МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: **Розробка ігрового додатку на JS**

Виконав студент групи **КН-5**
спеціальності 122 Комп'ютерні науки
Міогло Ілля Миколайович

Керівник **ст. викладач**
Штефан Наталія Зінов'ївна

Консультант **док. техн. наук,**
професор
Казакова Надія Феліксівна

Рецензент **к.т.н., доцент**
Сергієнко Андрій Володимирович

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

Платформер – ігровий процес в якому складається зі стрибків персонажа по різноманітним платформам

Tiled – це 2D-редактор рівнів WebGL

Tileset – це набір тайлів (англ. tile – плитка)

Angular – поалтформа для розробки веб-додатків

C++ – мова програмування

JavaScript – мова програмування

jQuery – набір функцій JavaScript, що фокусується на взаємодії JavaScript та HTML

Node.js – крос-платформне середовище виконання JavaScript

React – бібліотека JS з відкритим кодом

Vue.js – JavaScript-фреймворк з відкритим вихідним кодом для створення інтерфейсів користувача

ВСТУП

Відеоігри зробили великий стрибок з моменту появи майже 70 років тому. Графіка та ігровий процес перейшли від простих блоків до досвіду, максимально наближеного до реального. Простіші ігри, включно з багатьма двовимірними, в які легко грати та доступні в Інтернеті, все ще цінуються.

Багато з цих популярних ігор створено за допомогою JavaScript, наразі найпопулярнішої мови програмування у світі. Вона є чудовою мовою для створення гри та ідеально підходить для онлайн і мобільних ігор.

Деякі ігри є зменшеними версіями інших ігор, деякі є оригінальними. Є ігри, які дуже прості, але залишаються складними. Є також ігри, які пропонують чудову 2D-графіку з кількома викликами для користувача. Однією з чудових переваг багатьох із цих ігор є те, що розробники роблять їхній вихідний код доступним для інших творців. Багато з них безкоштовні, але для деяких є і платні версії [1].

JavaScript – мова дуже універсальна працює набагато повільніше, ніж такі мови, як C++, і споживає набагато більше пам’яті. Двовимірні ігри найчастіше асоціюються зі смартфонами, оскільки більшість ігор для мобільних пристроїв є двовимірними через те, що вони менш вимогливі, легші та легші для оптимізації.

Метою проекту є розробка ігрового додатку мовою JavaScript. В якості жанру гри обрано 2Д платформер.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Опис предметної області

Ігри на JavaScript можна грати в браузері або на мобільному телефоні. Використання платформ і інструментів може допомогти створювати як 2D, так і 3D ігри, які можна запускати безпосередньо у вашому браузері. Окрім лише веб-ігор, JavaScript стає все більш популярним у розробці мобільних ігор.

Жанри відеоігор – це категорії ігор, які мають подібні характеристики. Ігри-головоломки змушують в мозок працювати, шутери вимагають від гравця стріляти кулями та снарядами, а симулятори водіння розроблені, щоб перенести стрес і хаос ревучої дороги.

Багато сучасних ігор не просто належать до одного ігрового жанру, а часто містять елементи ігрового процесу з кількох жанрів одночасно. Одним з таких «змішанних» жанрів є платформер. Тут і робота на логіку, на швидкість, але при цьому, зазвичай т акі ігри не перевантажують мозок гравця та надають гарні емоції [2].

Платформери домінували на 2D-сцені з моменту появи класичної аркадної гри Donkey Kong 1981 року – першого в історії платформера. Зазвичай такі ігри характеризується: стрибками та лазінням, щоб досліджувати навколишнє середовище. Існує також багато жанрових відгалужень, зокрема:

- платформери collect-a-thon, де гравцеві потрібно збирати предмети колекціонування, наприклад дорогоцінні камені або шматочки пазла, щоб завершити гру;
- платформери-талісмани, які мають повторюваного головного персонажа та повторювані ігрові механізми;
- головоломки-платформери, де гравець розв’язує головоломки, щоб виконувати цілі або переміщатися між кімнатами [3].

1.2 Вибір мови програмування

JavaScript – це легка інтерпретована мова програмування. Він призначений для створення мережевих додатків. Він доповнює Java та інтегрований з нею. JavaScript дуже легко реалізувати, оскільки він інтегрований з HTML. Він відкритий і кросплатформний.

Javascript є скрізь, він встановлюється в кожному сучасному веб-браузері, тому для вивчення Javascript вам справді не потрібно налаштовувати спеціальне середовище. Наприклад, Chrome, Mozilla Firefox, Safari та всі браузери, які ви знаєте на сьогодні, підтримують Javascript.

Javascript допомагає створювати дійсно красиві та шалено швидкі веб-сайти. Можна розробити свій веб-сайт за допомогою консолі, схожої на вигляд і відчуття, і надати своїм користувачам найкращу графічну взаємодію з користувачем.

Тепер використання JavaScript поширилося на розробку мобільних додатків, розробку настільних додатків та розробку ігор. Це відкриває багато можливостей для вас як Javascript-програміста. Завдяки високому попиту є маса нових робочих місць і висока оплата для тих, хто знає JavaScript.

Чудова перевага Javascript полягає в тому, що розробник знайде безліч уже розроблених фреймворків і бібліотек, які можна використовувати безпосередньо в розробці програмного забезпечення, щоб скоротити час виходу на ринок.

Можуть бути тисячі вагомих причин вивчати програмування на Javascript. Доступно багато корисних фреймворків і бібліотек Javascript: Angular, React, jQuery, Vue.js, Ext.js, Ember.js, Meteor, Mithril, Node.js

Javascript є однією з найпоширеніших мов програмування (інтерфейсу, і серверу). Він присутній майже в усіх сферах розробки програмного забезпечення. Наприклад:

1. Перевірка на стороні клієнта. Це дуже важливо для перевірки будь-яких введених даних користувача перед надсиланням їх на сервер, і

Javascript відіграє важливу роль у перевірці цих введених даних у самому інтерфейсі.

2. Маніпулювання HTML-сторінками – Javascript допомагає маніпулювати HTML-сторінкою на льоту. Це допомагає дуже легко додавати та видаляти будь-які теги HTML за допомогою javascript і змінювати ваш HTML, щоб змінити його зовнішній вигляд на основі різних пристроїв і вимог.
3. Сповіщення користувачів. Ви можете використовувати Javascript, щоб викликати динамічні спливаючі вікна на веб-сторінках, щоб надавати різні типи сповіщень відвідувачам вашого веб-сайту.
4. Завантаження внутрішніх даних – Javascript надає бібліотеку Ajax, яка допомагає завантажувати внутрішні дані, поки ви виконуєте іншу обробку. Це справді дає чудовий досвід відвідувачам вашого веб-сайту.
5. Серверні програми – Node JS побудовано на середовищі виконання Javascript Chrome для створення швидких і масштабованих мережевих програм. Це бібліотека на основі подій, яка допомагає розробляти дуже складні серверні програми, включаючи веб-сервери.

Цей список можна продовжувати: існують різні сфери, де мільйони розробників програмного забезпечення із задоволенням використовують Javascript для розробки чудових веб-сайтів та іншого програмного забезпечення [4].

1.3 Огляд фреймворків для створення ігор на JavaScript

Розробка ігор, безсумнівно, виснажлива вправа, але можна полегшити біль за допомогою правильних інструментів. На щастя, двигуни JavaScript надають можливості розширеного рівня порівняно з іншими механізмами.

Вони пропонують широкий вибір інструментів і доповнень, які покращують якість процесу розробки гри.

Фреймворки – це бібліотеки коду, які містять деякий попередньо написаний код. Це дуже корисно для розробників, оскільки економить час, який можна витратити на рутинні завдання. Лише фреймворк має бути належним чином пов’язаний із IDE, яка буде використовуватися.

«Babylon.js» (рис. 1) – це 3D-технологія веб-рендерінгу нового покоління. Він відповідає за створення провідних галузевих ігор, таких як Space Invaders і Temple Run 2. Двигун має низку провідних галузевих функцій, зокрема камеру спостереження та джерела світла, які є зручними інструментами для створення великих і багатофункціональних ігор [5].

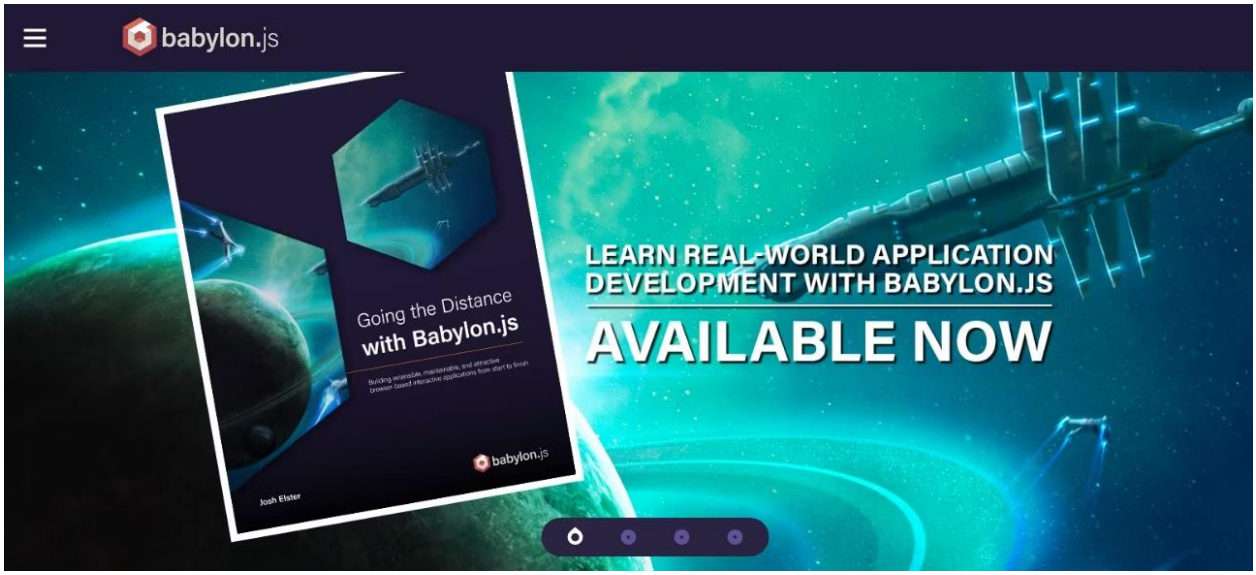

Рисунок 1 – Babylon.js

Крім того, це дозволяє розробникам писати, рендерити та відображати код на різних платформах браузера. Крім того, Babylon.js дозволяє відображати 3D-графіку у веб-браузері за допомогою HTML5.

«Babylon.js» має вбудований профайлер продуктивності, який спрощує керування продуктивністю та налагодження. Він також надає необмежений

доступ до цілей трансформації та має редактор кривих анімації, який дозволяє створювати та змінювати анімацію.

Іншою перевагою Babylon.js є його прагнення допомогти вам створити багаті графічні інтерфейси користувача. Механізм «Babylon.js» має набір інструментів змішаної реальності, який дає вам доступ до елементів XR/UR, 3D-повзунків і сенсорних голографічних кнопок для вдосконалення ваших ігрових сцен.

Більше того, розробники надають детальні посібники щодо поширених проблем, з якими стикаються більшість розробників, які тільки починають працювати з двигуном. Таким чином Babylon.js гарантує, що ви ніколи не почуватиметеся поза межами своєї глибини.

Phaser (рис. 2) – це двовимірний механізм із відкритим вихідним кодом із важливими функціями для створення ігор на основі WebGL та Canvas. Цей механізм спеціально призначений для розробників ігор, які прагнуть втілити свої ігрові ідеї в реальність [5].

Рисунок 2 – Phaser

За допомогою Phaser ви можете створити інтерактивну гру та розгорнути її в Інтернеті за допомогою таких інструментів, як спостереження за камерою, звук, масштабування пристрою, анімація та мобільний браузер.

Незважаючи на те, що Phaser є безкоштовним, він дозволяє створювати кросплатформні ігри, у які користувачі можуть грати в будь-якому сучасному веб-браузері, мобільному пристрої чи комп'ютері. Більше того, розпочати роботу з Phaser легко, оскільки існує багато посібників, які допоможуть розробникам ігор. Phaser також розпізнає кілька систем введення, має можливості підтримки звуку та є сумісним із WebGL.

Play Canvas – це ігровий движок з інфраструктурою, яка підтримує як 2D, так і 3D графіку в іграх (рис.3). Цей движок має функції полегшення та вбудований редактор для фізичного відтворення та інтеграції 3D-матеріалів. Головною перевагою Play Canvas є те, що він не обмежується лише розробниками ігор. Таким чином, бренди, яким потрібні анімовані візуальні ефекти для маркетингу або створення моделей нерухомості, знайдуть цей механізм корисним [6].

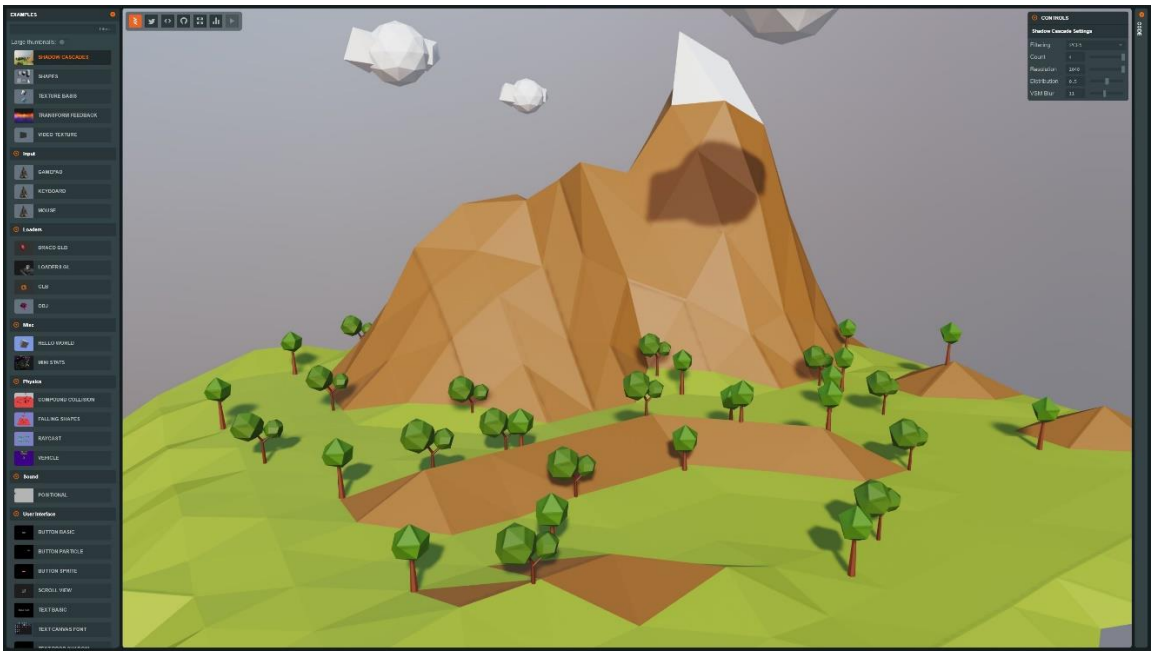

Рисунок 3 – Скрін Play Canvas

Можна використовувати Play Canvas для створення інтерактивних ігор і вмісту, написання та тестування коду, створення сцен і експорту вмісту на інші платформи. Веб-програма Play Canvas також дозволяє з легкістю створювати браузерні ігри.

Крім того, Play Canvas економить час, дозволяючи інтегрувати фізику у вашу гру. За допомогою цього механізму ви також можете писати сценарії та налаштовувати вміст відповідно до своїх уподобань. Більше того, не потрібно перезавантажувати браузер щоразу, коли вносяться зміни.

Новачкові корисно знати деякі основні особливості кількох різних фреймворків і вибрати один для роботи залежно від того, що ви хочете робити зі своєю грою. Як згадувалося у вступі, щоб створити гру на JavaScript, доступні як 2D, так і 3D двигуни. Також важливо вміти розрізняти фреймворки за HTML, що необхідно для розробки веб-ігор. Для початківців зазвичай вибирають безкоштовний фреймворк, оскільки для нього також є багато допомоги в Інтернеті [6].

1.3 Огляд аналогів на JS

«CrossCode» (рис. 4) з насиченою графікою був представлений у 2012 році та з тих пір розвивався, включивши розширення, і його можна було грати на ПК, Mac і консолях.

Гра розповідає про Леа, яка повинна пройти через «CrossWorlds» у пошуках, щоб відновити свої спогади. Ця двовимірна гра, створена за допомогою JavaScript, стала складнішою, додавши квести, головоломки, дерева навичок тощо. Його добре сприйняли з моменту появи.

Гра має безкоштовну демо-версію, яка доступна в Інтернеті, але різноманітні платні версії доступні за $19,99-$39,99 у Steam [7]

Рисунок 4 – Скрін сцени гри «CrossCode»

Наступний аналог – «OpHog» (рис. 5) тут гравцям у грі потрібно перемогти босів в інших світах, захищаючи портали у своєму світі. У грі також є можливість збирати діаманти, щоб купувати предмети, які допоможуть досягти цілей.

Рисунок 5 – Скрін сцени гри «OpHog»

«Tower Construction» – ще один приклад гри на JS (рис. 6). Будівництво вежі, нещодавнє доповнення, вимагає від вас побудувати вежу з кількома рівнями, з'єднаними з краном, який гойдається. він складний для гри на JavaScript і має чудову естетику. Нижче наведено стандартні правила гри.

Рисунок 6 – Скрін сцени гри «Tower Construction»

Кожного разу, коли блок вежі випадає, гравець вираховує 1 hp; гра закінчується, коли hp закінчується. Гравець отримує 25 балів за кожен успішно складений блок (успіх). Якщо блок ідеально (ідеально) укладено поверх попереднього, гравець натомість отримує 50 балів. Consecutive Perfects присуджує додаткові 25 балів [8].

«The Jump Guy» (рис. 7) створений за допомогою JavaScript, HTML& CSS(Front-end). Ця гра була розроблена з використанням мови програмування JavaScript..

Додаток веселий і простий, це може бути цікаво, якщо грати удвох. У гру можна грати за допомогою прив'язок клавіатури (клавіша зі стрілкою

вліво, щоб рухатися вліво, клавіша зі стрілкою вправо, щоб рухатися вправо, клавіша зі стрілкою вгору, щоб стрибати), тобто в грі заложено просту механіку.

Рисунок 7 – Скріни гри «The Jump Guy»

Основна мета гри – здобувати монети по дорозі й уникати удару ворогів, що наближаються. Щоразу, коли гравець отримує монету, він отримує величезну кількість очок. Також можна вбивати свого ворога, коли стрибаєш та топтаєш його головою. Гра складається з різних рівнів, які намагаються пройти всі рівні та отримати якомога більше очок.

Крім цього, гра передбачає елементарний туторіал при запуску, який допомогає зорієнтуватися щодо правил гри та кнопок управління [9].

Останній аналог буде взято за основу для дипломного проекту. З урахуванням того, що проект є першим на мові JavaScript та наявність короткого ліміту часу для його розробки, його буде реалізовано на JS, HTML & CSS без використання додаткових фреймворків.

1.4 Tiled Map Editor

Tiled – це 2D-редактор рівнів, який допомагає розвивати вміст гри (рис. 8). Його основною функцією є редагування плиткових карт різної форми, але він також підтримує безкоштовне розміщення зображень, а також ефективні способи додавання анотацій вашого рівня додатковою інформацією, яка використовується в грі. Tiled зосереджується на загальній гнучкості, намагаючись залишатися інтуїтивно зрозумілим.

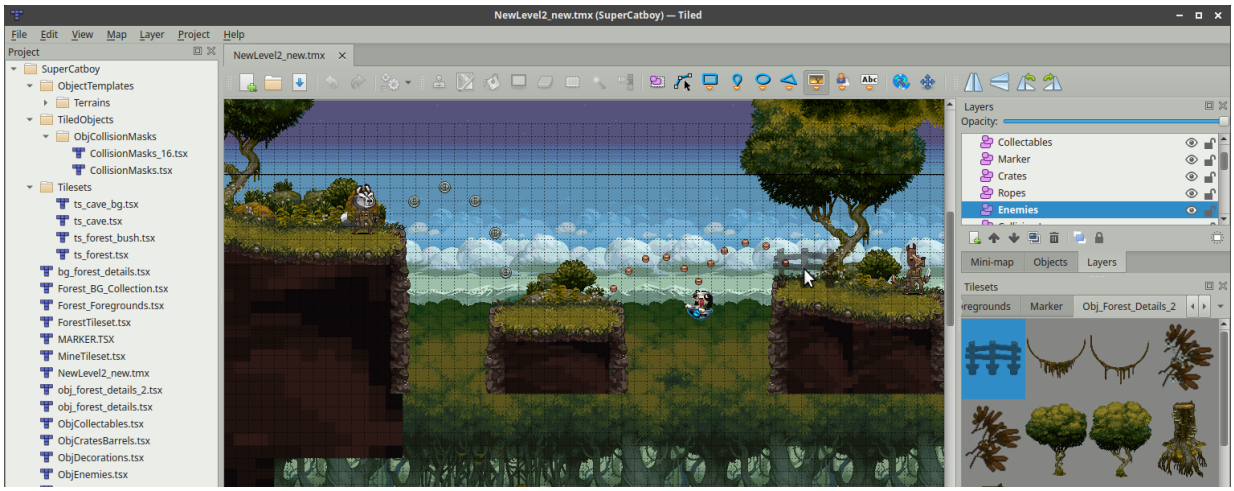

Рисунок 8 – Tiled Map Editor

З точки зору мозаїчних карт, він підтримує прямі прямокутні мозаїчні шари, а також проекційні ізометричні, змішані ізометричні та змішані шестикутні шари. Набір плиток може являти собою одне зображення, що

містить багато плиток, або це може бути колекція окремих зображень. Щоб підтримувати певні методи фальсифікації глибини, плитки та шари можна зміщувати на спеціальну відстань, а також можна налаштовувати порядок їх візуалізації.

Основним інструментом для редагування шарів плитки є щітка для штампу, яка дозволяє ефективно малювати та копіювати ділянки плитки. Він також підтримує малювання ліній і кіл. Крім того, є кілька інструментів вибору та інструмент, який виконує автоматичні переходи рельєфу. Нарешті, він може застосовувати зміни на основі зіставлення шаблонів для автоматизації частин вашої роботи.

Tiled також підтримує шари об’єктів, які традиційно використовувалися лише для анотування вашої карти інформацією, але останнім часом їх також можна використовувати для розміщення зображень. Ви можете додавати об’єкти прямокутник, точка, еліпс, багатокутник, полілінія та плитка. Рівні об’єктів пропонують велику гнучкість для додавання до вашого рівня майже будь-якої інформації, яка потрібна вашій грі.

Інші речі, які варто згадати, це підтримка додавання користувальницьких форматів карт або наборів плиток за допомогою плагінів, розширення Tiled за допомогою JavaScript, пам’ять штампів плиток, підтримка анімації плиток і редактор зіткнень плиток [10].

1.5 Постановка завдання

Аналіз предметної області, огляд існуючих аналогів та засобів розробки дають уявлення щодо вимог до дипломного проекту:

1. Проектування гри у стилі платформера (концепт-документ).
2. Підбір графічного контенту.
3. Моделювання ігрового простору.
4. Реалізація механіки засобами HTML та JS.
5. Тестування.

2 ПРОЕКТУВАННЯ

2.1 Концепт-документ

2.1.1 Вступ

Робоча назва проекту: «Сторожева вежа».

В цьому документі описані основні принципи розробки гри під назвою «Сторожева вежа». Тут буде розглянуто головні механіки гри, функціональні вимоги до розробки, а також детальний опис графіки та інтерфейсу користувача. Метою розробки є створення захоплюючої 2D-гри.

2.1.2 Жанр та аудиторія

«Сторожева вежа» – це платформер, де гравець через логічне мислення повинен розставити вдовж дороги до міста сторожеві вежі для захисту від гоблінів. Кожна вежа буде запускати ядра. Вона входить до категорії "хардкорних" аркадних ігор, тому що вимагає від гравця високої концентрації та швидкої реакції, щоб уникати зіткнень з перешкодами. Часу на роздуми буде обмежено, крім того, локації для можливої встанови вежі також обмежені.

Процес геймплею залишається актуальним незалежно від вікової групи або інших показників гравця. Аудиторією такої гри можуть бути як і маленькі діти, бо геймплей є дуже простим для розуміння, так і дорослі геймери, які хочуть відчути ностальгію олдскульних аркадних ігор.

Гра не містить вбивств, замість цього до кожного ворога буде прикріплено показчик його життевих ресурсів. Тож, цільова аудіторія від 5 років.

2.1.3 Основні особливості гри

«Сторожева вежа» має кілька ключових особливостей, які роблять її відмінною від інших ігор:

1. Простий геймплей: основна мета гри – захистити єдиний шлях до міста від нападків гоблінів. Гравець керує розміщенням веж, їх кількість

обмежена. Тож, гравцю необхідно збирати додаткові бали для будови нових захисних локацій. Бонуси будуть додаватить за кожне знищення ворога.

2. Гра вимагає від гравця великої уваги, концентрації та швидкості реакції, щоб уникнути ворогів. На початковому рівні на полі будуть помічені можливі локації для захистих веж, але ставити поряд дві заборонено. Тож гравцю прийдеться швидко аналізувати стан на ігровому просторі, дивитись напрям влучання зарядів веж у ворогів. Крім того, з часом швидкість гри буде прискорюватись и кількість гоблінів зростатиме.

При максимально успішному розміщенні захисних веж гравець перейде на наступнпий рівень, де карта дороги буде змінено і вже будуть відсутні помічені локації під розміщення захисних споруд. Тут гравцю прийдеться аналвзувати стан на основі власного досвіду з попереднього рівня.

3. Гра має простий, але ефектний дизайн, який дуже приваблює гравців. Анівмація спрайтів плавна та не буде перевантажувати зір гравця. Графічний контент буде взято з інтернет-ресурсів, які предоставляють широкий спектр спрайтів у вільному доступі.

4. Відсутність історії: гра не має складної сюжетної лінії, тож грати в неї можна буде скільки завгодно, намагаючись покращувати свій рекорд. Це сприяє залученню гравців до гри та збільшенню часу гри в цілому.

2.1.4 Опис гри

«Сторожева вежа» – це гра, де гравець керує захисними вежами на шляху до міста. Гравець мусить уникати проходження ворогів до ворот міста, які зявляються зі збільшенням кількості. Один з найбільших викликів полягає у відсутності можливості зупинитися на момент, коли починається наступ. Таким чином, гравець повинен швидко реагувати та правильно використовувати моменти, щоб оптимізувати механізм захисту.

Гра має дуже простий дизайн, але при цьому дуже ефектний. Кольори та оформлення нагадують піксельні ігри з 8-бітних ігрових консолей, що створює враження, що гравець знаходиться в дитинстві та грає на старій консолі.

Основна мета гри – збирати якомога більше балів, та розміщення сторожевих веж найоптімальнішим шляхом. Для зручності у верхній частіні ігрового простору буде розміщено інформацію щодо зароблених мбонусів та рівня «життевих ресурсів» самого гравця. З кожним гоблином, якому вдасться пройти через вежі, у гравця буде відніматися одне життя . Гра має дуже високу реплеєрність, оскільки гравець може грати скільки завгодно довго, пробуючи покращувати свій рекорд.

2.1.5 Порівняння та передумови створення

Концепт та ідею гри було сформовано на основі існуючих ігор у даному стилі. Вони не вимагають складної механіки чи авторського контенту, тож є гарним прикладом для перших проектів по розробці ігрових застосунків.

Проте, слід мати на увазі, що конкуренція в галузі інді-ігрової розробки дуже висока, і багато розробників вже створили свої версії схожих сценаріїв. Тому, щоб зробити свій проект унікальним та привабливим для гравців, потрібно ретельно продумати у подальшому дизайн та додати нові елементи геймплею, щоб розробка не була простим копіюванням вже існуючих проектів.

При розробці використовуються матеріали та ресурси, що знаходяться у відкритому доступі та є вільними для поширення та використання [11].

2.2 Опис дизайну

Для створення дизайну гри в жанрі 2D платформера потрібно враховувати декілька ключових факторів, таких як графіка, механіка, звукове оформлення та рівневий дизайн.

Графіка має бути чіткою та яскравою, щоб допомогти гравцям легко орієнтуватися в ігровому світі та швидко реагувати на те, що відбувається на екрані.

Механіка має бути простою та інтуїтивно зрозумілою. Звукове оформлення має доповнити графіку та механіку, щоб створити максимально повний та емоційний ігровий досвід.

Рівневий дизайн має бути унікальним та цікавим. Кожен рівень має бути продуманий та розроблений таким чином, щоб гравці могли отримати задоволення від проходження, але водночас залишатися викликом.

Персонажі мають бути цікавими та харизматичними. Вони повинні мати унікальні особливості та здібності, які допоможуть їм долати рівні та перемагати противників.

2.2.1 Призма елементарного зошиту

У гейм-дизайні є таке поняття, як «призми» – це набори правил або питань, відповіді на які допомагають максимально ефективно створювати ігрові застосунки. Перша з них, це призма елементарного зошита.

Одним із способів покращення дизайну є покращення деталізації графіки, оскільки вона може допомогти зробити гру більш привабливою та цікавою для гравців. Наприклад, можна додати більш детальні текстури, особливості персонажів, покращити фонові елементи та багато іншого.

Також можна додати анімації. Це може бути анімація бігу, атак, а також анімації оточення, такі як рух дерев, хвиль на воді та багато іншого. Не слід забувати про звуки та музику, які можуть посилити враження від гри. Можна додати звуки ударів, кроків, голоси персонажів та багато іншого. Музика допоможе створити потрібну атмосферу та передати емоції гравцям.

Також завжди можна покращити геймплей. Навіть якщо на перший погляд гейплей гарний, необхідно пам’ятати, що немає межі досконалості. Треба бути певним, що гра завжди буде цікавою та викликатиме у гравців бажання продовжувати грати.

2.2.2 Призма голографічного дизайну

При роботі з нею слід описати наступне:

1. Елементи, які роблять досвід приємним: плавне керування персонажем, гравець повинен мати повний контроль над своїм персонажем, і переміщення має бути плавним та природним.
   - простота та ясність геймплею: гравець повинен легко розуміти, що відбувається на екрані, і що він повинен робити, щоб пройти рівень.
   - можливість для гравця дослідити світ: гравцям має бути надана можливість проходити рівень та переходити на наступний.
   - захоплюючі рівні та завдання: гравцям мають бути запропоновані різні рівні, включаючи різноманітні механіки гри, які вони мають освоїти, щоб просуватися вперед (для наступних рівнім слід додати новий тип захисної вежі з унікальною механікою).
2. Елементи, які можуть негативно впливати на досвід у 2D платформерах:
   - погано налаштовані контролери: погане керування або несподівані зміни в управлінні можуть суттєво вплинути на ігровий досвід.
   - погано спроектовані рівні: рівні мають бути уважно спроектовані, щоб надати достатньо виклику, але не настільки складними, що гравець не зможе їх пройти.
   - перевантажений інтерфейс: гравцям може бути важко розібратися у складних чи перевантажених інтерфейсах, що може відштовхувати їхню відмінність від гри.
   - занадто високий рівень складності: якщо гра надто складна, то гравці можуть швидко втратити інтерес та перестати грати.

Щоб покращити досвід, можна: ретельно опрацювати управління, переконатись, що управління механікою плавне та природне, і що гравці мають повний контроль над своїм персонажем.

Зробити рівні цікавими та захоплюючими, додати різноманітних механік, секретів та викликів у рівні, щоб гравці не втрачали інтересу.

Зробити інтерфейс зрозумілим та інтуїтивний, простий у використанні та не перевантажений надмірними елементами. Ретельно продумати баланс складності, баланс між викликом та досяжністю рівнів, щоб гравці не відчували себе занадто обмеженими чи невпевненими у своїх здібностях.

2.2.3 Призма потреб

Гра забезпечує гравцю безпеку, свободу та самовираження, досягнення цілей та задоволення, вона задовольнятиме вищі рівні потреб, такі як самоактуалізація та самореалізація. Щоб зробити гру більш задовільною для гравців, можна звернутися до теорії основних потреб, яка стверджує, що люди мають низку базових потреб, які потрібно задовольняти, щоб досягти високої задоволеності та благополуччя.

Потреба безпеки: гравці повинні відчувати себе безпечно в грі, не відчуваючи тривоги або занепокоєння. Можна додати в гру механіки, які допоможуть гравцям відчувати себе у безпеці, наприклад, збереження прогресу у грі, резервні точки чи систему автозбереження.

Щоб гра задовольняла ці потреби ще краще, необхідно:

- зосередитись на покращенні механік та функцій, які вже є у грі;
- додати нові механіки та функції, які покращать досвід гри для гравців;
- переконатись, що гра досить різноманітна і має достатній рівень складності, щоб вона не ставала нудною чи надто легкою;
- забезпечити гравцям можливість самовиражатися та налаштовувати свій досвід гри, наприклад, шляхом вибору персонажа або налаштування складності гри (карта місцевості та різні вороги).

2.3 Візуальне проектування карти

Tiled допомагає проектувати ігровий простір, завантаживши в нього необхідні набори тайлів, та після інтерпретувати карти до ігрового рущія. Tileset – це набір тайлів (англ. tile – плитка), пов'язаних загальною тематикою і підходять один до одного без помітних швів у тих місцях, де тайли з'єднуються один з одним. Першим кроком є завантаження тайлсету до проекту (рис. 9)

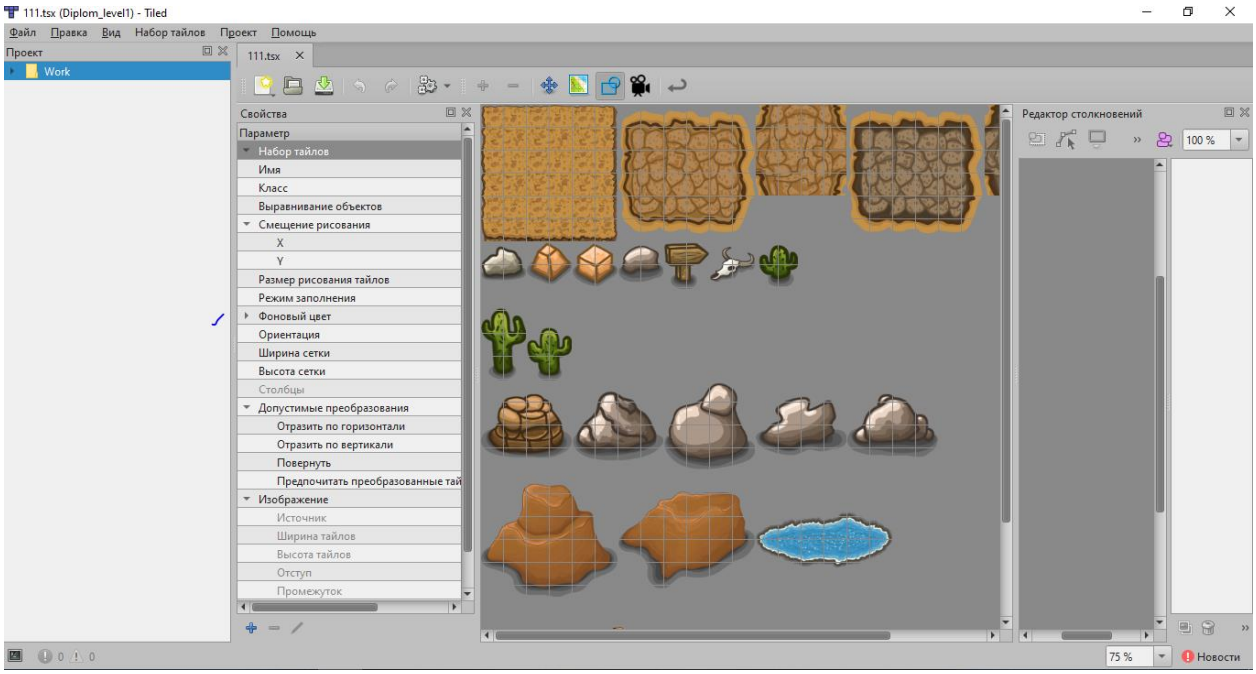

Рисунок 9 – Завантаження тайлів

Далі, використовуючі слої у Tiled проектуємо карту для ігрового простору (рис. 10). Це буде дорога на фоні пустелі з додаванням декількох додаткових елементів. Так як основна динаміка гри відбуватиметься на дорозі, слід не перевантажити фонову локацію.

Рисунок 10 – Проект дороги

Після того, як макет дороги сформовано, слід продумати про переміщення ворогів. Ідеально для відображення ефекту появи ворога на початку дороги відмальовувати його (розмістити стартовий спрайт) перед ігровою сценою (тут слід урахувати розміри спрайтів). На рисунку 11 показано, як на слой «Object Layers1» через об’єкт лінії було нанесено трек переміщення гоблінів. Він складається з 10 точок.

Рисунок 11 – Скрін нанесення основних точок треку для ворога

Новий слой зберігаємо як js-файл (рис. 12), він містить масив координат ключових точок для треку ворога. Це допоможе при програмуванні механіки гри.

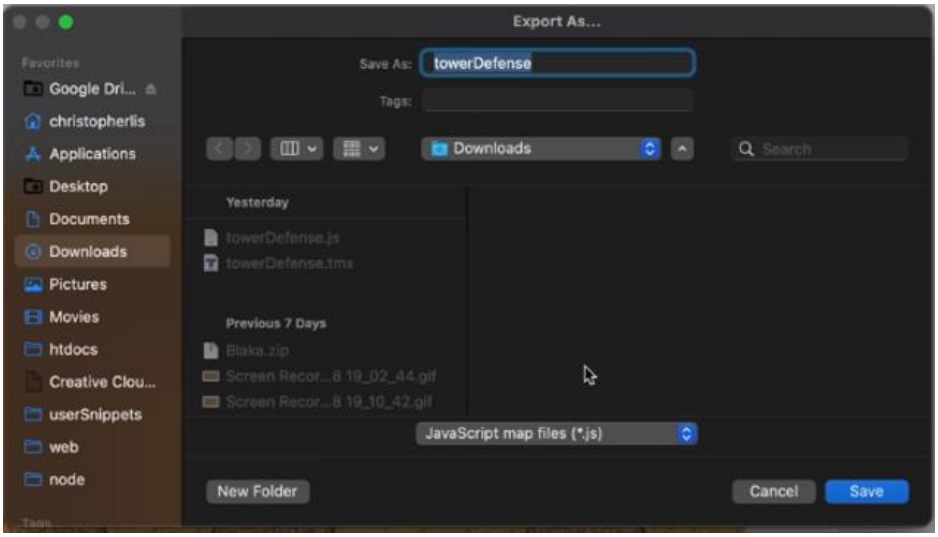

Рисунок 12 – Збереження координат треку

В результаті, файл «Waypoints.js» має наступний зміст:

```
const waypoints = [
  {
    x: -124.21331566744,
    y: 475.322954620735
  },
  {
    x: 276.581649552832,
    y: 472.01059953627
  },
  {
    x: 276.581649552832,
    y: 157.33686651209
  },
  {
    x: 735.342828751242,
    y: 158.993044054323
  },
  {
    x: 735.342828751242,
    y: 405.763497846969
  },
```

```
  {
    x: 606.160980457105,
    y: 407.419675389202
  },
  {
    x: 606.160980457105,
    y: 669.095727061941
  },
  {
    x: 1046.70420669096,
    y: 665.783371977476
  },
  {
    x: 1050.01656177542,
    y: 288.17489234846
  },
  {
    x: 1407.75091089765,
    y: 286.518714806227
  }
]
```

3 ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

Під час розробки гри будь-якою мовою програмування добре мати підготовлену папку з ресурсами. Це включатиме всі фонові текстури, гравців, ворожих (помічних) персонажів, анімацію, звукові ефекти тощо. Є багато сайтів, які пропонують безкоштовні або дешеві аркуші спрайтів, фонові блоки, звуки та деталі [12].

Якщо говорити про елементи керування цією грою на ПК, то все просто. Для пересування потрібно використовувати лише клавіші зі стрілками вліво, вправо, вгору та вниз (відповідно до концепт-документу). Це проста двовимірна гра з використанням JavaScript. Усі ігрові функції налаштовано з Javascript, тоді як HTML і CSS налаштовано для макетів. Для розробки цієї двовимірної гри були використані різні спрайти.

Так, структура проекту складається з наступних директорій та файлів:

```
Project/
      |-- img
      |-- js/
            |--classes/
                        |--Buildings.js
                        |--PlacementTile.js
                        |--ProjecTile.js
                        |--Sprite.js
            |--index.js
            |--placementTilesData.js
            |--waypoints.js
```

Для реалізації ігрової механіки необхідні: карта ігрового простору, спрайти ворогів, спрайти сторожевої вежі з відображенням анімації руху, а також анімація ядра вежі (зброя проти гоблінів).

3.1 Опис об’єктів

За ідеєю гри, гравцю на початку дається 10 життів (кожен пройшовши трек гоблін буде забирати одне життя) та 100 монет. Покажчики відображаються в верхньому правому кути екрану (рис. 13).

Рисунок 13 – Сцена ігрового простору

Наступний крок: слід попрацювати з листом спрайтів для ворога. Ідея використання зображення спрайту полягає в тому, щоб використовувати одне єдине зображення, яке містить усі анімації персонажа, замість того, щоб мати справу з кількома зверненнями сервера до кількох окремих файлів.

Таким чином, замість того, щоб зберігати кожен кадр анімації в окремих файлах і виконувати кілька викликів сервера для отримання кожного файлу для відтворення анімації, аркуш спрайтів може замість цього зберігати всю анімацію в одному файлі зображення, яке завантажується один раз (тільки один виклик сервера), коли сторінка завантажується.

Такий підхід забезпечує меншу затримку в Інтернеті, оскільки весь файл доступний браузеру під час першого завантаження сторінки, що забезпечує швидшу та плавнішу анімацію.

Для відображення руху гобліна було обрано такий спрайт-лист (рис. 14).

Рисунок 14 – Спрайти для анимації гоблинів

Для відображення динаміки сторожевої вежі у спрайт-листу знаходиться 10 зображень у png-фійлі (рис. 15):

Рисунок 15 – Спрайт-лист для сторожевої вежі

3.2 Анімація спрайтів

Щоб анімувати спрайт, слід просто відобразити на екрані кожен його кадр, використовуючи метод drawImage(). Цей метод, який дозволяє намалювати зображення або відео в Canvas. Нижче показані синтаксис drawImage():

```
context.drawImage(img, sx, sy, swidth, sheight, x, y, width,
height);
```

Далі пишемо перевірочну умову, якщо currentFrame менше, ніж загальна кількість кадрів, то робимо інкремент (збільшуємо на один) currentFrame. CurrentFrame використовується для визначення, який фрейм спрайту повинен бути показаний на canvas, використовуючи метод drawImage(). Нарешті ми використовуємо “setInterval()” для виконання функції малювання кожні 100 мілісекунд:

```
<script>
  var width = 100,
      height = 100,
      frames = 4,
      currentFrame = 0,
      canvas = document.getElementById("myCanvas");

  var ctx = canvas.getContext("2d");
  var image = new Image();
      image.src = 'sprite_tower.png';

  var draw = function() {
      ctx.clearRect(0, 0, width, height);
      ctx.drawImage(image, 0, height*currentFrame, width, height,
0, 0, width, height);

      if (currentFrame == frames) {
        currentFrame = 0;
      } else {
        currentFrame++;
      }
  }
   setInterval(draw, 100);
</script>
```

Тут:

- width: ширина елемента canvas;
- height: висота елемента canvas;
- frames: всього кількість фреймів у спрайті;
- currentFrame: поточний кадр у спрайті;
- canvas: для доступу до контекстного елемента візуалізації;
- ctx: візуалізація контексту canvas у 2D форматі;
- image: змінна, містить PNG файл.

На ігровому просторі гравець повинен розставити сторожеві вежі. Локація та їх кількість напряму будуть впливати на результат гри. Для підказки на поле було розміщено додаткові тайли (світлішого кольору), які маркирують можливі місця для веж (рис. 16).

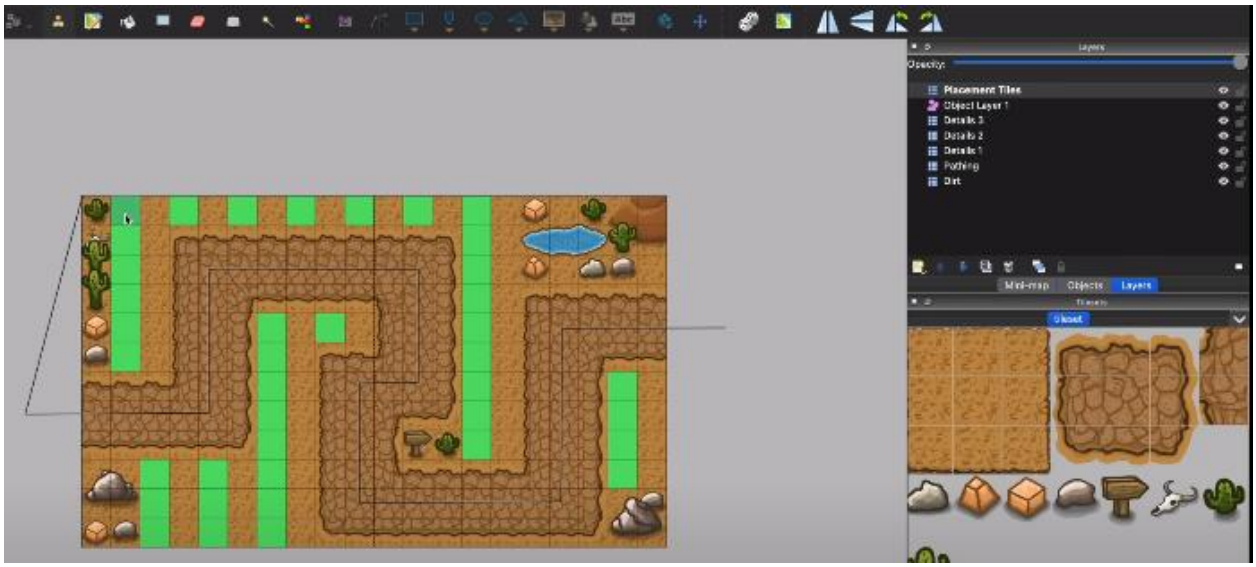

Рисунок 16 – Скрін поля з можливими локаціями для вежі

Для збереження координат таких локацій слід створити новий шар, нанести тайли та зберегти у js-файл «placementTilesData.js»:

```
const placementTilesData = [
  0, 14, 0, 14, 0, 14, 0, 14, 0, 14, 0, 14, 0, 14, 0, 0, 0,
0, 0, 0, 0, 14, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0, 14,
0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
       14, 0, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 14, 0, 14, 0, 0,
0, 0, 14, 0, 0, 0,
       0, 0, 0, 0, 14, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 14, 0, 0,
0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 14, 0, 0,
0, 0, 0, 0, 0, 14,
       0, 0, 0, 0, 0, 0, 14, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0,
14, 0, 0, 0, 0, 0,
       0, 14, 0, 0, 0, 0, 14, 0, 0, 0, 14, 0, 14, 0, 14, 0, 0, 0,
0, 0, 0, 0, 0, 0,
       0, 0, 14, 0, 0, 0, 14, 0, 14, 0, 14, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0,
       0, 14, 0, 14, 0, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
     ]
```

Таким чином, при спробі розмістити вежу буде перевірятись значення масиву: якщо 0-розміщення заборонено, якщо 14-можна ставити. Розглянемо файл «Building.js» більш детально:

```
class Building extends Sprite {
  constructor({ position = { x: 0, y: 0 } }) {
    super({
      position,
      imageSrc: './img/tower.png',
      frames: {
        max: 19
      },
      offset: {
        x: 0,
        y: -80
      }
    })
```

Тут йде звернення до файлу з тайлами «tower.png» та відображення сторожевої вежі:

```
    this.width = 64 * 2
    this.height = 64
    this.center = {
      x: this.position.x + this.width / 2,
      y: this.position.y + this.height / 2
```

```
        }
        this.projectiles = []
        this.radius = 250
        this.target
      }

      draw() {
        super.draw()  }
      update() {
        this.draw()
        if (this.target || (!this.target && this.frames.current
!== 0))
          super.update()

        if (
          this.target &&
          this.frames.current === 6 &&
          this.frames.elapsed % this.frames.hold === 0
        )
          this.shoot()
      }

      shoot() {
        this.projectiles.push(
          new Projectile({
            position: {
              x: this.center.x - 20,
              y: this.center.y - 110
            },
            enemy: this.target
          })
        )}
    }
```

3.3 Реалізація механіки ворогів

Далі слід урахувати відображення рівня життя ворога. Вежі будуть випускати снаряди, але як зрозуміти гравцю чи вдалося знищити ворога – буде показувати «health bar».

Для зручності його розміщуємо поверх спрайту ворога. Це допоможе слідкувати одночасно за станом багатьох гоблинів (рис. 17):

Рисунок 17 – Скрін стану «health bar»-ворогів

У файлі «Enemy.js» це реалізовано наступним чином:

```
class Enemy extends Sprite{
  constructor({ position = { x: 0, y: 0 } }) {
    super({
      position,
      imageSrc: 'img/orc.png',
      frames: {
        max: 7
      }
    })

    this.position = position
    this.width = 100
    this.height = 100
    this.waypointIndex = 0
    this.center = {
      x: this.position.x + this.width / 2,
      y: this.position.y + this.height / 2
    }
    this.radius = 50
    this.health = 100
    this.velocity = {
      x: 0,
      y: 0
    }
  }
```

Функція draw() відповідає за відображення стану життя ворогів:

```
draw(){
    super.draw()
```

```
    //health bar
    c.fillStyle = 'red'
    c.fillRect(this.position.x, this.position.y - 15,
this.width, 10)

    c.fillStyle = 'green'
    c.fillRect(
      this.position.x,
      this.position.y - 15,
      (this.width*this.health)/100,10)
  }

draw() {
    super.draw()
```

Фіксуємо розміри для «health bar» та його кольори:

```
c.fillStyle = 'red'
    c.fillRect(this.position.x, this.position.y - 15,
this.width, 10)

    c.fillStyle = 'green'
    c.fillRect(
      this.position.x,
      this.position.y - 15,
      (this.width * this.health) / 100,
      10
    )
  }

  update() {
    this.draw()
    super.update()
```

Так як вороги з’являються групами і кожного разу їх кількість збільшується на дві одиниці (це забезпечує динаміку розвитку ігрового процесу) слід відображати спрайти гоблинів з маленьким відступом, щоб зображення не накладалися один на одне:

```
    const waypoint = waypoints[this.waypointIndex]
    const yDistance = waypoint.y - this.center.y
    const xDistance = waypoint.x - this.center.x
    const angle = Math.atan2(yDistance, xDistance)
```

Швидкість руху ворогів буде дорівнювати трьом. Якщо взяти більший покажчик – рух спрайтів буде швидким і гравець просто не встигатиме обмірковувати свої дії.

```
const speed = 3
    this.velocity.x = Math.cos(angle) * speed
    this.velocity.y = Math.sin(angle) * speed
    this.position.x += this.velocity.x
    this.position.y += this.velocity.y
    this.center = {
      x: this.position.x + this.width / 2,
      y: this.position.y + this.height / 2
    }
    if (
      Math.abs(Math.round(this.center.x) -
Math.round(waypoint.x)) <
        Math.abs(this.velocity.x) &&
      Math.abs(Math.round(this.center.y) -
Math.round(waypoint.y)) <
        Math.abs(this.velocity.y) &&
      this.waypointIndex < waypoints.length - 1
    ) {
      this.waypointIndex++
    }
  }}
```

3.4 Реалізація функцій у «Index.js»

Розглянемо «index.js» більш детально. В ньому прописані три основні функції:

- func onload();
- func spawnEnemies();
- func animate().

Спочатку слід задати розмірі екрану та константу з масивом тайлів ігрового простору:

```
const canvas = document.querySelector('canvas')
const c = canvas.getContext('2d')

canvas.width = 1280
canvas.height = 768
```

```
c.fillStyle = 'white'
c.fillRect(0, 0, canvas.width, canvas.height)
const placementTilesData2D = []
```

Так як, на полі у нас по горизонталі і вертикалі всього 20 тайлів, то відповідні діапазоні вказуються у циклі:

```
for (let i = 0; i < placementTilesData.length; i += 20) {
  placementTilesData2D.push(placementTilesData.slice(i, i + 20))
}
const placementTiles = []
```

Наступний крок – перевірка локацій (тайлу) на дозвіл розміщення сторожової вежі (якщо значення елементу масиву буде 14):

```
placementTilesData2D.forEach((row, y) => {
  row.forEach((symbol, x) => {
    if (symbol === 14) {
      // add building placement tile here
      placementTiles.push(
        new PlacementTile({
          position: {
            x: x * 64,
            y: y * 64
          }
        })
      )
    }
  })
})
```

Генерація ворогів:

```
const image = new Image()
image.onload = () => {
  animate()
}
image.src = 'img/gameMap.png'
const enemies = []

function spawnEnemies(spawnCount) {
  for (let i = 1; i < spawnCount + 1; i++) {
    const xOffset = i *150
```

```
    enemies.push(
      new Enemy({
        position: { x: waypoints[0].x - xOffset, y: waypoints[0].y
}
      })
    )
  }
}
```

При запуску гри слід виставити значення за замовчуванням, а саме кількість монет та рівень життєвих ресурсів гравця.

```
const buildings = []
let activeTile = undefined
let enemyCount = 3
let hearts = 10
let coins = 100
const explosions = []
spawnEnemies(enemyCount)
```

Анімація ворогів, функція animate(). Тут йде перевірка на кількість ворогів, що встигли перейти дорогу, тобто «скрилися» за розмірами canvas. У цьому разі буде зменшено кількість життів для гравця на кількість одиниць ворогів. Якщо кількість сердець дорівнює нулю – гру завершено з програшом.

```
function animate() {
  const animationId = requestAnimationFrame(animate)
  c.drawImage(image, 0, 0)

  for (let i = enemies.length - 1; i >= 0; i--) {
    const enemy = enemies[i]
    enemy.update()

    if (enemy.position.x > canvas.width) {
      hearts -= 1
      enemies.splice(i, 1)
      document.querySelector('#hearts').innerHTML = hearts

      if (hearts === 0) {
        console.log('game over')
        cancelAnimationFrame(animationId)
        document.querySelector('#gameOver').style.display='flex'
      }
    }
  }
```

```
for (let i = explosions.length - 1; i >= 0; i--) {
  const explosion = explosions[i]
  explosion.draw()
  explosion.update()

  if (explosion.frames.current >= explosion.frames.max - 1) {
    explosions.splice(i, 1)
  }

  console.log(explosions)
}
```

Відстеження загальної кількості ворогів: тобто кожне нове появлення ворогів буде з більшою кількістю, ніж минулого разу.

```
if (enemies.length === 0) {
  enemyCount += 2
  spawnEnemies(enemyCount)
}

placementTiles.forEach((tile) => {
  tile.update(mouse)
})
```

Задача гравця – розставити вежі так, щоб було максимальне влучання у гоблінів. Коли снаряд сторожової вежі влучає у ворога, перераховується його життєва сила:

```
buildings.forEach((building) => {
  building.update()
  building.target = null
  const validEnemies = enemies.filter((enemy) => {
    const xDifference = enemy.center.x - building.center.x
    const yDifference = enemy.center.y - building.center.y
    const distance = Math.hypot(xDifference, yDifference)
    return distance < enemy.radius + building.radius
  })
  building.target = validEnemies[0]

  for (let i = building.projectiles.length - 1; i >= 0; i--) {
    const projectile = building.projectiles[i]

    projectile.update()
```

```
      const xDifference = projectile.enemy.center.x -
projectile.position.x
      const yDifference = projectile.enemy.center.y -
projectile.position.y
      const distance = Math.hypot(xDifference, yDifference)
   if (distance < projectile.enemy.radius + projectile.radius) {
```

Здоров'я ворога та видалення ворога:

```
    projectile.enemy.health -= 20
        if (projectile.enemy.health <= 0) {
          const enemyIndex = enemies.findIndex((enemy) => {
            return projectile.enemy === enemy
          })
```

За кожного видаленого ворога гравцю додається 25 монет, їх слід використовувати на покупку нової вежі для захисту дороги.

```
          if (enemyIndex > -1) {
            enemies.splice(enemyIndex, 1)
            coins += 25
            document.querySelector('#coins').innerHTML = coins
          }
        }

        console.log(projectile.enemy.health)
        explosions.push(
          new Sprite({
            position: { x: projectile.position.x, y:
projectile.position.y },
            imageSrc: './img/explosion.png',
            frames: { max: 4 },
            offset: { x: 0, y: 0 }
          })
        )
        building.projectiles.splice(i, 1)
      }
    }
  })
}
```

Підрахунок коштів при розміщенні нової вежі за обрану локацію:

```
canvas.addEventListener('click', (event) => {
  if (activeTile && !activeTile.isOccupied && coins - 50 >= 0) {
```

```
    coins -= 50

  document.querySelector('#coins').innerHTML = coins
    buildings.push(
      new Building({
        position: {
          x: activeTile.position.x,
          y: activeTile.position.y
        }
      })
    )


activeTile.isOccupied = true
    buildings.sort((a, b) => {
      return a.position.y - b.position.y
    })
```

Вибір локації і розміщення нової вежі:

```
window.addEventListener('mousemove', (event) => {
  mouse.x = event.clientX
  mouse.y = event.clientY

  activeTile = null
  for (let i = 0; i < placementTiles.length; i++) {
    const tile = placementTiles[i]
    if (
      mouse.x > tile.position.x &&
      mouse.x < tile.position.x + tile.size &&
      mouse.y > tile.position.y &&
      mouse.y < tile.position.y + tile.size
    ) {
      activeTile = tile
      break
    }
  }
})
```

У JavaScript клас є різновидом функції. Ключове слово «extends» використовується в оголошеннях класів або виразах класів для створення класу, який є нащадком іншого класу. Діаграму класів проекту представлено на рисунку 18:

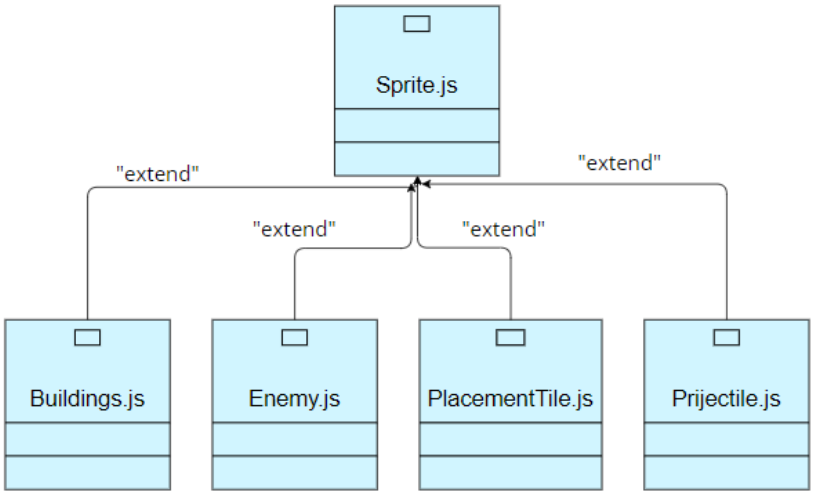

Рисунок 18 – Класи проекту

Скрін сцени гри (рис. 19) відображає приклад розміщення сторожових веж на дозволених локаціях, наступ групи ворогів та атаку. У верхній частині екрану можна слідкувати за станом життєвих ресурсів гравця, а також кількість накопичувальних коштів.

Рисунок 19 – Скрін сцени гри

Задача гравця: мислити логічно і швидко реагувати на події у грі. Кожного разу кількість ворогів збільшується, вони йдуть рандомним темпом. Тож слід уважно стежити за напрямом вилучення снарядів у сторожових веж і обдумувати максимально ефективне розміщення об’єктів по карті.

На першому етапі гравець може придбати тільки дві вежі. На полі є фіксовані локації для розміщення об’єктів гравця. Далі він буде намагатись знищити ворогів та отримати за це винагороду. Це дозволить придбати нове обладнання.

На даний час у проекті реалізовані тільки один рівень гри, тобто одна карта.

Якщо кількість ворогів, які змогли пройти до кінця дороги перевищить покажчик життєвих ресурсів гравця – гру буде зупинене та висвітиться повідомлення «Game Over!» (рис. 20).

Рисунок 20 – Приклад завершення гри

У подальшому планується розвивати проект щодо ускладнення умов, а саме більша кількість ворогів, додаткові перешкоди для гравця. Для підвищення зацікавленості цільової аудиторії слід додати звукові ефекти та нові конструкції сторожових веж. І головне, проектування нових карт ігрового простору.

3.4 Тестування

Тестування ігор, важлива частина розробки ігор, – це процес тестування програмного забезпечення, який дозволяє розробникам виявляти, документувати та виправляти помилки програмного забезпечення, щоб запобігти їх перешкоджанню ігровому досвіду. Це допомагає контролювати якість гри, створюючи оптимальну продуктивність у каналах розподілу.

Основне значення тестування ігор включає:

Оцінка якості – кожен розробник ігор прагне надати клієнтам найкращий досвід користувача. Найпростіший спосіб досягти цього – найняти кваліфікованого тестувальника ігор. Тестер ефективно виявить лазівки та виявить ймовірні дефекти.

Рейтинг відеоігор – тестування гри може безпосередньо впливати на рейтинг відеоігри. Коли має бути запущена нова гра, тестування суттєво впливає на те, як її бачать клієнти. Знову ж таки, коли випробувачі працюють, вони зазвичай розглядають детальний перелік параметрів, перш ніж винести остаточний вердикт.

Визначте сфери вдосконалення – більшість відеоігор підлягають частому оновленню протягом свого існування на ринку. Більшість із них також планують стати серіалами та запустити продовження. Тестування ігор дозволяє розробникам визначити сфери, які необхідно вдосконалити, щоб ігри довго жили на ринку [13].

Серед найпоширеніших помилок гри:

Помилки інтерфейсу: вони виникають, коли ігрова графіка неправильно тлумачиться, коли елементи гри знаходяться в неправильному місці, а простір, відведений для тексту, не підходить належним чином.

Технічні помилки: вони стосуються ситуацій, коли гра не працює належним чином через нестабільний Інтернет або мережеві з'єднання.

Помилки продуктивності: такі випадки, як низька продуктивність і помітне зниження частоти кадрів у грі (FPS) на пристроях високого класу під час анімації атаки персонажа, є одними з помилок продуктивності.

Помилки сумісності: вони стосуються проблем, які виникають, коли ігри не функціонують належним чином на різних пристроях чи ОС, незважаючи на те, що вони створені для підтримки всіх.

Баланс і логічні помилки: ці помилки не дозволяють гравцям виконувати завдання.

Протестуємо гру на відповідність до концепт-документу (табл. 1)

Таблиця 1 – Тестування

Назва	Очікуваний результат	Реальний результат	
Запус гри з бунусами та рівнем життевих сил за замовчув.	Кількість бонусів= 50 Кількість життів= 10	Кількість бонусів= 50 Кількість життів= 10	+
Напрям і кількість появи ворогів	Вороги з'являються зліва, 3 гобліна	Вороги з'являються зліва, 3 гобліна	+
Збільшення кількості ворогів	При наступній генерації кількість ворогів збільшується на 2 одиниці	При першій генерації – 3 ворога, При другій – 5	+

Продовження таблиці 1

Зменшення «health bar»	При влучанні у ворога його покажчик зменшується	При влучанні у ворога його покажчик зменшується	+
Зменшення покажчика життя гравця	При проходженні одного ворога, забирається 1 життя	При проходженні одного ворога, забирається 1 життя	+
Запуск зброї веж	Анімація динаміки вежі і запуск снаряду	Анімація динаміки вежі і запуск снаряду	+
Локація для вежі	Розміщення тільки у поміченій на карті локації	Розміщення тільки у поміченій на карті локації	+
Нова вежа	Можливість розмістити нову вежу тільки при наявності 50 монет	Можливість розмістити нову вежу тільки при наявності 50 монет	+
Кінець гри	Кількість життів меньша за кількість вижившних ворогів	Кількість життів меньша за кількість вижившних ворогів	+

Під час тестування були перевірені основні механіки алгоритму гри. Все реалізовано коректо. Але, виявлено один недолік, якій не був передбачений під час складання вимог до проекту: у випадку, коли існуючі вежі справляються з кількістю ворогів і відпадає необхідність купувати та встановлювати нову вежу, гра продовжує свій процес. Тобто, не передбачено вихід з ігрового циклу у разі досягнення мети для рівня.

Для ліквідації такого багу слід у подальшому розвитку проекту додати функцію перевірки на «оптимальну умову» та при наявності, наприклад, достатнього рівня бонусів, відкривати для гравця новий левел гри.

Крім того, в грі відсутній покажчик часу, але його можна додати, тим самим з’явиться можливість вести статистику щодо успіхів гравця по проходженню кожного левела.

ВИСНОВКИ

JavaScript – це мова, яку легко підібрати та вивчити шляхом практичних експериментів і великої кількості доступних ресурсів. У сукупності ці активи розробки JavaScript роблять його потужним інструментом для розгортання для створення виняткових ігор. Такі проекти допомагають опонувати основні функції і можливості мови на прикладі маленької гри.

В якості об’єкту розробки було обрано гру-платформер. Додаткові рущії не використовувались, тільки HTML, Canvas та JS. На першому кроці роботи над проектом сформовано концепт-документ, за вимогами якого вже проектувався ігровий простір та механіки.

В якості графічного контенту були взяти спрайт-листи, які є у вільному доступі у мережі. Через JS вдалося реалізувати анімацію спрайтів. В роботі наведено опис всіх основних функцій коду, а також структура всього проекту.

Крім того, виконано тестування щодо основних механік та правил гри. За результатами виявлено один недолік, якій не був врахований на початку роботи над проектом. В дипломі наведено опис подальшого розвитку проекту для отримання більш цікавого та придатного для реалізації серед цільової аудиторії програмного продукту.

СПИСОК ВИКОРИСТОВАНИХ ДЖЕРЕЛ

1. Ultimate List of Best Javascript Games for Kids in 2022. URL: https://brightchamps.com/blog/javascript-games/ (дата звернення 23.04.2023)
2. The ultimate guide to 2D game genres. URL: https://gamemaker.io/ blog/2d-game-genres (дата звернення 23.04.2023)
3. Ultimate Guide to JavaScript Game Development: Best JavaScript Games and How to Code Your Own. URL: https://www.codewizardshq.com/javascript-games/ (дата звернення 27.04.2023)
4. Tutorialspoint. URL: https://www.tutorialspoint.com/javascript/index.htm (дата звернення 28.04.2023)
5. 10 Best JavaScript Frameworks to Use in 2023. URL: https://hackr.io/blog/best-javascript-frameworks (дата звернення 28.04.2023)
6. Top JavaScript Frameworks and Technology 2023. URL: https://medium.com/javascript-scene/top-javascript-frameworks-and-technology-2023-4e4a06d6be93 (дата звернення 30.04.2023)
7. The top 10 JavaScript games you can play right now. URL: https://www.galvanize.com/blog/the-top-10-javascript-games-you-can-play-right-now/ (дата звернення 30.04.2023)
8. JavaScript. URL: https://brightchamps.com/blog-вк/javascript-games/(дата звернення 05.05.2023)
9. The Jump Guy Game in JavaScript Free Source Code. URL: https://www.sourcecodester.com/javascript/15499/jump-guy-game-javascript-free-source-code.html (дата звернення 05.05.2023)
10. Introduction About Tiled. Tutorial for using. URL: https://doc.mapeditor.org/en/stable/manual/introduction/ (дата звернення 05.05.2023)

11. Popular free sprites for games. URL: https://opengameart.org/art-search-advanced?keys=&field_art_type_tid%5B%5D=9&sort_by=count&sort_order=DESC (дата звернення 21.05.2023)
12. Skills for JavaScript game development. URL: https://www.codebrainer.com/blog/skills-for-javascript-game-development (дата звернення 27.05.2023)
13. A Complete Guide to Game Testing – Its Types and Processes. URL: https://www.headspin.io/blog/game-testing-a-complete-guide-to-its-types-and-processes (дата звернення 27.05.2023)