МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

До захисту допускається:
завідувач кафедри ІТП
Лифар В.
«_____» _____2023 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА
ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

Інтерактивна гра
для навчання

Освітній ступінь – «бакалавр»

Спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Керівник проекту: _____ Доцент к. т. н. Іванов В.Г.
(підпис) (ініціали, прізвище)

Здобувач вищої освіти: _____ Ємелін В.Д.
(підпис) (ініціали, прізвище)

Група: ІПЗ-19д

Київ 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Інформаційних технологій та програмування
Освітній ступінь бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
завідувач кафедри ІТП
Лифар В.О.
«_____» _____2023 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА

Ємеліна Владислава Денисовича
(прізвище, ім'я, по батькові)

1.Тема роботи: Інтерактивна гра для навчання

Керівник проекту (роботи) Іванов В. Г., к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "26 " 04 2023 р. № 240/15-ОД

2.Термін подання роботи здобувача вищої освіти 05.06.2023

3. Вихідні дані до роботи матеріали переддипломної практики.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): а) аналіз предметної області;
б) вибір програмних засобів для розробки продукту;
в) розробка продукту;
г) висновки.

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

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання 24.03.2023

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Ознайомлення з проблематикою досліджуваної галузі	03.04.2023-24.04.2023	
2	Аналіз існуючих ігор	25.04.2023-07.05.2023	
3	Ознайомлення з необхідною літературою	03.05.2023-03.06.2023	
4	Розробка базової частини інтерфейсу	08.05.2023-20.05.2023	
5	Написання основних функцій гри	20.05.2023-28.05.2023	
6	Тестування фінальної версії продукту	28.05.2023-30.05.2023	
7	Оформлення пояснювальної записки	30.05.2023-04.06.2023	

Здобувач вищої освіти _____ Ємелін. В.Д.
(підпис) (ініціали, прізвище)

Керівник _____ Іванов В.Г.
(підпис) (ініціали, прізвище)

Р Е Ф Е Р А Т

Пояснювальна записка до дипломного проєкту (роботи) бакалавра: 67 с., 35 мал., 15 бібліографічних джерел, 1 додаток.

Об'єкт розробки: інтерактивна гра у жанрі Rogue-like RPG з елементами навчання.

Мета роботи: методом дослідження можливостей фреймворку Rot.js розробити інтерактивну браузерну гру у жанрі Rogue-like RPG з текстовим інтерфейсом та елементами навчання. Реалізувати проєкт з використанням HTML, CSS та JavaScript.

В проєкті виконано:

1. Аналіз процесу розробки інтерактивної гри даного жанру, аналіз ігор-вчителів.
2. Аналіз існуючих проєктів на основі фреймворку Rot.js та аналогічних фреймворків для мови програмування JavaScript.
3. Відтворення інтерфейсу проєкту для запуску в браузері засобами мов HTML та CSS.
4. Реалізація функціональної частини проєкту за допомогою мови JavaScript та фреймворку Rot.js.
5. Впровадження рішень ігрового дизайну для проєкту, створення базових елементів ігрового світу, створення ігрового балансу, впровадження процесу навчання в процес гри тощо.

JAVASCRIPT, HTML, CSS, ROT.JS, ВЕБ-СТОРІНКА, БРАУЗЕР, ІНТЕРАКТИВНА ГРА, RPG, ROGUE-LIKE

ЗМІСТ

ВСТУП

Ігрова індустрія як один з економічних секторів, напряму пов’язаний з розробкою та просуванням програмного забезпечення (далі – ПЗ), зародилася ще в середині 70-х років, і з тих пір виросла з невеликих локальних ринків в мейнстрімну індустрію, яка тільки за останні 3 роки вдвічі виростила свій економічний потенціал. Через конкуренцію та запровадження нових технологій в сфері персональних комп’ютерів, необхідність в розробці ПЗ для цієї індустрії зростає разом з її річним прибутком, таким чином цей ринок створює безліч нових робочих місць для усіх різновидів спеціальностей в сфері інформаційних технологій. Перспективним напрямком є створення розвиваючих ігор, що можна прослідкувати на прикладі багатьох проєктів, які були не предзначені для навчання, але адаптовані розробниками для використання в школах, університетах тощо (наприклад Minecraft). В жанрі rogue-like, який став помітно популярніше серед молоді за останні 5 років кількість ігор-вчителів дуже обмежена. Рішення цієї проблеми і є метою цього дослідження.

Суть проєкту полягає у створенні застосунку, який за допомогою веб-сторінки браузеру буде виконувати функцію інтерфейсу інтерактивної гри на основі популярного в сучасності жанру Rogue-like RPG та мати в собі розвиваючі елементи, які можуть бути корисні для основної авдиторії обраного жанру – тобто студентам та школярам. Відображення ігрового світу буде виконуватися за допомогою тексту, таким чином виграна економія ресурсів персонального комп’ютеру дозволить користуватися застосунком на слабких за продуктивністю машинах.

Виконання цієї роботи буде проводитися в середі для розробки веб-застосунків з використанням необхідних для відображення інтерфейсу та реалізації функціональної частини гри мов програмування та обраних бібліотек та/або фреймворків.

Слідуючи за метою дослідження, фінальний продукт повинен виконувати функції інтерактивної гри з елементами навчання та буде відповідати умовам нижче:

- Інтерактивна гра буде реалізована в межах веб-браузеру.
- Ігровий процес буде мати ціль та прогрес до цілі.
- Мета ігрового процесу – навчити користувача новій інформації.
- Застосунок буде використовувати обмежено невелику кількість програмних ресурсів персонального комп’ютеру.
- Веб-сторінка буде коректно відображатися в різних веб-браузерах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Ігрова індустрія одна з провідних галузей-спонсорів інформаційних технологій і займає велику частину інформаційного ринку багатьох країн світу, в тому числі й України. За даними звіту компанії Newzoo світовий ринок відеоігор (враховується дохід від продажу) на початку 2022 року становив 180,3 мільярда доларів США, що демонструє значну динаміку з двохтисячних років 21-ого століття та дає привід вважати, що розробка ігрових застосунків це перспективна спеціалізація, яка непомірно важлива на сьогодні через постійний розвиток підходу до розробки відеоігор.

Один з перспективних але поки недосліджених напрямків розвитку ігрової індустрії це ігри-вчителі, які заохочують увагу користувачів простим та цікавим ігровим дизайном, та пропонують поєднати розваги та навчання.

На сьогодні кількість проєктів такого типу дуже обмежена, особливо в жанрі Rogue-like, який стає все популярніше серед молоді: студентів та школярів. Через збільшення складності навчального процесу в школах та університетах, все популярніше стають додаткові способи мотивації студентів до позашкільного навчання. Одним з таких способів може стати подібний проєкт, та зайняти нішу серед такого специфічного продукту, як rogue-like вчитель.

Цей дипломний проєкт націлений на дослідження процесу розробки та реалізацію браузерної відеогри з метою поєднання ігрового дизайну з процесом навчання в спільний продукт.

1.2 Аналіз процесу розробки інтерактивної гри

Перед аналізом процесу розробки даної інтерактивної гри необхідно визначити, що мається на увазі під Rogue-like RPG та гра-вчитель.

Rogue-like – піджанр рольових відеоігор (далі – RPG) головними особливостями якого зауважуються випадкове (або процедурне) створення світу, генерація ворогів, артефактів (ігрових предметів) тощо. Частіше за все в іграх подібного жанру реалізований покроковий ігровий процес та перманентність таких дій, як «смерть» ворогів або гравця. Назва «Rogue-подібні ігри» походить від початківця піджанру, який вийшов в світ у 1980-ому році під назвою Rogue.

Частіше за все ігри подібного жанру мають ряд технічних особливостей, від чого можна відштовхуватися під час розробки ігрового застосунку. Це такі особливості, як:

- Реалізація ігрового процесу в консольному застосунку
- Генерація ігрової графіки та інтерфейсу за допомогою ASCII символів
- Керування ігровим інтерфейсом виключно клавіатурою
- Мінімальне (або відсутнє) використання звуків або музики

Хоча дизайн ігрового процесу не входить в мету даного дослідження, його можливо реалізувати на основі класичних Rogue-like ігор та інших представників жанру RPG.

Грою-вчителем в даному випадку є гра, фінальною метою якої для здобувача стає засвоєння якоїсь інформації, яку гра передає через специфічний ігровий дизайн, який не властивий іншим іграм. Представників таких ігор для персональних комп’ютерів можна перерахувати по пальцях, більшість з них розраховано на дошкільну авдиторію, та допомагає вчити базові знання та навички. Проєктів, які розраховані на більш дорослу авдиторію дуже мало, та їх більшість сконцентрована в секторі мобільного геймінгу.

Щоб запровадити ігровий процес по’єднаний з навчальним, необхідно не стільки відштовхуватись від технічних засобів та рішеннь, які можуть

знадобитися в процесі, скільки від ігрового дизайну, балансу гри та реалізації навчального процесу тощо. Тому огляд існуючих проєктів важливий тільки як спосіб пошуку технічних рішень для подібної гри. Ігровий дизайн буде здійснюватися за допомогою спеціалізованої літератури та відштовхуватися від власних спостережень за тим, як зацікавити в навчанні школярів та студентів (хто є основною авдиторію ігор жанру rogue-like).

1.2.1 Огляд існуючих rogue-like проєктів

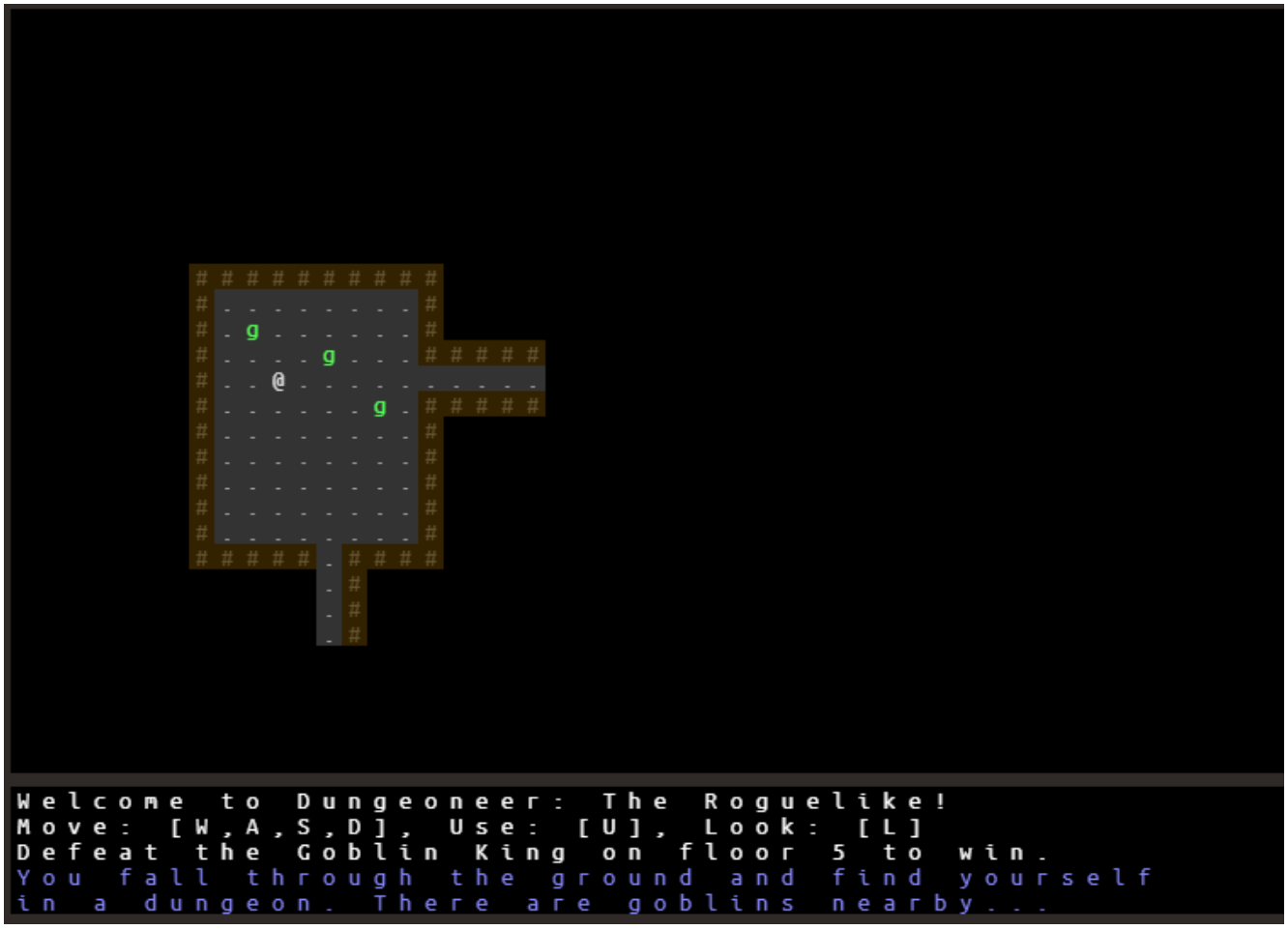
Для успішної розробки проєкту необхідно розглянути аналогічні даному застосунки, для реалізації яких використовувався фреймворк Rot.js. Було обрано три застосунки, два з яких були розроблені на обраному фреймворку, третій був обраний незалежно від використаних засобів розробки, для дослідження ігрового дизайну Rogue-like ігор. Було вирішено не розглядати розвиваючі ігри для персонального комп’ютеру через їх примітивність, та натомість обрати за референс до проєкту настільні ігри з елементами навчання.

1.2.2 «Dungeoneer: The Roguelike» (Rot.js)

Один з найпопулярніших ігрових продуктів на фреймворку Rot.js це «Dungeoneer: The Roguelike». Цей проєкт також можна вважати останньою версією «Rogue» - першої Rogue-like гри, де були збережені більшість автентичних елементів ігрових та технічних рішень. Одне з таких рішень це відкритий код.

Відкритим кодом (open source code) прийнято називати модель розробки різноманітного ПЗ (не тільки ігрового продукту), де вихідний код проєкту вільно ділиться серед користувачів, і кожний може адаптувати код під свої потреби та поширювати його у зміненому вигляді без ризиків потрапити під авторське право.

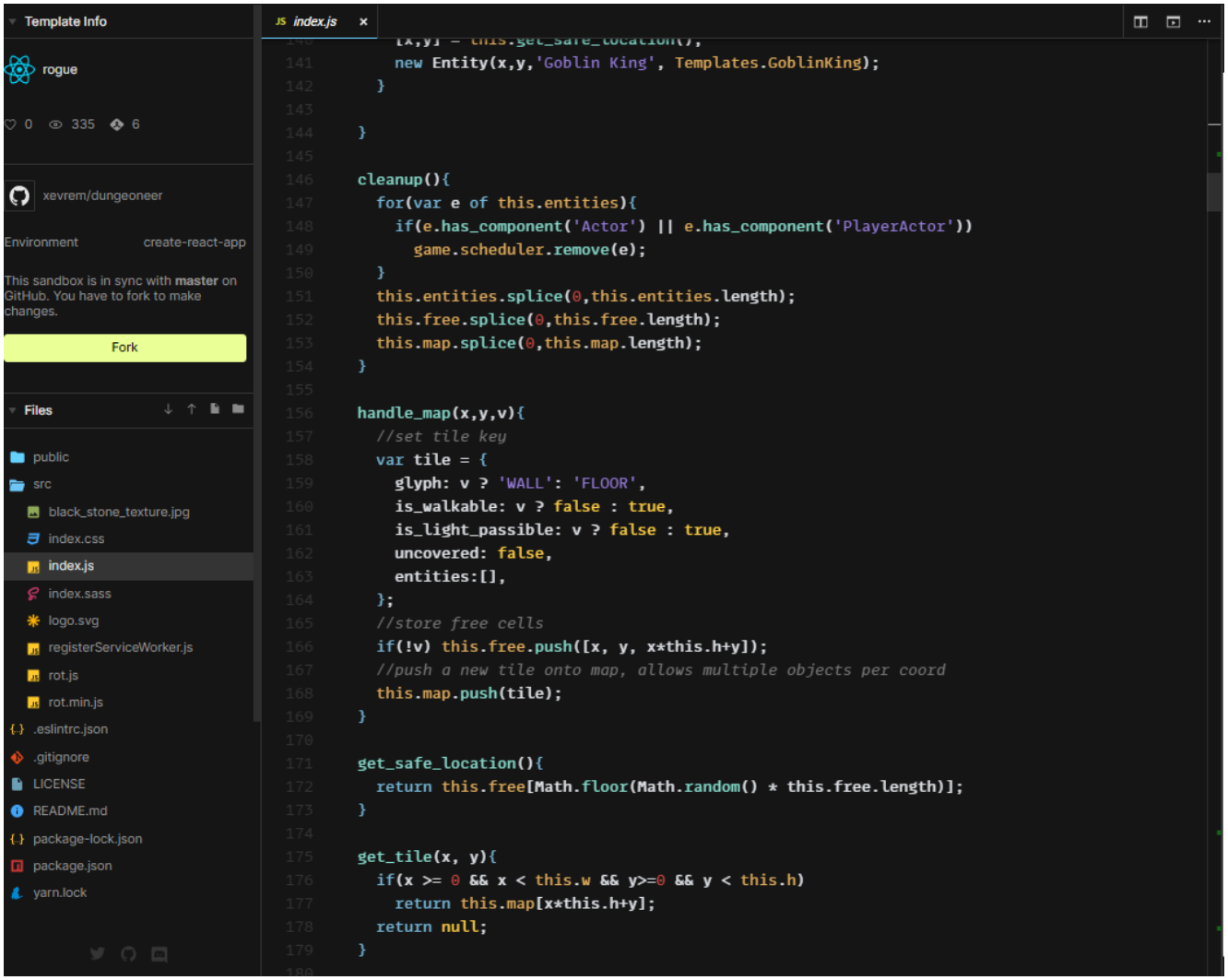
Інтерфейс Dungeoneer (малюнок 1.1) реалізований в імітованому під консоль середовищі, але саме ПЗ адаптовано під запуск на персональному комп’ютері як окремий застосунок, так і як веб-сторінка в браузері, що надає продукту кросс-платформерність та зручність в використанні.



Малюнок 1.1 – Вигляд інтерфейсу Dungeoneer

Окрім ігрового поля на інтерфейсі також можна побачити текстову консоль, яка повідомляє гравцю інформацію про навколишній світ та його зміни. В грі використані усі можливості Rot.js, такі як динамічне світло, покроковий бій, взаємодія з навколишнім світлом, процедурна генерація рівнів тощо. Стиль та реалізація графіки дозволяє сфокусуватися на технічних рішеннях та ігнорувати проблеми ігрового та візуального дизайну. Ця концепція інтерактивної гри підходить для дослідження процесу розробки подібних продуктів. Задля

розуміння роботи з Rot.js присутня можливість дослідження джерела коду (малюнок 1.2).



Малюнок 1.2 – Open source Dungeoneer

1.2.3 Dungeon Crawl Stone Soup (Rot.js)

Dungeon Crawl Stone Soup – ще один представник інтерактивних ігор на Rot.js, який натомість від Dungeoneer: The Roguelike має реальний попит та базу гравців, що дозволило розробникам інтегрувати в проєкт графіку (малюнок 1.3), звукове супроводження та комплексний ігровий дизайн. Відкритий код дозволив тисячам гравців приєднатися до вдосконалення та розробки проєкту.

Основні переваги гри очевидні при спостереженні за інтерфейсом застосунку – більш прийнятний для великої маси гравців зовнішній вигляд гри, звукове супроводження та вивід характеристик гравця, візуальне відображення всієї ігрової мапи у форматі minimap. Важливо відмітити, що навантаженість проєкту ускладнює кросс-платформерність, тим не менш завдяки відкритому коду небайдужі гравці адаптували гру під більшість ОС та інших від персональних комп'ютерів платформ, таких як телефони, консолі тощо. Тому відкритий код важлива перевага для більшості інді-ігор (інді від independent – «незалежний»), так як дозволяє розробнику сфокусуватися на процесі технічної реалізації задуманої функціональної частини, а спільнота бере на себе роль адаптації продукту для мас.

Малюнок 1.3 – Інтерфейс Dungeon Crawl Stone Soup

Хоча і ця гра має більше переваг за Rogue, досі залишилися й певні недоліки, які зумовлені самим фреймворком Rot.js, тому існує необхідність в порівнянні з іншими представниками жанру, функціональна частина яких реалізована на основі інших засобів.

1.2.4 Caves of Qud

Caves of Qud – типовий представник жанру Rogue-like ігор, який візуально та концептуально нагадує Dungeoneer та Dungeon Crawl Stone Soup, але натомість вона була реалізована за допомогою мови програмування Unity, однієї з найпопулярніших мов програмування для розробки відеоігор. Unity надає потужні інструменти та фреймворки для створення графічних ігор різних жанрів, включаючи roguelike.

Існують такі переваги використання Unity для створення будь-яких ігор:

- Unity підтримує розробку для різних платформ, таких як Windows, macOS, Linux, Android та iOS. Це робить гру доступною для широкого кола користувачів на різних пристроях.

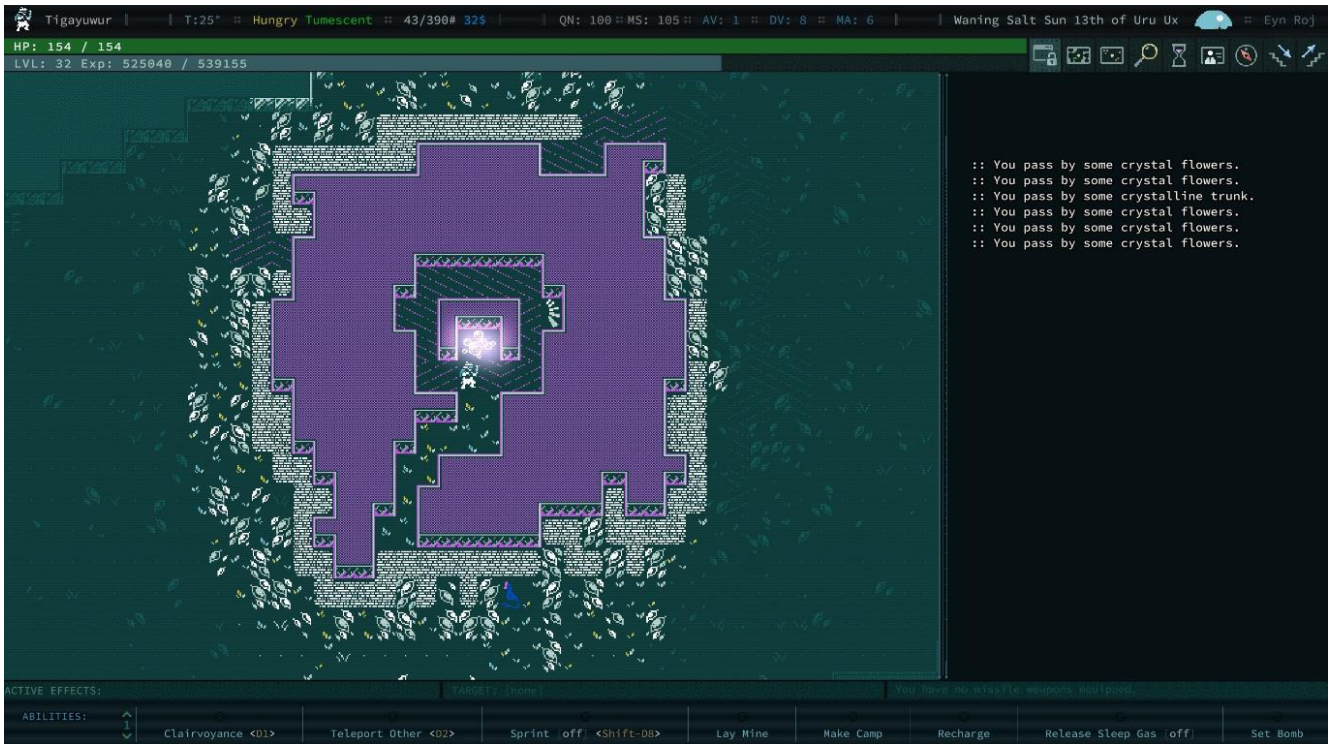
- Unity має велику спільноту розробників, яка активно ділиться знаннями, документацією та рішеннями. Це дозволяє швидко отримати підтримку та вирішити будь-які проблеми, які можуть виникнути під час розробки гри.

- Unity надає потужний візуальний редактор, який спрощує розробку графіки, анімації та розміщення об’єктів у грі. Це дозволяє розробникам швидко створювати прототипи та налаштовувати елементи гри.

- Unity надає потужні інструменти для створення 2D і 3D графіки. Це дозволяє створювати графічно насичені та деталізовані ігрові світи.

- Unity підтримує використання сторонніх розширень і плагінів, які можуть додати функціональність грі. Це дає розробникам більше гнучкості та швидкості під час розробки.

- Ця мова програмування надає розробникам потужні інструменти для створення графічних ігор для кількох платформ. Unity також має велику спільноту та великі ресурси, щоб допомогти розробникам втілити свої ідеї в життя та отримати бажані результати.



Малюнок 1.4 – Caves of Qud – загальний вигляд

В цій грі головною відмінністю від проєктів, реалізованих на Rot.js є складний механізм процедурної генерації квестових ланцюгів, створення якого стало можливим завдяки додатковим фреймворкам та плагінам, які Unity дає можливість швидко та «безболісно» впроваджувати в свій продукт навіть під час розробки.

Але побудова та реалізація навіть невеликого проєкту подібного плану на Unity окрім великої кількості ресурсів вимагає досвіду в розробці ігрових проєктів та додаткових витрат часу. Найрозумніший спосіб реалізації концепту rogue-like гри постає в виборі засобів, які були розроблені навмисно для покращення досвіду розробника при створенні такого роду продуктів. Саме тому

Unity та Caves of Qud залишаються ціллю, перед подоланням якої доведеться створити не одну схожу гру з використанням іншого ряду засобів.

1.2.5 Аналіз процесу створення ігрового дизайну гри-вчителя

Розглянувши три приклади з іграми в жанрі rogue-like можна прийти до висновку, що такий продукт є гарною платформою для реалізації різноманітних ігрових механік, в тому числі і механік, пов’язаних з навчальним процесом. Проблемним питанням постає необхідність об’єднати ігровий процес з процесом навчання ненав’язливим чином. Можна звернути увагу на декілька ігрових проєктів, які ставлять собі за мету вчити гравця.

Такі проєкти можна поділити на дві групи:

- Ігри, задумані як ігри вчителі
- Ігри, можуть використовуватися для навчання

До першої групи відносяться навчальні ігри для дошкільнят та мобільні ігри створені для вивчення мов, такі як Duolingo. До другої групи відноситься велика кількість sandbox ігор, таких як Minecraft. Концепт цікавої гри з ненав’язливим процесом навчання більше підходить до жанру rogue-like, тому під час подальшої розробки будемо відштовхуватися від подібної концепції, щоб не віднімати в гравця ціль гри та не заміняти її на «грати заради навчання».

1.3 Висновки до аналізу та постановка наступного завдання

Після огляду трьох представників жанру Rogue-like, постають наступні проблеми, які необхідно вирішити в процесі розробки інтерактивної гри:

- відтворення схожого інтерфейсу;
- використання основних дизайнерських прийомів для створення схожого ігрового досвіду;
- демонстрація основних функцій Rot.js та переваг при розробці гри жанру Rogue-like;
- оптимізація та реалізація можливості запуску продукту з використанням різноманітних веб-браузерів.

Ці кроки допоможуть створити гру, яка може зацікавити авдиторію підлітків. Відштовхуючись від не цифрових проєктів (настільних ігор), постає завдання додати до ігрового процесу елементи навчання. Вони можуть вміститися в базові механіки rogue-like гри, такі як бій з ворогами або пересування мапою гри, або додані як винагорода за виконання певних дій.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ

2.1 Мова HTML

HTML (від англ. HyperText Markup Language) [13] – це мова гіпертекстової розмітки, яка використовується для структурування та відображення веб-сторінок та їхнього вмісту.

HTML базується на концепції тегів, які використовуються для об'єднання та форматування різних частин веб-сторінки. Теги визначають структуру і семантику вмісту і вказують браузеру, як правильно відображати вміст. Наприклад, теги <h1> позначають заголовки першого рівня, теги <p> використовуються для абзаців тексту, а теги <img> - для вставки зображень.

HTML також має атрибути, які можна додавати до тегів, щоб надати елементам додаткові властивості або функціональність. Наприклад, атрибут src тегу <img> використовується для вказівки шляху до зображення, а атрибут href тегу <a> - для створення посилань на інші веб-сторінки.

2.2 Мова CSS

CSS (від англ. Cascading Style Sheets) [14] - є основним інструментом для реалізації дизайну веб-сторінок і заслуговує на увагу при виборі для розробки проєктів. Ця мова стилізації має кілька переваг, які роблять її привабливим варіантом для дизайнерів і розробників.

По-перше, CSS дозволяє розділити зовнішній вигляд веб-сторінок від їхньої структури. Це означає, що зміни в дизайні можуть бути внесені безпосередньо в CSS-файл, не впливаючи на HTML-структуру сторінки. Таке розділення полегшує управління дизайном і дозволяє швидко змінювати вигляд всіх сторінок проєкту.

Крім того, CSS забезпечує консистентність дизайну. За допомогою стилів можна одночасно застосовувати зміни до багатьох елементів, забезпечуючи єдиний вигляд для всіх сторінок проєкту. Це сприяє впізнаваності бренду і поліпшує користувацький досвід.

Гнучкість CSS дозволяє налаштовувати різні аспекти дизайну, включаючи кольори, шрифти, відступи, тіні та анімацію. Використання CSS-препроцесорів, таких як Sass або Less, збільшує ефективність написання CSS-коду та додає додаткові можливості для стилізації.

CSS також є широко підтримуваним стандартом у веб-розробці, що забезпечує сумісність з більшістю сучасних браузерів і пристроїв. Використання CSS гарантує, що веб-сторінки будуть відображатися правильно на різних пристроях і забезпечує єдиний вигляд для користувачів незалежно від того, яким браузером вони користуються.

2.3 Мова JavaScript

JS [15] є кросплатформовою скриптовою мовою, яка надає можливість керувати елементами веб-сторінки за допомогою програмного коду. Вона пропонує стандартні фреймворки (або бібліотеки), а також можливість використання сторонніх бібліотек або створення власних об'єктів. Код на мові JS можна розташовувати в окремих файлах з розширенням .js, що сприяє структуризації та поділу коду на логічні модулі або блоки, що в свою чергу додає зручність роботи з всім кодом веб-сторінки, та дозволяє корегувати функціональну частину в окремому файлі.

JS має європейську стандартизацію від Ecma International, що забезпечує єдність мови у всіх програмах, що підтримують цей стандарт. Компанії мають можливість використовувати відкритий стандарт мови для адаптації реалізації JS під свої власні потреби або потреби користувачів.

JS є потужним інструментом, який додає інтерактивність та динаміку до веб-сторінок. Завдяки його можливостям, розробники можуть створювати складні функції, обробляти події, взаємодіяти з сервером і забезпечувати багатофункціональність. JS є необхідним інструментом для розробки сучасних веб-додатків, оскільки він дозволяє створювати багатофункціональні та інтерактивні інтерфейси користувача.

Завдяки стандартизації і відкритості, JS забезпечує портативність і сумісність з різними платформами та пристроями. Ця мова є основним інструментом для реалізації динамічного функціоналу веб-сторінок та додатків, і вибір її для розробки проектів є обґрунтованим рішенням.

2.4 Фреймоворк Rot.js

Rot.js – фреймворк для JS з відкритим кодом, спеціально розроблений для створення ігор в жанрі Roguelike. Він надає розробникам потужні та універсальні інструменти і можливості для створення унікальних і захоплюючих ігрових світів та надає автентичний досвід як розробки, так і користування готовим ігровим продуктом.

Одна з головних переваг Rot.js полягає в спрощеному підході до розробки. Фреймворк надає широкий спектр готових компонентів, таких як готові класи та функції для реалізації генерації рівнів, підземель, світів тощо. Це дозволяє розробникам швидко створювати ігровий інтерфейс, обробляти події і взаємодіяти зі світом. Це також забезпечує ефективне використання ресурсів розробника та прискорює процес створення гри.

Rot.js забезпечує високу кросбраузерність, що означає, що ігрові додатки, розроблені з використанням цього фреймворку, працюватимуть на різних браузерах без необхідності вносити значні зміни. Це відкриває широкий спектр можливостей для розповсюдження гри та охоплення більшої аудиторії користувачів.

Однією з найбільш сильних сторін Rot.js є його гнучкість і розширюваність. Фреймворк надає можливість використовувати різноманітні модулі і плагіни, які допомагають розширити функціональність гри. Розробники можуть використовувати готові компоненти або створювати власні, відповідно до потреб проекту. Це дозволяє створювати унікальні гральні механіки та різноманітні ігрові світи.

Rot.js також надає зручний синтаксис та читабельність коду, що сприяє простоті розробки та обслуговування проектів. Фреймворк стимулює розробників до створення якісного та модульного коду, що спрощує розробку, пришвидшує тестування та покращує якість збереження проектів у майбутньому.

Rot.js є добре задокументованим фреймворком, який надає розробникам докладну документацію, пропонує приклади коду та має велику підтримку спільноти. Це значно спрощує вивчення і освоєння фреймворку для новачків, а також дозволяє розробникам ефективно використовувати всі його можливості.

Rot.js - це потужний і гнучкий JS-фреймворк, який допомагає розробникам створювати захоплюючі ігри без витрат ресурсів часу на вивчення фреймворку та специфіки роботи з складними функціями. Його простота використання, кросбраузерність, розширюваність та зручний синтаксис роблять його привабливим вибором для розробки ігрових проектів різного масштабу.

Малюнок 2.1 – сайт розробників Rot.js

2.5 Середа розробки (VSCode)

VSCode (від Visual Studio Code) - це потужне та популярне кросплатформове інтегроване середовище розробки (IDE), яке здобуло широку популярність серед веб-розробників. Завдяки своїй універсальності та розширюваності, VSCode відкриває розробникам купу можливостей для ефективної роботи над будь-яким ПЗ.

Однією з найбільших переваг перед іншими IDE VSCode є його розширюваність. Інтегроване середовище розробки підтримує велику кількість розширень та плагінів, які дозволяють розробникам налаштовувати інтерфейс, додавати нові функціональні можливості та інтегрувати зовнішні інструменти (в тому числі поєднувати функціонал різного ПЗ). Наявність широкого спектру розширень сприяє зручності роботи в даній IDE, дозволяючи вибрати інструменти, які найкраще відповідають потребам розробника.

VSCode також має потужний інтерфейс з великою кількістю функціональних можливостей. Він надає можливість швидко переходити між файлами, використовувати автодоповнення, виділяти синтаксис, розгортати термінали тощо. Ці можливості допомагають підвищити продуктивність розробки та забезпечити зручний та швидкий робочий процес.

Ще однією перевагою VSCode є його інтеграція з іншими інструментами та платформами. Він має вбудовану підтримку для різних мов програмування, фреймворків і технологій, що дозволяє легко працювати з веб-стеком. Крім того, VSCode дозволяє з легкістю інтегруватися з системами контролю версій, веб-серверами, віддаленими обліковими записами і багатьма іншими інструментами, що розширює його функціональні можливості та полегшує розробку веб-сайтів.

Існує велика спільнота розробників VSCode, яка створює та підтримує розширення, надає допомогу та ділиться досвідом. Це створює підтримувану середу, де розробники можуть спілкуватися, вчитися один від одного та розвиватися разом.

VSCode - це IDE, що пропонує широкі можливості, розширюваність та зручний інтерфейс. Завдяки цим перевагам, воно є відмінним вибором для розробки веб-сайтів, допомагаючи розробникам працювати ефективно, швидко і зручно.

2.6 Висновок до другого розділу

В другому розділі оглянуто засоби розробки даної інтерактивної гри: мови програмування та їх фреймворки, програмне забезпечення. Кожен засіб розробки був описаний в подробицях, що дає можливість використати сильні сторони кожного засобу у рамках дипломної роботи.

Маючи окреслений список завдань та обрані засоби розробки, що оглянуті в даному розділі, можна почати процес розробки проєкту.

3 РОЗРОБКА ІНТЕРАКТИВНОЇ ГРИ

3.1 Створення ігрового інтерфейсу

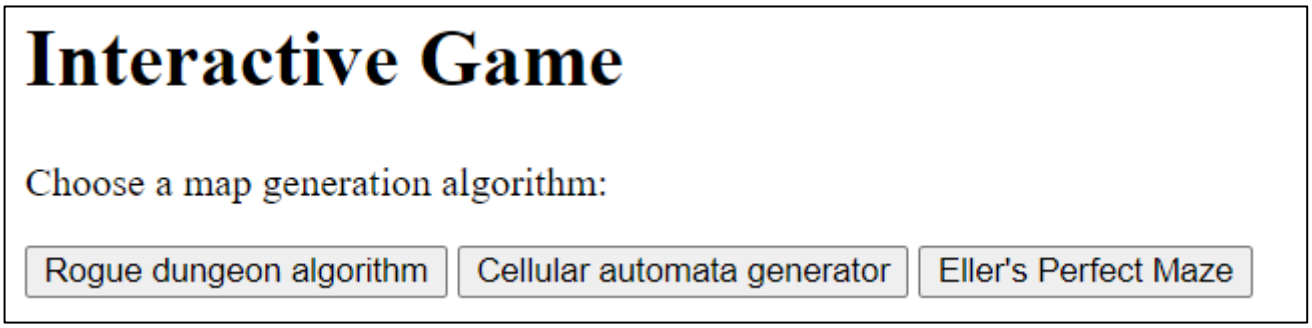
Першим етапом розробки даного проєкту буде створення HTML-шаблону, який буде відповідати за зовнішній інтерфейс веб-сторінки. HTML-файл буде виконувати функцію каркасу гри, через який буде реалізована візуалізація інтерфейсу.

Досліджуючи текст файлу, можна побачити використання декількох базових тегів для відображення інформації на веб-сторінці. Всередині «тіла» нашого застосунку знаходиться декілька контейнерів, кожний з яких виконує свою функцію:

1. зберігає стартовий екран гри. На стартовому екрані можлива реалізація багатьох функцій, які можуть знадобиться в майбутньому при розвитку проєкта в повноцінний ігровий продукт. Зараз start-screen зберігає базову інформацію. Щоб реалізувати можливість вибору генерації рівня в майбутньому, на стартовий екран додано три кнопки з назвою генерації:

```
    <button onclick="startGame('rogue')">Rogue dungeon algorithm</button>
    <button onclick="startGame('cellular')">Cellular automata
generator</button>
    <button onclick="startGame('eller')">Eller's Perfect Maze</button>
```

Зараз кнопки (малюнок 3.2) не несуть ніякої функціональної частини, вони знадобляться в майбутньому.



Малюнок 3.1 – Вигляд екрану вибору генерації без доданих стилів

2. повинен вмістити в собі увесь ігровий інтерфейс, який буде реалізований в JS-файлі застосунку. Зараз він немає візуального вигляду, та існує як «скелет» майбутнього інтерфейсу.

Для подальшого використання фреймворку Rot.js, його необхідно «під'єднати» до HTML-файлу програми. Так як Rot.js технічно є js-скріптом, його можна підключити простою стрічкою коду. Разом Rot.js існує потреба під'єднати майбутній JS-файл:

```
<script src="rot.js"></script>
<script src="game.js"></script>
```

За допомогою стрічки

```
<link rel="stylesheet" type="text/css" href="style.css">
```

підключається майбутній файл з CSS кодом, який створює стилі для HTML-тегів. На цьому створення HTML-файлу для застосунку закінчено.

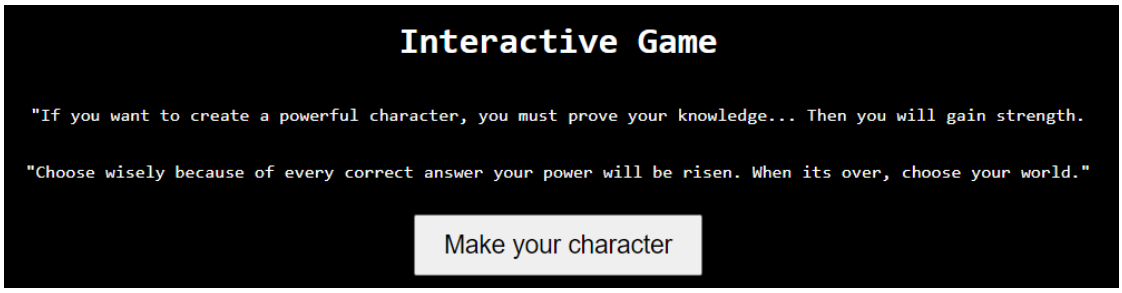
3.2 Створення CSS-файлу

Щоб додати прийнятний для користувача вигляд сайту до HTML-файлу був під'єднаний файл зі стилями розширення .css. Так як фреймворк Rot.js буде відповідати за налаштування зовнішнього вигляду інтерфейсу гри окремо від

стилів в CSS-файлі, додаємо тільки базові стилі для початкового екрану та «тіла» HTML-файла.

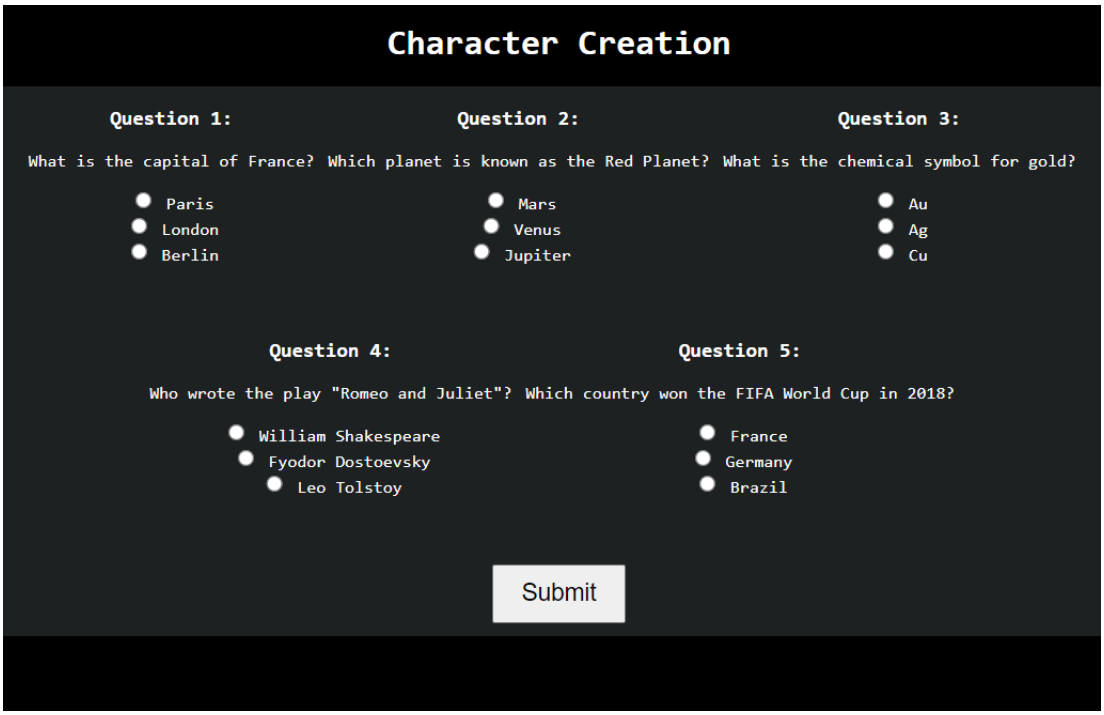
На виході отримаємо мінімалістичний прийнятний дизайн початкового екрану без зайвої інформації та/або кольорів.

Фінальний дизайн початкового екрану та екранів створення персонажу, вибору генерації представлено на мал. 3.3-3.5,



Малюнок 3.2 – Початковий екран

Малюнок 3.3 – Екран вибору генерації



Малюнок 3.4 – Екран створення персонажу

На початковому екрані присутній текст з шаблоном для назви гри (на даний момент Interactive Game), та текст з пропозицією обрати майбутній алгоритм генерації рівня і три кнопки до цього. Кожна з кнопок буде відповідати за названу генерацію. Функціональна частина буде реалізована в JS-файлі застосунку.

3.3 Вміст файлу rot.js

Фреймворк rot.js вмістить в собі велику кількість коду, яка існує автономно, та не потребує додаткових кореляцій від користувача. Тим не менш, якщо виникає необхідність змінити функції rot.js, така можливість присутня, в коді залишена велика кількість коментарів, а на сайті розробників існує детальна документація до кожної частини коду. В файлі rot.js реалізовано велику кількість складних

задач, таких як рандомізація результату, велика кількість різновидів процедурної генерації світу, генерація квестів, артефактів (ігрових предметів) тощо.

3.4 Розробка логіки гри з використанням JavaScript

3.4.1 Запровадження генерації світу та його наповнення

В першу чергу створимо логіку зміни початкового екрану на екран створення персонажу та на вибір генерації (мал. 3.5)

```
function showCharacterCreationScreen() {
    var startScreen = document.getElementById('start-screen');
    var characterCreationScreen = document.getElementById('character-creation-screen');

    startScreen.style.display = 'none';
    characterCreationScreen.style.display = 'block';
}

var knowledge = 0;
function showMapCreationScreen() {
    var characterCreationScreen = document.getElementById('character-creation-screen');
    var mapCreationScreen = document.getElementById('map-creation-screen');

    characterCreationScreen.style.display = 'none';
    mapCreationScreen.style.display = 'block';
```

Малюнок 3.5 – Логіка кнопок showCharacterCreationScreen та showMapCreationScreen

При натисканні кнопок гравця буде переносити через екрані, дизайн яких був розглянутий на малюнках 3.2-3.4.

При натисканні кнопки з вибраним алгоритмом генерації, буде відбуватися створення світу та його наповнення. Перед тим як реалізувати логіку взаємодії гравця зі світом, ворогами, предметами, прогрес тощо, необхідно запровадити функції, які будуть відповідати за генерацію світу. Почнемо зі створення функції function Game(), яка буде вмістити в собі масиви мапи, ворогів, босів, об’єкт гравця та дисплею.

Важливо відмітити, що світ буде зберігатися в масиві map[], який в свою чергу зберігає масиви з рядками та стовпцями тайлів (клітинки), які в майбутньому будуть наповнені різними елементами, такими як прохідні тайли, стіни, вороги, предмети тощо.

Запровадимо генерацію самого світу методом generateMap у прототипі об’єкта Game. Задаємо обрані розміри мапи

```
this.mapWidth = 70;
this.mapHeight = 35;
```

Залежно від вказаного алгоритму, який був обраний при натисканні кнопки з його назвою, цей метод повинен генерувати мапу гри. Створюється порожній масив ‘map’, який як вже було зазначено вище, буде зберігати інформацію про клітини.

Наступним кроком в методі задається змінна ‘generator’, яка і буде головним чином відповідати за створення мапи. Залежно від переданого алгоритму за допомогою оператору if (малюнок 3.5) обирається відповідний генератор

```
var generator;
if (algorithm === 'rogue') {
    generator = new ROT.Map.Rogue(this.mapWidth, this.mapHeight);
} else if (algorithm === 'cellular') {
    generator = new ROT.Map.Cellular(this.mapWidth, this.mapHeight);
    generator.randomize(0.5);
    var totalIterations = 3;
    for (var i = 0; i < totalIterations - 1; i++) {
        generator.create();
    }
} else if (algorithm === 'eller') {
    generator = new ROT.Map.EllerMaze(this.mapWidth, this.mapHeight);
}
```

Малюнок 3.6 – Реалізація вибору алгоритму

Якщо algorithm === 'rogue', використовується алгоритм Rogue для створення карти. Відповідний генератор створюється за допомогою

new ROT.Map.Rogue(this.mapWidth, this.mapHeight).

Якщо algorithm === 'cellular', використовується клітинний алгоритм для створення карти. Відповідний генератор створюється за допомогою

new ROT.Map.Cellular(this.mapWidth, this.mapHeight).

Після створення генератора викликаються методи randomize(0.5) для випадкового початкового заповнення клітинок та create() для створення карти.

Додаткові ітерації цих методів виконуються в циклі for залежно від значення totalIterations.

Якщо algorithm === 'eller', використовується алгоритм Eller для створення карти. Відповідний генератор створюється за допомогою

new ROT.Map.EllerMaze(this.mapWidth, this.mapHeight).

В даному випадку код JS-файлу звертається до rot.js, де шукає метод, який відповідає за генерацію за названим алгоритмом. На малюнку 3.6 можна побачити, що алгоритм EllerMaze, до прикладу, реалізований за допомогою циклів:

```
for (j = 1; j + 3 < this._height; j += 2) {
  /* one row */
  for (var _i5 = 0; _i5 < w; _i5++) {
    /* cell coords (will be always empty) */
    var x = 2 * _i5 + 1;
    var y = j;
    map[x][y] = 0;
    /* right connection */

    if (_i5 != L[_i5 + 1] && RNG$1.getUniform() > rand) {
      addToList(_i5, L, R);
      map[x + 1][y] = 0;
    }
    /* bottom connection */


    if (_i5 != L[_i5] && RNG$1.getUniform() > rand) {
      /* remove connection */
      removeFromList(_i5, L, R);
    } else {
      /* create connection */
```

Малюнок 3.7 – Генерація EllerMaze в фреймворку rot.js

Наступним кроком викликається метод create() генератора, передаючи

```
generator.create(function(x, y, v) {
    map[y] = map[y] || [];
        if (v === 1) {
            map[y][x] = {
                character: '#',
                color: '#343434',
                walkable: false
```

```
                };
            } else {
                map[y][x] = {
                    character: '.',
                    color: '#1B1B1B',
                    walkable: true
                };
            }
    });
```

йому функцію зворотнього виклику. Ця функція отримує задані координати x та y та значення ‘v’, яке відповідає стану клітинки на мапі (прохідна або ні, тобто 0 або 1). Залежно від цього значення встановлюються відповідні характеристики клітинки, які будуть в подальшому зберігатися в масиві map[].

Після виконання цього методу, Game матиме створену мапу для гри залежно від вказаного алгоритму, а кожна клітинка в масиві map[], буде мати ряд збережених характеристик (‘character’), (‘color‘) та (‘walkable’).

Додаємо масив mathProblem[], який буде містити в собі питання, які в майбутньому будуть частиною бойової системи гри.

Визначимо метод generatePlayer у прототипі об’єкта Game. Його функція буде полягати в генерації гравця та його початкового розташування на мапі.

Метод визначає змінну ‘player’, яка посилається на об’єкт гравця this.player, отримує розміри ігрової мапи в змінні mapWidth та mapHeight. Викликається метод getFreeCells() запроваджений на малюнку нижче:

```
Game.prototype.getFreeCells = function() {
    var freeCells = [];
    for (var y = 0; y < this.map.length; y++) {
        for (var x = 0; x < this.map[y].length; x++) {
            var tile = this.map[y][x];
            if (tile.walkable && x !== this.player.x && y !== this.player.y) {
                freeCells.push({ x: x, y: y });
            }
        }
    }
    return freeCells;
};
```

Малюнок 3.8 – Метод getFreeCells()

Представлений метод повертає список не зайнятих іншими об’єктами клітинок. Через те, що гравець генерується першим, клітинки можуть бути зайняті тільки стінами, які генеруються одночасно з прохідними тайлами. Отримане значення зберігається в масиві ‘freeCells’. Цей метод буде використаний при усіх наступних генераціях об’єктів.

Далі метод generatePlayer перевіряє чи є вільні клітинки на мапі, та генерує випадкове число в змінній index, яка представляє індекс вільної клітинки в масиві ‘freeCells’. Цей процес визначає вільну клітинку з масиву, на якій буде знаходитися персонаж користувача. Метод оновлює координати гравця на координати заданої клітинки.

Таким чином персонаж пересувається на вільну клітинку на мапі, що дозволяє починати новий забіг в грі з різних місць.

Схожим чином реалізуємо генерацію зілля здоров’я та підказки, які представляють в грі предмети, з яким може взаємодіяти гравець (малюнки 3.9 та 3.10).

```
Game.prototype.generateHealthPotion = function() {
    var freeCells = this.getFreeCells();

    if (freeCells.length > 0) {
        var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
        this.healthPotion = freeCells[index];
    }
};
```

```
Game.prototype.generateHint = function() {
    var freeCells = this.getFreeCells();

    if (freeCells.length > 0) {
        var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
        this.hint = freeCells[index];
    }
};
```

Малюнки 3.9 та 3.10 – метод генерації зілля здоров’я та підказки

Метод генерації ворогів в свою чергу буде реалізований іншим способом. В методі generateEnemies теж буде відбуватися перевірка на вільні клітини, але сам процес генерації буде створений за допомогою цикла for, що допоможе генерувати відразу декілька супротивників за одну активацію методу. Цикл for буде приймати задане в методі generateEnemies значення змінної enemyCount.

```
for (var i = 0; i < enemyCount; i++)
```

Цикл створює об’єкт enemy, який зберігається в масиві enemies[], що був заданий в об’єкті Game. Таким чином кожен ворог має власні координати, та окрему характеристику запасу здоров’я.

Схожим чином реалізуємо генерацію босів, змінимо характеристики здоров’я, щоб складність супротивників різного типу відрізнялась.

3.4.2 Методи запуску та оновлення відображення гри

Після створення методів генерації необхідно запровадити їх спільну активацію, а також знайти рішення для оновлення відображення гри, яке буде виконувати відразу декілька важливих функцій:

1. Таке рішення повинно задавати символьне відображення для згенерованих об’єктів.
2. В цей метод можливо додати відображення тексту з інформацією про гравця, стадію гри тощо.
3. Метод повинен виконувати функцію “оновлювача” ігрового поля. Після рухів гравця та ворогів, поле повинно змінюватися, за що і буде відповідати це рішення.

Метод refresh і буде вирішувати усі поставлені задачі. Відповідно поточного стану гри він буде оновлювати інтерфейс, та відображати усі елементи на екрані, такі як мапу, гравця, ворогів, босів, зілля та текст стану гравця.

На початку за допомогою двох вкладених циклів for метод проходиться по усіх елементах масиву map[], таким чином представляючи кожну клітинку ігрового поля. У кожній ітерації даного циклу

```
    for (var y = 0; y < this.map.length; y++) {
        for (var x = 0; x < this.map[y].length; x++)
```

отримується клітинка з масиву, та за допомогою методу draw об’єкта display, який береться з фреймворку rot.js, та буде пізніше заданий в методі init, візуалізується символ та колір клітинки на заданих координатах. Наступним кроком метод відображає поетапно спочатку гравця, ворогів, босів і зілля (якщо вони існують) (малюнок 3.11).

```
var player = this.player;
this.display.draw(player.x, player.y, '@', 'white');

var enemies = this.enemies;
for (var i = 0; i < enemies.length; i++) {
    var enemy = enemies[i];
    this.display.draw(enemy.x, enemy.y, 'E', 'red');
}

var bosses = this.bosses;
for (var i = 0; i < bosses.length; i++) {
    var boss = bosses[i];
    this.display.draw(boss.x, boss.y, 'B', '#FFE330');
}

if (this.healthPotion) {
    this.display.draw(this.healthPotion.x, this.healthPotion.y, '+', '#5DF023');
}
```

Малюнок 3.11 – Змінні з характеристиками та функцією відображення елементів

Метод також виконує функцію відображення тексту стану гравця під основним полем для гри, для чого використовує функцію drawText об’єкта display. Текст розміщується послідовно.

Далі відбувається перевірка на стан здоров'я гравця. Якщо здоров'я знизилося до 0 одиниць, виводиться повідомлення «Гра закінчена», та сторінка перезавантажується, даючи можливість почати гру знову.

Визначимо метод update у прототипі об'єкта Game. Цей метод викликається з певною періодичністю, інтервал якої буде заданий в методі ініціалізації гри нижче, та необхідний для виконання різних дій, пов'язаних з рухом ворогів та босів, та переходу на наступну стадію гри.

Метод проходиться по кожному елементу масива enemies та генерує для кожного окремого ворога випадкові значення координат, які вказують напрямок руху ворога (що можливий вгору, вниз, вліво або вправо). Отримавши випадкові значення, метод обчислює нові координати ворогів, та перевіряє, чи знаходяться ці координати в межах дозволеної області гри та чи прохідна за цими координатами клітинка. Якщо умови виконані, то метод оновлює координати ворога на нові значення. На малюнку нижче зображений цикл, відповідний за описані вище дії.

```
for (var i = 0; i < this.enemies.length; i++) {
    var enemy = this.enemies[i];
    var dx = Math.floor(ROT.RNG.getUniform() * 3) - 1;
    var dy = Math.floor(ROT.RNG.getUniform() * 3) - 1;
    var newX = enemy.x + dx;
    var newY = enemy.y + dy;

    if (this.isWithinMap(newX, newY) && this.map[newY][newX].walkable) {
        enemy.x = newX;
        enemy.y = newY;
    }
}
```

Малюнок 3.12 – Змінні з характеристиками та функцією відображення елементів

Аналогічна логіка руху виконується і для босу, або босів, які знаходяться на мапі. Важливо зазначити, що саме метод update відповідає за генерацію ворогів та босів, та оновлення стадії гри.

Якщо кількість ворогів на полі битви дорівнює 0, метод генерує нову партію ворогів, та оновлює стадію на i+1. Коли гравець перемагає 5 стадій поспіль, метод викликає метод generateBoss().

Наступним кроком буде створення методу запуску гри init. Цей метод буде відповідати за ініціалізацію гри і налаштовувати прослуховувач подій, який буде обробляти введення користувачем натискань на кнопки керування персонажем.

Метод init викликає усі описані вище методи, такі як

```
    this.generateMap(algorithm);
    this.generatePlayer();
    this.generateEnemies();
    this.generateHealthPotion();
```

Можна побачити, що метод generateBoss не викликається при ініціалізації гри, що обумовлено базовим ігровим дизайном. Бос буде генеруватися на пізніших стадіях гри, що буде зазначено далі.

Додається об’єкт display типу ROT.Display, який береться з rot.js та відповідає за відображення всього графічного інтерфейсу і використовується в методі refresh.

‘display’ додається до елемента з ідентифікатором ‘map-container’ в HTML-файлі.

Наступним кроком викликається метод refresh, який був розглянутий вище, і який відповідає за оновлення графічного інтерфейсу гри. Також встановлюється інтервал виклику функції update кожні 1500 мілісекунд (або 1,5 секунди).

На остачу додається прослуховувач подій keydown, що викликає функцію handleInput

```
Game.prototype.handleInput = function(e) {
    var player = this.player;
    var dx = 0;
    var dy = 0;

    switch (e.keyCode) {
        case 37: // Left arrow key
        dx = -1;
        break;
        case 38: // Up arrow key
        dy = -1;
        break;
```

```
        case 39: // Right arrow key
        dx = 1;
        break;
        case 40: // Down arrow key
        dy = 1;
        break;
        case 32: // Space key
        this.attackEnemy();
        this.attackBoss();
        this.drinkPotion();
        break;
    }

    var newX = player.x + dx;
    var newY = player.y + dy;

    if (this.isWithinMap(newX, newY) && this.map[newY][newX].walkable) {
        player.x = newX;
        player.y = newY;
    }

    this.refresh();
};
```

Функція відповідальна за обробку введення користувачем дій, які викликаються натисканням на задані клавіші клавіатури.

3.4.3 Методи взаємодії з ігровим світом

Тепер для закінчення роботи над функціональною частиною гри залишилось додати методи, які будуть відповідати за взаємодію персонажа гравця з ворогами та предметами. На основі цих методів можна буде зробити будь-яку взаємодію з будь-якими предметами, NPC (від англ. Non playable character), або ворогами, які можна реалізувати за допомогою rot.js, та саме в цьому методі буде задійний масив mathProblem, який додає складність бойової системи.

Додамо метод attackEnemy, який буде відповідати за атаку ворогів гравцем та викликатися натисканням на клавішу ‘Пробіл’.

Код метода буде виконувати декілька основних функцій. В першу чергу створюються змінні player та enemies, які посилаються на об’єкт гравця та масив enemies[]. Метод проходиться по кожному ворогу в масиві enemies окремо, та

наступним кроком за допомогою оператору if перевіряє, чи знаходяться ворог та гравець на одній клітинці. Якщо координати сутностей співпадають та гравець натискає «Пробіл», запускається функція alert, яка випадковим чином обирає завдання з масиву mathProblem та передає його в вспливаюче вікно з input (стрічка для вводу тексту). Якщо користувач вірно відповідає на питання, ворог та гравець “обмінюються” ударами, що зменшує здоров’я гравця на задану кількість, та здоров’я ворога на значення damage, яке береться з об'єкту player. Якщо гравець помиляється, то він тільки отримує шкоду, але не завдає жодної ворогу. В межі однієї гілки if відбувається дві важливих перевірки: перевірка запасу здоров’я ворога та гравця. Якщо показник ворога зменшується до нуля, він зникає, за що відповідальна стрічка нижче,

```
enemies.splice(i, 1);
```

а гравець отримує досвід, який додається до його загального показника, та відображується та інтерфейсі під ігровим полем. Якщо навпаки, то гра закінчується, функцією alert() викликається повідомлення “Гра закінчена!”, і сторінка перезапускається, дозволяючи користувачу спробувати знову.

В коді даного методу також обумовлені зміни характеристик, представлені на малюнку нижче:

```
player.experience += 20;
if (player.experience >= player.requiredExperience) {
    var rEscail = 1;
    rEscail += 1;
    player.level += 1;
    player.experience -= player.requiredExperience;
    player.requiredExperience += 20 * rEscail;
    player.damage += 2;
    player.health += 30;
this.generateHealthPotion();
}
```

Малюнок 3.13 – зростання рівня

Таким чином, перемога над супротивником має ігровий сенс та створює перешкоду – для перемоги над супротивником гравець повинен збирати підказки або використовувати зовнішні джерела інформації, щоб знаходити відповіді на

питання та таким чином наносити шкоду ворогам. Також гравець може отримувати досвід персонажу, який в свою чергу веде гравця до нових рівнів. Нові рівні збільшують запас здоров'я та шкоду, яку персонаж користувача здатний нанести ворогам та босам при правильній відповіді, що робить доступ до підказок та боротьбу з ворогами легшою.

Для реалізації взаємодії з босами був створений аналогічний код. Різниця між методами міститься в стрічці

```
player.experience += 50;
```

що відповідальна за кількість отриманого гравцем досвіду після перемоги над босом.

Метод взаємодії з зіллям здоров'я був реалізований іншим способом, представленим на малюнку 3.14:

```
Game.prototype.drinkPotion = function() {
    var player = this.player;
    var potion = this.healthPotion;

    if (player.x === potion.x && player.y === potion.y) {
        player.health += 50;
        delete this.healthPotion;
    }
}
```

Малюнок 3.14 – метод drinkPotion

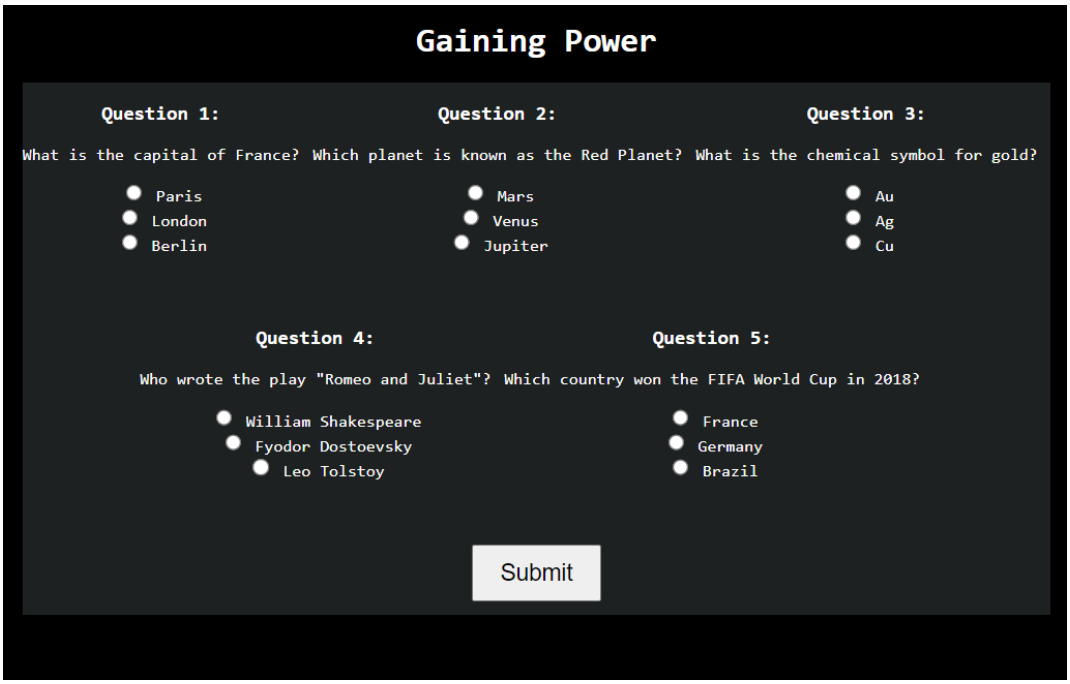
3.5 Відображення та процес гри

Тепер необхідно перевірити роботу усіх елементів гри, та за потреби створити додаткові методи, або відремонтувати існуючи, якщо вони функціонують неналежним чином.

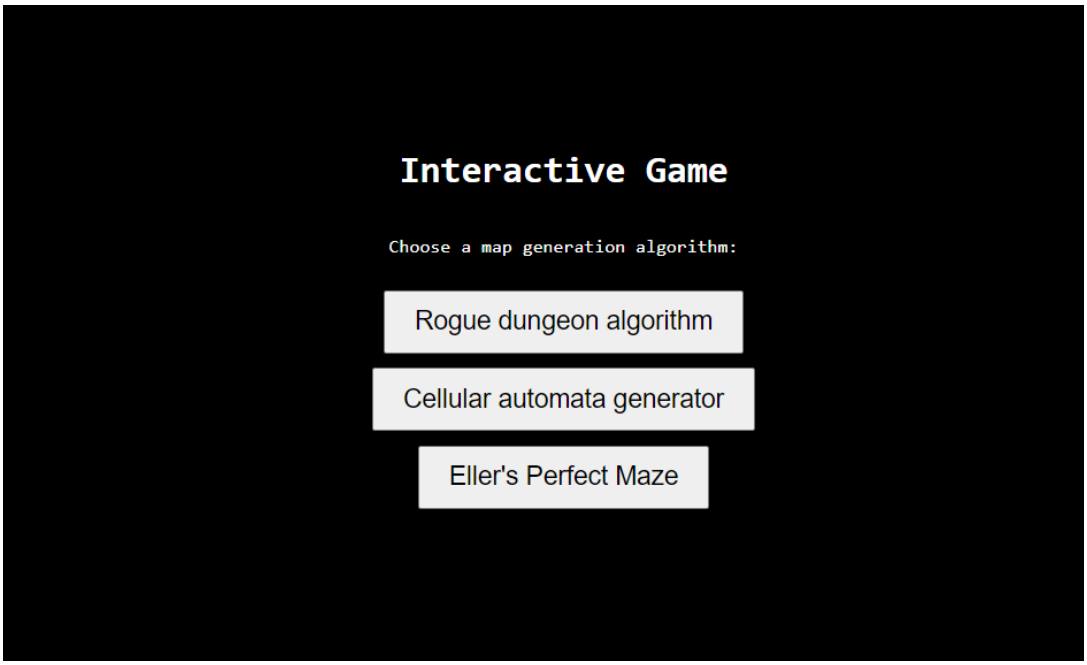
3.5.1 Відображення ігрового світу. Вибір генерації.

Заходячи на ігровий сайт, користувач відразу потрапляє до вибору генерації світу. Розглянемо початкові інтерфейси ще раз (малюнок 3.15-3.17):

Малюнок 3.15 – початковий екран

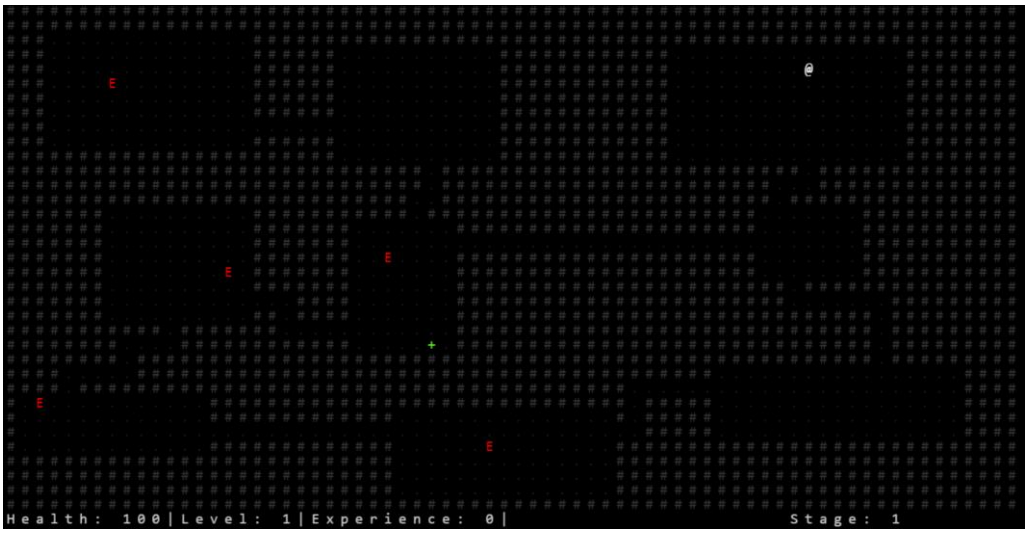


Малюнок 3.16 – екран створення персонажу

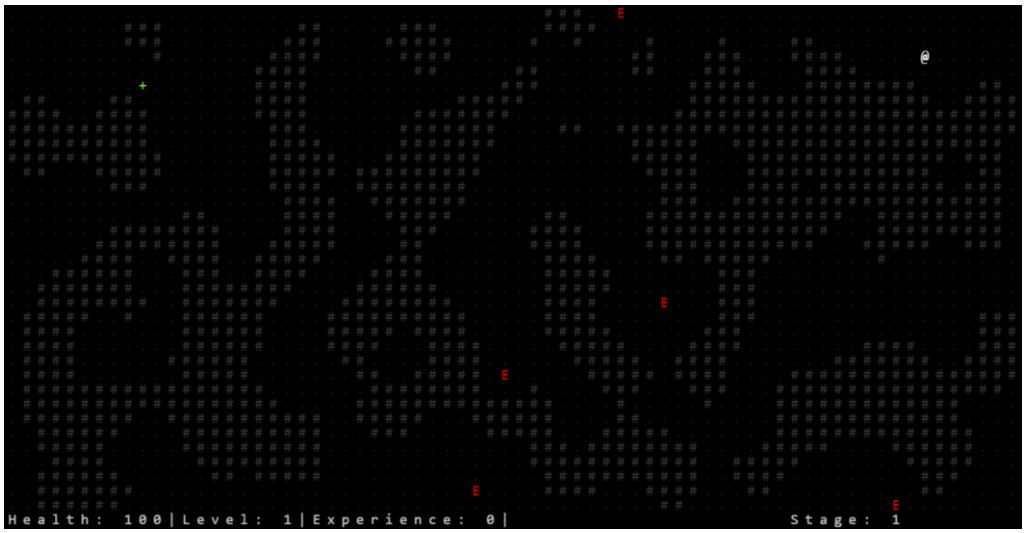


Малюнок 3.17 – екран вибору генерації

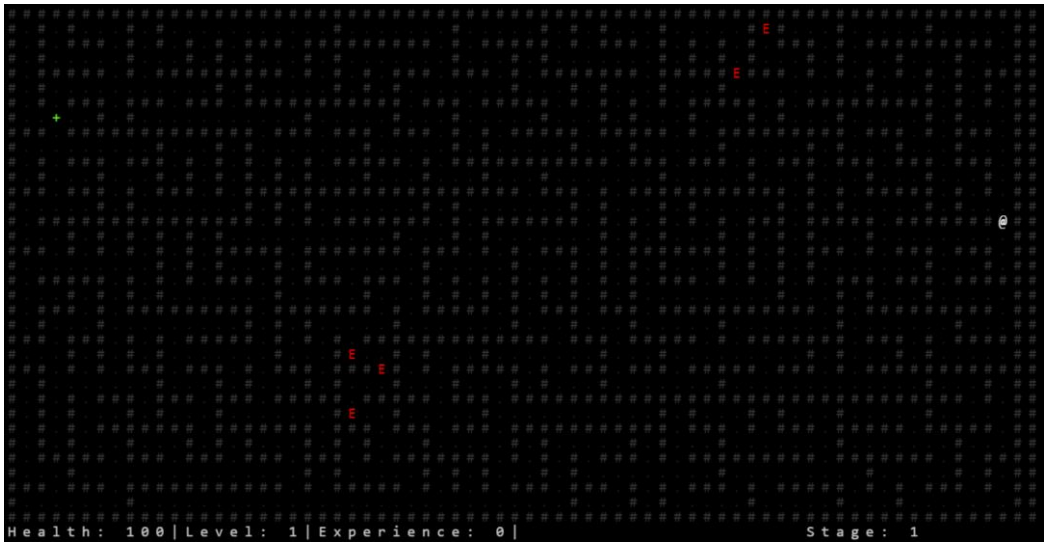
Перевіримо всі три типи заданої генерації на малюнках нижче:



Малюнок 3.18 – генерація Rogue



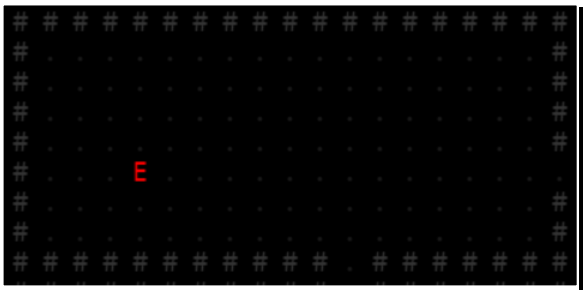
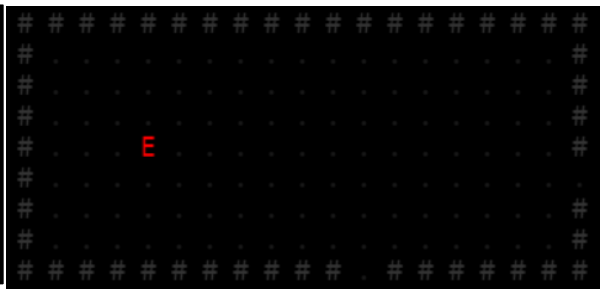
Малюнок 3.19 – генерація Cellular



Малюнок 3.20 – генерація Cellular

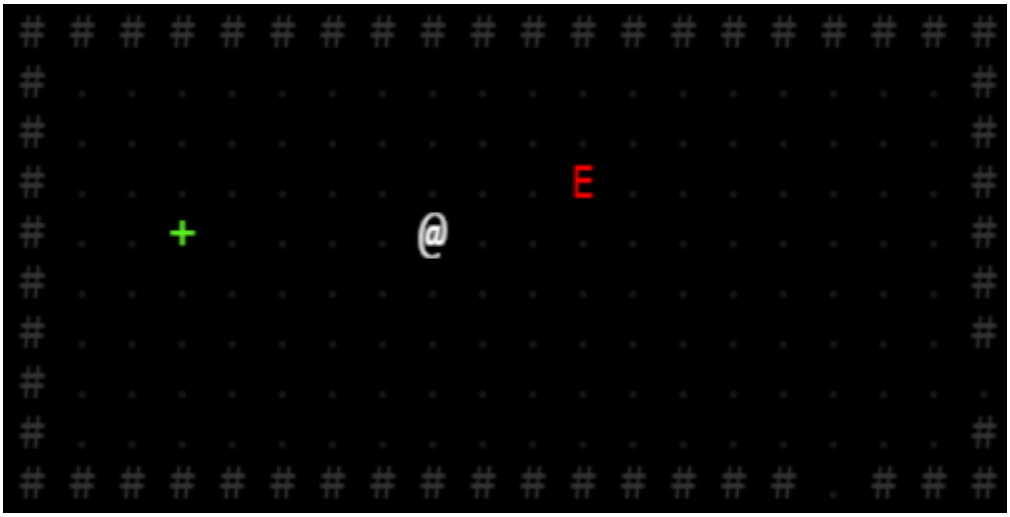
Як можна побачити на представлених малюнках, всі три типи генерації функціонують нормально, та створюють світ заданого типу. Разом з цим можна побачити інші елементи інтерфейсу – текстовий інтерфейс стану гравця, елементи ворогів позначені червоною літерою ‘E’ зілля здоров’я, позначене зеленим символом ‘+’ та підказку, яка зберігає в собі випадкову інформацію про одне з можливих питань, позначену блідо-жовтим символом ‘?’

При спостереженні за грою можна помітити пересування ворогів (малюнки 3.21 та 3.22)

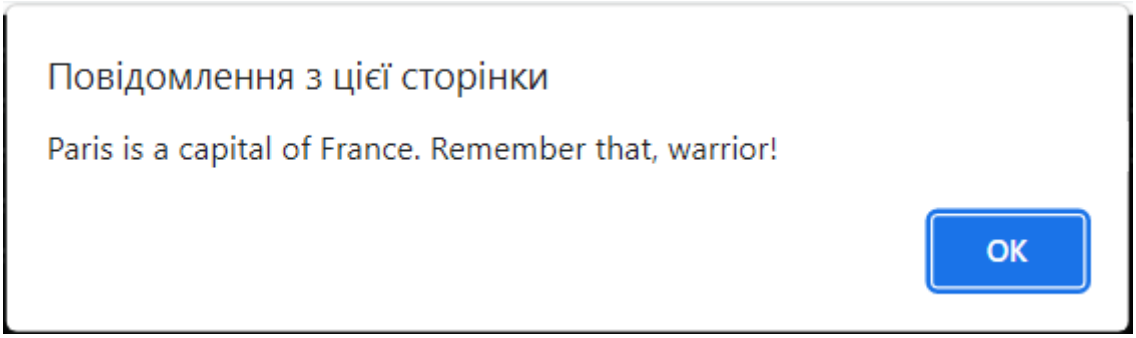
Малюнки 3.21 та 3.22 – пересування ворога з інтервалом 1,5 секунди

На малюнку 3.23 зображені елемент гравця, елемент ворога та елемент зілля здоров’я в одній кімнаті. Перевіримо функціональну частину взаємодії цих елементів один з одним.



Малюнок 3.23 – елементи в кімнаті

Використовуючи підказку гравець отримує alert з випадковою інформацією про одне з можливих запитань, що задіяні в бойовій системі, або в створенні персонажа. Таким чином зігравши партію, та зібравши велику кількіть підказок, гравець збільшує шанси на створення більш сильного персонажу зі збільшеним запасом здоров’я та збільшеною наносимою шкодою (мал 3.24)

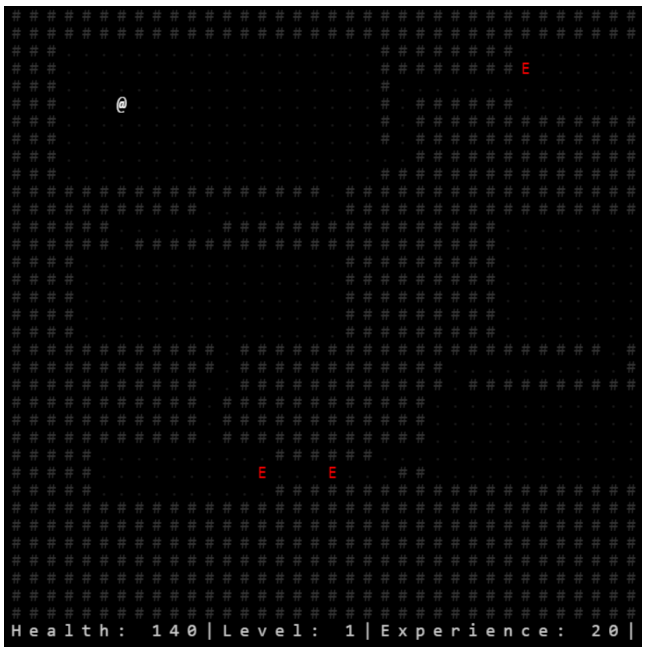


Малюнок 3.24 – alert викликаний підказкою

Після битви з ворогом гравець перемагає, та отримує 10 одиниць шкоди, при тому ворог гине. Гравець отримує досвід та шкоду, перевіримо, чи відобразилися зміни стану гравця в інтерфейсі (малюнок 3.25)

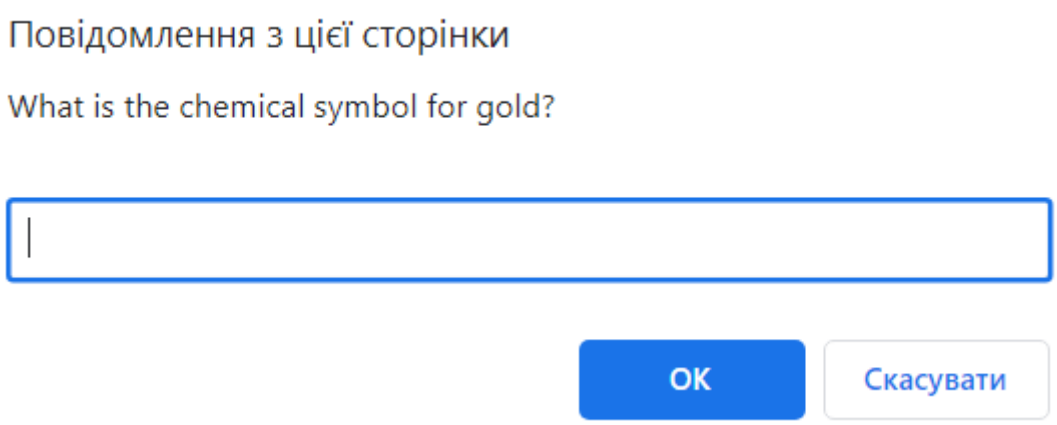
Малюнок 3.25 – Інтерфейс стану гравця зі змінами

Так, можемо побачити зміни, які внеслися до інтерфейсу. Перевіримо, чи працює метод взаємодії з зіллям здоров'я (малюнок 3.26).



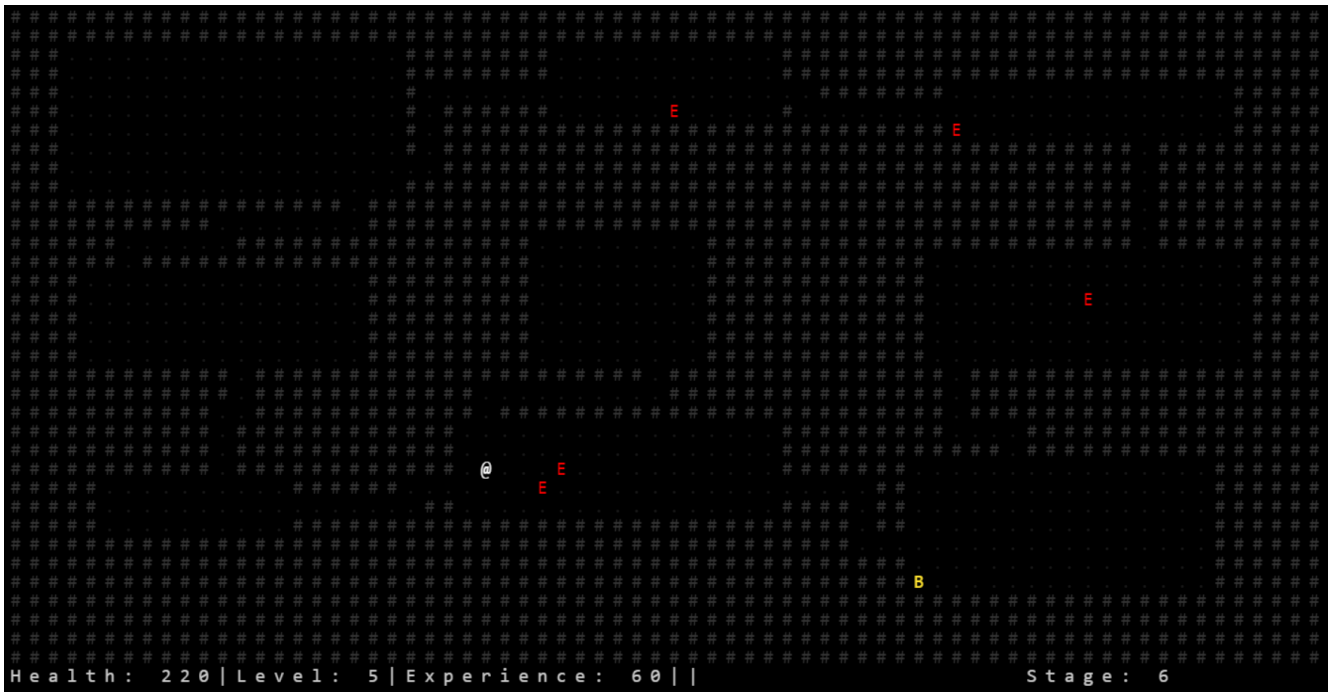
Малюнок 3.26 – загальний вид поля битви та інтерфейсу.

При початку бійки з ворогом гравець отримує alert з випадковим питанням там стрічкою для відповіді. Увесь пул питань знаходиться в масиві mathProblem. Приклад alert можна побачити на малюнку 3.27.



Малюнок 3.27 – alert для відповіді на питання.

Взаємодія з зіллям здоров’я теж є коректною. Перевіримо, якого рівня гравець зможе досягти до стадії, коли з’явиться перший бос (малюнок 3.28).

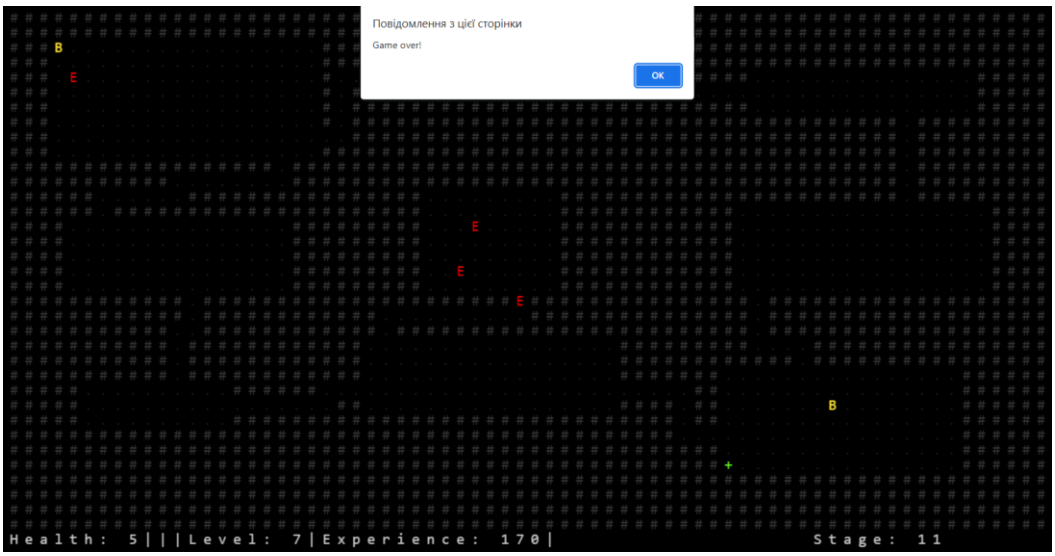


Малюнок 3.28 – 6 стадія гри

Пройдено 5 стадій. Гравець отримав 5 рівень персонажу та в залишку має 60 досвіду та 220 одиниць здоров’я. Математика вірна. Після п’ятої стадії, з’явився бос (відображений на малюнку 3.28 жовтою літерою ‘B’). Метод генерації босу працює вірно, після 5-ти стадій метод update успішно згенерував нового супротивника для гравця.

Перевіримо, що чекає на гравця, якщо в битві з босом його запас здоров’я впаде до 0 (малюнки 3.29 та 3.30).

Малюнок 3.29 – гравець з двома босами в кімнаті одночасно



Малюнок 3.30 – гравець програв босу в правій нижній кімнаті.

3.6 Висновки до третього розділу

Третій розділ даного дипломного проєкту розглядає внутрішню будову розробленої інтерактивної гри, а також досліджує функціональну та візуальну частину продукту.

Було досліджено фінальний інтерфейс програми, та розкрито процес створення каркасу гри за допомогою HTML та CSS.

Після створення інтерфейсу додатка було розроблено функціональну частину гри, тобто код JS-файлу. Були враховані елементи ігрового дизайну, та реалізовані основні функції, які були розглянуті в минулих розділах та присутні іграм даного жанру та стилістики. Окрім візуалізації ігрового поля була врахована необхідність візуалізації інтерфейсу стану гравця для зручності та розуміння подальших дій користувача на полі гри.

Були реалізовані механіки пересування гравця ту супротивників, а також підлаштовано керування гравцем за допомогою клавіш клавіатури. Механіка бойової системи була реалізована на основі звичайної вікторини, правильні відповіді в якій заохочуються перемогою над ворогами та зростанням рівня персонажа.

В результаті роботи над дипломним проєктом був отриманий ігровий продукт-вчитель жанру Rogue-like. Усі необхідні функції, які вимагалися від цього проєкту, були реалізовані.

Таким чином, була створена функціонуюча ігрова система на базі веб-сторінки жанру Rogue-like RPG в ретро-стилістиці в бойову систему та рольову систему створення персонажу якої були додані навчальні механіки типу вікторини (популярний жанр навчальних ігор серед настільних ігор).

ВИСНОВКИ

Представлений дипломний проєкт досліджує питання розробки (реалізації) ігрового продукту, який поєднує в собі мету розваги та навчання користувача в межах веб-сторінки на HTML, JavaScript, CSS з використанням фреймворку Rot.js. В результаті аналізу предметної області та дослідження аналогічних продуктів була розроблена функціональна гра. Всі завдання, які були поставлені як мета цього проєкту були успішно виконані.

В процесі розробки гри була проведена дослідницька робота, яка розглядає створення такого роду ПЗ з різних сторін – ігровий та візуальний дизайн та технічна частина були розглянуті окремо та разом в одному цілому продукті. Основною метою ставилося реалізація основних технічних засобів для функціонування подібного продукту за допомогою фреймворку Rot.js, та здатність такого продукту виконувати конкретне завдання – функціонувати як гра для навчання заданих тем. Другою метою ставилася реалізація гри у жанрі Rogue-like із збереженням стилістики та класичного ігрового дизайну.

Інтерфейс та функціональна частина були реалізовані за допомогою таких засобів (мов програмування та бібліотек), як JavaScript (фреймворк rot.js), HTML та CSS. В процесі розробки використовувалася середа для розробки ПЗ (IDE) Visual Studio Code із рядом QOL (від англ. Quality of Life) плагінів та розширень, для пришвидшення процесу розробки.

Фінальна версія продукту виконує функції простої гри-вчителя жанру Rogue-like, якому присутні такі характеристики:

- Процедурна генерація світу
- Випадкове пересування та генерація ворогів
- Випадкова генерація предметів (в даному випадку зілля здоров’я та підказок)

А також гра має елементи класичної гри-вікторини, де гравець заохочується до навчання винагородами у вигляді ігрового прогресу та подолання стадій. Як елемент ігрового дизайну гри жанру Rogue-like була створена реграбельність на основі системи створення персонажу. Гравець має пограти в гру, щоб в підказках

винайти відповіді на запитання, які при створенні роблять персонажа більш потужним.

Цей дипломний проєкт має можливості для активного розвитку та подальшого вдосконалення функціональної та візуальної частини, на що і розрахований будь-який проєкт з використанням rot.js фреймворку, але вже на цій стадії розвитку він виконує базові функції гри-вікторини.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Фланаган, Д. (2008). "JavaScript: Повний посібник". O'Reilly Media. // Режим доступу: https://www.oreilly.com/library/view/javascript-the-definitive/9780596805531/

2. Раушмаєр, А. (2014). "JavaScript у деталях". O'Reilly Media. // Режим доступу: https://www.oreilly.com/library/view/javascript-the-good/9780596517748/

3. Дукетт, Дж. (2011). "HTML та CSS: Дизайн та створення веб-сайтів". John Wiley & Sons. // Режим доступу: https://www.wiley.com/en-us/HTML+and+CSS%3A+Design+and+Build+Websites-p-9781118008188

4. Онлайн-посібник W3Schools. (2021). Посібник з HTML. // [Електронний ресурс] URL: https://www.w3schools.com/html/

5. Онлайн-посібник W3Schools. (2021). Посібник з CSS. // [Електронний ресурс] URL: https://www.w3schools.com/css/

6. Мережева документація Mozilla Developer Network (MDN). (2021). Посібник по JavaScript. // [Електронний ресурс] URL: https://developer.mozilla.org/uk/docs/Web/JavaScript/Guide

7. Матц, Дж., & Кінер, Д. (2012). "Кулінарна книга з HTML5 Canvas". Packt Publishing. // [Електронний ресурс] URL: https://www.packtpub.com/product/html5-canvas-cookbook/9781849691369 (дата звернення 31.05.2021).

8. Остерло, М., & Стоян, С. (2012). "Навіть швидші веб-сайти: Практичні поради щодо продуктивності для веб-розробників". O'Reilly Media. // Режим доступу: https://www.oreilly.com/library/view/even-faster-web/9781449336202/

9. Пірсон, Р. (2017). "Оволодіння патернами проектування на JavaScript". Packt Publishing. // Режим доступу: https://www.packtpub.com/product/learning-javascript-design-patterns-third-edition/9781785882620

10. Етемад, М. (2018). "Розробка ігор на HTML5 з нуля за допомогою Construct 2". Apress. // Режим доступу: https://www.apress.com/gp/book/9781484229098

11. HTML5 Rocks. (2021). Посібник для початківців з використання кешу програми. [Електронний ресурс] URL: https://www.html5rocks.com/uk/tutorials/appcache/beginner/

12. Де Луна, Дж. (2018). "Розробка гри у стилі Rogue-like з використанням JavaScript і Canvas". Apress. // Режим доступу: https://www.apress.com/gp/book/9781484233545

13. W3C. (2021). HTML Living Standard. [Електронний ресурс] // URL: https://html.spec.whatwg.org/multipage/

14. CSS Working Group. (2021). CSS Snapshot 2018. [Електронний ресурс] // URL: https://www.w3.org/TR/CSS22/

15. Хавербеке, М. (2012). "Елегантний JavaScript: Сучасне введення в програмування". // Режим доступу: https://eloquentjavascript.net/

ДОДАТОК А

Лістинги коду

1. Файл game.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Interactive Game</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>

  <div id="start-screen">
    <h1>Interactive Game</h1>
    <p>"If you want to create a powerful character, you must prove your knowledge... Then you will gain strength.</p>
    <p>"Choose wisely because of every correct answer your power will be risen. When its over, choose your world."</p>
    <button onclick="showCharacterCreationScreen()">Make your character</button>
  </div>

  <div id="character-creation-screen">
    <h1>Gaining Power</h1>
    <div class="character-list">
      <div class="row">
        <div id="question1">
          <h3>Question 1:</h3>
          <p>What is the capital of France?</p>
          <label><input type="radio" name="q1" value="paris"> Paris</label><br>
          <label><input type="radio" name="q1" value="london"> London</label><br>
          <label><input type="radio" name="q1" value="berlin"> Berlin</label><br>
        </div>

        <div id="question2">
          <h3>Question 2:</h3>
          <p>Which planet is known as the Red Planet?</p>
          <label><input type="radio" name="q2" value="mars"> Mars</label><br>
          <label><input type="radio" name="q2" value="venus"> Venus</label><br>
          <label><input type="radio" name="q2" value="jupiter"> Jupiter</label><br>
        </div>

        <div id="question3">
          <h3>Question 3:</h3>
          <p>What is the chemical symbol for gold?</p>
          <label><input type="radio" name="q3" value="au"> Au</label><br>
          <label><input type="radio" name="q3" value="ag"> Ag</label><br>
```

```
          <label><input type="radio" name="q3" value="cu"> Cu</label><br>
        </div>
      </div>

      <div class="row">
        <div id="question4">
          <h3>Question 4:</h3>
          <p>Who wrote the play "Romeo and Juliet"?</p>
          <label><input type="radio" name="q4" value="shakespeare"> William Shakespeare</label><br>
          <label><input type="radio" name="q4" value="dostoevsky"> Fyodor Dostoevsky</label><br>
          <label><input type="radio" name="q4" value="tolstoy"> Leo Tolstoy</label><br>
        </div>

        <div id="question5">
          <h3>Question 5:</h3>
          <p>Which country won the FIFA World Cup in 2018?</p>
          <label><input type="radio" name="q5" value="france"> France</label><br>
          <label><input type="radio" name="q5" value="germany"> Germany</label><br>
          <label><input type="radio" name="q5" value="brazil"> Brazil</label><br>
        </div>
      </div>
      <button onclick="showMapCreationScreen()">Submit</button>
      </div>
  </div>

  <div id="map-creation-screen">
    <p>Choose a map generation algorithm:</p>
    <button onclick="startGame('rogue')">Rogue dungeon algorithm</button>
    <button onclick="startGame('cellular')">Cellular automata generator</button>
    <button onclick="startGame('eller')">Eller's Perfect Maze</button>
  </div>

  <div id="map-container"></div>

  <script src="https://cdn.jsdelivr.net/gh/ondras/rot.js@latest/rot.js"></script>
  <script src="game.js"></script>
</body>
</html>
```

2. Код файлу style.css

```
body {
font-family: monospace;
```

```
3. background-color: black;
4. color: white;
5. text-align: center;
6. }
7. #start-screen {
8. display: flex;
9. flex-direction: column;
10.align-items: center;
11.justify-content: center;
12.height: 100vh;
13.}
14.#start-screen button {
15.margin-top: 10px;
16.padding: 10px 20px;
17.font-size: 18px;
18.}
19.#character-creation-screen {
20.display: flex;
21.flex-direction: column;
22.align-items: center;
23.justify-content: center;
24.height: 100vh;
25.}
26.#character-creation-screen button {
27.margin-top: 10px;
28.padding: 10px 20px;
29.font-size: 18px;
30.margin-bottom: 10px;
31.}
32.#map-creation-screen {
33.display: flex;
34.flex-direction: column;
35.align-items: center;
36.justify-content: center;
37.height: 100vh;
38.}
39.#map-creation-screen button {
40.margin-top: 10px;
41.padding: 10px 20px;
42.font-size: 18px;
43.}
44.#character-creation-screen .row {
45.display: flex;
46.justify-content: center;
47.}
48.#character-creation-screen .row div {
49.margin-right: 10px;
50.}
51.#character-creation-screen div {
52.margin-bottom: 20px;
```

```
53.}
54.#character-creation-screen .character-list {
55.background-color: rgb(30, 33, 33);
56.}
```

79. Код файлу game.js

```
1. function showCharacterCreationScreen() {
2. var startScreen = document.getElementById('start-screen');
3. var characterCreationScreen = document.getElementById('character-creation-
   screen');

4. startScreen.style.display = 'none';
5. characterCreationScreen.style.display = 'block';
6. }

7. var knowledge = 0;
8. function showMapCreationScreen() {
9. var characterCreationScreen = document.getElementById('character-creation-
   screen');
10.var mapCreationScreen = document.getElementById('map-creation-screen');

11.characterCreationScreen.style.display = 'none';
12.mapCreationScreen.style.display = 'block';

13.function submitAnswers() {
14.var q1Answer = document.querySelector('input[name="q1"]:checked');
15.var q2Answer = document.querySelector('input[name="q2"]:checked');
16.var q3Answer = document.querySelector('input[name="q3"]:checked');
17.var q4Answer = document.querySelector('input[name="q4"]:checked');
18.var q5Answer = document.querySelector('input[name="q5"]:checked');

19.if (q1Answer && q1Answer.value === "paris") {
20.knowledge++;
21.}
22.if (q2Answer && q2Answer.value === "mars") {
23.knowledge++;
24.}
25.if (q3Answer && q3Answer.value === "au") {
26.knowledge++;
27.}
28.if (q4Answer && q4Answer.value === "shakespeare") {
29.knowledge++;
30.}
31.if (q5Answer && q5Answer.value === "france") {
32.knowledge++;
33.}
34.}
35.}
```

```
36.function Game() {
37.this.map = [];
38.this.player = {
39.x: 0,
40.y: 0,
41.health: 100,
42.level: 1,
43.experience: 0,
44.damage: 25,
45.requiredExperience: 50
46.};

47.this.hint = {
48.x: 0,
49.y: 0
50.}
51.this.enemies = [];
52.this.bosses = [];
53.this.display = null;
54.}

55.var stage = 1;

56.var mathProblems = [
57.{
58.question: "2 + 2",
59.correctAnswer: "4"
60.},
61.{
62.question: "8 - 3",
63.correctAnswer: "5"
64.},
65.{
66.question: "What is the capital of France?",
67.correctAnswer: "Paris"
68.},
69.{
70.question: "Which is the largest ocean on Earth?",
71.correctAnswer: "Pacific Ocean"
72.},
73.{
74.question: "What is the process of photosynthesis?",
75.correctAnswer: "The process by which green plants and some other organisms use
   sunlight to synthesize foods with the help of chlorophyll"
76.},
77.{
78.question: "What is the formula for the area of a triangle?",
79.correctAnswer: "A = (base * height) / 2"
80.},
81.{
```

```
82.question: "Who is known as the father of genetics?",
83.correctAnswer: "Gregor Mendel"
84.},
85.{
86.question: "What is the main function of the respiratory system?",
87.correctAnswer: "To facilitate the exchange of oxygen and carbon dioxide between
   the body and the environment"
88.},
89.{
90.question: "Solve the equation: x² + 5x + 6 = 0",
91.correctAnswer: "x = -2 or x = -3"
92.},
93.{
94.question: "What is the formula for the volume of a sphere?",
95.correctAnswer: "V = (4/3) * π * r³"
96.},
97.{
98.question: "What is the largest organ in the human body?",
99.correctAnswer: "Skin"
100.  },
101.  {
102.  question: "What is the chemical symbol for gold?",
103.  correctAnswer: "Au"
104.  },
105.  {
106.  question: "What is the capital of Australia?",
107.  correctAnswer: "Canberra"
108.  },
109.  {
110.  question: "What is the process of mitosis?",
111.  correctAnswer: "The process of cell division that results in two genetically
   identical daughter cells"
112.  },
113.  {
114.  question: "What is the formula for the perimeter of a rectangle?",
115.  correctAnswer: "P = 2 * (length + width)"
116.  },
117.  {
118.  question: "What is the theory of evolution?",
119.  correctAnswer: "The theory that species change over time through the process of
   natural selection"
120.  },
121.  {
122.  question: "Who discovered penicillin?",
123.  correctAnswer: "Alexander Fleming"
124.  },
125.  {
126.  question: "What is the function of the nervous system?",
127.  correctAnswer: "To coordinate and control the actions and reactions of the
   body"
```

```
128.  },
129.  {
130.  question: "Solve the equation: 3x + 7 = 22",
131.  correctAnswer: "x = 5"
132.  },
133.  {
134.  question: "What is the formula for the area of a circle?",
135.  correctAnswer: "A = π * r²"
136.  },
137.  {
138.  question: "What is the largest rainforest in the world?",
139.  correctAnswer: "Amazon Rainforest"
140.  },
141.  {
142.  question: "What is the process of meiosis?",
143.  correctAnswer: "The process of cell division that results in the formation of
      gametes with half the number of chromosomes"
144.  },
145.  {
146.  question: "What is the formula for the perimeter of a triangle?",
147.  correctAnswer: "P = a + b + c"
148.  },
149.  {
150.  question: "What is the structure of DNA?",
151.  correctAnswer: "A double helix composed of nucleotides"
152.  },
153.  {
154.  question: "Who proposed the theory of relativity?",
155.  correctAnswer: "Albert Einstein"
156.  },
157.  {
158.  question: "What is the function of the circulatory system?",
159.  correctAnswer: "To transport oxygen, nutrients, hormones, and other substances
      throughout the body"
160.  },
161.  {
162.  question: "Simplify the expression: (4 + 7) * (8 - 3)",
163.  correctAnswer: "55"
164.  }
165.  ];

166.  Game.prototype.generatePlayer = function() {
167.  var player = this.player;
168.  var mapWidth = this.map[0].length;
169.  var mapHeight = this.map.length;
170.  var freeCells = this.getFreeCells();

171.  if (freeCells.length > 0) {
172.  var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
173.  var cell = freeCells[index];
```

```
174.  player.x = cell.x;
175.  player.y = cell.y;
176.  }
177.  };


178.  Game.prototype.generateHealthPotion = function() {
179.  var freeCells = this.getFreeCells();

180.  if (freeCells.length > 0) {
181.  var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
182.  this.healthPotion = freeCells[index];
183.  }
184.  };

185.  Game.prototype.generateHint = function() {
186.  var freeCells = this.getFreeCells();

187.  if (freeCells.length > 0) {
188.  var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
189.  this.hint = freeCells[index];
190.  }
191.  };


192.  Game.prototype.init = function(algorithm) {
193.  this.generateMap(algorithm);
194.  this.generatePlayer();
195.  this.generateEnemies();
196.  this.generateHealthPotion();
197.  this.generateHint();

198.  this.display = new ROT.Display({
199.  width: this.map[0].length,
200.  height: this.map.length + 1,
201.  fontSize: 20,
202.  forceSquareRatio: true
203.  });
204.  document.getElementById('map-
   container').appendChild(this.display.getContainer());

205.  this.refresh();

206.  var game = this;

207.  var gameInterval = setInterval(function() {
208.  game.update();
209.  }, 1500);

210.  window.addEventListener('keydown', function(e) {
```

```
211.  game.handleInput(e);
212.  });
213.  };


214.  Game.prototype.refresh = function() {
215.  for (var y = 0; y < this.map.length; y++) {
216.  for (var x = 0; x < this.map[y].length; x++) {
217.  var tile = this.map[y][x];
218.  this.display.draw(x, y, tile.character, tile.color);
219.  }
220.  }

221.  var player = this.player;
222.  this.display.draw(player.x, player.y, '@', 'white');

223.  var enemies = this.enemies;
224.  for (var i = 0; i < enemies.length; i++) {
225.  var enemy = enemies[i];
226.  this.display.draw(enemy.x, enemy.y, 'E', 'red');
227.  }

228.  var bosses = this.bosses;
229.  for (var i = 0; i < bosses.length; i++) {
230.  var boss = bosses[i];
231.  this.display.draw(boss.x, boss.y, 'B', '#FFE330');
232.  }

233.  if (this.healthPotion) {
234.  this.display.draw(this.healthPotion.x, this.healthPotion.y, '+', '#5DF023');
235.  }

236.  var hint = this.hint;
237.  this.display.draw(this.hint.x, this.hint.y, '?', '#F0E68C');

238.  var healthText = 'Health: ' + player.health + '|';
239.  var levelText = 'Level: ' + player.level + '|';
240.  var experienceText = 'Experience: ' + player.experience + '|';
241.  var stageText = 'Stage: ' + stage;
242.  this.display.drawText(0, this.map.length, healthText);
243.  this.display.drawText(12, this.map.length, levelText);
244.  this.display.drawText(21, this.map.length, experienceText);
245.  this.display.drawText(54, this.map.length, stageText);

246.  if (player.health <= 0) {
247.  clearInterval(gameInterval);
248.  alert('Game over!');
249.  location.reload();
250.  }
251.  };
```

```
252.  Game.prototype.generateMap = function(algorithm) {
253.  this.mapWidth = 70;
254.  this.mapHeight = 35;

255.  var map = [];

256.  var generator;
257.  if (algorithm === 'rogue') {
258.  generator = new ROT.Map.Rogue(this.mapWidth, this.mapHeight);
259.  } else if (algorithm === 'cellular') {
260.  generator = new ROT.Map.Cellular(this.mapWidth, this.mapHeight);
261.  generator.randomize(0.5);
262.  var totalIterations = 3;
263.  for (var i = 0; i < totalIterations - 1; i++) {
264.  generator.create();
265.  }
266.  } else if (algorithm === 'eller') {
267.  generator = new ROT.Map.EllerMaze(this.mapWidth, this.mapHeight);
268.  }

269.  if (generator) {
270.  generator.create(function(x, y, v) {
271.  map[y] = map[y] || [];
272.  if (v === 1) {
273.  map[y][x] = {
274.  character: '#',
275.  color: '#343434',
276.  walkable: false
277.  };
278.  } else {
279.  map[y][x] = {
280.  character: '.',
281.  color: '#1B1B1B',
282.  walkable: true
283.  };
284.  }
285.  });
286.  }

287.  this.map = map;
288.  };

289.  Game.prototype.generateEnemies = function() {
290.  var enemyCount = 5;

291.  for (var i = 0; i < enemyCount; i++) {
292.  var enemy = {
293.  x: 0,
294.  y: 0,
```

```
295.  health: 50
296.  };
297.  this.enemies.push(enemy);

298.  var freeCells = this.getFreeCells();
299.  if (freeCells.length > 0) {
300.  var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
301.  var cell = freeCells[index];
302.  enemy.x = cell.x;
303.  enemy.y = cell.y;
304.  }
305.  }
306.  };

307.  Game.prototype.generateBoss = function() {
308.  var bossCount = 1;

309.  for (var i = 0; i < bossCount; i++) {
310.  var boss = {
311.  x: 0,
312.  y: 0,
313.  health: 100
314.  };
315.  this.bosses.push(boss);

316.  var freeCells = this.getFreeCells();
317.  if (freeCells.length > 0) {
318.  var index = Math.floor(ROT.RNG.getUniform() * freeCells.length);
319.  var cell = freeCells[index];
320.  boss.x = cell.x;
321.  boss.y = cell.y;
322.  }
323.  }
324.  };

325.  Game.prototype.isWithinMap = function(x, y) {
326.  return x >= 0 && x < this.map[0].length && y >= 0 && y < this.map.length;
327.  };

328.  Game.prototype.getFreeCells = function() {
329.  var freeCells = [];
330.  for (var y = 0; y < this.map.length; y++) {
331.  for (var x = 0; x < this.map[y].length; x++) {
332.  var tile = this.map[y][x];
333.  if (tile.walkable && x !== this.player.x && y !== this.player.y) {
334.  freeCells.push({ x: x, y: y });
335.  }
336.  }
337.  }
338.  return freeCells;
```

```
339.  };

340.  Game.prototype.attackEnemy = function() {
341.  var player = this.player;
342.  var enemies = this.enemies;

343.  for (var i = 0; i < enemies.length; i++) {
344.  var enemy = enemies[i];

345.  if (player.x === enemy.x && player.y === enemy.y) {
346.  var problem = mathProblems[Math.floor(Math.random() * mathProblems.length)];
347.  var answer = window.prompt(problem.question + "\n");
348.  if (problem.correctAnswer === answer) {
349.  enemy.health -= player.damage;
350.  player.health -= 5;
351.  if (enemy.health <= 0) {
352.  enemies.splice(i, 1);
353.  player.experience += 20;
354.  if (player.experience >= player.requiredExperience) {
355.  var rEscail = 1;
356.  rEscail += 1;
357.  player.level += 1;
358.  player.experience -= player.requiredExperience;
359.  player.requiredExperience += 20 * rEscail;
360.  player.damage += 2;
361.  player.health += 30;
362.  this.generateHealthPotion();
363.  }
364.  }
365.  if (player.health <= 0) {
366.  alert('Game over!');
367.  location.reload();
368.  }
369.  break;

370.  } else {
371.  player.health -= 5;
372.  }
373.  }
374.  }
375.  };

376.  Game.prototype.attackBoss = function() {
377.  var player = this.player;
378.  var bosses = this.bosses;

379.  for (var i = 0; i < bosses.length; i++) {
380.  var boss = bosses[i];

381.  if (player.x === boss.x && player.y === boss.y) {
```

```
382.  boss.health -= player.damage;
383.  player.health -= 5;
384.  if (boss.health <= 0) {
385.  bosses.splice(i, 1);
386.  player.experience += 50;
387.  if (player.experience >= player.requiredExperience) {
388.  var rEscail = 1;
389.  rEscail += 1;
390.  player.level += 1;
391.  player.experience -= player.requiredExperience;
392.  player.requiredExperience += 20 * rEscail;
393.  player.damage += 2;
394.  player.health += 30;
395.  this.generateHealthPotion();
396.  }
397.  }

398.  if (player.health <= 0) {
399.  alert('Game over!');
400.  location.reload();
401.  }
402.  break;
403.  }
404.  }
405.  };

406.  Game.prototype.drinkPotion = function() {
407.  var player = this.player;
408.  var potion = this.healthPotion;

409.  if (player.x === potion.x && player.y === potion.y) {
410.  player.health += 50;
411.  delete this.healthPotion;
412.  }
413.  }

414.  Game.prototype.useHint = function() {
415.  var player = this.player;
416.  var hint = this.hint;

417.  if (player.x === hint.x && player.y === hint.y) {
418.  alert('Paris is a capital of France. Remember that, warrior!')
419.  delete this.hint;
420.  }
421.  }

422.  Game.prototype.handleInput = function(e) {
423.  var player = this.player;
424.  var dx = 0;
425.  var dy = 0;
```

```
426.  switch (e.keyCode) {
427.  case 37: // Left arrow key
428.  dx = -1;
429.  break;
430.  case 38: // Up arrow key
431.  dy = -1;
432.  break;
433.  case 39: // Right arrow key
434.  dx = 1;
435.  break;
436.  case 40: // Down arrow key
437.  dy = 1;
438.  break;
439.  case 32: // Space key
440.  this.attackEnemy();
441.  this.attackBoss();
442.  this.drinkPotion();
443.  this.useHint();
444.  break;
445.  }

446.  var newX = player.x + dx;
447.  var newY = player.y + dy;

448.  if (this.isWithinMap(newX, newY) && this.map[newY][newX].walkable) {
449.  player.x = newX;
450.  player.y = newY;
451.  }

452.  this.refresh();
453.  };

454.  Game.prototype.update = function() {
455.  for (var i = 0; i < this.enemies.length; i++) {
456.  var enemy = this.enemies[i];
457.  var dx = Math.floor(ROT.RNG.getUniform() * 3) - 1;
458.  var dy = Math.floor(ROT.RNG.getUniform() * 3) - 1;
459.  var newX = enemy.x + dx;
460.  var newY = enemy.y + dy;

461.  if (this.isWithinMap(newX, newY) && this.map[newY][newX].walkable) {
462.  enemy.x = newX;
463.  enemy.y = newY;
464.  }
465.  }

466.  for (var i = 0; i < this.bosses.length; i++) {
467.  var boss = this.bosses[i];
468.  var dx = Math.floor(ROT.RNG.getUniform() * 3) - 1;
```

```
469.  var dy = Math.floor(ROT.RNG.getUniform() * 3) - 1;
470.  var newX = boss.x + dx;
471.  var newY = boss.y + dy;

472.  if (this.isWithinMap(newX, newY) && this.map[newY][newX].walkable) {
473.  boss.x = newX;
474.  boss.y = newY;
475.  }
476.  }

477.  this.refresh();

478.  if (this.enemies.length === 0) {
479.  this.generateEnemies();
480.  stage++;
481.  if (stage > 4 & stage % 5) {
482.  this.generateBoss();
483.  }
484.  this.generateHint();
485.  }
486.  };

487.  function startGame(algorithm) {
488.  var game = new Game();
489.  game.init(algorithm);
490.  document.getElementById('map-creation-screen').style.display = 'none';
491.  }
```