
        this.container.addChild(this.ClassText);

            this.leftArrow = createCustomButton(Utils.calculateTileRectangle({x: 10, y: 24}, 16, 16, 2),
Utils.calculateTileRectangle({x: 7, y: 24}, 16, 16, 2));
            this.rightArrow = createCustomButton(Utils.calculateTileRectangle({x: 11, y: 24}, 16, 16, 2),
Utils.calculateTileRectangle({x: 8, y: 24}, 16, 16, 2));
        this.container.addChild(this.leftArrow.button.view, this.rightArrow.button.view);

            this.leftArrow2 = createCustomButton(Utils.calculateTileRectangle({x: 10, y: 24}, 16, 16, 2),
Utils.calculateTileRectangle({x: 7, y: 24}, 16, 16, 2));
            this.rightArrow2 = createCustomButton(Utils.calculateTileRectangle({x: 11, y: 24}, 16, 16, 2),
Utils.calculateTileRectangle({x: 8, y: 24}, 16, 16, 2));
        this.container.addChild(this.leftArrow2.button.view, this.rightArrow2.button.view);
        this.selectButtons();

        this.confirm = createButton();
        this.confirm.addText("Confirm");
        this.confirm.scale_button(this.container.width/3, this.container.height/10);
            this.confirm.button.view.setTransform(this.container.width/2-this.container.width/6, this.container.height -
this.container.height/10*2);
                this.confirm.text.setTransform((this.confirm.button.view.width-this.confirm.text.width)/2,
(this.confirm.button.view.height-this.confirm.text.height)/2);
        this.container.addChild(this.confirm.button.view);
    }

    //wait till create button is pressed
    create(){
        return new Promise((resolve) => {
            this.confirm.button.onUp.connect(() => {
                resolve();
            });
        })
    }

    //select character appearance
    characterSelect(){
        var texture = new PIXI.Texture(Assets.characters[this.currentCharacter], Utils.calculateTileRectangle({x: 0, y:
0}, 16, 16, 1));
        Utils.scaleTexture(texture);
        var sprite = new PIXI.Sprite(texture);
        sprite.x = this.container.width / 2;
        sprite.y = this.container.height / 2;
        sprite.anchor.set(0.5);
        sprite.setTransform( this.container.width / 2, this.container.height / 2, Utils.scale+3, Utils.scale+3);

        return sprite;
    }

    classSelect(){
```

```
        return createText(this.classes[0], this.characterSprite.x, this.characterSprite.y + 16*Utils.scale);
    }

    //functionality for arrow buttons
    selectButtons() {
        this.leftArrow.button.view.x = this.characterSprite.x - 16*2*Utils.scale;
        this.leftArrow.button.view.y = this.characterSprite.y;
        this.rightArrow.button.view.x = this.characterSprite.x + 16*Utils.scale;
        this.rightArrow.button.view.y = this.characterSprite.y;

        this.leftArrow.scale_button(Utils.scale*16, Utils.scale*16);
        this.rightArrow.scale_button(Utils.scale*16, Utils.scale*16);

        this.leftArrow2.button.view.x = this.characterSprite.x - 16*3*Utils.scale;
        this.leftArrow2.button.view.y = this.characterSprite.y+ 16*Utils.scale;
        this.rightArrow2.button.view.x = this.characterSprite.x + 16*2*Utils.scale;
        this.rightArrow2.button.view.y = this.characterSprite.y+ 16*Utils.scale;

        this.leftArrow2.scale_button(Utils.scale*16, Utils.scale*16);
        this.rightArrow2.scale_button(Utils.scale*16, Utils.scale*16);

        this.leftArrow.button.onUp.connect(() => {
            this.currentCharacter = (this.currentCharacter === 0) ? Assets.characters.length - 1 : this.currentCharacter - 1;
            this.updateCharacterSprite();
        });

        this.rightArrow.button.onUp.connect(() => {
             this.currentCharacter = (this.currentCharacter === Assets.characters.length - 1) ? 0 : this.currentCharacter +
1;
            this.updateCharacterSprite();
        });

        this.leftArrow2.button.onUp.connect(() => {
            this.currentClass = (this.currentClass === 0) ? this.classes.length - 1 : this.currentClass - 1;
            this.updateClassText();
        });

        this.rightArrow2.button.onUp.connect(() => {
            this.currentClass = (this.currentClass === this.classes.length - 1) ? 0 : this.currentClass + 1;
            this.updateClassText();
        });
    }

    //update sprite on screen
    updateCharacterSprite() {
        const newTexture = new PIXI.Texture(
            Assets.characters[this.currentCharacter],
            Utils.calculateTileRectangle({ x: 0, y: 0 }, 16, 16, 1)
        );
        Utils.scaleTexture(newTexture);
        this.characterSprite.texture = newTexture;
    }

    //update class name on screen
    updateClassText() {
        const text = this.classes[this.currentClass];
        this.ClassText.text = text;
    }

    //input line for character name
    createInput(){
        const desiredWidth = this.container.width / 3;
        const height = this.container.height/12;
```

```
        let border = 5;
        const input = new Input({
            bg: new PIXI.Graphics()
                .beginFill('#DCB000')
                .drawRoundedRect(0, 0,desiredWidth, height, 10 + border)
                .beginFill('#F1D583')
                .drawRoundedRect(border, border, desiredWidth - (border * 2), height - (border * 2), 10),
            placeholder: 'Enter name',
            padding: 11,
            textStyle: new PIXI.TextStyle({
                fontFamily: "Courier New",
                fontSize: 8*Utils.scale,
                fontVariant: "small-caps",
                fontWeight: "800"
            }),
            maxLength: 15,
        });

        input.x = (this.container.width - desiredWidth) / 2;
        input.y = this.container.height / 12;

        return input;
    }
}

export { CreateCharacter };
```

HealthBar.js

```
import * as PIXI from 'pixi.js';
import * as Utils from '../utils.js'

//health and exp bar (can be any color)
class HealthBar {
    constructor(maxHealth, currentHealth, x, y, width, height, color=0x00FF00) {
        this.maxHealth = maxHealth;
        this.currentHealth = currentHealth;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;

        this.container = new PIXI.Container();
        this.container.position.set(x, y);

        //grey bg
        this.background = new PIXI.Graphics();
        this.background.beginFill(0x999999);
        this.background.drawRoundedRect(0, 0, width, height, 10);
        this.background.endFill();
        this.container.addChild(this.background);

        //border
        this.border = new PIXI.Graphics();
        this.border.lineStyle(2, 0x000000);
        this.border.drawRoundedRect(0, 0, width, height, 10);//round
        this.container.addChild(this.border);

        const style = new PIXI.TextStyle({
            fontFamily: "Courier New",
            fontSize: 8*Utils.scale,
            fontVariant: "small-caps",
```

```
            fontWeight: "800"
        });
        this.textHealth = new PIXI.Text(this.currentHealth+"/"+this.maxHealth, style);
        this.textHealth.x = (4+16)*Utils.scale;
        this.textHealth.y = 0*Utils.scale;
        this.textHealth.resolution = 10;
        this.textHealth.zIndex = 2;

        //colored bg
        this.healthBar = new PIXI.Graphics();
        this.updateHealthBar();
        this.container.addChild(this.healthBar);
        this.container.addChild(this.textHealth);

    }

    setColor(color){
        this.color = color;
    }

    updateHealthBar() {
        //count bar length
        const healthPercentage = this.currentHealth / this.maxHealth;
        const barWidth = Math.max(0, this.width * healthPercentage);

        //redraw
        this.healthBar.clear();
        this.healthBar.beginFill(this.color);
        this.healthBar.drawRoundedRect(1, 1, barWidth-2, this.height-2, 10);
        this.healthBar.endFill();


        this.textHealth.text = this.currentHealth+"/"+this.maxHealth;
    }

    setCurrentHealth(newHealth, newMaxHealth) {
        //update healthbar
        this.maxHealth = newMaxHealth;
        this.currentHealth = Math.max(0, Math.min(newHealth, newMaxHealth));
        this.updateHealthBar();
    }
}
export {HealthBar}
```

Journal.js

```
import * as PIXI from 'pixi.js';
import * as Assets from '../Assets.js'
import { createBackground, createButton, createScrollBox, createItemButton, createItemList, createText,
createCloseButton } from './uiUtils';
import * as Utils from '../utils';
import { Item } from '../game/Item';
import { Inventory } from '../game/Inventory';

//info about quests
class Journal {
    constructor(hero) {
        this.hero = hero;
        this.container = new PIXI.Container();
        this.container.visible = false;
        this.container.setTransform(100, 100);
        this.container.width = Utils.width - 200;
        this.container.height = Utils.height - 200;
```

```
        this.container.addChild(createBackground());

        this.close = createCloseButton(this.container.width);
        this.container.addChild(this.close.button.view);
        this.close.button.onPress.connect(() => {
            this.container.visible = false;
            Assets.sound.play('openMenu');
        });

        this.container.addChild(createText("Journal", this.container.width / 2, 16));

        this.questInfoContainer = new PIXI.Container();
        this.questInfoContainer.setTransform(this.container.width / 2, 100);
        this.questInfoContainer.width = this.container.width / 2;
        this.questInfoContainer.height = Utils.height - 200;
        this.questInfoContainer.addChild(createBackground(this.container.width / 3, this.container.height - 200));
        this.container.addChild(this.questInfoContainer);

        var active = new Item('Active');
        var completed = new Item('Complete');

            this.activeButton = createItemButton(active.text, active.Texture, 100, 32, this.container.width/2, 100*Utils.scale);
          this.completedButton = createItemButton(completed.text, completed.Texture, this.container.width/5+50, 32, this.container.width/2+50, 100*Utils.scale);

        this.activeButton.button.onUp.connect(()=>{
            this.createQuestList(this.hero.questManager.activeQuests);
        })
        this.completedButton.button.onUp.connect(()=>{
            this.createQuestList(this.hero.questManager.completedQuests);
        })
        this.container.addChild(this.activeButton.button.view, this.completedButton.button.view);
        this.createQuestList();
    }

    createQuestList(activeQuests = this.hero.questManager.activeQuests) {
        //refresh quest list
        if (this.questList) {
            this.container.removeChild(this.questList);
        }
        this.questInfoContainer.removeChildren();

        this.activeQuestsInventory = new Inventory();
        activeQuests.forEach(quest => {
            const questItem = new Item(quest.name);
            this.activeQuestsInventory.addItem(questItem);
        });

        //buttons for each ques
        const items = createItemList(this.activeQuestsInventory.getItems(), this.container.width, this.container.height);
        const buttons = [];
        items.forEach((item, index) => {
            const button = item.button.view;
            item.button.onPress.connect(() => {
                this.displayQuestInfo(activeQuests[index]);//show detailed info onclick
            });
            buttons.push(button);
        });

        this.questList = createScrollBox(buttons, this.container.width / 3 + 16, this.container.height - 200);
        this.questList.x = 100;
        this.questList.y = 100;
```

```
        this.container.addChild(this.questList);
    }

    displayQuestInfo(quest) {
        this.questInfoContainer.removeChildren();

        this.questInfoContainer.addChild(createBackground(this.container.width / 3, this.container.height - 200));

        this.questInfoContainer.addChild(createText(quest.name, this.questInfoContainer.width / 2, 16));

        quest.objectives.forEach((objective, index) => {
            const objectiveText = `${index + 1}. ${objective.type} - ${objective.target}\n`;
            this.questInfoContainer.addChild(createText(objectiveText, 3*16*Utils.scale, (index + 1) * 32*2));
        });

        let rewardText = 'Rewards:\n';
        rewardText += `Gold:  ${quest.rewards.gold},\n`;
        rewardText += `Exp:  ${quest.rewards.exp},\n`;
        this.questInfoContainer.addChild(createText(rewardText, 16*Utils.scale, (quest.objectives.length + 2) * 32));

        this.questInfoContainer.addChild(createText("Status: " + quest.status, 2*16*Utils.scale, (quest.objectives.length
+ 5) * 32));
    }

}

export { Journal };
```

ScreenUI.js

```
import * as PIXI from 'pixi.js';
import {createText } from './uiUtils';
import * as Utils from '../utils';
import {HealthBar} from './HealthBar'

//screen ui - health bar, exp bar, money
class ScreenUI{
    constructor(hero) {
        this.hero = hero;
        this.container = new PIXI.Container();

        this.barWidth = Utils.width/6;
        this.barHeight = Utils.height/25;
                this.health = new HealthBar(this.hero.maxHP, this.hero.currentHP, Utils.width-this.barWidth-16,
1*16+0*this.barHeight, this.barWidth, this.barHeight);
                    this.exp = new HealthBar(this.hero.nextLvlExp, this.hero.exp, Utils.width-this.barWidth-16,
2*16+1*this.barHeight, this.barWidth, this.barHeight, 0x1099bb);
        this.money = createText(this.hero.money + "G",  Utils.width-16*Utils.scale, 3*16+2*this.barHeight)

        this.container.addChild(this.health.container, this.exp.container, this.money);
    }

    //update stats and ui
    update(){
        this.money.text = this.hero.money + "G";
        if(this.health.currentHealth != this.hero.currentHP || this.health.maxHealth != this.hero.maxHP)
            this.health.setCurrentHealth(this.hero.currentHP, this.hero.maxHP);

        if(this.exp.currentHealth != this.hero.exp || this.exp.maxHealth != this.hero.nextLvlExp)
            this.exp.setCurrentHealth(this.hero.exp, this.hero.nextLvlExp);
    }
}
```

```
export {ScreenUI}
```

ShopUI.js

```
import * as PIXI from 'pixi.js';
import { createBackground, createCloseButton, createItemList, createText, createScrollBox } from './uiUtils';
import * as Utils from '../utils';
import * as Assets from '../Assets'

//shop UI
class ShopUI {
   constructor(shop, hero) {
      this.container = new PIXI.Container();
      this.container.visible = true;
      this.container.setTransform(100, 100);
      this.container.width = Utils.width - 200;
      this.container.height = Utils.height - 200;
      this.container.addChild(createBackground());

      this.close = createCloseButton(this.container.width);
      this.container.addChild(this.close.button.view);
      this.closeButtonPromise = new Promise((resolve) => {
         this.close.button.onPress.connect(() => {
            this.container.visible = false;
            resolve();
         });
      });

      this.container.addChild(createText("Buy", 64, 16));
      this.container.addChild(createText("Sell", this.container.width / 2 + 64, 16));

      this.shop = shop;
      this.hero = hero;

      this.updateUI();
   }

   updateUI() {
      if (this.inventory) this.container.removeChild(this.inventory);
      if (this.inventory2) this.container.removeChild(this.inventory2);
      if (this.shopMoneyText) this.container.removeChild(this.shopMoneyText);
      if (this.heroMoneyText) this.container.removeChild(this.heroMoneyText);

      const shopInventory = this.shop.inventory.getItems();
      const heroInventory = this.hero.inventory.getItems();
      console.log(shopInventory);
      console.log(heroInventory);
      //add seller and layers inventory
      const shopList = createItemList(shopInventory, this.container.width, this.container.height);
      const heroList = createItemList(heroInventory, this.container.width, this.container.height);

      const shopButtons = [];
      const heroButtons = [];

      //add prices
      shopList.forEach((item, index) => {
         shopButtons.push(item.button.view);
         item.button.onPress.connect(() => this.buyItem(shopInventory[index]));
         item.addEndText(shopInventory[index].buyPrice + "G");
      });

      heroList.forEach((item, index) => {
         heroButtons.push(item.button.view);
```

```
            item.button.onPress.connect(() => this.sellItem(heroInventory[index]));
            item.addEndText(heroInventory[index].sellPrice + "G");
        });

        //money and items lists
         this.shopMoneyText = createText("Shop money: " + this.shop.money + "G", 200, this.container.height - 16 *
Utils.scale);
          this.heroMoneyText = createText("Your money: " + this.hero.money + "G", this.container.width / 2 + 200,
this.container.height - 16 * Utils.scale);
        this.container.addChild(this.shopMoneyText);
        this.container.addChild(this.heroMoneyText);

        this.inventory = createScrollBox(shopButtons, this.container.width / 3 + 16, this.container.height - 128);
        this.inventory.x = 32;
        this.inventory.y = 64;
        this.container.addChild(this.inventory);

        this.inventory2 = createScrollBox(heroButtons, this.container.width / 3 + 16, this.container.height - 128);
        this.inventory2.x = this.container.width / 2 + 32;
        this.inventory2.y = 64;
        this.container.addChild(this.inventory2);
    }

    async  buyItem(item) {
        //remove item from shop
        if(this.hero.money>=item.buyPrice){
            Assets.sound.play('money');
            this.shop.inventory.removeItem(item);
            this.hero.inventory.addItem(item);

            this.hero.money -= item.buyPrice;
            this.shop.money += item.buyPrice;
        }
        else{
            await this.flashMoneyText(this.heroMoneyText);
        }
        this.updateUI();
    }

    async sellItem(item) {
        //remove item from players inventory
        if(this.shop.money>=item.sellPrice){
            Assets.sound.play('money');
            this.hero.inventory.removeItem(item);
            this.shop.inventory.addItem(item);

            this.shop.money -= item.sellPrice;
            this.hero.money += item.sellPrice;
        }
        else{
            await this.flashMoneyText(this.shopMoneyText);
        }
        this.updateUI();
    }

    //highlight money red if not enough
    async flashMoneyText(textElement) {
        textElement.style.fill = 'red';
        await this.wait(1000);
        textElement.style.fill = 'black';
    }

    wait(ms) {
```

```
        return new Promise(resolve => setTimeout(resolve, ms));
    }

    waitForClose() {
        return this.closeButtonPromise;
    }
}

export { ShopUI };
```

Stats.js

```
import * as PIXI from 'pixi.js';
import { createBackground, createScrollBox, createCloseButton, createItemButton, createItemList, createText } from
'./uiUtils';
import * as Assets from '../Assets';
import * as Utils from '../utils';

//player stats and inventory
class Stats {
    constructor(hero) {
        this.hero = hero;
        this.container = new PIXI.Container();
        this.container.visible = false;
        this.container.setTransform(100, 100);
        this.container.width = Utils.width - 200;
        this.container.height = Utils.height - 200;
        this.container.addChild(createBackground());

        this.close = createCloseButton(this.container.width);
        this.container.addChild(this.close.button.view);
        this.close.button.onPress.connect(() => {
            this.container.visible = false;
            Assets.sound.play('openMenu');
        });

        this.container.addChild(createText("Player Stats", this.container.width / 2, 16));
        this.addStats(); //renew player stats
    }

    addStats() {
        if (this.inventory) this.container.removeChild(this.inventory);

        const items = createItemList(this.hero.inventory.getItems(), this.container.width, this.container.height);
        const buttons = [];
        //load inventory
        var heroItems = this.hero.inventory.getItems();
        items.forEach((button, index) => {
            const isEquipped = Object.values(this.hero.equippedItems).includes(heroItems[index]);

            if (isEquipped) {
                button.addEndText('E');
            }
        });
        items.forEach((button, index) => {
            button.button.onPress.connect(() => {
                const item = heroItems[index];
                const previousEquippedItem = this.hero.toggleEquip(item);

                //if equipped
                const isEquipped = Object.values(this.hero.equippedItems).includes(item);

                if (isEquipped) {
```

```
                button.addEndText('E');
                if (previousEquippedItem) {
                    //unequip previous item
                    const previousIndex = heroItems.findIndex(i => i === previousEquippedItem);
                    if (previousIndex !== -1) {
                        items[previousIndex].removeEndText();
                    }
                }
            } else {
                button.removeEndText();
            }

            this.updateStatsDisplay();
        });
        buttons.push(button.button.view);
    });


    this.inventory = createScrollBox(buttons, this.container.width / 3 + 16, this.container.height - 200);
    this.inventory.x = 100;
    this.inventory.y = 100;
    this.container.addChild(this.inventory);

    this.stat = new PIXI.Container();
    this.stat.setTransform(this.container.width / 2, 100);
    this.stat.width = this.container.width / 2;
    this.stat.height = Utils.height - 200;
    this.stat.addChild(createBackground(this.container.width / 3, this.container.height - 200));
    this.container.addChild(this.stat);

    this.updateStatsDisplay();
  }

  updateStatsDisplay() {
    //update info
    this.stat.removeChildren();
    this.stat.addChild(createBackground(this.container.width / 3, this.container.height - 200));

    var sprite = new PIXI.Sprite(this.hero.sprite.texture);
    sprite.setTransform(0, this.stat.height / 3);
    sprite.anchor.set(0, 0.25);
    Utils.scaleTexture(sprite.texture);
    sprite.scale.set(Utils.scale * 2, Utils.scale * 2);
    this.stat.addChild(sprite);

    this.stat.addChild(createText(this.hero.name, this.stat.width / 2, 0));

    const startY = 8 * Utils.scale;
    const verticalOffset = 8 * Utils.scale;

    // player stats
    this.stat.addChild(createText("LVL  " + this.hero.lvl, this.stat.width / 3 * 1.5, startY + verticalOffset * 1, false));
    this.stat.addChild(createText("Atk  " + this.hero.atk, this.stat.width / 3 * 1.5, startY + verticalOffset * 2, false));
    this.stat.addChild(createText("Sp. Atk  " + this.hero.spAtk, this.stat.width / 3 * 1.5, startY + verticalOffset * 3,
false));
    this.stat.addChild(createText("Def  " + this.hero.def, this.stat.width / 3 * 1.5, startY + verticalOffset * 4, false));
    this.stat.addChild(createText("Sp. Def  " + this.hero.spDef, this.stat.width / 3 * 1.5, startY + verticalOffset * 5,
false));
  }
}

export { Stats };
```

ДОДАТОК В

Код серверної частини

Server.js

```
import express from 'express';
import mongoose from 'mongoose';
import bcrypt from 'bcryptjs';
import session from 'express-session';
import bodyParser from 'body-parser';
import cors from 'cors';

const app = express();

app.use(bodyParser.json());
app.use(cors({
   origin: 'http://localhost:5173',
   credentials: true
}));

app.use(session({
   secret: 'your_secret_key',
   resave: false,
   saveUninitialized: false,
   cookie: { secure: false }
}));

//connect mongodb
mongoose.connect('mongodb+srv://santorcism:melovin18@chattest.dfclsqt.mongodb.net/game', {
   useNewUrlParser: true,
   useUnifiedTopology: true
})
.then(() => console.log('MongoDB Connected'))
.catch(err => console.log(err));

// user model
const UserSchema = new mongoose.Schema({
   username: { type: String, required: true, unique: true },
   email: { type: String, required: true, unique: true },
   password: { type: String, required: true },
   date: { type: Date, default: Date.now }
});

const User = mongoose.model('User', UserSchema);

//game save data model
const GameSaveSchema = new mongoose.Schema({
   userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
   gameData: { type: Object, required: true },
   date: { type: Date, default: Date.now }
});

const GameSave = mongoose.model('GameSave', GameSaveSchema);

//routes and functions
app.post('/register', async (req, res) => {
   const { username, email, password } = req.body;
```

```
    //if user already exists
    const userExists = await User.findOne({ email });
    if (userExists) {
        return res.status(400).json({ message: 'User already exists' });
    }

    const newUser = new User({
        username,
        email,
        password
    });

    //hash the password
    bcrypt.genSalt(10, (err, salt) => {
        bcrypt.hash(newUser.password, salt, (err, hash) => {
            if (err) throw err;
            newUser.password = hash;

            newUser.save()
                .then(user => res.json({ message: 'User registered successfully', user }))
                .catch(err => res.status(500).json({ error: err.message }));
        });
    });
});

app.post('/login', async (req, res) => {
    const { email, password } = req.body;

    //if user exists
    const user = await User.findOne({ email });
    if (!user) {
        return res.status(400).json({ message: 'Invalid email or password' });
    }

    //compare passwords
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
        return res.status(400).json({ message: 'Invalid email or password' });
    }

    req.session.user = {
        id: user._id,
        username: user.username
    };
    console.log(req.session);
    res.json({ message: 'Login successful', user: req.session.user });
});

app.post('/logout', (req, res) => {
    req.session.destroy(err => {
        if (err) {
            return res.status(500).json({ message: 'Logout failed' });
        }
        res.json({ message: 'Logout successful' });
    });
});

// app.get('/protected', (req, res) => {
//     if (!req.session.user) {
//         return res.status(401).json({ message: 'Unauthorized' });
//     }

//     res.json({ message: 'This is a protected route', user: req.session.user });
```

```
// });

app.post('/save-game', async (req, res) => {
    console.log(req.session);
    if (!req.session.user) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    const { gameData } = req.body;
    try {
        const existingSave = await GameSave.findOne({ userId: req.session.user.id });

        if (existingSave) {
            existingSave.gameData = gameData;
            await existingSave.save();
        } else {
            const newSave = new GameSave({
                userId: req.session.user.id,
                gameData
            });
            await newSave.save();
        }

        res.json({ message: 'Game saved successfully' });
    } catch (err) {
        res.status(500).json({ message: 'Failed to save game', error: err.message });
    }
});

app.get('/load-game', async (req, res) => {
    if (!req.session.user) {
        return res.status(401).json({ message: 'Unauthorized' });
    }

    try {
        const save = await GameSave.findOne({ userId: req.session.user.id });
        if (!save) {
            return res.status(404).json({ message: 'No saved game found' });
        }

        res.json({ gameData: save.gameData });
    } catch (err) {
        res.status(500).json({ message: 'Failed to load game', error: err.message });
    }
});

//run server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Register.js

```
import axios from "axios";
axios.defaults.baseURL = 'http://localhost:5000';

const registerForm = document.getElementById('register-form');
const registerMessage = document.getElementById('register-message');
axios.defaults.withCredentials = true;

//register
registerForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const username = document.getElementById('register-username').value;
```

```
    const email = document.getElementById('register-email').value;
    const password = document.getElementById('register-password').value;

    try {
        //request for register
        const response = await axios.post('/register', { username, email, password });
        registerMessage.textContent = response.data.message;

        //redirect to login
        if (response.data.user) {
            window.location.href = 'login.html';
        }
    } catch (error) {
        registerMessage.textContent = error.response ? error.response.data.message : 'Error registering user';
    }
});
```

Login.js

```
import axios from "axios";
axios.defaults.baseURL = 'http://localhost:5000';
axios.defaults.withCredentials = true;

const loginForm = document.getElementById('login-form');
const loginMessage = document.getElementById('login-message');
//login
loginForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const email = document.getElementById('login-email').value;
    const password = document.getElementById('login-password').value;

    try {
        //request for login
        const response = await axios.post('/login', { email, password });
        if (response.data.user) {
            localStorage.setItem('isLoggedIn', 'true');
            localStorage.setItem('username', response.data.user.username);
            //redirect to game
            window.location.href = 'load.html';
        } else {
            loginMessage.textContent = response.data.message;
        }
    } catch (error) {
        loginMessage.textContent = error.response ? error.response.data.message : 'Error logging in';
    }
});
```

Load.js

```
import axios from "axios";
axios.defaults.baseURL = 'http://localhost:5000';
axios.defaults.withCredentials = true;

document.getElementById('load-game').addEventListener('click', async () => {
    try {
        //request to lad gamedata
        const response = await axios.get('/load-game', { withCredentials: true });

        if (response.data.gameData) {
            //save to localstorage, redirect to game and pass loaded data
            localStorage.setItem('gameData', JSON.stringify(response.data.gameData));
            window.location.href = 'game.html';
        } else {
```

```
            document.getElementById('game-message').textContent = 'No saved game found';
        }
    } catch (error) {
        console.log(error);
        document.getElementById('game-message').textContent = 'Error loading game';
    }
});

//start new game
document.getElementById('new-game').addEventListener('click', () => {
    localStorage.removeItem('gameData');
    window.location.href = 'game.html';
});
```

Game.js

```
import axios from "axios";
axios.defaults.baseURL = 'http://localhost:5000';
axios.defaults.withCredentials = true;
import * as Utils from './utils.js';

const logoutButton = document.getElementById('logout');
const saveButton = document.getElementById('save-game');

//check if user is logged in
window.addEventListener('load', async () => {
    const isLoggedIn = localStorage.getItem('isLoggedIn') === 'true';

    if (!isLoggedIn) {
        window.location.href = 'login.html';
    } else {
        //load gamedata if its existing
        try {
            const gameData = localStorage.getItem('gameData') ? JSON.parse(localStorage.getItem('gameData')) : null;

            if (gameData) {
                //send data to game
                const mainModule = await import('../main.js');
                if (mainModule && mainModule.setup) {
                    mainModule.setup(gameData);
                }
            } else {
                //start new game if no save data
                const mainModule = await import('../main.js');
                if (mainModule && mainModule.setup) {
                    mainModule.setup();
                }
            }
        } catch (error) {
            console.error('Error loading main.js:', error);
        }
    }
});

//logout
logoutButton.addEventListener('click', () => {
    localStorage.removeItem('isLoggedIn');
    localStorage.removeItem('username');
    window.location.href = 'login.html';
});

//save game
saveButton.addEventListener('click', async () => {
```

```
    try {
        const gameData = Utils.save;

        const response = await axios.post('/save-game', { gameData });
        if (response.data.message) {
            alert(response.data.message);
        }
    } catch (error) {
        console.log(error)
        alert('Failed to save the game' , error);
    }
});
```

App.js

```
//called after start game button pressed
document.getElementById('start').addEventListener('click', () => {
    //check if user logged in
    const isLoggedIn = localStorage.getItem('isLoggedIn') === 'true';

    if (isLoggedIn) {
        window.location.href = 'game.html'; //redirect to game if logged in
    } else {
        window.location.href = 'login.html'; //redirect to login page if not
    }
});
```