**Пояснювальна записка  
до курсової роботи**

на тему: Розробка гри в жанрі Tower Defense з штучним інтелектом

КПІ.ІП-0115.045490.02.81

Київ – 2022

Зміст

[Перелік умовних позначень 4](#_Toc115992741)

[Вступ 5](#_Toc115992742)

[1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6](#_Toc115992743)

[1.1 Загальні положення 6](#_Toc115992744)

[1.2 Змістовний опис і аналіз предметної області 6](#_Toc115992745)

[1.3 Аналіз існуючих технологій та успішних IT-проєктів 6](#_Toc115992746)

[1.3.1 Аналіз відомих алгоритмічних та технічних рішень 6](#_Toc115992747)

[1.3.2 Аналіз допоміжних програмних засобів та засобів розробки 7](#_Toc115992748)

[1.3.3 Аналіз відомих програмних продуктів 7](#_Toc115992749)

[1.4 Аналіз вимог до програмного забезпечення 7](#_Toc115992750)

[1.4.1 Розроблення функціональних вимог 8](#_Toc115992751)

[1.4.2 Розроблення нефункціональних вимог 9](#_Toc115992752)

[1.5 Постановка задачі 9](#_Toc115992753)

[Висновки до розділу 10](#_Toc115992754)

[2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 11](#_Toc115992755)

[2.1 Моделювання та аналіз програмного забезпечення 11](#_Toc115992756)

[2.2 Архітектура програмного забезпечення 11](#_Toc115992757)

[2.3 Конструювання програмного забезпечення 11](#_Toc115992758)

[2.4 Аналіз безпеки даних 13](#_Toc115992759)

[Висновки до розділу 13](#_Toc115992760)

[3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 14](#_Toc115992761)

[3.1 Аналіз якості ПЗ 14](#_Toc115992762)

[3.2 Опис процесів тестування 14](#_Toc115992763)

[3.3 Опис контрольного прикладу 15](#_Toc115992764)

[Висновки до розділу 15](#_Toc115992765)

[4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16](#_Toc115992766)

[4.1 Розгортання програмного забезпечення 16](#_Toc115992767)

[4.2 Підтримка програмного забезпечення 16](#_Toc115992768)

[Висновки до розділу 17](#_Toc115992769)

[ВИСНОВКИ 18](#_Toc115992770)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 19](#_Toc115992771)

Перелік умовних позначень

|  |  |  |
| --- | --- | --- |
| IDE | – | Integrated Development Environment – інтегроване середовище розробки. |
| API | – | Application programming interface, прикладний програмний Інтерфейс |
| SDK | – | Software development kit |
| IT | – | Інформаційні технології |
| ER | – | Entity-Relation diagram |
| OC | – | Операційна система. |
| БД | – | База даних. |

Вступ

В сучасному світі ігрова індустрія стрімко розвивається, впроваджуючи передові технології та концепції для створення захопливих та інноваційних геймплей-вражень для користувачів. Одним із популярних жанрів виділяється Tower Defense — стратегічна гра, в якій гравець має захищати визначену територію від хвиль ворогів, використовуючи обмежені ресурси для побудови веж. Однак зростання очікувань гравців поставило завдання перед розробниками створювати продукти, які не лише відповідають вимогам класичного жанру, але й надають нові можливості за допомогою штучного інтелекту (ШІ).

Жанр Tower Defense своєю історією налічує коріння в іграх стратегії, де основною метою було ефективно управляти обмеженими ресурсами для вибудовування веж та збереження бази від вторгнення ворожих сил. Перші представники цього жанру виникли на початку 2000-х років і швидко завоювали популярність серед геймерської аудиторії.[1]

Одним із перших визначних представників став продукт "Rampart", випущений у 1990 році. Ця аркадна стратегія поєднувала елементи будівництва оборонних стін і захисту території від нападів ворогів. Хоча вона не входила в жанр Tower Defense у сучасному розумінні, її концепції взяли на озброєння подальшого розвитку цього жанру.[2]

З появою персональних комп'ютерів і мобільних платформ, Tower Defense став популярним серед гравців, які шукали простий, але захоплюючий геймплей. Гра "Desktop Tower Defense", випущена в 2007 році, стала еталоном для багатьох наступних представників жанру.[3]

Разом із стрімким розвитком індустрії, особливо завдяки впровадженню передових технологій та концепцій, відзначається зростання очікувань гравців. Це ставить завдання перед розробниками не лише відповідати стандартам класичного жанру Tower Defense, але й надавати нові можливості через застосування штучного інтелекту (ШІ).

Актуальність проведення даної роботи визначається саме цією потребою вдосконалення ігор за допомогою ШІ. Впровадження штучного інтелекту в гру Tower Defense може створити більш інтелектуальних та адаптивних ворогів, розширити можливості стратегічного взаємодії гравця та розкрити нові перспективи в індустрії розваг та навчання.

Метою курсової роботи є створення ігрового програмного забезпечення в жанрі Tower Defense з штучним інтелектом для генерації хвиль ворогів в залежності від розміщення гравцем башт на ігровій мапі та можливістю автоматичної та мануальної генерації ігрової мапи.

# АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Загальні положення

Відеогра - це електронна гра, яка передбачає взаємодію з користувацьким інтерфейсом або пристроєм введення (наприклад, джойстиком, контролером, клавіатурою або пристроєм, що реагує на рух) для отримання візуального зворотного зв'язку від пристрою відображення, який найчастіше відображається у відеоформаті на телевізорі, комп'ютерному моніторі, пласкому дисплеї або сенсорному екрані на портативних пристроях, або гарнітурі віртуальної реальності.[4]

Однокористувацька відеогра - це відеогра, в якій протягом ігрового сеансу очікується участь лише одного гравця. Однокористувацька гра - це зазвичай гра, в яку може грати лише одна людина, тоді як "однокористувацький режим" - це зазвичай режим гри, призначений для гри одного гравця, хоча гра також містить багатокористувацькі режими.[5]

Більшість сучасних консольних ігор та аркадних ігор розроблені так, що в них може грати один гравець; хоча багато з цих ігор мають режими, які дозволяють грати двом або більше гравцям, дуже мало з них насправді вимагають більше одного гравця для гри. Серія Unreal Tournament є одним з таких прикладів.[5]

Одним із найпоширеніших видів архітектури для розробки відеоігор є компонентно-орієнтоване програмування. Компонентно-орієнтована програмна інженерія - це стиль програмної інженерії, метою якого є створення програмного забезпечення з слабко пов'язаних між собою модульних компонентів. Він наголошує на розділенні завдань між різними частинами програмної системи.[6]

Перевагами компонентно-орієнтованого програмування є:

* Швидкість розробки. Створюючи компоненти з багаторазових бібліотек, доступних у будь-який час, командам не потрібно починати роботу над своїм програмним забезпеченням з нуля, вони можуть безпосередньо вибирати з цієї бібліотеки, не турбуючись про нефункціональні вимоги, такі як безпека, зручність використання або продуктивність.[7]
* Легкість обслуговування. Кожен компонент є незалежним і може бути використаний повторно, це допомагає розкласти моноліт фронтенду на менші та керовані компоненти, що значно полегшує будь-які оновлення або модифікації. Замість того, щоб щоразу змінювати код, потрібно лише один раз оновити відповідні компоненти.[7]
* Можливість розробки незалежними командами. Компоненти є самодостатніми, але не впливають на систему. Це призводить до формування автономних команд, оскільки вони мають більше свободи, гнучкості та підзвітності, щоб вирішувати, як підтримувати безперебійну роботу над своїми проектами.[7]
* Краще перевикористання. Багаторазове використання має багато переваг, включаючи написання меншої кількості коду для бізнес-додатків. Працюючи з фреймворком на основі компонентів, розробникам не потрібно повторно реєструвати одні й ті ж рядки коду, і вони можуть зосередитися на основній функціональності.[7]
* Покращена масштабованість. Якщо продукт новий і люди реєструються, система, швидше за все, повинна бути готовою до зростання. Компонентна розробка дозволяє спеціально створеним елементам працювати разом, як шматочки пазла.[7]

Походження компонентного підходу не зовсім зрозуміле, оскільки різні розробники ігор могли використовувати подібні ідеї незалежно один від одного. Однак є кілька ранніх прикладів ігор, які використовували компонентний підхід:

* Rampart (1990). Вона використовувала компонентну систему для представлення ігрових об'єктів, таких як стіни, гармати та кораблі. Кожен об'єкт мав набір компонентів, які визначали його властивості та поведінку, а логіку гри реалізовували системи, що оперували компонентами;[2]
* Dungeon Siege (2002). Рольова гра, яка використовувала компонентний рушій Dungeon Siege Engine, що дозволяв створювати складні та різноманітні ігрові об'єкти та оточення. Кожен ігровий об'єкт складався з компонентів, які можна було додавати, видаляти або модифікувати під час виконання, а логіка гри визначалася даними, а не кодом.[8][9]

Tower Defense - це жанр стратегічних відеоігор, в яких гравець повинен будувати і вдосконалювати вежі, що захищають базу від хвиль ворогів, які намагаються прорватися. Основними компонентами гри Tower Defense є:

* Вежі: Це оборонні споруди, які мають різні типи, вартість, характеристики та ефекти. Вежі можуть атакувати, зупиняти, послаблювати, посилювати або взаємодіяти з ворогами чи іншими вежами. [1][10]
* Вороги: Це атакуючі об'єкти, які мають різні типи, швидкість, силу, здоров'я та здібності. Вороги можуть рухатися за певним маршрутом, літати, бити, ухилятися, зцілюватися або взаємодіяти з іншими ворогами чи вежами. [1][10]
* База: Цільовий об'єкт, який гравець повинен захищати від ворогів. База має певну кількість здоров'я, яке зменшується, коли ворог досягає її. Якщо здоров'я бази досягає нуля, гравець програє гру. [1][10]
* Гроші: Це ресурс, який гравець отримує за знищення ворогів або іншим чином. Гроші можна використовувати для будівництва, модернізації, ремонту або продажу веж. [1][10]
* Хвилі: Послідовні групи ворогів, які з'являються з одного або декількох напрямків і рухаються до бази. З кожною хвилею ворогів стає більше і вони стають сильнішими.[1][10]

## Змістовний опис і аналіз предметної області

Жанр Tower Defense є стратегічною грою, в якій гравець має захищати визначену територію від вторгнення ворожих сил. Головна мета полягає в розташуванні захисних веж та перешкод для ефективного стримування хвиль ворогів. Гра поділяється на різні етапи, включаючи будівництво та апгрейд веж, ремонт та фазу захисту.

На сучасний момент існує багато різновидів гри Tower Defense. Вони можуть відрізнятися графікою, типами ворогів, системами взаємодії та типами ігрових карт. Деякі варіації дозволяють встановлювати барикади, інші мають значно більш розвинуту економічну систему, коли гравець отримує гроші не тільки за смерть ворогів, а й за внутрішньоігрові досягнення або ж існують башти, що приносять гравцеві валюту. Також існують варіації з грою від першого обличчя, наявністю сюжету.

Серед основних існуючих недоліків можна відзначити передбачуваність рівнів і хвиль ворогів. Повторюючи той самий рівень, гравець може спостерігати однотипність карт та ворогів, що може викликати втому та зниження інтересу до продукту.

Для подолання недоліків жанру та зроблення гри більш динамічною та непередбачуваною, пропонується використання штучного інтелекту (ШІ) для генерації шляхів для ворогів та створенню хвиль на основі розміщених на ігровому полі веж гравця.

Для збільшення різноманітності гри та підвищення рівня несподіваності буде реалізований алгоритм автоматичного генерування мапи. Цей алгоритм буде базуватися на методах бектрекінгу, що дозволить створювати під час кожної нової гри унікальні та цікаві ігрові обстановки.

Застосування ШІ в грі Tower Defense призводить до генерації різноманітних хвиль ворогів та вдосконалення їхньої взаємодії з гравцем. Це може зробити гру більш захопливою та викликаючою для користувача, оскільки хвиля ворогів буде підлаштовуватися під стиль гри гравця.

## Аналіз існуючих технологій та успішних IT-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації однокористувацької гри в жанрі Tower Defense зі штучним інтелектом “Neural Siege”. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

### Аналіз відомих алгоритмічних та технічних рішень

У випадку однокористувацької гри, де весь ігровий процес обмежений взаємодією лише з одним гравцем, використання клієнт-серверної архітектури може виявитися недоцільним. Така архітектура призначена для випадків, коли гра має багато гравців, які взаємодіють один з одним у режимі реального часу.[13]

У випадку однокористувацької гри, де важлива лише індивідуальна взаємодія гравця з грою та обчислення локальних аспектів, введення клієнт-серверної архітектури може призвести до зайвих складнощів та затрат на обмін даними між клієнтом та сервером, через що дану архітектуру недоцільно застосовувати в контексті даної роботи.[13]

Мікросервісна архітектура також не підійде у даному випадку, оскільки передбачає розділення функціональності програмного продукту на невеликі, автономні мікросервіси, що працюють над конкретною функцією чи завданням. Однак, враховуючи особливості гри Tower Defense та концепції її розробки, мікросервісна архітектура може виявитися неефективною. Основними проблемами мікросервісної архітектури є:[11][12]

* Складність управління. Розбиття ігрового процесу на окремі мікросервіси може призвести до ускладнення управління та координації між ними;
* Вплив на продуктивність. Збільшення кількості мікросервісів може вплинути на продуктивність гри через збільшення обміну даними між ними. У випадку, коли обчислення та взаємодія відбуваються локально на пристрої гравця, вибір мікросервісної архітектури може стати надто складним та непрактичним;
* Ускладнення розробки та тестування. Вибір більш монолітного підходу до розробки та тестування може спростити взаємодію між компонентами гри, забезпечуючи більш зручний та швидкий процес розробки та вдосконалення;

Відкинувши клієнт-серверну та мікросервісну архітектури можемо дійти висновку що в даному випадку найкращим вибором архітектури буде монолітна архітектура, оскільки вона проста та зручна в розробці, більш продуктивна та має менші затримки в обміні даним.[12]

У контексті даної роботи доцільним буде використання компонентного підходу. Цей підхід передбачає поділ функціональності гри на невеликі автономні компоненти, які можна розробляти і тестувати незалежно, а потім легко інтегрувати в основний продукт. [6]

Даний підхід дає модульність і розширюваність, оскільки дозволяє розділити функціонал гри на окремі модулі, що полегшує її розширення та модифікацію. Наприклад, окремі компоненти можуть відповідати за генерацію мапи, розміщення ворогів та стратегію ШІ.

Також великою перевагою використання компонентного підходу є незалежність та ізоляція помилок, простота тестування: кожен компонент може працювати незалежно, що зменшує взаємодію між ними і полегшує виявлення та виправлення помилок.

У випадку розробки гри, використання шаблону Finite State Machine (FSM) є дуже корисним для контролю загального стану гри. FSM можна використовувати для моделювання різних станів гри, таких як "старт", "ігровий процес", "поразка".[14]

Використання FSM для керування станом гри забезпечує чітку структуру для визначення можливих переходів між різними фазами гри. Наприклад, вона вказує, як гра переходить від стадії "старт" до "ігрового процесу", а потім до "програшу". Це забезпечує систематичний і легкий контроль над відображенням і поведінкою гри в різних сценаріях.[14]

Простота розширення та зміни ігрових станів за допомогою FSM робить архітектуру гнучкою та пристосованою до змін в ігровому процесі. Додавання нових фаз або зміна послідовності переходів може бути зроблено ефективно без значних змін у загальному коді гри.

Використання патерну Observer також є доцільним стратегічним вибором. Цей патерн дозволяє компонентам гри ефективно реагувати на зміни в стані інших об'єктів, не прив'язуючись до конкретних класів.[16]

Патерн Observer забезпечує систему сповіщень, де спостерігачі можуть реагувати на події та зміни в стані інших об'єктів. У випадку даної гри це означає можливість ефективного взаємодії різних її елементів. Observer полегшує додавання нових спостерігачів та зміну їх поведінки без необхідності впливу на інші частини системи. [16]

Крім того, патерн Observer вже реалізований у мові програмування C#, що робить його зручним і легко використовуваним в даній роботі.[25]

Ін'єкція залежностей (Dependency Injection, DI) - це розумний і корисний аспект розробки гри Tower Defense з використанням штучного інтелекту. DI полегшує управління залежностями в коді, дозволяючи відокремити створення об'єктів та їхніх залежностей від класів, які використовують ці об'єкти.[15]

Такий підхід зменшує кількість жорстких залежностей у коді, роблячи його більш гнучким і легшим для змін. Використання DI допомагає підвищити тестованість коду. Інтерфейси залежностей дозволяють проводити ефективне модульне тестування окремих частин системи, що робить розробку стабільною та надійною.

DI також полегшує передачу залежностей через конструктори або властивості класів, обмежуючи залежності в класах. Це важливо для створення чистого та організованого коду, особливо коли важливо уникати прямих залежностей.

Для автоматичної генерації шляху для ворогів в даній роботі використовується алгоритм бектрекінгу. [17] Попри те, що алгоритм бектрекінгу не дозволяє знайти оптимальний шлях у кожному випадку, він чудово підійде для генерації випадкових лабіринтів у грі жанру Tower Defense. Його особливість полягає в тому, що він створює різноманітні та унікальні карти, що призводить до постійної динаміки та виклику для гравців під час кожної гри. Такий підхід забезпечує необхідний рівень випадковості, роблячи ігровий процес більш різноманітним.

Проблему генерації хвиль можна розглядати як задачу про рюкзаки[19], де потрібно ефективно розподілити обмежені ресурси (ворожі поінти) між різними хвилями, щоб створити складний та інтенсивний ігровий процес.

Для генерації хвиль у даній роботі використовується генетичний алгоритм, оскільки цей алгоритм може ефективно моделювати та оптимізувати характеристики ворожих хвиль. Алгоритм є метаевристичним та простим в реалізації та модифікації, що робить його хорошим кандидатом для поставленої задачі.[18]

Генетичний алгоритм може адаптуватися до мінливих умов і забезпечувати безперервне вдосконалення ворожих хвиль, що робить ігровий процес більш динамічним і захоплюючим для гравців. Такий підхід дозволяє досягти балансу між викликом і задоволенням від гри, створюючи позитивний ігровий досвід.

### Аналіз допоміжних програмних засобів та засобів розробки

На момент написання роботи існує велика кількість ігрових рушіїв, проте більшість з них є закритими та переважно використовуються всередині компанії, що їх створила. Незважаючи на це, існують два основних рушія, доступні для загалу: Unity, Unreal Engine.[20][21]

Unity – один із найпопулярніших ігрових рушіїв на ринку, який дозволяє створювати ігри та інтерактивні візуалізації у 2D та 3D. Використовується та підтримується на великій кількості платформ.[20]

Основні переваги Unity:

* Простота у використанні та вивченні
* Підтримка понад 25 платформ, включаючи iOS, Android, Windows, Mac, Linux, PS4, Xbox One, Switch та інші
* Безкоштовна версія для невеликих проектів
* Велика спільнота розробників і безліч ресурсів для навчання
* Магазин ресурсів, куди можна імпортувати готові моделі, текстури, скрипти та інші компоненти
* Система анімації, що дозволяє створювати природні та динамічні рухи
* Система частинок, що дозволяє імітувати такі ефекти, як вогонь, дим, вода та інші [22]
* Система інтерфейсів, що дозволяє створювати інтерфейси для гравців [23]

Основними недоліками Unity є:

* Низька якість графіки в порівнянні з Unreal Engine
* Порівняно низька продуктивність для важких 3D ігор



Рисунок 1.1 – Інтерфейс в Unity

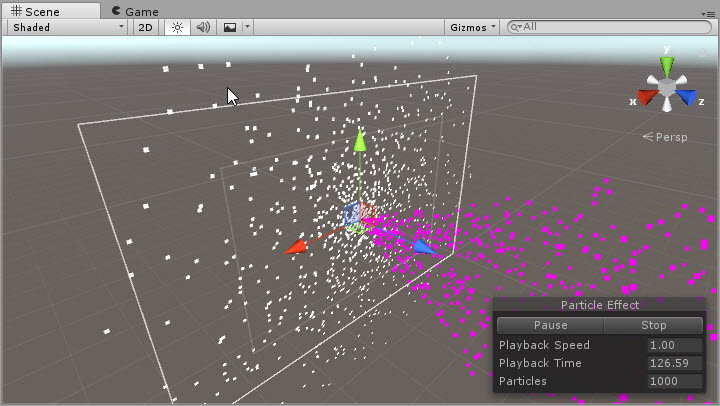


Рисунок 1.2 – Система частинок в Unity

Приклади ігор, написаних на Unity:

* Cities: Skylines 1, 2
* Marvel Snap
* Among Us



Рисунок 1.3 – Приклад гри на Unity. Cities Skylines

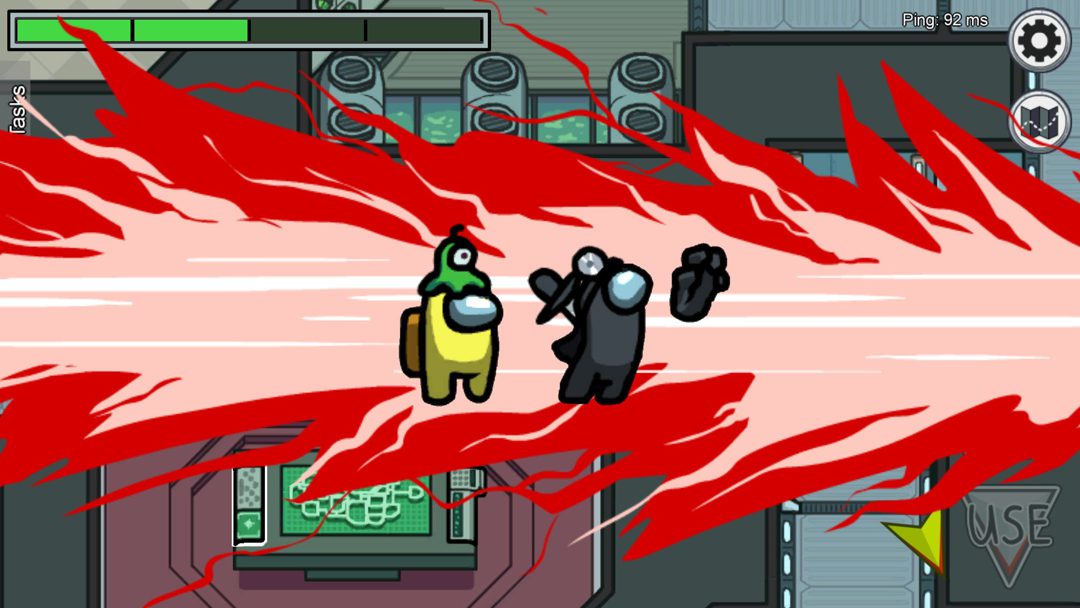


Рисунок 1.4. – Приклад гри на Unity. Among Us



Рисунок 1.5. – Приклад гри на Unity. Marvel Snap

Unreal Engine - це потужний ігровий рушій, розроблений компанією Epic Games. Unreal Engine дозволяє створювати ігри та інтерактивні можливості з високою якістю графіки та реалістичністю. Через свою високу якість графіки, даний рушій широко використовується в сфері кінематографу та AAA ігор. Також компанія має гру Fortnite, що одразу наочно показує можливості рушія Unreal Engine.[21]

Основні переваги Unreal Engine:

* Висока якість графіки та рендерингу, що перевершує більшість інших ігрових рушіїв
* Гнучкість та адаптивність, що дозволяє розробникам налаштовувати ігровий процес та ресурси за допомогою візуальних скриптів Blueprint або коду на C++ [24]
* Широкий спектр функцій та інструментів, які включають фізичний рушій, систему анімації, систему частинок, систему користувацького інтерфейсу, підтримку VR та AR та багато іншого
* Безкоштовна версія для некомерційних проектів та освітніх цілей
* Велика спільнота розробників та багато ресурсів для навчання

Основні недоліки Unreal Engine:

* Висока складність і крута крива навчання, що вимагає від розробників глибоких знань і досвіду
* Високі вимоги до ресурсів, які можуть обмежувати продуктивність та сумісність з деякими платформами
* Нестабільність та помилки, які можуть виникати через невирішені конфлікти або несумісність плагінів

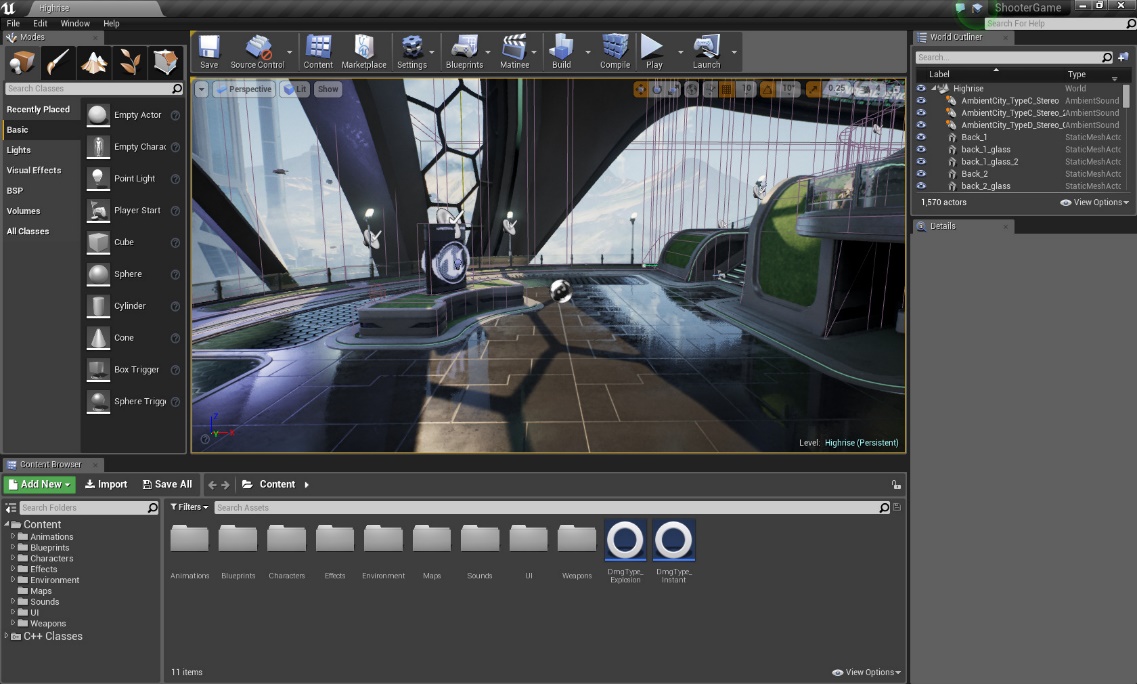


Рисунок 1.6. – Інтерфейс в Unreal Engine

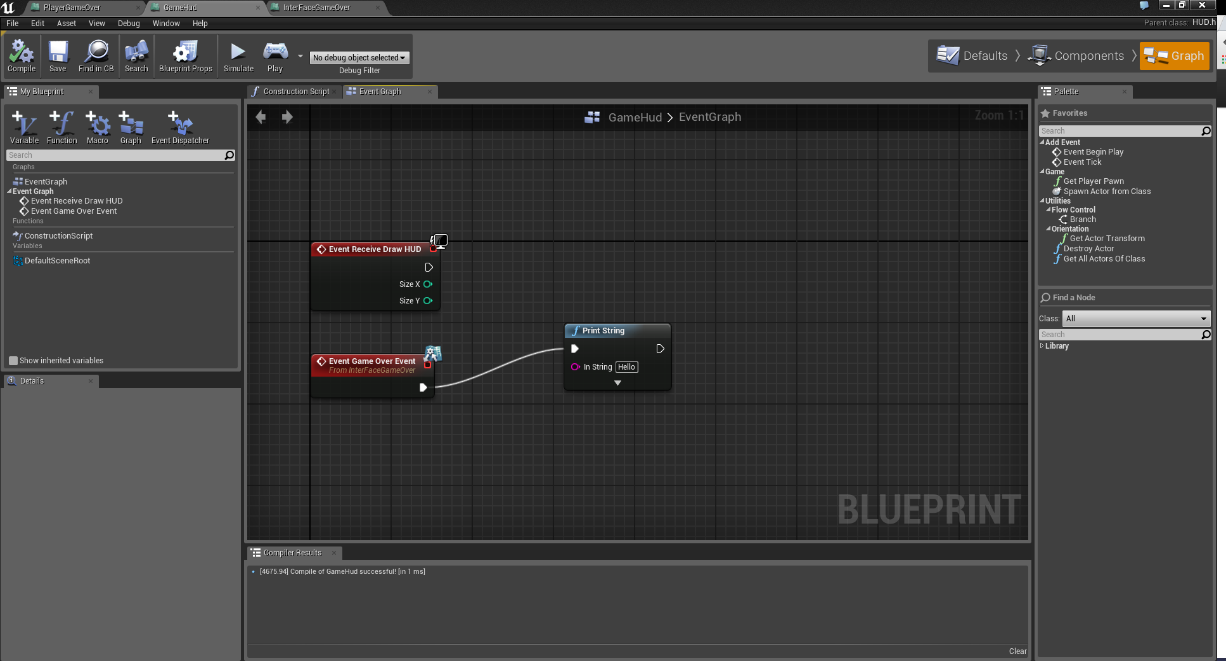


Рисунок 1.7. – Blueprints в Unreal Engine

Приклади ігор, написаних на Unity:

* Fortnite
* Sea of Thieves
* Aliens: Fireteam Elite



Рисунок 1.8. – Приклад гри на Unreal Engine. Fortnite



Рисунок 1.9. – Приклад гри на Unreal Engine. Sea of Thieves



Рисунок 1.10. – Приклад гри на Unreal Engine. Aliens: Fireteam Elite

При виборі між Unity та Unreal Engine для розробки даної гри був обраний рушій Unity, оскільки він відзначається своєю простотою вивчення та користуванням, використовує мову C#, яка є популярною та має велику спільноту розробників. Також Unity має активну спільноту та простоту експорту гри на різні платформи (наприклад мобільні телефони).

При виборі IDE для Unity було порівняно Visual Studio та Rider як інтегровані середовища розробки для мови програмування C#. Для обох засобів можна виділити ряд характеристик.[26][27]

Visual Studio, завдяки своїй великій популярності та розширеним функціоналом, є широко використовуваним серед розробників. Велика кількість розширень та плагінів дозволяє розширити можливості інструменту. Проте, деяким розробникам може здатися важким налаштовування для оптимальної роботи з Unity.

У свою чергу, Rider спеціально адаптований для роботи з проектами Unity та мовою програмування C#. Це середовище надає вбудовану підтримку Unity [28] та можливість редагувати код прямо в редакторі Unity. Це забезпечує оптимальну взаємодію з платформою та високу продуктивність.

Обираючи Rider, розробник отримує інструмент, який спрощує та оптимізує роботу з Unity. Його спеціалізація на Unity, вбудована підтримка та висока продуктивність роблять його ефективним вибором для розробки гри жанру Tower Defense. Саме тому для даної розробки був обраний Rider.

### Аналіз відомих програмних продуктів

На час написання роботи існує велика кількість ігор жанру Tower Defense, що мають різні модифікації, змішання стилів та різні цільові платформи. Для порівняння з курсовою роботою були обрані два популярних продукти цього жанру: Plants vs Zombies [29] та Bloons TD6 [30]. Обидві ці гри вважаються знаковими для свого жанру та індустрії в цілому.

Bloons TD6 - це гра в жанрі Tower Defense, розроблена компанією Ninja Kiwi для мобільних телефонів, проте згодом портована і на інші платформи, такі як PC, XBOX, PS. В грі гравець повинен захищати свою територію від нападу повітряних кульок під назвою Bloons. Гравець може будувати різні види веж, які мають різні здібності, щоб знищити кульки. У грі є багато режимів, карт, подій та іншого контенту, що робить гру цікавою та різноманітною.

Основними характеристиками гри Bloons TD6 є велика кількість різноманітних веж(23), кожна з яких має покращення. Покращення для башт гравець може придбати за внутрішньо-ігрову валюту або ж отримати їх через певний час після встановлення вежі. Гра має велику кількість ворогів та кооперативний режим для 4 гравців. Кожен рівень в грі має наперед створену ігрову мапу та наперед створені хвилі ворогів, через що гравець точно знає чого чекати, заходячи на внутрішньо-ігровий рівень. Гра відзначається унікальним візуальним стилем, величезною кількістю ворогів в хвилях.



Рисунок 1.11. – Внутрішньо-ігровий інтерфейс гри Bloons TD6

Plants vs Zombies - це гра в жанрі Tower Defense, розроблена компанією PopCap Games для PC, Mac, iOS, Android та інших платформ. В грі гравець повинен захищати свій будинок від нападу зомбі, які хочуть з’їсти його мозок. Гравець може садити різні види рослин, які мають різні здібності, щоб зупинити зомбі. Гра має нестандартну для жанру Tower Defense лінійну мапу, проте все ще вважається представником цього жанру.

Основними характеристиками гри Plants vs Zombies є велика кількість росли-веж (49 рослин), кожна з яких має свою особливу здібність, 20+ видів ворогів з різними особливостями, систему досягень, різні режими гри, такі як Виживання, Город, Міні-ігри, Головоломки та інші. Всі мапи в грі мають лінійний характер, а всі хвилі ворогів наперед створені для кожного рівня. Гра відзначається простотою та інтуітивністю геймплею та великою кількістю режимів.



Рисунок 1.12. – Внутрішньо-ігровий інтерфейс гри Plants vs Zombies

Для порівняння курсової роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Функціонал | Дипломний проєкт (Neural Siege) | Аналог для порівняння 1 Bloons TD6 | Аналог для порівняння 2 Plants vs Zombies | Пояснення |
| Випадково створена мапа | + | - | - | Мапа генерується випадково для кожного запуску гри |
| Можливість створити власну мапу | + | - | - | Гравець може створити власну ігрову мапу |
| Автоматична генерація хвиль | + | - | - | Склад хвиль генерується під час гри на основі певних факторів |
| Наявність рівнів | - | + | + | Можливість обирати наперед визначений рівень |
| Покращення веж | + | + | - | Можливість покращувати характеристики веж |
| Продаж веж | + | + | + | Можливість видаляти вежі з ігрового поля |

Продовження таблиці 1.1. – Порівняння з аналогами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наявність кооперативу | - | + | - | Можливість грати по кооперативу |
| Наявність інших режимів | - | + | + | Наявність ігрових режимів, що модифікують стандартний |

## Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є генерація хвиль ворогів в ході ігрового процесу, автоматичне випадкове створення ігрової мапи, більше функцій можна побачити на рисунку 1.13.

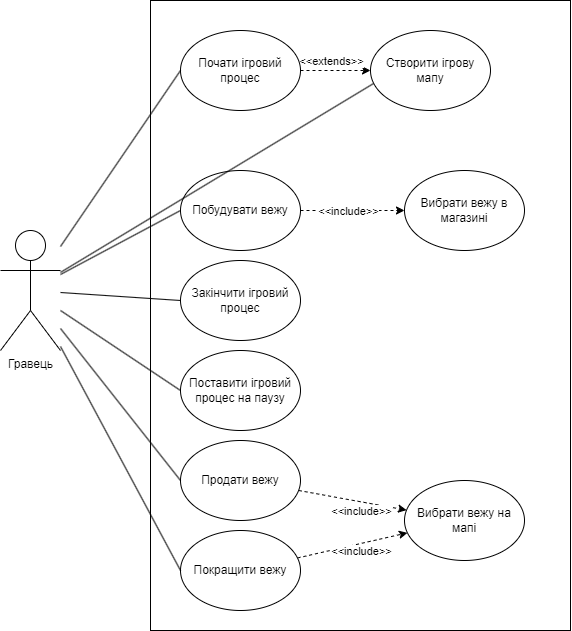


Рисунок 1.13 – Діаграма варіантів використання

В таблицях 1.2 - 1.10 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-01

|  |  |
| --- | --- |
| Use case name | Почати ігровий процес |
| Use case ID | UC-01 |
| Goals | Почати ігровий процес |
| Actors | Гравець |
| Trigger | Гравець хоче почати ігровий процес |
| Pre-conditions | - |
| Flow of Events | 1. Гравець запускає гру.  2. Гравець обирає опцію "Start Game".  3. Система генерує ігрову мапу.  4. Основне вікно гри відкривається з згенерованою мапою. |
| Extension | - |
| Post-Condition | Гравець знаходиться в основному вікні гри зі згенерованою мапою. |

Таблиця 1.3 - Варіант використання UC-02

|  |  |
| --- | --- |
| Use case name | Створити ігрову мапу |
| Use case ID | UC-02 |
| Goals | Почати ігровий процес з власно створеною ігровою мапою |
| Actors | Гравець |
| Trigger | Гравець хоче почати ігровий процес з власно створеною ігровою мапою |
| Pre-conditions | - |
| Flow of Events | 1. Гравець запускає гру.  2. Гравець обирає опцію "Create Map" у головному меню.  3. Система відкриває редактор мапи.  4. Гравець розміщує елементи мапи за своїм вибором.  5. Гравець підтверджує створення мапи. |

Продовження таблиці 1.3 - Варіант використання UC-02

|  |  |
| --- | --- |
| Extension | Якщо мапа побудована некоректно, гравець не зможе почати гру на ній. |
| Post-Condition | Гравець знаходиться в основному вікні гри з новоствореною гравцем мапою. |

Таблиця 1.4 - Варіант використання UC-03

|  |  |
| --- | --- |
| Use case name | Побудувати вежу |
| Use case ID | UC-03 |
| Goals | Побудувати вежу |
| Actors | Гравець |
| Trigger | Гравець хоче розмістити вежу на мапі |
| Pre-conditions | Гравець знаходиться в основному вікні гри зі згенерованою мапою, гравець обрав вежу в магазині |
| Flow of Events | 1. Гравець натискає по місцю на мапі, де він хоче розмістити вежу |
| Extension | Гравець може вибрати "Скасувати". |
| Post-Condition | Система розміщує вибрану вежу на мапі гри. |

Таблиця 1.5 - Варіант використання UC-04

|  |  |
| --- | --- |
| Use case name | Вибрати вежу в магазині |
| Use case ID | UC-04 |
| Goals | Вибрати вежу в магазині |
| Actors | Гравець |
| Trigger | Гравець хоче обрати тип вежі для будівництва |
| Pre-conditions | Гравець знаходиться в основному вікні гри зі згенерованою мапою. |
| Flow of Events | 1. Система відображає доступні типи веж для будівництва.  2. Гравець обирає конкретний тип вежі для розміщення, натискаючи на його іконку. |
| Extension | - |
| Post-Condition | Система встановлює вибрану вежу як активну для подальшого розміщення на мапі. |

Таблиця 1.6 - Варіант використання UC-05

|  |  |
| --- | --- |
| Use case name | Закінчити ігровий процес |
| Use case ID | UC-05 |
| Goals | Повернутися в головне меню |
| Actors | Гравець |
| Trigger | Гравець хоче завершити поточну гру та повернутися в головне меню |
| Pre-conditions | Гравець перебуває в основному вікні гри зі згенерованою мапою та поточним ігровим процесом, має бути увімкнена павза або екран закінчення гри |
| Flow of Events | 1. Гравець натискає кнопку переходу в головне меню  2. Закривається основне вікно гри, з’являється вікно головного меню |
| Extension | - |
| Post-Condition | Гравець повертається до головного меню гри. |

Таблиця 1.7 - Варіант використання UC-06

|  |  |
| --- | --- |
| Use case name | Поставити ігровий процес на паузу |
| Use case ID | UC-06 |
| Goals | Поставити ігровий процес на паузу |
| Actors | Гравець |
| Trigger | Гравець хоче призупинити поточну гру |
| Pre-conditions | Гравець перебуває в основному вікні гри зі згенерованою мапою та поточним ігровим процесом. |
| Flow of Events | 1. Гравець натискає «Esc» на клавіатурі  2. Система призупиняє гру та виводить відповідне повідомлення. |
| Extension | - |
| Post-Condition | Гра призупиняється, гравець перебуває у режимі паузи. |

Таблиця 1.8 - Варіант використання UC-07

|  |  |
| --- | --- |
| Use case name | Продати вежу |
| Use case ID | UC-07 |

Продовження таблиці 1.8 - Варіант використання UC-07

|  |  |
| --- | --- |
| Goals | Продати вежу |
| Actors | Гравець |
| Trigger | Гравець хоче продати встановлену вежу |
| Pre-conditions | Гравець перебуває в основному вікні гри зі згенерованою мапою та має вибрану вежу для продажу. |
| Flow of Events | 1. Гравець вибирає опцію "Sell Tower" у вікні вибраної вежі. |
| Extension | - |
| Post-Condition | Вибрана вежа зникає з мапи, гравцеві нараховуються гроші за башту |

Таблиця 1.9 - Варіант використання UC-08

|  |  |
| --- | --- |
| Use case name | Покращити вежу |
| Use case ID | UC-08 |
| Goals | Покращити вежу |
| Actors | Гравець |
| Trigger | Гравець хоче покращити встановлену вежу |
| Pre-conditions | Гравець перебуває в основному вікні гри зі згенерованою мапою та має вибрану вежу для покращення. |
| Flow of Events | 1. Гравець вибирає опцію "Upgrade Tower" у вікні вибраної вежі. |
| Extension | - |
| Post-Condition | Вибрана вежа має покращені характеристики, з ігрових коштів знімаються кошти для покращення башти |

Таблиця 1.10 - Варіант використання UC-09

|  |  |
| --- | --- |
| Use case name | Вибрати вежу на мапі |
| Use case ID | UC-09 |
| Goals | Вибрати вже побудовану межу на мапі |
| Actors | Гравець |
| Trigger | Гравець хоче вибрати вже встановлену вежу на мапі |
| Pre-conditions | Гравець перебуває в основному вікні гри зі згенерованою мапою |

Продовження таблиці 1.10 - Варіант використання UC-09

|  |  |
| --- | --- |
| Flow of Events | 1. Гравець клікає на вежу на ігровій мапі, яку хоче вибрати. |
| Extension | Гравець може вибрати "Скасувати". |
| Post-Condition | Система встановлює вибрану вежу як активну для подальших дій, гравець бачить вікно вибраної вежі, радіус атаки вежі |

### Розроблення функціональних вимог

<У пункті викладають:

модель вимог, їх опис та матрицю трасування.>

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 1.11 наведено загальну модель вимог, а в таблицях 1.18 – 1.27 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.5.

Таблиця 1.11 – Модель вимог у загальному вигляді

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Name | Priority | Risk |
| Розміщення веж на мапі | FR-1 | High | Medium |
| Валідація розміщення веж на мапі | FR-2 | Medium | Low |
| Перевірка ігрового балансу гравця | FR-3 | High | Medium |
| Продаж вежі | FR-4 | Medium | Low |
| Покращення вежі | FR-5 | Medium | Medium |
| Режим паузи | FR-6 | Medium | Low |
| Генерація мапи | FR-7 | High | Medium |
| Створення мапи власноруч | FR-8 | Medium | Low |
| Валідація мапи | FR-9 | Medium | Low |
| Генерація хвиль ворогів | FR-10 | High | Medium |
| Закінчення ігрового процесу | FR-11 | High | Low |

Рисунок 1.4 – Модель вимог у загальному вигляді

Таблиця 1.16 – Функціональна вимога FR-1

|  |  |
| --- | --- |
| Назва | Реєстрація користувача |
| Опис | Система повинна надавати можливість реєстрації користувачеві шляхом введення пошти, паролю, підтвердження паролю. |

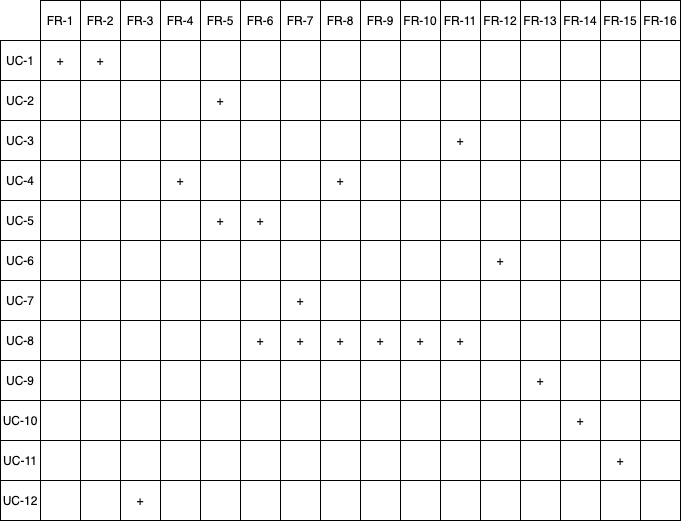


Рисунок 1.11 – Матриця трасування вимог

### Розроблення нефункціональних вимог

<У пункті викладають:

опис нефункціональних вимог.>

## Постановка задачі

<У підрозділі викладають:

що саме ви збираєтесь реалізувати. Це не зовсім вимоги і варіанти використання. Це достатньо узагальнене поняття з метою, цілями та задачами, що підлягають розв’язанню у результаті розробки програмного забезпечення на курсову роботу. Обсяг 1 сторінка, чи більше. >

## Висновки до розділу

< Необхідно стисло описати усе, що було виконано у даному розділі. Обсяг 0,75-1 сторінка>

# МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Моделювання та аналіз програмного забезпечення

<У підрозділі викладають:

опис бізнес процесів вашої розробки та їх наочне представлення за допомогою засобів моделювання (BPMN та інші). Обсяг 2 сторінки, чи більше. >

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1).

Опис послідовності створення облікового запису користувача:

* користувач переходить на сторінку реєстрації;
* користувач заповнює поля реєстрації;
* якщо введені поля, не відповідають шаблону заповнення на клієнтській стороні, відповідні поля підсвічуються помилкою;

## Архітектура програмного забезпечення

<У підрозділі викладають:

втілення архітектурного паттерну для вашого ПЗ та деталізація його компонентів за необхідності. Необхідно виконати повний опис архітектури (схеми, таблиці та ін.). Обсяг 2 сторінки, чи більше.>

## Конструювання програмного забезпечення

<У підрозділі викладають:

опис оригінальних алгоритмів чи модифікацій існуючих. Опис структур даних, програмних структур та ін. Опис бази даних з представленням концептуальної, логічної чи фізичної моделі та з описом сутностей чи таблиць. Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці. Аналіз системних вимог. Обсяг 4 сторінки, чи більше.>

Бібліотека xpring, яка виконує запити в мережу блокчейн – однопоточна, тобто не може оброблювати паралельні запити. Через таку специфіку бібліотеки, довелось розробляти рішення яке б не блокувало запити з різних потоків, або інших частин коду. За основу ідеї алгоритму була взята багатопоточність самої мови програмування java. Java для вирішення проблем з потоками представляє ділянки коду які можна синхронізувати, що унеможливлює одночасний доступ до нього з різних потоків. Ця синхронізація проходить за допомогою передачі управління об’єкта-монітора. Алгоритм вирішення проблеми з доступом до однопоточної бібліотеки наведено на рисунку 2.3.

В якості системи управління базами даних використовується Postgres. База даних серверу призначена для зберігання користувачів, а також даних про їх …. Опис таблиць бази даних наведено у таблицях 2.11 - 2.14. Модель бази даних наведена на рисунку 2.12.

Таблиця 2.11 – Опис таблиці user

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця | Назва поля | Тип даних | Опис |
| user | id | serial | ідентифікаційний номер користувача |
| email | varchar | електронна пошта користувача |
| password\_id | int | посилання на запис у таблиці password, де зберігається пароль користувача |

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.22.

Таблиця 2.22 – Опис утиліт

|  |  |  |
| --- | --- | --- |
| №  п/п | Назва утиліти | Опис застосування |
| 1 | IntelliJ IDEA | Головне середовище розробки програмного забезпечення серверної частини курсової роботи. |
| 2 | Postman | Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів. |
| 3 | MySQL Workbench | Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних. |

## Аналіз безпеки даних

<У підрозділі викладають:

аналіз вразливостей ПЗ та будь-які питання пов’язані з безпекою даних. >

## Висновки до розділу

< Необхідно стисло описати усе, що було виконано у даному розділі. Обсяг 0,75-1 сторінка>

# АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз якості ПЗ

<У підрозділі викладають:

аналіз якості ПЗ за певними метриками. Обсяг 2 сторінки, чи більше.>

## Опис процесів тестування

<У підрозділі викладають:

опис процесів тестування та приклади тестів. Обсяг 4 сторінки, чи більше.>

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.3 – 3.30.

Таблиця 3.3 – Тест 1.1

|  |  |
| --- | --- |
| Тест | Реєстрація користувача |
| Модуль | Реєстрація користувача |
| Номер тесту | 1.1 |
| Початковий стан системи | Користувач знаходиться на сторінці реєстрації |
| Вхідні данні | Електронна пошта, пароль, підтвердження паролю |
| Опис проведення тесту | У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 10 до 64 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ, і який не входить у топ 10000 найпопулярніших паролей, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього … |
| Очікуваний результат | Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації. |
| Фактичний результат | Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації. |

## Опис контрольного прикладу

<У підрозділі викладають:

повний опис контрольного прикладу з усіма можливими розгалуженнями та особливостями. Кроки доповнюють ілюстраціями. Не обов’язковий розділ.>

## Висновки до розділу

< Необхідно стисло описати усе, що було виконано у даному розділі. Обсяг 0,75-1 сторінка>

# ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Розгортання програмного забезпечення

<У підрозділі викладають:

повний опис покрокового розгортання ПЗ. Кроки доповнюють ілюстраціями. Обсяг 2 сторінки, чи більше.>

Клієнтську і серверну частини програмного забезпечення було вирішено розгорнути на платформі Heroku. Для розгортання було використано сервіс GitHub Actions, який надає можливості для постійної інтеграції і розгортання [13].

Розгортання починається коли новий код застосунку доставляється у репозиторій у гілку main. Тоді у середовищі GitHub Actions створюється Docker image за допомогою Dockerfile, що знаходиться у проекті. Цей image розгортається у Heroku за допомогою пакету heroku-deploy. Інформацію про розгортання клієнтської і серверної частини проекту можна побачити на рисунках 4.1 і 4.2.

Рисунок 4.1 - Інформація про розгортання клієнту

## Підтримка програмного забезпечення

<У підрозділі викладають:

опис того, як буде виконуватись підтримка програмного забезпечення. Ілюстрації. Обсяг 1 сторінка, чи більше.>

Користувачі повинні мати можливість отримати нову версію консольного застосунку з кожною версією. До того ж кожна нова версія консольного застосунку повинна бути опублікована в npm. Для автоматизації цього процесу був використаний сервіс GitHub Actions.

Створення нового випуску починається, коли нова версія консольного застосунку доставляється у репозиторій у гілку main, тобто коли commit має tag формату “v\*.\*.\*.”, де замість “\*” знаходиться число. Тоді у середовищі GitHub Actions встановлюється NodeJS. Після цього для проекту встановлюються залежності і проект збирається. Bash скрипт за допомогою бібліотеки pkg генерує виконувані файли (executables) для Linux і для Windows, та пакує файл для Linux у .deb пакет. Після цього .deb пакет і файл для Windows архівуються,

## Висновки до розділу

< Необхідно стисло описати усе, що було виконано у даному розділі. Обсяг 0,75-1 сторінка>

# ВИСНОВКИ

У висновках викладають найважливіші наукові й практичні результати роботи та наводять:

* оцінку одержаних результатів і їх відповідність сучасному рівню наукових і технічних знань;
* ступінь впровадження та можливі галузі або сфери використання результатів роботи;
* наукову, науково-технічну, соціально-економічну значущість роботи;
* доцільність продовження досліджень за відповідною тематикою тощо.

Також у висновках необхідно відобразити стан вирішення усіх поставлених в курсовій роботі задач.

В результаті виконання курсової роботи було спроєктовано …

В якості середовища розробки обрано …

У якості БД використано …

Після реалізації застосунку він був протестований на пристроях з різними версіями Android, з різними розмірами екранів щоб переконатися, що додаток акуратно відображається на різних пристроях.

Наукова новизна роботи (якщо вона є) полягає в наступному (достатньо вказати щось одне).

Вперше:

* реалізовано можливість запитів від пацієнта до лікаря;
* використано те-то, що дозволило те-то.

Модифіковано:

* те-то, що дозволило те-то.

Набуло подальший розвиток:

* те-то, що дозволило те-то.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://en.wikipedia.org/wiki/Tower\_defense
2. <https://en.wikipedia.org/wiki/Rampart_%28video_game%29>
3. <https://en.wikipedia.org/wiki/Desktop_Tower_Defense>
4. https://en.wikipedia.org/wiki/Video\_game
5. <https://en.wikipedia.org/wiki/Single-player_video_game>
6. <https://en.wikipedia.org/wiki/Component-based_software_engineering>
7. <https://medium.com/geekculture/an-in-depth-guide-to-component-based-architecture-d023f02f4147>
8. https://en.wikipedia.org/wiki/Dungeon\_Siege
9. <https://github.com/OpenSiege>
10. <https://plarium.com/uk/games/genre/tower-defense-games/>
11. <https://cloud.google.com/learn/what-is-microservices-architecture#:~:text=A%20microservices%20architecture%20is%20a,architecture%20diagrams%20and%20services%20independently>.
12. <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a,monolith%20architecture%20for%20software%20design>.
13. <https://www.simplilearn.com/what-is-client-server-architecture-article#:~:text=The%20client-server%20architecture%20refers,model%20or%20client%20server%20network>.
14. <https://en.wikipedia.org/wiki/Finite-state_machine>
15. https://en.wikipedia.org/wiki/Dependency\_injection#:~:text=Dependency%20injection%20aims%20to%20separate,how%20to%20construct%20those%20services.
16. <https://refactoring.guru/design-patterns/observer>
17. <https://www.geeksforgeeks.org/backtracking-algorithms/>
18. <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>
19. <https://en.wikipedia.org/wiki/Knapsack_problem>
20. <https://unity.com>
21. <https://www.unrealengine.com/en-US>
22. https://docs.unity3d.com/Manual/class-ParticleSystem.html
23. <https://docs.unity3d.com/Manual/UsingTheEditor.html>
24. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/>
25. <https://learn.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern>
26. <https://visualstudio.microsoft.com>
27. https://www.jetbrains.com/rider/
28. <https://www.jetbrains.com/lp/dotnet-unity/>
29. https://en.wikipedia.org/wiki/Plants\_vs.\_Zombies\_(video\_game)
30. https://en.wikipedia.org/wiki/Bloons\_TD\_6