ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ГРИ «OUTDEF» У ЖАНРІ TOWER DEFENSE НА UNITY C#»**

Виконав: студент 5 курсу, групи ППЗ-51
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)
Мягких М.О.
(прізвище та ініціали)
Керівник Негоденко О.В.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки -121 - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
“___”_____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
_____ МЯГККИХ МИКИТІ ОЛЕКСАНДРОВИЧУ _____
(прізвище, ім‘я, по батькові)

1.Тема роботи: «Розробка гри «OutDef» у жанрі tower defensена Unity C#»
_____

Керівник роботи: Негоденко О.В., к.т.н., доцент
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від“18”лютого 2022 року №22.

2. Строк подання студентом роботи 03.06.2022

3. Вхідні дані до роботи:

3.1 Положення розробки гри;

3.2 Методи розробки гри;

3.3 Розробка алгоритмів, моделі гри;

3.4 Науково-технічна література.

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

4.1 Загальні положення та методи створення ігор;

4.2 Аналіз системних засобів та систем програмування;

4.4 Аналіз алгоритмів та методів для гри;

4.5 Висновки.

5. Перелік графічного матеріалу

1. Ознайомлення з середовищами для розробки програмних додатків;
2. Засоби розробки;
3. Опис функціоналу;
4. Тестування програмного додатку.

6. Дата видачі завдання 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Ознайомлення з документами та матеріалами для проходження переддипломної практики.	11.04-14.04	
2.	Постановка задач практики. Формування календарного плану.	15.04-17.04	
3.	Аналіз предметної галузі у жанрі «Tower defense»	18.04-21.04	
4.	Аналіз та порівняння можливих системних засобів для реалізації програмного додатку	22.04-.25.04	
5.	Розробка програмного додатку, опис алгоритмів	26.04-05.05	
6.	Тестування розробленого додатку	05.05-07.05	
7.	Визначення об’єкта, предмета, мети кваліфікаційної роботи бакалавра та постановка завдань	07.05-10.05	
8.	Оформлення звітної документації(звіт, щоденник практики)	11.05-15.05	
9.	Підготовка презентаційних матеріалів	16.05-22.05	
10.	Попередній захист роботи	16.05-01.06	
11.	Подання роботи в деканат	03.06	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 65 стр., 23 рис., 35 джерел

Ключові слова: ГРА, ВІДЕОІГРИ, TOWER DEFENSE, UNITY, мова програмування C#

Об'єкт дослідження – процес створення гри у жанрі «Tower defense».

Предмет дослідження – програмне забезпечення у жанрі «Tower defense».

Мета роботи – створення програмного додатку (гри) за рахунок використання ігрового рушія Unity на платформі Android

Методи дослідження – порівняння програмних засобів, метод оптимального кодування, обробка та аналіз інформації.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Проаналізувати існуючі ігри в жанрі «Tower defense», знайти їх переваги та недоліки;
2. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового додатку;
3. Розробити вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків;
4. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з жанром «Tower defense» з використанням ігрового двигуна UNITY та мови програмування C#. Також реалізовану гру можна поширювати серед гравців, виклавши її до магазинів ігор.

В роботі розглянуто основні етапи створення ігор і досліджено можливості технічних засобів для розробки гри.

Розроблено модель застосунку та функціональні вимоги до гри та підібрані методи реалізації механік.

Галузь використання – завдяки розвитку IT індустрії, будь-який користувач може завантажити та використовувати гру в жанрі «Tower defense».

ЗМІСТ

ВСТУП

Актуальність теми. Перші відеоігри з’являлися на комп’ютери та приставки. Та й зараз при слові «відеогра» на думку спадає саме комп’ютерна гра. Але останнім часом мобільні ігри набирають шалену популярність. Це пов’язано з тим, що в останні роки спостерігається зростання попиту на мобільні пристрої, через доступність всіх можливостей, включаючи вихід в Інтернет, не тільки через стаціонарні ПК, а й через планшети, смартфони і звичайні телефони. Потужний комп'ютер для роботи потрібен далеко не всім, а ось мобільний пристрій, який займає мало місця, завжди з вами і все може – зовсім інша справа. Цілком природно, що постійно з'являються нові застосунки для Android, IOS і Windows, що роблять процес експлуатації пристроїв ще більш комфортним і простим.

У зв’язку з цим розвивається й ринок розваг для мобільних пристроїв. Кожного дня в магазини мобільних ігор виходить 1-2 гри. Тому щоб створювати конкурентоспроможні та якісні ігри – потрібно генерувати нові ідеї та мати багато досвіду для втілення їх у життя. Потрібно розроблювати універсальні модулі, що можуть використовуватись у майбутніх розробках, покращувати візуальну складову ігор задля привернення більшої аудиторії до ігрового застосунку. Саме тому темою дипломного проекту я обрав розробку мобільної гри. Адже ринок ігор сьогодні один із самих прибуткових. Гаджети стрімко розвиваються щоб полегшити життя людини та зайняти дозвілля. Популярність мобільних телефонів зростає.

Мета та завдання. Виходячи з актуальності завдання, метою є створення програмного додатку(гри) за рахунок використання ігрового рушія Unity на платформі Android.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Проаналізувати існуючі ігри в жанрі «Tower defense», знайти їх переваги та недоліки**;**
2. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового додатку;

3. Розробити вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків;

4. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

Об'єкт дослідження – процес створення гри у жанрі «Tower defense»

Предмет дослідження – програмне забезпечення у жанрі «Tower defense»

Мета роботи – створення програмного додатку(гри) за рахунок використання ігрового рушія Unity.

Методи дослідження – порівняння програмних засобів, метод оптимального кодування, обробка та аналіз інформації.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з жанром «Tower defense» з використанням ігрового двигуна UNITY та мови програмування C#. Також реалізовану гру можна поширювати серед гравців, виклавши її до магазинів ігор.

Результати роботи. Матеріали дипломного проекту можуть сприяти вдосконаленню ігрових механік, підвищенню конкуренції, цим чином підвищить якість нових додатків, створенню більш просунутих ігрових світів.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ІГОР ЖАНРУ «TOWER DEFENSE» НА РИНКУ. ЗАСТОСУВАННЯ В ОСВІТНЬОМУ ПРОЦЕСІ

Інформаційні технології (ІТ) зростали та розвивалися за останні 50 років; Ви не можете думати та планувати проект, бізнес чи іншу ініціативу без використання цієї технології. Коли ми говоримо про інформаційні технології, то мають на увазі не лише персональні комп'ютери чи смартфони, а й сучасне обладнання на фабриках, автомобільній промисловості, авіаційній промисловості, різноманітну побутову техніку тощо, це так чи інакше не тільки полегшило наше повсякденне життя, але й також зменшується вартість та час загалом.

Дослідження показують, що чверть працівників країни, значну частину року працюють вдома, а ще чверть працюють «мобільними» - на ходу. Це відображає великі можливості, які надають інформаційні технології та Інтернет як важливий інструмент для впровадження в організаціях та державних установах. Економісти високо цінують важливість інформаційних технологій для зростання бізнесу, зниження витрат і просування найкращих продуктів.

Протягом останніх років глобалізація та комп'ютеризація змінили промисловість, політику, культуру та соціальний порядок. Глобалізація означає остаточну інтеграцію економічних і культурних інститутів. Ця інтеграція відбувається в результаті використання інформаційних технологій. технологічна революція передбачає глобальні комп'ютеризовані мережі та вільний рух товарів, інформації та людей через національні кордони. Отже, Інтернет та глобальні комп'ютерні мережі роблять можливим глобалізацію, створюючи технологічну інфраструктуру для світової економіки. Комп'ютеризовані мережі, системи супутникового зв'язку, програмне та апаратне забезпечення пов'язують разом і сприяють розвитку світової економіки.

1.1. Опис предметного середовища у жанрі «Tower defense»

“Tower defense”, як ігровий жанр, з'явилися на початку 1980-х років, коли ігрові консолі були досить потужними. Дія гри не виходила за межі одного екрану.

Персонаж лазив вгору і вниз сходами або стрибав з платформи на платформу, часто борючись із противниками та долаючи перешкоди. Незабаром процес проходження рівня перестав бути в основному вертикальним і став горизонтальним з появою довгих багатоекранних ігрових світів, що прокручуються.

Теоретична значимість даної роботи полягає в тому, що набутий мною досвід на всіх етапах розробки даного мобільного додатка дозволить у майбутньому набагато впевненіше ставити собі за мету проектування більш складних ігор і досягати їх. До того ж дана робота зможе привернути увагу інших студентів до програмування, покаже, що процес створення гри захоплюючий і пізнавальний, дозволяє втілити в життя всі свої ідеї та задуми. І зрештою скористатися всіма корисними властивостями кінцевого продукту – гри.

Пристосованість гри до різних мобільних пристроїв, і, відповідно до найпоширеніших платформ (Android, iOS, MacOS, Windows та ін) є чудовою перспективою для її успішного просування, залучення великої кількості аудиторії при реалізації кінцевого користувача.

Спочатку можна розробити гру для однієї операційної системи (ОС), у нашому випадку для Android. Надалі, проаналізувавши всі недоліки, щоб уникнути їхнього повторення, портувати гру на інші ОС.

Комп'ютерні ігри мають неослабний інтерес. Вони приносять чимало користі. Багато навчальних програм у своїй основі містять ігрову складову. Ігри вчать людину швидко реагувати, приймати рішення, відчуваючи свою відповідальність за це. Вони сприяють розвитку образного мислення, критичного, стратегічного мислення. Розвивають дрібну моторику, вчать планувати свої події. Можуть допомогти із вибором професії. Крім розвиваючої та навчальної функції, ігри дозволяють людині провести дозвілля, відволіктися від справ і просто відпочити.

Мова програмування C#

C# (вимовляється як «сі шарп») – проста, сучасна об'єктно-орієнтована мова програмування, але підтримує також компонентно-орієнтоване програмування. Розробка сучасних програм все більше тяжіє до створення програмних компонентів

у формі автономних та самоописових пакетів, що реалізують окремі функціональні можливості. Важливою особливістю таких компонентів є модель програмування на основі властивостей, методів і подій. Кожен компонент має атрибути, що надають відомості про компонент і вбудовані елементи документації. C# надає мовні конструкції, які безпосередньо підтримують таку концепцію роботи. Завдяки цьому C# відмінно підходить для створення та застосування програмних компонентів.

Ось кілька функцій мови C#, що забезпечують надійність та стійкість додатків:

• Складання сміття автоматично звільняє пам'ять, зайняту знищеними та невикористовуваними об'єктами;

• Обробка винятків дає структурований та розширюваний спосіб виявляти та обробляти помилки;

• Сувора типізація мови не дозволяє звертатися до неініціалізованих змінних, виходити за межі масиву або виконувати неконтрольоване наведення типів.

У C# існує єдина система типів. Усі типи C# успадковують від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій, і значення будь-якого типу можна зберігати, передавати та обробляти так.

Платформа Unity

Unity – міжплатформне середовище розробки комп'ютерних ігор. Unity дозволяє створювати програми, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, Інтернет-програми та інші.

Випуск Unity відбувся у 2005 році і з того часу триває постійний розвиток. Спочатку Unity призначався виключно для комп'ютерів Mac, але потім поступово виходили поновлення, що дозволяє працювати під Windows та інші ОС.

Основними перевагами Unity є наявність візуального середовища розробки, міжплатформної підтримки та модульної системи компонентів. До недоліків

відносять появу складнощів при роботі з багатокомпонентними схемами та утрудненні при підключенні зовнішніх бібліотек.

На Unity написані сотні ігор, додатків та симуляцій, які охоплюють безліч платформ та жанрів. Разом з тим, Unity використовується як великими розробниками, так і незалежними студіями.

Редактор Unity має простий Drag&Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна налагоджувати гру прямо в редакторі. Двигун підтримує дві скриптові мови: C#, JavaScript (модифікація).

Проект в Unity ділиться на сцени (рівні) – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв та налаштувань. Сцени можуть містити як, власне, об'єкти (моделі), і порожні ігрові об'єкти – об'єкти, які мають моделі («пустушки»). Об'єкти, у свою чергу, містять набори компонентів, з якими і взаємодіють скрипти. До об'єктів можна застосовувати колізії (в Unity т. зв. колайдери – collider). У редакторі є система успадкування об'єктів – дочірні об'єкти повторюватимуть усі зміни позиції, повороту та масштабу батьківського об'єкта. Скрипти у редакторі прикріплюються до об'єктів як окремих компонентів.

При компіляції проекту створюється файл гри (для Windows), що виконується (.exe), а в окремій папці – дані гри (включаючи всі ігрові рівні та бібліотеки, що динамічно підключаються).

Середовище розробки MonoDevelop

MonoDevelop IDE (інтегроване середовище розробки) (http://monodevelop.com) – вільне мультиплатформене середовище розробки, призначене для створення програм на мовах C#, C, C++, Java, Visual Basic.NET, CIL, Nemerle, Boo. Вбудований у дистрибутив Unity3D як написання скриптів.

Можливості:

1. Підсвічування синтаксису – виділення синтаксичних конструкцій тексту з використанням різних кольорів, шрифтів та зображення. Застосовується полегшення читання вихідного тексту комп'ютерних програм, поліпшення візуального сприйняття.

2. Згортання коду, або фолдинг (англ. folding) – одна з функцій текстового редактора, що дозволяє приховувати певний фрагмент коду або тексту, що редагується, залишаючи лише один рядок.

Наприклад, фолдинг функції призводить до згортання всього коду функції в один рядок таким чином, що буде видно лише назву функції. Зазвичай, щоб згорнути фрагмент, потрібно натиснути символ «?» ліворуч від нього. Щоб побачити весь фрагмент, тобто розгорнути його, потрібно натиснути на символ +, що з'являється у згорнутих фрагментів.

3. Автодоповнення коду – функція у програмах, що передбачають інтерактивне введення тексту (редактори, оболонки командного рядка, браузери тощо) за доповненням тексту за введеною його частиною.

4. Вбудований налагоджувач – використовується на етапі налагодження комп'ютерної програми, на якому виявляють, локалізують та усувають помилки. Щоб зрозуміти, де виникла помилка, доводиться:

- впізнавати поточні значення змінних;

- з'ясовувати, яким шляхом виконувалася програма.

5. Модульне тестування – одиничне тестування, або модульне тестування (англ. unit testing) – процес у програмуванні, що дозволяє перевірити на коректність одиниці вихідного коду, набори з одного або більше програмних модулів разом із відповідними керуючими даними, процедурами використання та обробки.

Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє досить швидко перевірити, чи не призвела чергова зміна коду до регресії, тобто появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок.

Редактор скелетної анімації Spriter

Скелетна анімація – спосіб анімування двовимірних та тривимірних моделей у мультиплікації та комп'ютерних іграх.

Полягає в тому, що мультиплікатор або модельєр створює скелет, що являє собою зазвичай деревоподібну структуру кісток, в якій кожна наступна кістка «прив'язана» до попередньої, тобто повторює за нею рухи та повороти з

урахуванням ієрархії в скелеті. Далі кожна вершина моделі «прив'язується» до будь-якої кістки кістяка. Таким чином, під час руху окремої кістки рухаються і всі вершини, прив'язані до неї. Завдяки цьому завдання аніматора сильно спрощується, тому що відпадає необхідність анімувати окремо кожну вершину моделі, а достатньо лише задавати положення та поворот кісток скелета.

Також завдяки такому методу скорочується обсяг інформації, необхідної для анімування. Досить зберігати інформацію про рух кісток, а рухи вершин обчислюються вже з них.

Spriter – програма для створення кісткової анімації, що працює з растровою графікою. Використовує модульний метод створення плавних анімацій. Цей спосіб має безліч переваг:

1. Економить час – неважливо, чи є ви досвідченим чи початківцем художником з дизайну ігор, ви зможете витрачати набагато менше часу на налаштування та налагодження персонажів, що анімуються, оскільки це дозволить вам повторно використовувати лише кілька модульних зображень.

2. Миттєва ітерація. Припустимо, необхідно змінити конфігурацію голови ігрового персонажа на етапі остаточної розробки. Якщо у вас є Spriter у 2D-панелі інструментів, вам потрібно буде лише змінити невелику частину зображень голови, тому що модулі налаштовані для використання у всіх кадрах анімації.

3. Налаштування користувача. Spriter набагато спрощує роботу дизайнера щодо створення будь-яких трюків, оскільки модульні зображення (частини тіла персонажів) можна вільно підштовхувати або повертати.

4. Необмежені зміни персонажів. Цей метод забезпечує надшвидке та безболісне створення нових персонажів на основі даних вже створеного персонажа. Також це ефективний з погляду пам'яті спосіб створення інструментів персонажа, які можуть змінюватися протягом гри (наприклад: збір бонусів, нова зброя тощо).

Аналіз популярних ігрових додатків показує, що на сьогоднішній день найбільшою популярністю користуються продукти, які мають простий візуальний стиль. Обчислювальна потужність сучасних смартфонів, звичайно, дозволить намалювати гарну картинку з величезною кількістю дрібних деталей, але

маленький розмір екрану призведе до того, що об'єкти почнуть накладатися один на одного. Звідси випливає перше рішення щодо проекту, що розробляється. Візуальну складову гри буде мінімалізовано.

1. 2. Методи та основні етапи розробки

Аналіз популярних додатків показав, що зараз між собою конкурують 3 жанри: головоломки, аркади та екшн-ігри. Для продукту, що розробляється, був обраний жанр - Tower defense.

В результаті розробки повинен вийти продукт, орієнтований на людей, які використовують мобільні пристрої на базі ОС Android, та призначений для розваги користувачів. Він буде 2D ігровий додаток у жанрі аркада з геймплеєм Tower defense. Візуальна складова програми буде реалізована наступним чином: як тло виступає сірий екран, а як персонажі — темні контури людей. Такий стиль не сильно навантажуватиме оперативну пам'ять пристрою, що збільшить кількість пристроїв, здатних запустити додаток, а також, в силу простоти виконання картинки, він дозволить розробнику приділити більше часу іншим аспектам розробки як рефакторинг коду. Більш того, такий стиль надаватиме найменший негативний вплив на очі. Досвід показує, що яскрава картинка з великою кількістю дрібних елементів добре виглядає на екранах з великою роздільною здатністю, але на екранах з невеликою роздільною здатністю вона починає швидко втомлювати очі користувача.

Геймплей проекту, що розробляється, злегка відрізнятиметься від канонічного Tower defense, де завданням гравця є подолання якомога більшої відстані та подолання перешкод за допомогою одного лише стрибка, за подолання певної відстані нараховується певна кількість очок, якщо гравець не долає перешкоду, гра закінчується. У проекті, що розробляється, крім функції стрибка буде присутня функція удару. Завданням гравця буде подолання розривів в опорній поверхні та знищення супротивників. За подолання певної відстані нараховуватиметься певна кількість очок. За знищення противника також нараховуватимуться очки. Таким

чином, основна мета гравця залишиться незмінною — набрати якнайбільше очок, але ігровий процес при цьому ускладниться через те, що за знищення противника нараховуватиметься більше очок, ніж за проходження однієї одиниці відстані. У проекті буде реалізовано автоматичну генерацію світу, іншими словами ігрові об'єкти (платформи та противники) генеруватимуться автоматично на певній відстані перед персонажем. Швидкість руху персонажів постійно збільшуватиметься, що ускладнить досягнення високих результатів. Також збільшення швидкості не дасть гравцеві увійти в ритм гри і ідеально вибирати момент.

1995 рік був особливо цікавим роком у світі комп'ютерного програмування. Саме в цьому році було випущено чотири нові мови програмування, які вплинули на світове співтовариство програмування таким чином, що не передбачалося на момент їх офіційного оголошення. До речі, це був також час, коли Інтернет тільки починав робити хвилі, а Інтернет був на межі вибуху в основну цифрову культуру.

Чотири мови, які дебютували в 1995 році, були Java, JavaScript, PHP і Ruby. Незважаючи на те, що їхні відповідні релізи не супроводжували особливої фанфари, ці мови з часом виростуть і стануть повсюдними інструментами програмування для більшості розробників програмного забезпечення. До цього часу домінуючими мовами були C і C++. Незважаючи на те, що ці двоє були дуже потужними, вони за своєю суттю не підходили для всесвітньої мережі. Крім того, для початківців програмістів (особливо C++) вони часто вважалися дещо складними та залякуючими.

Серед чотирьох, Java виявилася шаленим успіхом. З його часто цитованим гаслом «Напишіть один раз, запустіть будь-де» Java стала миттєвим хітом, оскільки її було набагато легше вивчити та опанувати (порівняно з C++). Java також представила нову ідею віртуальної машини (JVM), яка дозволила писати програми, які запускалися б на різних платформах без необхідності перекомпіляції.

В останні роки JavaScript витіснив Java як провідну мову програмування у світі. Постійне зростання JavaScript значною мірою пов'язано з впровадженням Node.js, технології, яка зробила можливим запуск JavaScript на стороні сервера.

PHP став домінуючою силою у сфері веб-програмування, особливо в поєднанні з іншими популярними технологіями з відкритим вихідним кодом, такими як Linux, Apache та MySQL. Разом вони утворили те, що зазвичай називають стеком LAMP.

Ruby здобув популярність серед веб-розробників після випуску шалено популярного веб-фреймворку Ruby on Rails у 2005 році датським програмістом Девідом Хайнемайєром Ханссоном (DHH).

Для великих (або командних) проектів в середу розробки повинні бути включені файловий менеджер, інтегроване середовище розробки програмного забезпечення, PlSql (використовується і для роботи з Системою Управління БД і як інструмент звітів), Cristal Reports (створення звітів), StarTeam (ведення журналу версій продукту, що розробляється).

Підводячи підсумки потрібно сказати про те, що інтегровані середовища розробки ПО дозволяють програмістам скоротити час на написання додатків, знизити затратність на написання (розробку), підвищити зручність розробки - що і є однією з основних цілей програмної інженерії.

Інтегровані середовища розробки зручні для командних проектів, остільки оскільки в таких середовищах можна виробляти весь цикл створення програмного забезпечення.

Інтегровані середовища зручні в написанні програм.

1. 3. Ознайомлення з середовищами для розробки програмного додатку

Середовища розробки ПО (програмного забезпечення) є об'єднанням програмних засобів, які призначені для написання (створення) програмних продуктів. -Середовище розробки включає в свій зміст: компілятор, інтерпретатор, відладчик, засоби автоматизації збирання, а також редактор тексту.

Компілятор - це така програма, яка зчитує вихідні коди, написані програмістом і перетворює ці коди в програму.

Інтерпретатор - це програма яка зчитує команди, що знаходяться в вихідних кодах, відразу виконуючи їх.

Коли в середовищі розробки ПО присутні всі вищеназвані компоненти, тоді таке середовище називають інтегрованою. Такі середовища розробки збільшують темп, а також зручність розробки за рахунок: автоматизації, можливості виробляти весь цикл створення і розробки ПЗ.

Зазвичай середовище розробки ПО призначена для розробки тільки на одній мові програмування. А таке середовище розробки як інтегрована, надає право вибрати творцеві програми мову програмування для розробки, зручний розробнику (з мов підтримуваних цим середовищем). Прикладом цього є: Visual Studio, Komodo, Geany, Kylix, NetBeans, Eclipse.

MicrosoftVisualStudio - одна з інтегрованих середовищ розробки, розроблена на C ++ і C #, підтримується Windows OS. Дане середовище розробки переведена на десять мов (також і на російську мову). У Visual Studio творець може вести розробку веб-сайтів, веб-служб, писати консольні додатки, а також додатки з графічним інтерфейсом. Також VS підтримує різного роду доповнень. Найзнаменитіші доповнення - це Re Sharper (виконує пошук помилок в коді під час написання коду програми розробником, до компіляції); Visual Assist (на відміну від Re Sharper підтримує також і C ++); AnkhSVN (використовує в Visual Studio систему контролю версій, яка носить назву Subversion).

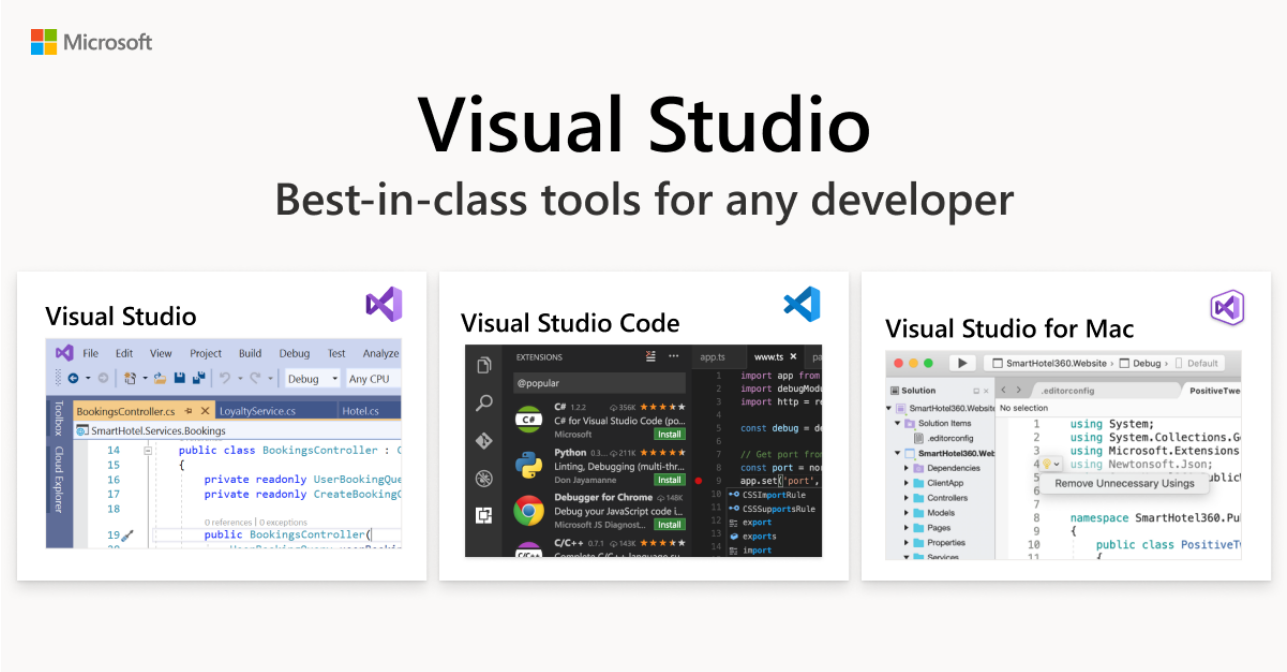

Рис. 1.2 – Visual studio

Переваги: Зрозумілий інтерфейс середовища розробки, зручність, автоматичне виявлення помилок в коді.

Недоліки: Складно для початківців програмістів.

Середовище особливо поширена в англомовних країнах, Росії, Китаї, Німеччині, Франції, Португалії, Італії, Японії, Іспанії та Кореї.

Geany також інтегроване середовище розробки програмного забезпечення. Підтримується на ОС Linux, а також на Mac Os і на Windows. Працює з тридцятьма двома мовами (також і з російською мовою). У складі Geany відсутня компілятор. Компілятор можна встановити як доповнення. Підтримує досить багато мов програмування, серед яких присутні класичний С. С ++ і С #.

Переваги: Простота і зручність, підсвічування вихідного коду, можливість підключати доповнення.

Недоліки: не включає до свого складу компілятор.

Серед розповсюджувалися в багатьох країнах (Більш ніж у тридцяти).

Komodo або ActiveStateKomodo - була написана на JavaScript, XUL, Python. Інтерфейс даного середовища тільки на англійською мовою. Работает на тех же операціонних система як Geany: на Os Linux, Windows і Mac Os.

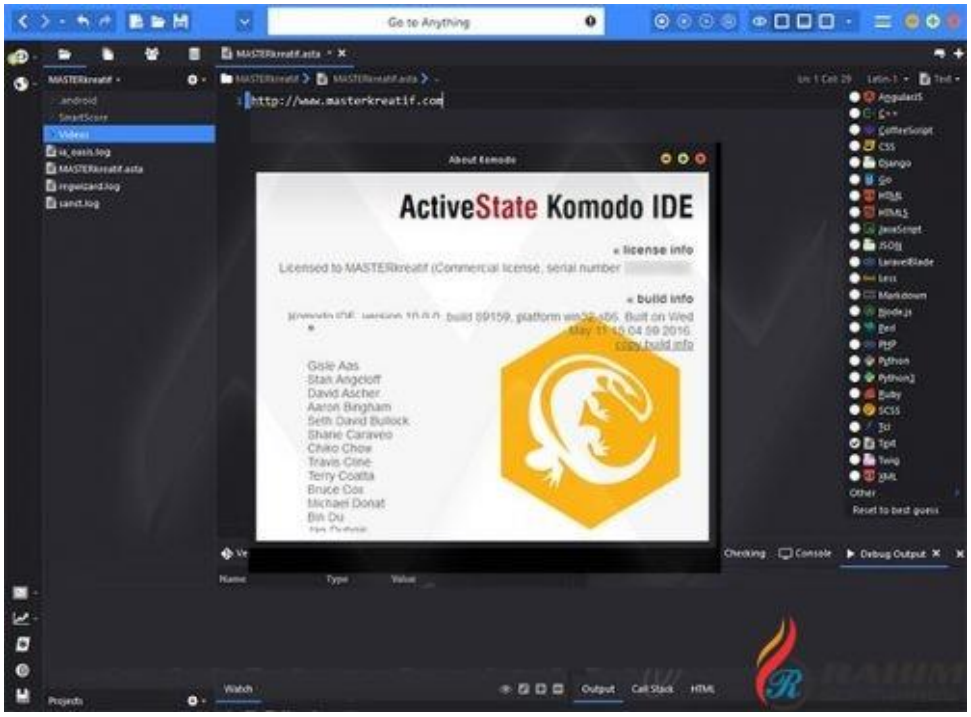

Рис. 1.3 - ActiveStateKomodo

Підтримує десять мов програмування, серед які присутні: PHP, Ruby, HTML5.

Переваги: Доповнення Code Explorer дозволяє переглядати об'єктне дерево скрипта або бібліотеки, середовище є кроссплатформенной, зручний відладчик з можливістю вилученого налагодження, можливість налаштувати інтерфейс середовища «під себе».

Недоліки: Висока вартість, підтримує мало мов програмування, сильно завантажує комп'ютер (а саме оперативну пам'ять), є складним для розуміння.

Поширена в основному в англомовних країнах.

Kylix - інтегроване середовище. Функціонує на OS Linux. Працює з C, C ++ і ObjectPascal.

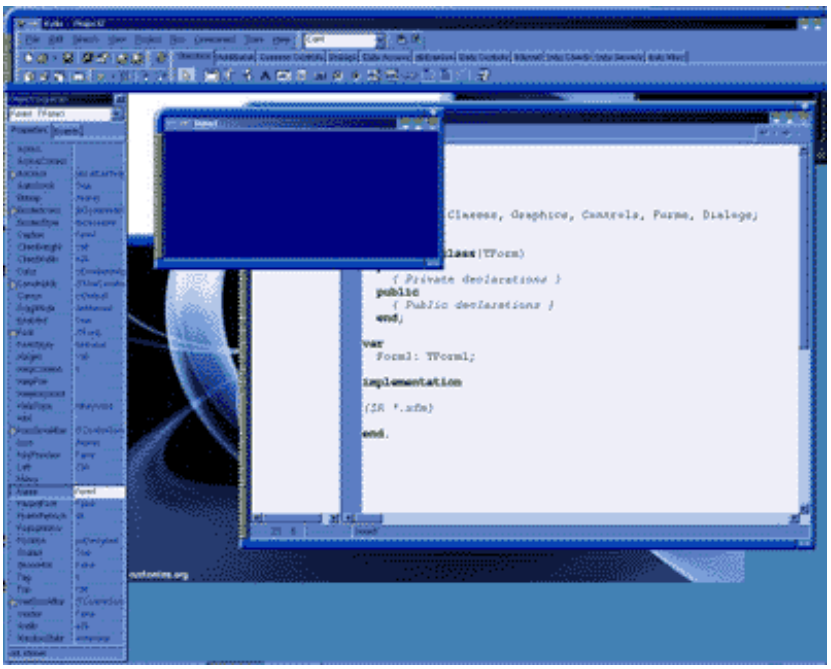

Рис. 1.4 - Kylix

В даному середовищі є можливість писати програми веб-служб.

Kylix випускався в трьох пакетах. Ці пакети: Enterprise Edition - включав в себе сто дев'яносто компонентів (був найбільшим і самим дорогим пакетом програми); Professional Edition (дешевший варіант, який включав в себе близько 165 компонентів); Open Edition - безкоштовний пакет програми, що містить в собі 75 компонентів, в ньому відсутній кошти для роботи з базами даних.

Оновлена версія Kylix 2, на відміну від Kylix працювала набагато швидше. Наприклад, Kylix 1 здійснював сортування бульбашкою масиву з 115 елементів півтори хвилини, Kylix 2 - одну секунду.

У 2002 році дане середовище розробки припинив підтримувати розробник.

Переваги: Зручний в перенесенні написаного з однієї операційної системи на іншу.

Недоліки: Дане середовище більше не підтримується розробником.

Поширена в основному в Європейських країнах і США, через те що розробник (Borland) перестав підтримувати Kylix - стає все менш популярною і не затребуваною.

Netbeans - інтегроване середовище розробки програмного забезпечення. Була реалізована на програмному мовою Java. Це середовище розробки високої якості. Вміє працювати на декількох операційних системах, тобто є кроссплатформенной. Працює більш ніж з п'ятьма програмними мовами.

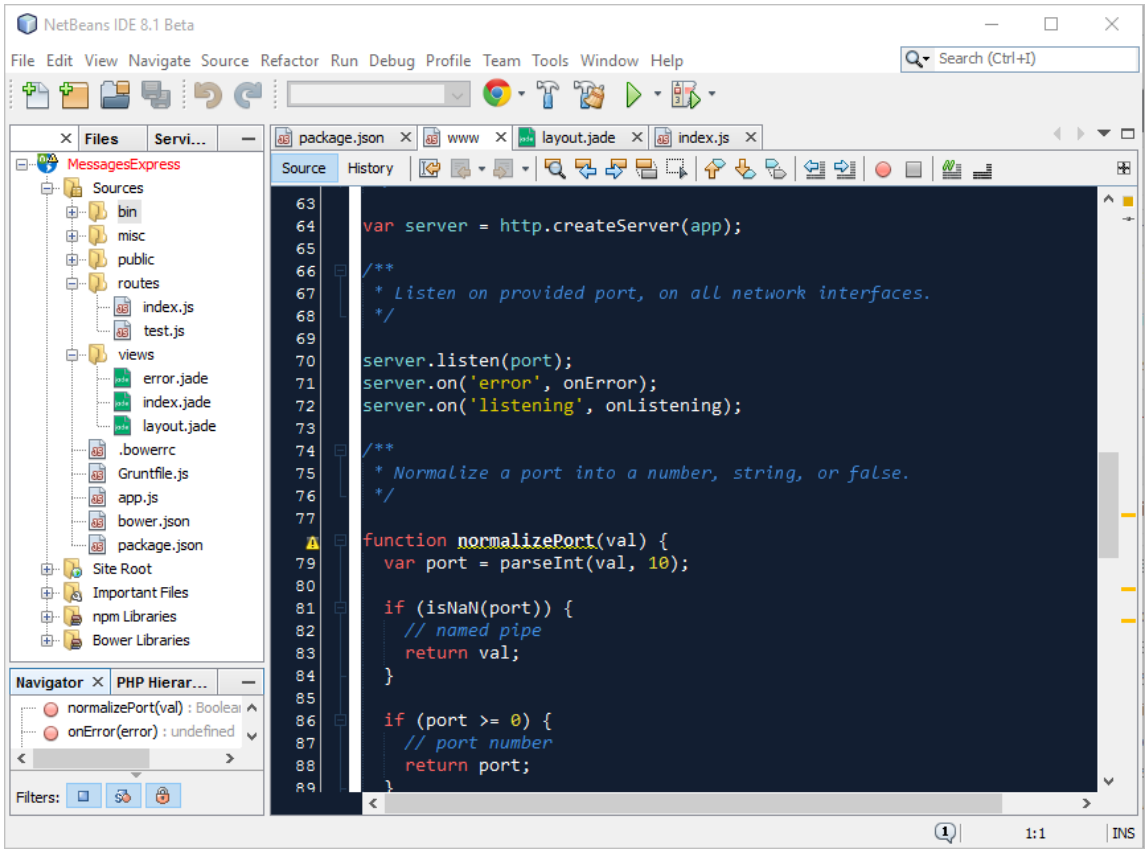

Рис. 1.5. - Netbeans

Переваги: Є безкоштовною, присутній система контролю версій, підсвічування синтаксису, можливо перейменовувати змінну / клас одним кліком, в тому випадку якщо вручну перейменовувати занадто довго (автоматизоване перейменування), є можливість форматування коду по CodeStyle, розробником середовище постійно вдосконалюється, поліпшується.

Недоліки: Часом в середовищі розробки виникають проблеми з кодуванням, довгий запуск програми.

Поширена в багатьох країнах, в силу того що є зручною і безкоштовною.

Eclipse - ще одна інтегрована середовище розробки ПО. Написана на мові Java в дві тисячі третьому році. Також є кроссплатформенной. За рахунок приєднуються до цього середовища доповнень - є можливість створювати програмні продукти більш ніж на п'яти мовах програмного коду.

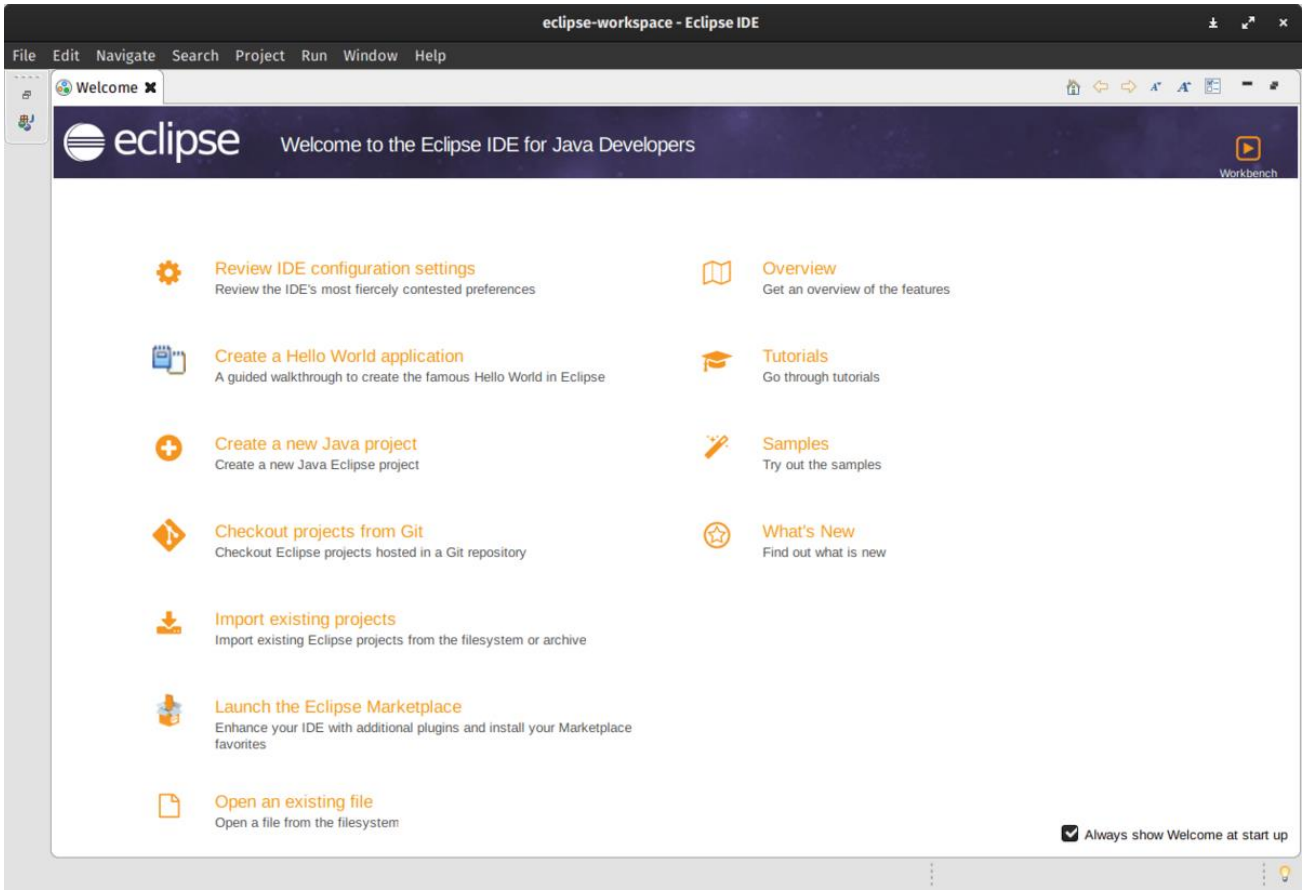

Рис. 1.6 - **Eclipse**

Переваги: Постійне оновлення версій середовища розробки, підтримка багатьох мов (включно з російською), безкоштовно, підтримка багатьох мов програмування, середовище має промисловий рівень, гнучка - що легко налаштовується як під будь-яку платформу, так і під будь-який користувач.

Недоліки: Сильно навантажує ОЗУ комп'ютера, довго запускається, проте якщо комп'ютер досить потужний - ця проблема легко вирішується.

Поширений у багатьох країнах, популярний.

РОЗДІЛ 2. ОПИС ПРОГРАМНОГО ДОДАТКУ

2.1 Опис технічного завдання

Завданням дипломного проекту є створення гри при створенні ряду класів, об’єкти яких будуть взаємодіяти між собою. Графічний інтерфейс гри повинен відповідати сучасним вимогам до таких проектів.

У реалізованому проекті можна розширити кількість класів, генеруючи класи, успадковані від класів Вежі та Ворога, тим самим збільшуючи різноманітність як наступальних, так і оборонних засобів.

2.2 Опис середовища та системи

Unity - це мультиплатформний інструмент для розробки дво- та 3D-програми та ігри під керуванням Windows та OS X. Програми на базі Unity працюють під керуванням Windows, OS X, Android, Apple iOS, Linux, а також ігрових консолей Wii, PlayStation 3 та XBox 360. Спеціальний плагін для браузера Unity, а також шляхом експериментальної реалізації у модулі Adobe Flash Player. Програми, створені за допомогою Unity, підтримують DirectX та OpenGL. Редактор Unity має простий інтерфейс Drag&Drop, який легко налаштовується та складається з різних вікон, тому ви можете налаштовувати гру прямо у редакторі. Двигун підтримує три скриптові мови: C#, JavaScript (модифікація). Проект Unity поділений на сцени (рівні) – окремі файли, які містять свої ігрові світи з власним набором об'єктів, сценаріїв та налаштувань. Сцени можуть містити як об'єкти (моделі), так і порожні ігрові об'єкти, тобто ті, які не мають моделі. Об'єкти, своєю чергою, містять набори компонентів, із якими взаємодіють скрипти. Об'єкти з видимою геометрією мають компонент Mesh Renderer, який робить їх модель видимою.

Unity також підтримує фізику твердого тіла та тканин, фізику Ragdoll (ганчіркова лялька). У редакторі є система наслідування об'єктів; дочірні об'єкти

будуть повторювати всі зміни положення, повороту та масштабу батьківського об'єкта. Скрипти у редакторі прикріплюються до об'єктів як окремі компоненти.

Коли ви імпортуєте текстуру в двигун, ви можете згенерувати альфа-канал, рівні MIP, карту нормалей, карту освітлення, карту мапінгу, але ви не можете прикріпити текстуру безпосередньо до моделі - ви створите матеріал, з якого шейдер буде призначено, а потім матеріал буде прикріплений до моделей. Редактор Unity підтримує написання та редагування шейдерів. Крім того, він містить компонент для створення анімації, анімацію також можна попередньо створити у 3D-редакторі та імпортувати разом із моделлю, а потім розбити на файли.

Unity має вбудовану підтримку мережі. Двигун гри повністю інтегрований із середовищем розробки. Це дозволяє протестувати гру прямо в редакторі. Має вбудований ландшафтний генератор.

2.3 Засоби розробки

Написання безпечного коду, що забезпечує безпеку без уразливостей, є критичним викликом для кібербезпеки. Написання коду без уразливостей вже давно принаймні так само складно, як написання коду без помилок. Хоча в програмному забезпеченні є багато інших потенційних джерел ризику безпеки, розробка коду без відомих класів уразливостей завжди здавалася посильною метою. Він покладається на людей-розробників, які використовують інструменти, методи та процеси для створення програмного забезпечення, яке не має особливо відомих типів дефектів.

Один з найефективніших підходів — дослідження мов та інструментів програмування — приніс технології, які, як показано, протистоять категоріям уразливостей, в основному, не допускаючи їх. Безпечні мови пам'яті, які керують виділенням та звільненням пам'яті, замість того, щоб вимагати від програміста це робити, унеможливлюють для розробників створення вразливостей переповнення буфера та деяких інших типів впливу через відсутні перевірки меж масиву, використання нульового покажчика та даних. витік через повторне використання

пам'яті. Потокобезпечні мови можуть вирішувати випадки, коли умови гонки можуть бути використані, щоб підірвати перевірки, пов’язані з безпекою в програмі.

Рис. 2.1 –Методи розробки

У межах спільноти розробників програмного забезпечення групи та організації, які мають на меті безпечну розробку програмного забезпечення, включили інструменти та методи у свій життєвий цикл розробки програмного забезпечення, щоб включити життєвий цикл безпечної розробки. Раннє програмне забезпечення з високою надійністю використовувало формальні методи для визначення властивостей безпеки системи, а також перегляд коду, щоб використовувати людину для пошуку таких недоліків на рівні кодування.2 Корпорація Майкрософт створила свій життєвий цикл розвитку безпеки, додавши аналіз першопричин, навчання безпеки, моделювання загроз, конкретні вимоги до безпечного кодування та тестування безпеки, яке включало тестування на проникнення та нечіткое тестування. Практика, як правило, впроваджується на

основі потреб бізнесу, відчутного впливу на безпеку та відповідає усталеній або розвивається практиці розвитку.

Необхідні дослідження, які впливають на те, що працює, а що може працювати для безпечного розвитку. Здається, що нинішні дослідження відіграють, на жаль, обмежену роль у створенні, пропонуванні, оцінці та підтвердженні інструментів, методів і процесів, які використовуються на практиці для безпечного розвитку. Зокрема, дослідження рідко стосуються безпосередньо інструментів і методів, оскільки вони використовуються, у контексті, в якому вони використовуються. Нам потрібні додаткові дослідження ефективності та результатів безпечних інструментів, методів і процесів розробки. Це дослідження можна судити про його вплив на те, як розробка програмного забезпечення працює на практиці. Властивості дослідження впливають на ймовірність такого впливу.

Суворість дослідницького наукового експериментування вимагає ряду вимог до процесу, включаючи формулювання гіпотези, що перевіряється, контроль змінних експерименту, щоб переконатися, що експеримент фактично перевіряє гіпотезу, і аналіз експериментальних даних і результатів для математичного підтвердження гіпотези (або спростувати нульову гіпотезу). Хоча ці процеси можуть стати основою важливих фундаментальних досліджень безпечної розробки, вони часто уникають безладних реалій, пов’язаних із впровадженням техніки на практиці, саме тому, що ці безладні реалії ускладнюють проектування експериментів.

Негативні результати дослідження, які не підтверджують, що безпечна техніка розробки підвищує безпеку, хоча й важливі для галузі досліджень, навряд чи вплинуть на безпечний розвиток на практиці. Перший урок розробника безпеки у великій технологічній компанії полягав у тому, що вказувати розробникам не робити чогось майже завжди було неефективним, якщо це не було поєднане з альтернативою, яку вони могли б використати для досягнення мети застарілої практики. «Не кидайте свою власну криптовалюту» має постати разом із криптобібліотекою, яку слід використовувати. Крім того, виявити, що інструмент або техніка експериментально неефективні для забезпечення безпеки, не доводить,

що вони неефективні за межами контрольованого експерименту, у більш широкому, безладному та різноманітнішому контексті розробки програмного забезпечення.

Які з тих речей, які робляться в дослідженні, сподіваються на практичне перенесення в безпечний розвиток? Дві сучасні тенденції досліджень безпеки дають певну надію на безпечний розвиток. Одна з них полягає в тому, що безпечна розробка стала темою на конференціях з дослідження безпеки, охоплюючи такі теми, оцінка інструментів, які допомагають розробникам уникати вразливостей, і вимірювання здатності розробників кодувати функціональні можливості, що стосуються безпеки. .

Інша обнадійлива тенденція — оцінка артефактів. Багато розробок програмного забезпечення базується на існуючому програмному забезпеченні, використовуючи фреймворки, бібліотеки та відкритий код. Пропонування артефакту, який використовується для встановлення та підтвердження ідеї дослідження, зменшує бар'єри для передачі цієї ідеї в розробку програмного забезпечення. Доступ до коду з відкритим кодом з умовами ліцензії, зручними для повторного використання, може збільшити його потенціал для використання. Деякі дослідницькі стимули змінюються, щоб заохочувати подання артефактів у рамках процесу подання та публікації наукової роботи на конференціях з безпеки, таких як USENIX Security і ACSAC.

2.4 Вимоги до технічного забезпечення

Запуск комп'ютерних програм на голій машині – не нова концепція. Так почалися обчислення десятиліття тому, коли програма завантажується вручну за допомогою перемикачів і запускається натисканням кнопки запуску. У цій статті ми про це не говоримо! Однак наші комп'ютерні системи складні, і вони виросли за межами пропорції, створюючи великий семантичний розрив між додатками та обладнанням. Наприклад, остання операційна система (ОС) Microsoft XP має 40 мільйонів рядків коду. Настав час переглянути еволюцію обчислювальної техніки

та шукати рішення, використовуючи нові парадигми. ОС і середовище швидко змінювалися з останніх 30 років, що робить речі застарілими, перш ніж вони зможуть стати продуктивними протягом свого життя. Наприклад, Microsoft випустила понад 20 основних випусків ОС за останні 25 років. Кожна мова та середовище випускають новий випуск кожні шість місяців. Коли ОС або її середовище змінюються, це має ефект пульсації, що призводить до застарілих програм. Як ми можемо зупинити таке поширення продуктів і технологій і зробити програми стабільними? Не забезпечуючи жодної перевірки, одним із можливих підходів є уникнення операційної системи та її середовища. У деяких областях обчислень це робиться повільно, непомітно.

Після завантаження ПК він перебуває в реальному режимі, де адресація обмежена 1 МБ і немає захисту вашого коду. Це просто в режимі дискової операційної системи (DOS). Щоб завантажувати та запускати більші програми та мати доступ до більшої пам’яті, вам потрібно навчитися переходити в захищений режим за допомогою інструкцій на мові асемблера. Однак усі виклики базової системи введення/виводу (BIOS) доступні лише в реальному режимі. Якщо ви хочете використовувати BIOS для вводу-виводу, вам потрібно мати механізми перемикання із захищеного режиму в реальний для виконання викликів BIOS. Таким чином, ваша програма на C++ або якась керуюча програма повинні виконувати це перемикання, прозоре для користувача. Якщо ви плануєте використовувати програмні переривання (їх 255) замість переривань BIOS, то вам потрібно написати власний код на зборці для вирішення всіх операцій введення-виводу. Ми використовували як BIOS, так і програмні переривання і написали перемикач захищеного режиму в режим реального для роботи в обох режимах. Це нетривіальна річ, яку потрібно зробити під час складання, яка вимагає від вас ознайомлення з документом специфікації архітектури Intel, який доступний на їх веб-сайті. Цей документ є корисним ресурсом для розуміння та реалізації переривань, схем адресації, засобів завдань, пасток і винятків.

2.5 Опис архітектури програмного забезпечення

Архітектура програмного забезпечення дає пояснення того, як ваші системи поводяться на структурному рівні. Системи, які ви використовуєте, мають набір компонентів, розроблених для виконання певного завдання або набору завдань. Архітектура програмного забезпечення забезпечує фундамент, на якому все програмне забезпечення, яке є у компанії, можна змінити, створити або вилучити з експлуатації.

Архітектура програмного забезпечення впливає на якість, продуктивність, обслуговування та успіх системи на основі дизайну. Не розглядаючи архітектуру програмного забезпечення на регулярній основі, компанія відкриває себе для довгострокових наслідків, і проблеми, які можуть поставити їх системи під загрозу поломки, злому або низької продуктивності.

У сучасних системах існують загальні шаблони в архітектурі програмного забезпечення, які називаються архітектурними системами для програмного забезпечення. У більшості випадків для створення цілісної системи використовується кілька різних архітектурних систем, особливо для систем, які створювалися роками або працювали, або тих, які були побудовані різними розробниками.

Архітектор не повинен втрачати зв'язку з тим, що розробники вважають за важливе. Ви повинні говорити мовою розробників. Ремонтопридатність є найбільшим фактором витрат на IT. Ви маєте проектувати з думкою про розробників. Архітектори, які забувають своє коріння, втрачають повагу своїх розробників. Вони фактично стають уславленими менеджерами проектів і нікому не потрібен додатковий менеджер.

2.5.1 Опис класів

Клас у Java є будівельним блоком, який веде до об'єктно-орієнтованого програмування. Це визначений користувачем тип даних, який містить власні члени даних і функції-члени, до яких можна отримати доступ і використовувати, створивши екземпляр цього класу. Клас Java схожий на план для об'єкта.

Наприклад: розглянемо клас автомобілів. Може бути багато автомобілів з різними назвами та марками, але всі вони будуть мати деякі спільні властивості, як-от усі вони матимуть 4 колеса, обмеження швидкості, діапазон пробігу тощо. Отже, автомобіль – це клас і колеса, обмеження швидкості, пробіг. їх властивості.

Клас — це визначений користувачем тип даних, який має члени даних і функції-члени.

Члени даних — це змінні даних, а функції-члени — це функції, які використовуються для маніпулювання цими змінними, і разом ці члени даних і функції-члени визначають властивості та поведінку об’єктів у класі.

На прикладі класу Car членом даних буде обмеження швидкості, пробіг тощо, а функціями члена можуть бути гальмування, збільшення швидкості тощо.

Об'єкт - це екземпляр класу. Коли клас визначено, пам’ять не виділяється, але коли він створюється (тобто створюється об’єкт), пам’ять виділяється.

Визначення класу та оголошення об'єктів

Клас визначається в Java за допомогою ключового слова class, за яким слідує ім'я класу. Тіло класу визначається всередині фігурних дужок і закінчується крапкою з комою в кінці.

Оголошення об'єктів: коли визначено клас, визначається лише специфікація об'єкта; пам'ять або сховище не виділено. Для використання даних і функцій доступу, визначених у класі, потрібно створити об’єкти.

Синтаксис:

ClassName ObjectName;

Доступ до членів даних і функцій-членів: До членів даних і функцій-членів класу можна отримати доступ за допомогою оператора dot(‘.’) з об’єктом.

Наприклад, якщо ім’я об’єкта – obj, і ви хочете отримати доступ до функції-члена з ім’ям printName(), тоді вам доведеться написати obj.printName() .

Доступ до членів даних

Доступ до загальнодоступних членів даних також здійснюється таким же чином, але об’єкт не може мати прямий доступ до членів приватних даних. Доступ до члена даних залежить виключно від контролю доступу цього члена даних.

Цей контроль доступу надається модифікаторами доступу в Java. Існує три модифікатори доступу: відкритий, приватний і захищений.

Клас - це шаблон або план, який зв'язує властивості та функції сутності. Ви можете помістити всі сутності або об’єкти, що мають подібні атрибути, під одним дахом, відомим як клас. Класи далі реалізують основні концепції, такі як інкапсуляція, приховування даних та абстракція. У Java клас діє як тип даних, який може мати кілька об’єктів або екземплярів типу класу.

Розглянемо в якості прикладу створений нами клас Products:

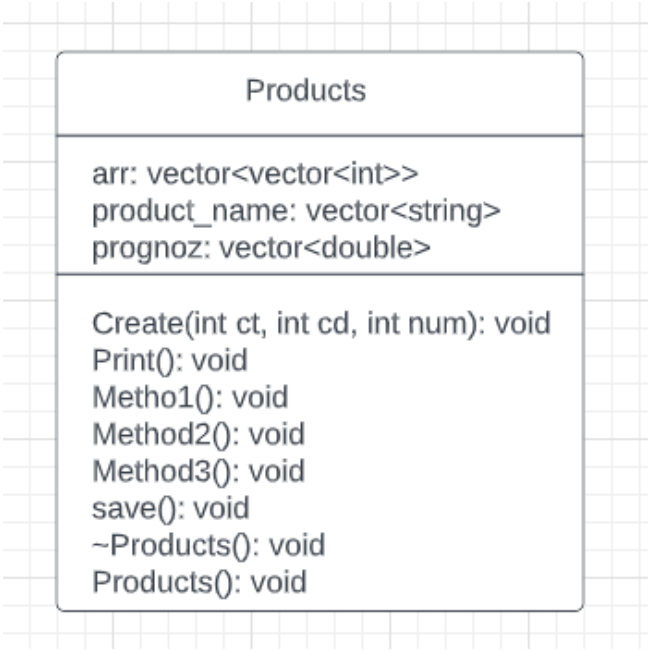

Рис 2.2 – Діаграма класу Products

В ньому є конструктор, деструктор. У private частині зберігаються змінні.

Також в ньому є декілька функцій які зберігаються у public секції.

2.5.2 Опис методів

Відзначимо основні завдання прототипу для детальнішого аналізу та порівняння вибраних інструментів. Серед обмежень, пов'язаних із вибором середовища розробки, варто відзначити використання 2-D графіки та фізики.

Насамперед, прототип гри повинен показати можливості фізичного двигуна та обробки зіткнень. Під час аналізу слід враховувати співвідношення отриманого результату зі складністю його реалізації.

По-друге, нам потрібно продемонструвати роботу з графікою, обробку анімації, ефекти тощо. Для подальшого аналізу кожен прототип буде побудований на тих самих графічних матеріалах.

По-третє, потрібно виділити роботу з логікою гри, її взаємодію зі сценою, предметами. Таким чином, особливу увагу слід приділити аналізу складності реалізації логіки у різних частинах ігрового проекту.

Нарешті, необхідно проаналізувати можливості взаємодії гравця із грою, тобто. підтримка периферії та її обробки, робота з елементами графічного інтерфейсу та ін.

Виходячи з визначення проблеми та обмежень, приступаємо до формування опису прототипу. Спочатку визначимося з жанром майбутніх ігрових проектів. Класичними представниками 2-D графіки є аркадні платформери, рольові ігри та шутери з видом зверху (камера висвітлює сцену зверху). Рольові ігри можна виключити через відсутність можливості продемонструвати роботу фізики та колізій, а також зайвих вимог до логіки та сценарію. Але шутери дають нам можливість гармонійно використовувати кілька пристроїв введення, один для переміщення персонажа, а інший для переміщення прицілу. Вони не прив'язані до складної логіки, мають простий сценарій або не мають і не мають певних вимог для обробки зіткнень, особливо куль, снарядів тощо.

Платформери, як правило, мають спрощене управління, але не обтяжені глибокою логікою, фокусуючись на фізиці та зіткненнях, зосереджуючи увагу гравця на цій частині взаємодії об'єктів. Вертикальне та горизонтальне переміщення.

РОЗДІЛ 3. ЯКІСТЬ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Основні алгоритми для реалізації гри

Завдяки бурхливому розвитку науки інформатики і проникненню їх у різні галузі народного господарства слово " алгоритм " стало найпоширенішим і найбільш вживаним у життєвому плані поняттям для широкого кола фахівців. Більше того, з переходом до інформаційного суспільства алгоритми стають одним з найважливіших факторів цивілізації.

Відомо, що математична теорія алгоритмів склалася зовсім у зв'язку з бурхливим розвитком інформатики та обчислювальної техніки, а виникла надрах математичної логіки на вирішення її проблем. Вона, перш за все, дуже вплинула на світогляд математиків і їх науку.

Тим не менш, взаємовплив теоретичних областей, пов'язаних з обчислювальною технікою, та теорії алгоритмів також, безсумнівно.

Теорія алгоритмів вплинула на теоретичне програмування. Зокрема, велику роль теоретичному програмуванні відіграють моделі обчислювальних автоматів, які, за суттевістю, є обмеженнями тих представницьких обчислювальних моделей, створених раніше у теорії алгоритмів. Трактування програм, як об'єктів обчислення, оператори, що використовуються для складання структурованих програм (послідовне виконання, розгалуження, повторення), прийшли в програмування з теорії алгоритмів. Зворотний вплив виявилося, наприклад, у тому, що виникла потреба у створенні та розвитку теорії обчислювальної складності алгоритмів. Отже, можна сказати, що теорія алгоритмів застосовується у інформатиці, а й у інших галузях знань.

Поняття алгоритму та його властивості

Слово «алгоритм» означає «процес або набір правил, яких слід дотримуватися під час обчислень або інших операцій з вирішення проблем». Тому Алгоритм

відноситься до набору правил/інструкцій, які крок за кроком визначають, як має виконуватися робота, щоб отримати очікувані результати.

Це можна зрозуміти, взявши приклад приготування за новим рецептом. Щоб приготувати новий рецепт, потрібно читати інструкції та кроки та виконувати їх по черзі в заданій послідовності. Отриманий таким чином результат – нове блюдо, приготоване ідеально. Аналогічно, алгоритми допомагають виконати завдання в програмуванні, щоб отримати очікуваний результат.

Розроблений алгоритм не залежить від мови, тобто це прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Оскільки не слід дотримуватись жодних письмових інструкцій, щоб приготувати рецепт, а лише стандартний. Так само не всі письмові інструкції з програмування є алгоритмами. Для того, щоб деякі інструкції були алгоритмом, вони повинні мати такі характеристики:

Чітко і однозначно: алгоритм повинен бути зрозумілим і недвозначним. Кожен із його кроків має бути зрозумілим у всіх аспектах і вести лише до одного значення.

Добре визначені вхідні дані: якщо алгоритм каже приймати вхідні дані, це повинні бути чітко визначені вхідні дані.

Добре визначені результати: алгоритм повинен чітко визначити, який вихід буде отримано, і він також повинен бути чітко визначений.

Скінченність: Алгоритм повинен бути скінченним, тобто він не повинен опинитися в нескінченних циклах або тому подібних.

Можливість: Алгоритм повинен бути простим, загальним і практичним, щоб його можна було виконувати за наявності доступних ресурсів. Він не повинен містити якусь майбутню технологію чи щось таке.

Незалежний від мови: розроблений алгоритм повинен бути незалежним від мови, тобто це повинні бути просто прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Переваги алгоритмів:

- це легко зрозуміти;

- алгоритм — це поетапне представлення рішення заданої задачі;
- в алгоритмі проблема розбивається на менші частини або кроки, отже, програмісту легше перетворити її на справжню програму.

Недоліки алгоритмів:

- написання алгоритму займає багато часу, тому займає багато часу.
- розгалуження та циклічні оператори важко показати в алгоритмах.

Як розробити алгоритм?

Для того, щоб написати алгоритм, необхідні такі речі як передумова:

- задача, яку потрібно вирішити за допомогою цього алгоритму.
- обмеження проблеми, які необхідно враховувати при розв'язуванні задачі;
- вхідні дані, які потрібно зробити для вирішення проблеми;
- результат, якого слід очікувати, коли проблема буде вирішена;
- рішення цієї задачі, в заданих обмеженнях;

Потім алгоритм записується за допомогою вищевказаних параметрів так, щоб він вирішував задачу.

Приклад: Розглянемо приклад, щоб додати три числа та надрукувати суму.

Крок 1: Виконання попередніх умов

Як обговорювалося вище, для того, щоб написати алгоритм, повинні бути виконані його передумови.

Задача, яку потрібно розв'язати за цим алгоритмом: Додайте 3 числа та виведіть їх суму.

Обмеження задачі, які необхідно враховувати при розв'язуванні задачі: числа повинні містити тільки цифри і ніяких інших символів.

Вхідні дані, які потрібно зробити для розв'язання задачі: три числа, які потрібно додати.

Вихід, якого слід очікувати, коли задачу буде розв'язано: сума трьох чисел, взятих як вхідні дані.

Розв'язання цієї задачі за наведених обмежень: Рішення складається з додавання 3 чисел. Це можна зробити за допомогою оператора «+», або порозрядно, або будь-яким іншим способом.

Крок 2: Розробка алгоритму

Тепер давайте розробимо алгоритм за допомогою наведених вище умов:

Алгоритм додавання 3 чисел і виведення їх суми:

СТАРТ

Оголосити 3 цілі змінні num1, num2 і num3.

Візьміть три числа, які потрібно додати, як вхідні дані у змінних num1, num2 та num3 відповідно.

Оголосіть цілу змінну sum, щоб зберегти підсумкову суму трьох чисел.

Додайте 3 числа та збережіть результат у сумі змінної.

Вивести значення змінної суми

КІНЕЦЬ

Крок 3: Тестування алгоритму шляхом його реалізації.

3.1.1 Алгоритмізація гри у жанрі «Tower defense»

Для початку необхідно розробити клас Camera для відображення картини:

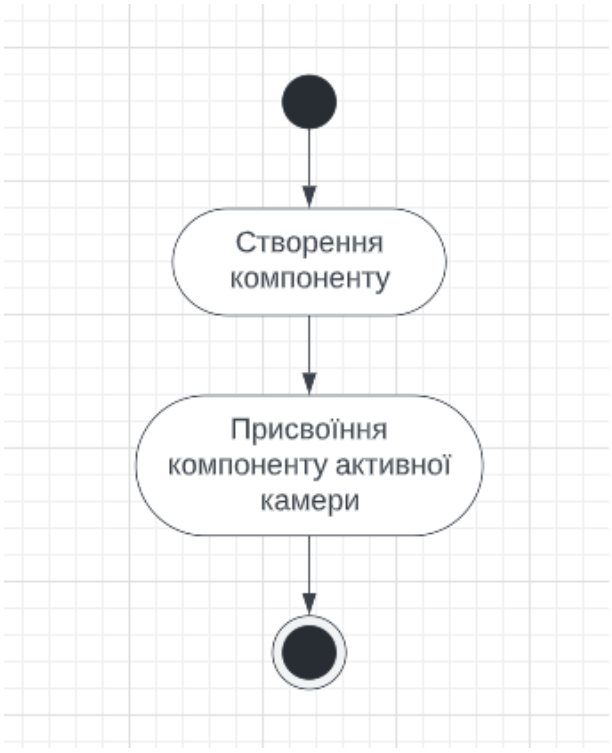

Рис 3.1 – Алгоритм камери

Наступним етапом розробки є створення веж:

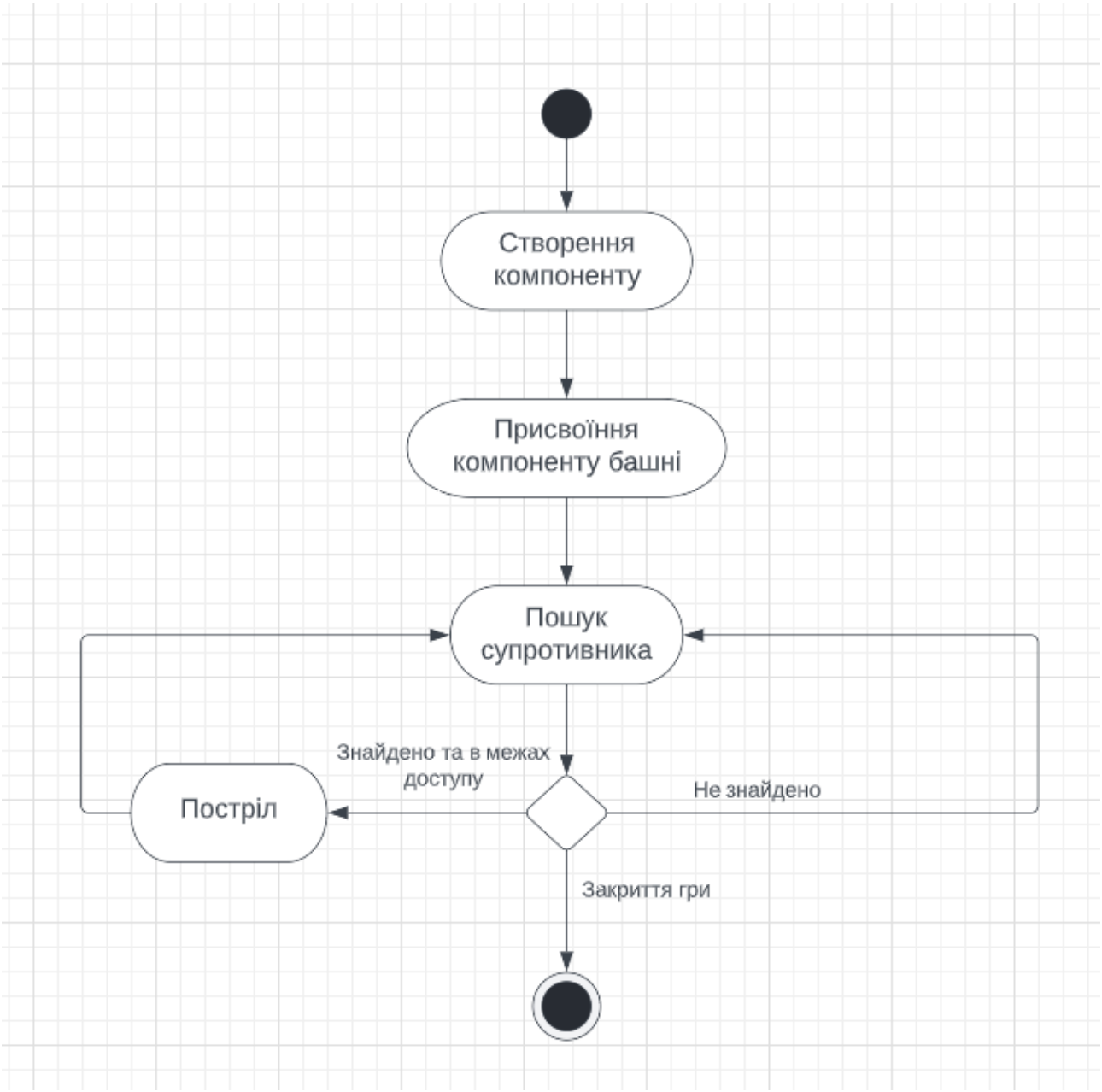

Рис 3.2 – Алгоритм вежі

Для пошук противників була написана наступна функція:

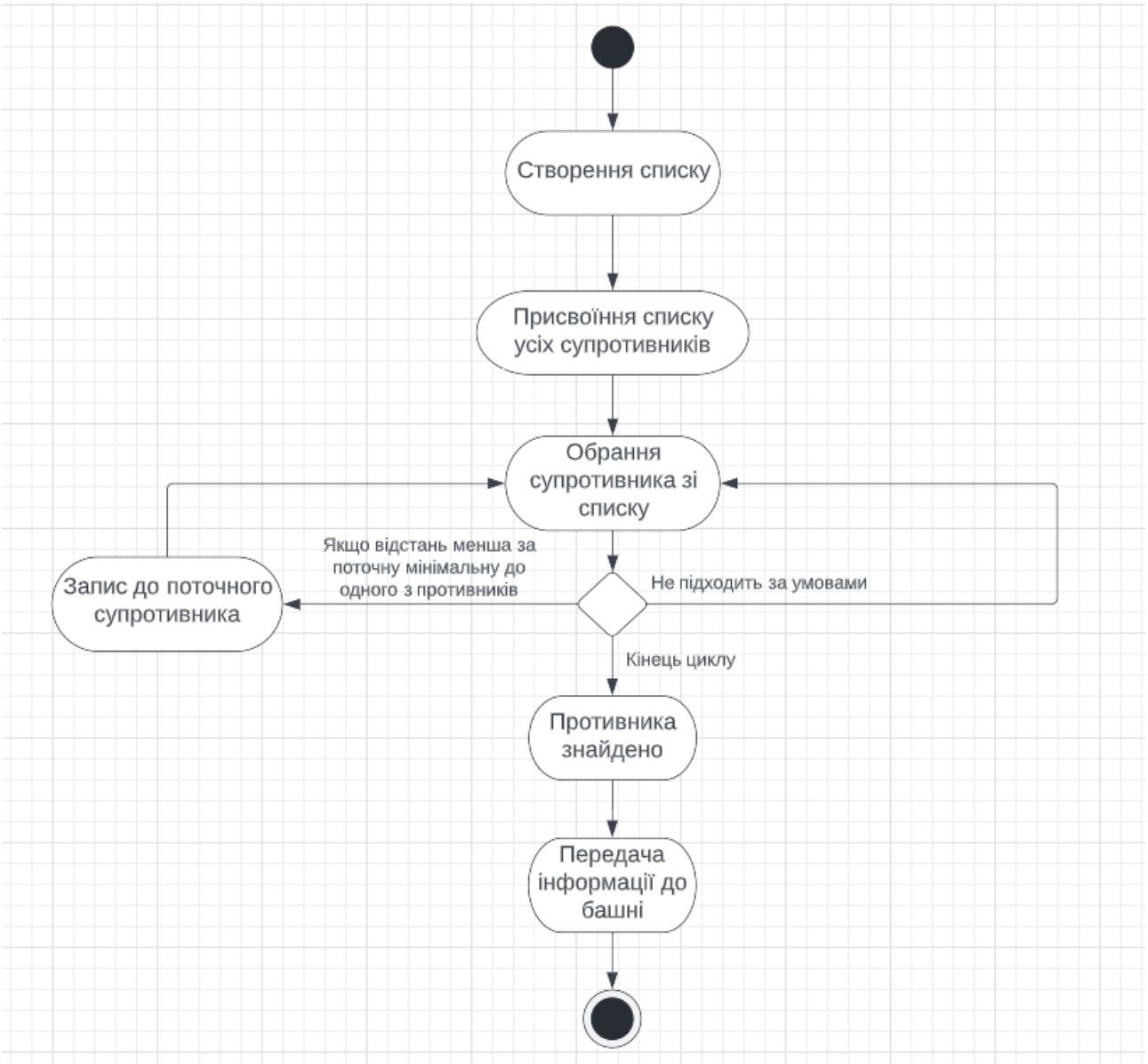

Рис 3.3 – Алгоритм пошуку найближчого супротивника

Наступним кроком було створення ворогів. Задавши їм швидкість та життя наступним чином:

Рис 3.4 – Алгоритм створення ворога

Для смерті ворогів створили наступну функцію:

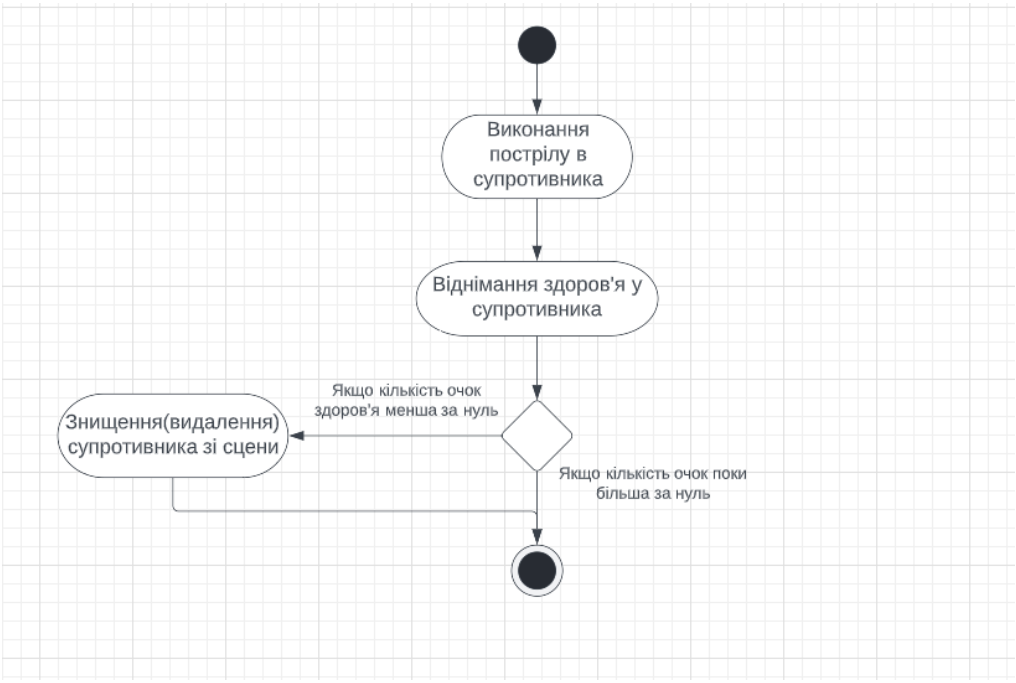

Рис 3.5 – Алгоритм смерті супротивника від пострілу вежі

3.3 Функціонал розробленої гри

Заходячи до гри, перед нами з’являється наступна картина:

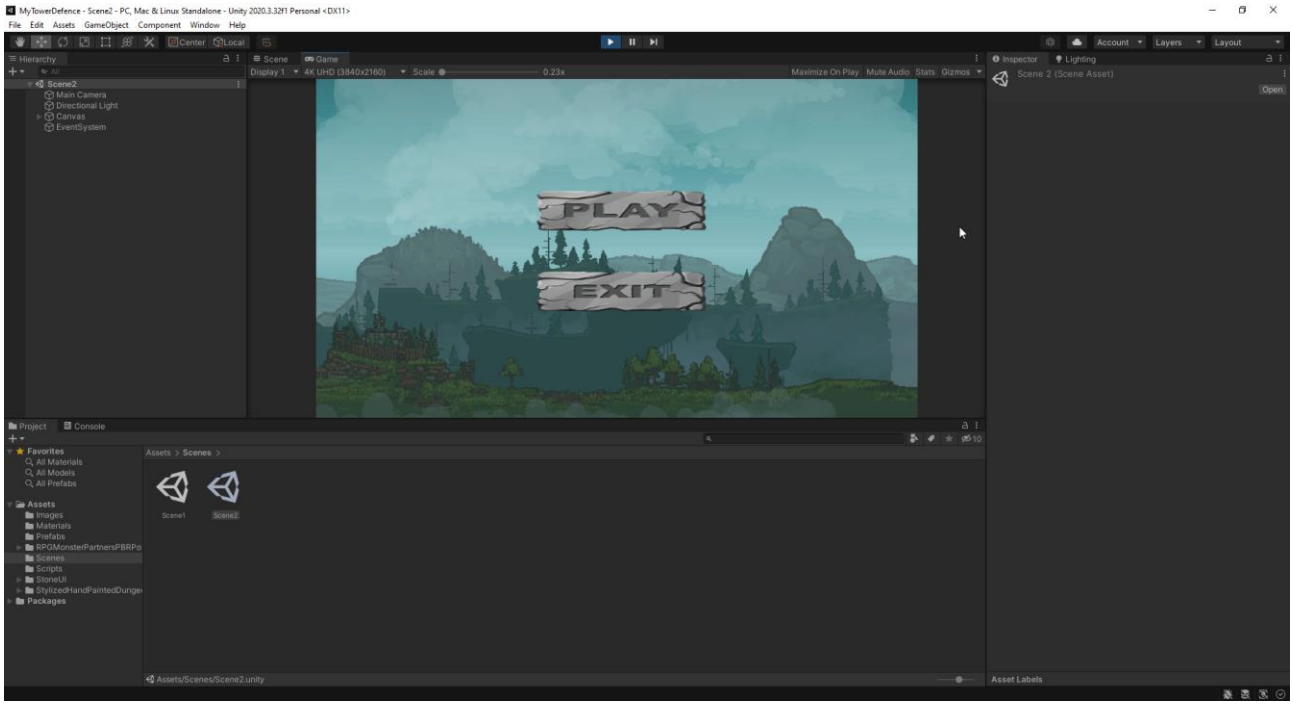

Рис 3.5 – Меню гри

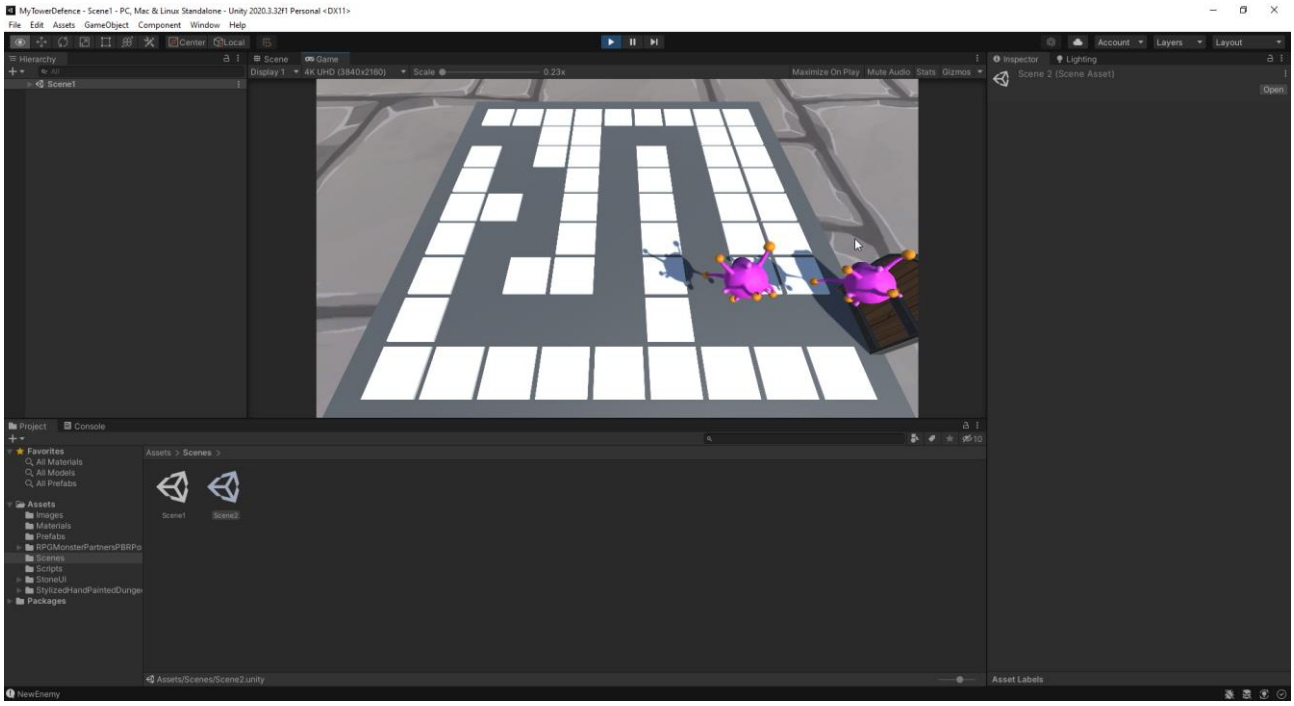

Рис. 3.6 - Початок гри

Після запуску гри починають з’являтися вороги

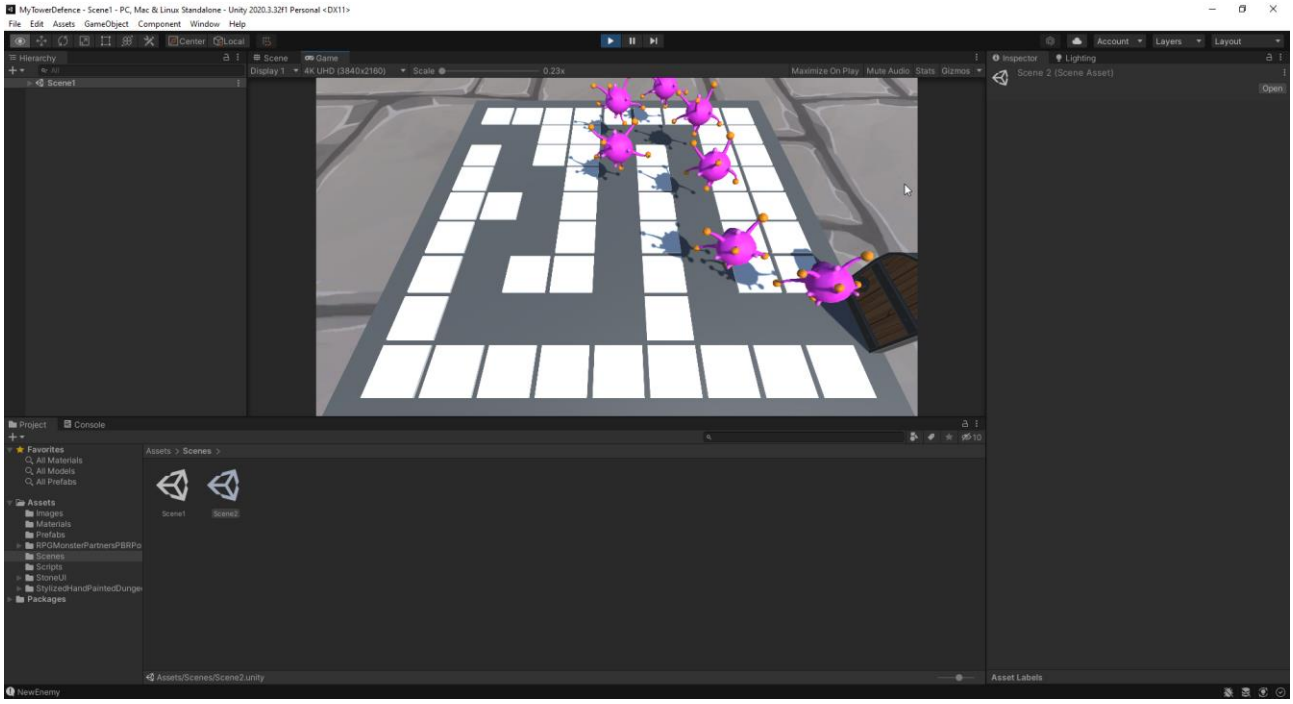

Рис. 3.7 – Поява ворогів

Далі можна обрати місце для розташування вежі

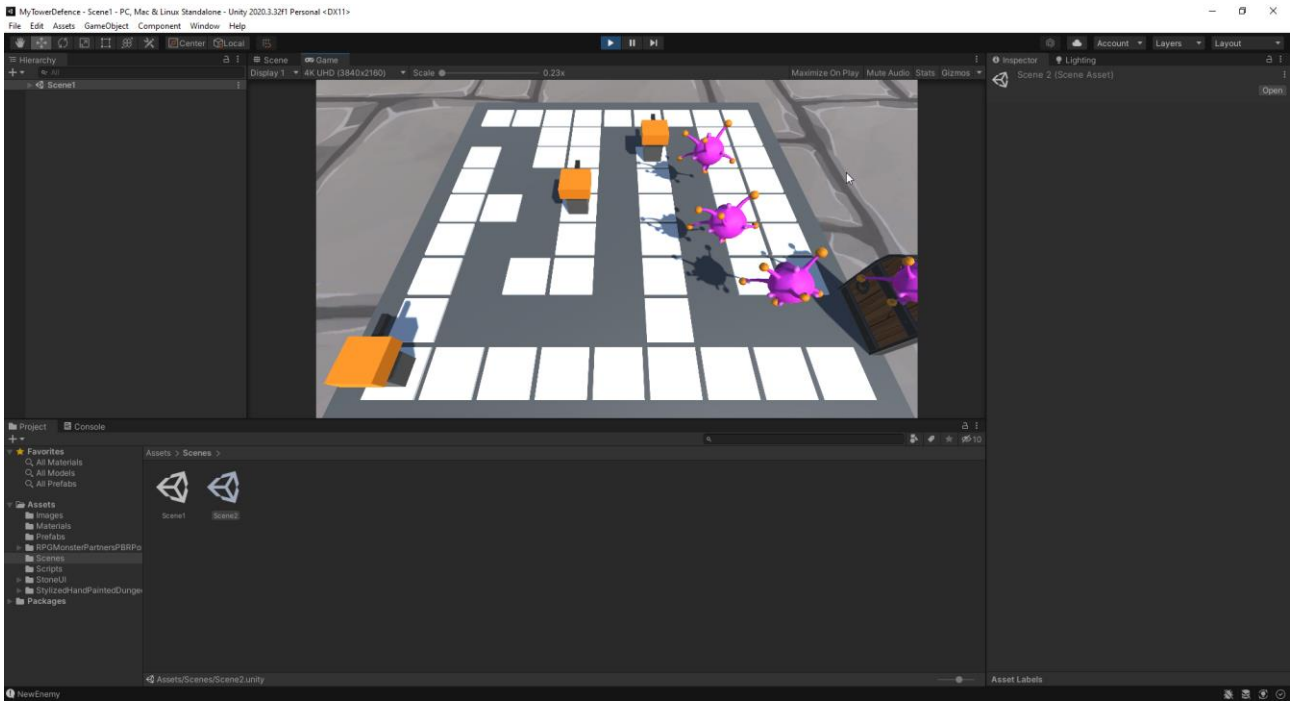

Рис. 3.9 – Додавання вежі

Вежі роблять вистріли по ворогам та знищують їх

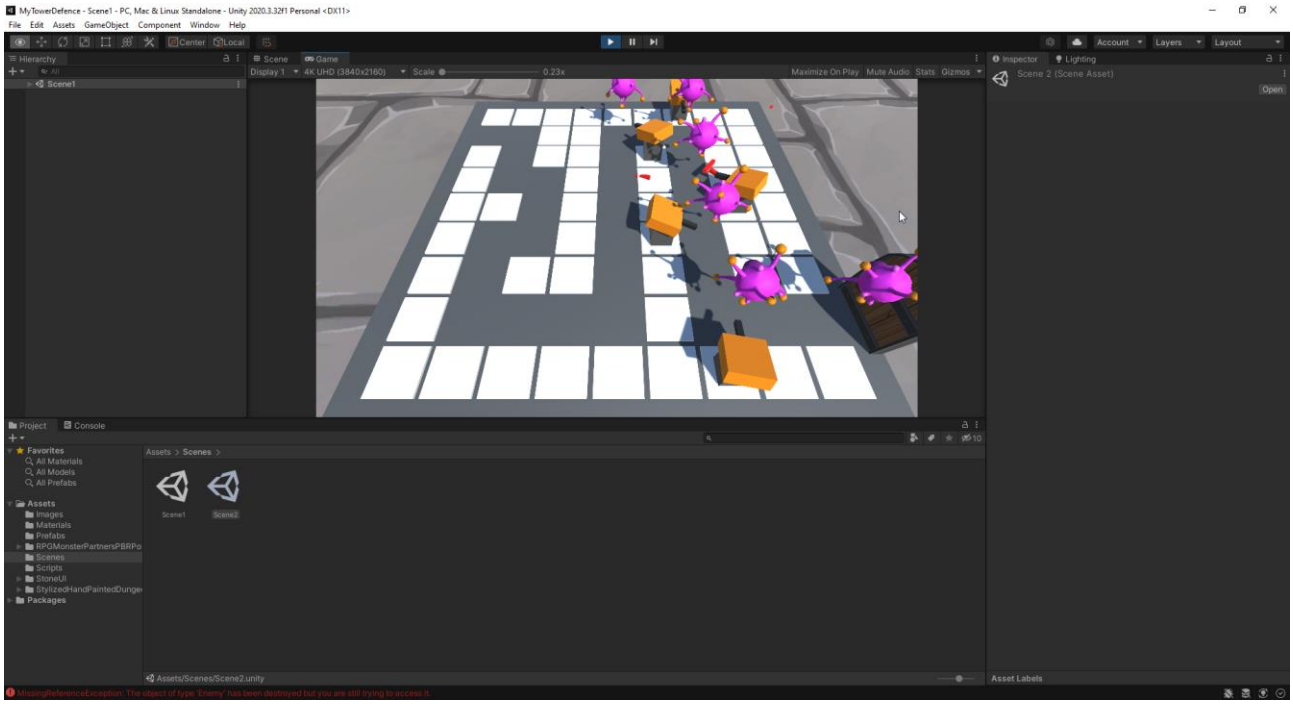

Рис. 3.8 – Вистріли веж

Гра закінчується, коли вороги(кульки) доходять до кінця

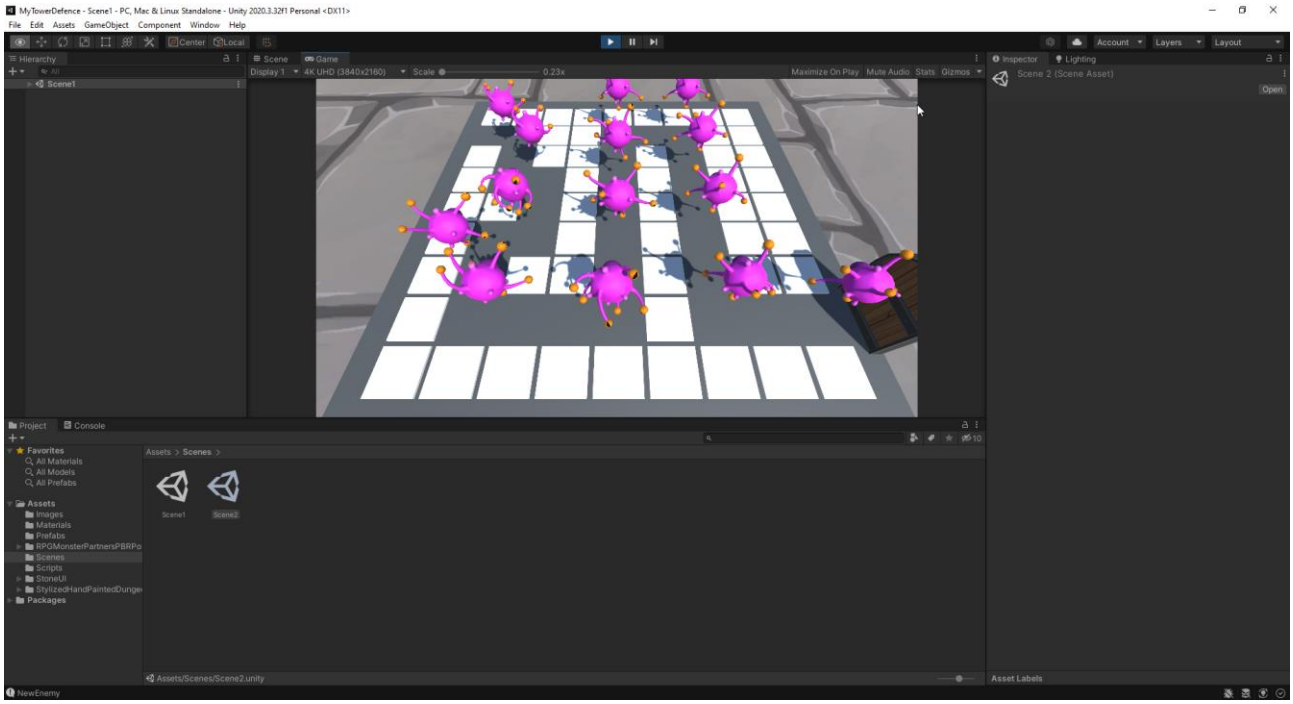

Рис. 3.10 - Кінець гри

3.4 Тестування програмного продукту

Під час тестування, проблеми та баги які виникали одразу ж виправлялися.

Першим чином попробуємо запустити гру без веж, щоб побачити, чи всі кульки можуть дійти до кінця мапи.

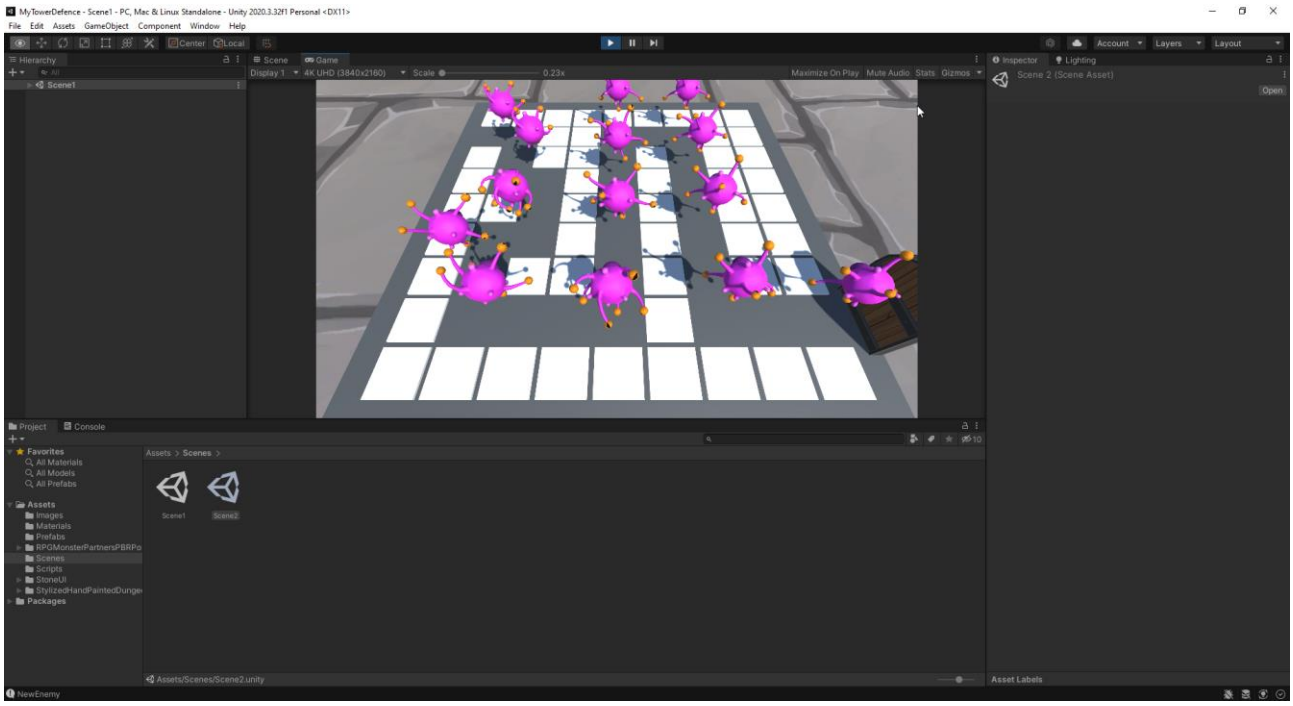

Рис. 3.11 – Тест №1

Перший тест успішно пройдено. Тепер попробуємо запустити гру з 6-ма вежами, щоб побачити, наскільки далеко можуть зайти кульки.

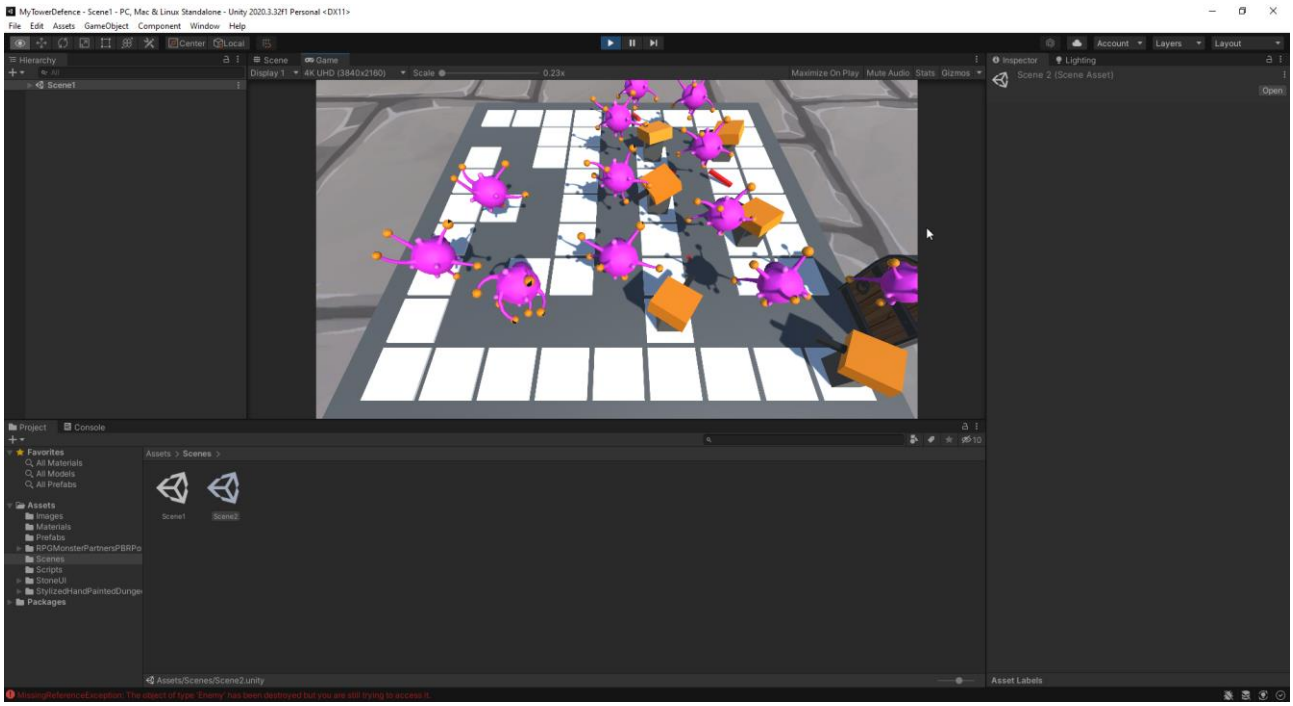

Рис. 3.12 – Тестування гри

Таким чином, на рисунку вище видно, що 6 веж не хватає, щоб зупинити усю потужність кульок. Саме на рисунку видно, що деякі все ж таки пропадають після вистрілу, але поодиноко вони можуть проходити до кінця мапи – верхній лівий куток рисунку. Тому наступним тестуванням буде спроба з 10-ма вежами.

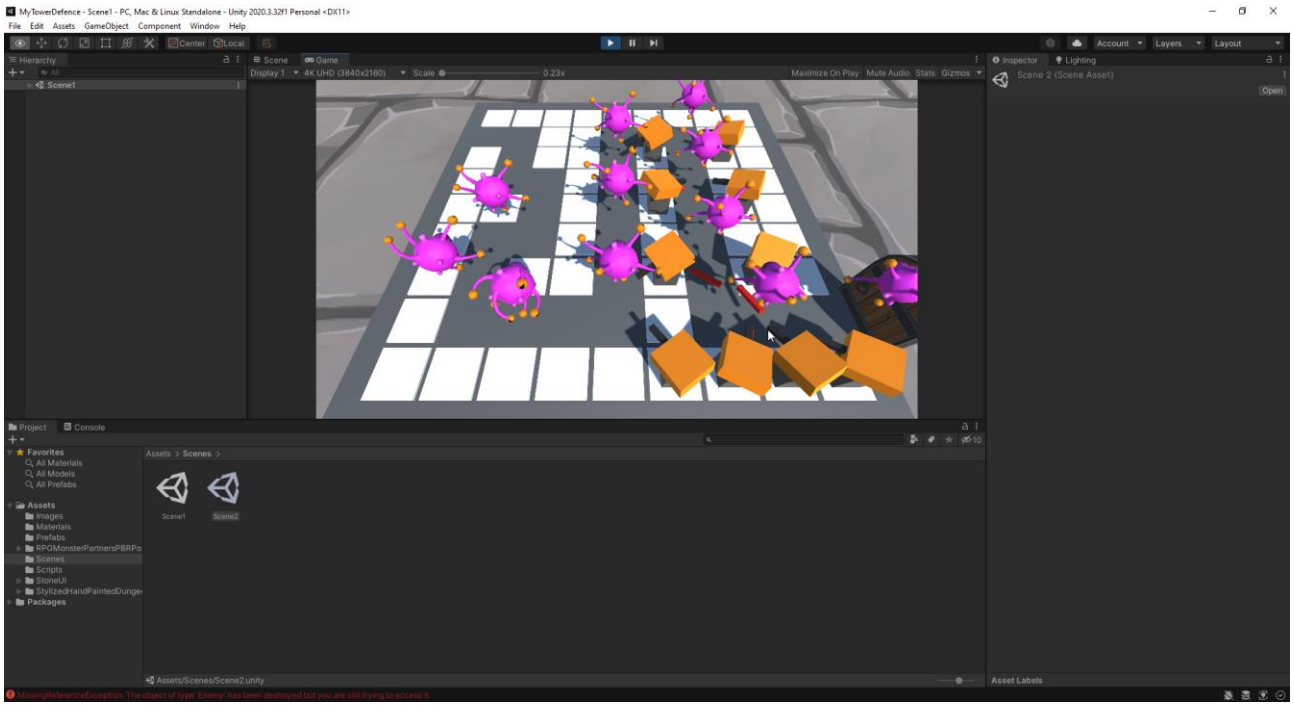

Рис 3.13 – Тестування гри з 10-ма вежами

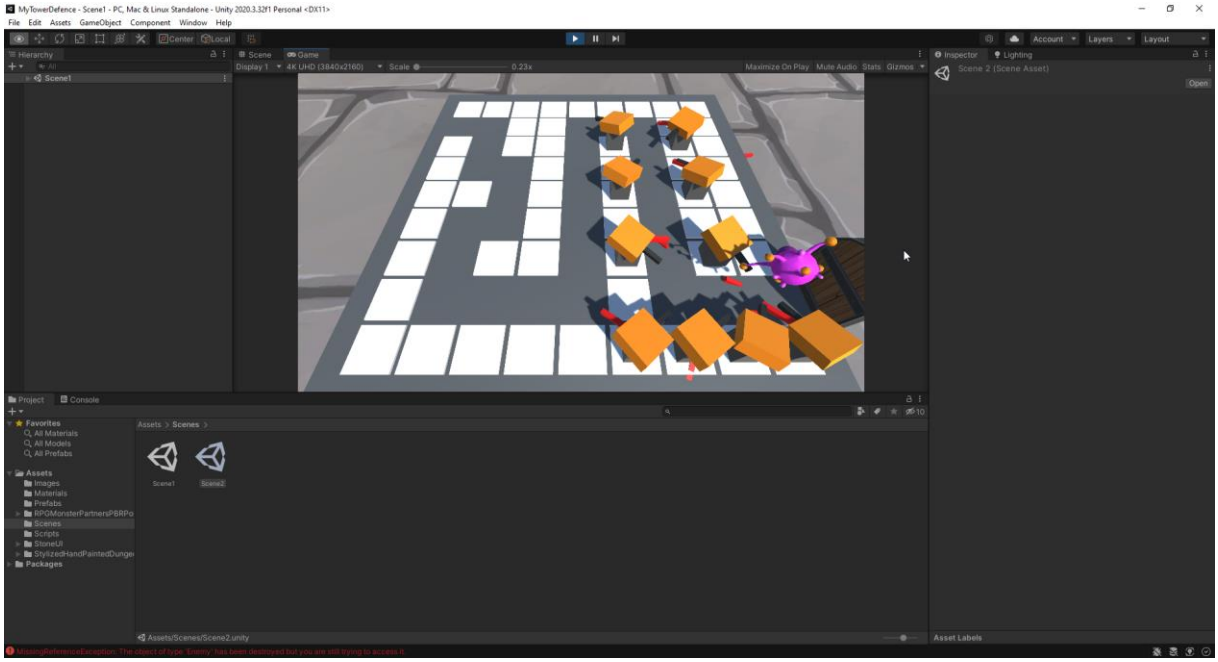

Рис 3.14 Тестування гри з 10-ма вежами

Оскільки гра запускається без веж, то відповідно потрібен час на їх розставляння. Тому на першому рисунку зображено максимальну точку, куди могла спрямувати кулька. Після стабілізації ситуації, коли всі 10 веж працювали на повну потужність, пройшло ~30 секунд, ситуація конкретно змінюється. Тепер кульки не можуть дійти навіть до 5-6 клітинки. Тому можна зробити висновок, що 10 веж вистачає аби подавити хвилю супротивників.

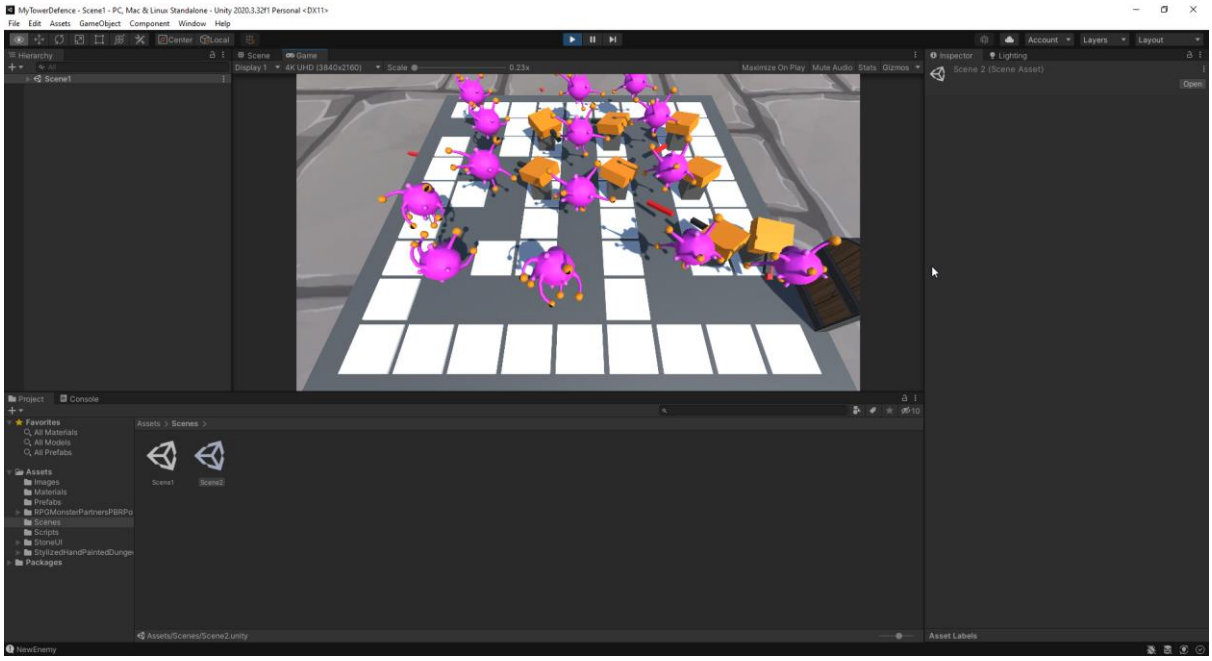

Рис. 3.15 – Тестування гри з 8 вежами

Для перевірки чи залежно від місця вежі буде змінюватись результат було вирішено зробити тест з 8 вежами. Оскільки вежі розташовані таким чином, що вони концентруються на центральному напрямку, то вони не можуть остаточно вбити ворога. У цій ситуації: 4 вежі дивляться за центральним напрямком та 4 інші за правим напрямком. Оскільки, на кожен напрямок цього буде мало, то й виходить, що кульки з мінімальним здоров’ям можуть дістатись кінця мапи. Виходячи з висновків попередніх тестів можна заявити, що потрібно не менше 6 веж хоча б на одному з напрямків, тоді супротив кульок буде подавлений.

ВИСНОВКИ

Робота присвячена розробці ігрового додатку у жанрі «Tower defense», що підвищить комерційну конкуренцію та приведе до покращення майбутніх ігрових механік, ігрових світів цього жанру.

Проведено дослідження котрі обґрунтовують актуальність роботи та наукову новизну. А саме рівень зацікавленості(популярності) ігрового жанру «Tower defense», конкурентоспроможність ринку, проаналізовано переваги та недоліки ігрових застосунків цього жанру.

Враховуючи переваги та недоліки існуючих ігрових додатків, було проаналізовано вимоги та спроектовано гру та її структурні елементи. Спроектований додаток відповідає усім виявленим вимогам.

Проведено аналіз існуючих програмних засобів для розробки додатків для платформ Android та Windows. Таким чином для розробки додатку було обрано мову програмування C# та середовище розробки Unity, що працює також і з середовищем Visual Studio.

Для перевірки правильності роботи додатку та відповідності усім вимогам було створено тест кейси.

Розроблено ігровий додаток у жанрі «Tower defense» з урахуванням усіх виявлених вимог. Розроблений додаток може використовувати кожен користувач, що встановить ігровий застосунок на свою платформу. За допомогою розробленого додатку користувачі можуть користуватись усіма механіками вище згаданого жанру, поєднуючи їх роботу в ігровому світі.

Результати досліджень бакалаврської роботи апробовані на науково-технічній конференції: Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технолоніях», м.Київ: ДУТ, 20 квітня 2022 року.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Єдність у дії. Багатоплатформна розробка на C#. - М .: Петро, 2018 .-- 608 с.

2. Арстанова, Л.Г. Заняття та розваги з дітьми старшого дошкільного віку. Розробка занять, бесід, ігор та розваг на моральні теми / Л.Г. Арстанов. - М .: Учитель, 2017 .-- 324 с.

3. Архангельська, М. Д. Діловий етикет, або гра за правилами / М. Д. Архангельська. - М .: Ексмо, 2015 .-- 160 с.

4. Вакуленко, Ю.А. Весела граматика. Розробка занять, завдань, ігор / Ю.А. Вакуленко. - М .: Учитель, 2017 .-- 780 с.

5. Джейсон, Фінканон Flash Ads. Розробка мікросайтів, рекламних ігор та фірмових додатків за допомогою Adobe Flash / Finkanon Jason. - М .: Група Рід, 2012 .-- 945 с.

6. Джейсон, Фінканон Flash Ads. Розробка мікросайтів, рекламних ігор і фірмових додатків з pom / Фінканон Джейсон. - М .: ТОВ РІД ГРУП Москва, 2012 .-- 288 с.

7. Любанова, Т.П. Бізнес-план: досвід, проблеми. Зміст бізнес-плану, приклад розробки / Т.П. Любанова, Л.В. Мясоедова, Т.А. Грамотенко та ін .. - М .: Пріор, 2012 .-- 204 с.

8. Паласіос, Хорхе Юніті 5.x. Програмування штучного інтелекту в іграх / Хорхе Паласіос. - М .: ДМК Прес, 2016 .-- 849 с.

9. Платов В.Я. Ділові ігри. Розробка, організація, впровадження. Підручник / В.Я. Платова. - М .: Профиздат, 2010 .-- 192 с.

10. Таран, В.А. Чи легко грати на біржі?! / В.А. ОЗП. - М .: СПб: Петро, 2016 .-- 272 с.

11. Тарп, Ван; Бертон Д.Р. Стратегії обміну. Ігри без ризику / Тарп, Ван; ЛІКАР. Бартон, С. Саггеруд. - М .: СПб: Петро, 2010 .-- 400 с.

12. Фінні, К. 3D-ігри. Все про розвиток (+ CD-ROM) / К. Фінні. - М .: Біном. Лабораторія знань, 2011 .-- 976 с.

13. Фінні, К. 3D-ігри: Все про розробку (+ CD-ROM) / К. Фінні. - М .: Біном. Лабораторія знань, 2015 .-- 133 с.

14. Хорхе, Palacios Unity 5.x. Програмування штучного інтелекту в іграх. Путівник / Паласіос Хорхе. - М .: ДМК Прес, 2017 .-- 427 с.

15. Шабельникова, Є.Ю. Англійська мова. Навчання дітей 5-6 років. Розробка занять, лінгвокультурологічного матеріалу, заходів, ігор / Є.Ю. Шабельникова. - М .: Учитель, 2015 .-- 557 с.

16. Язєв Ю.В. Магія моменту обертання. Мистецтво розробки ігор на движку Torque 2D, включає опис версій 3.2 і 3.3 / Ю. Язєв. - М .: Солон-Прес, 2016 .-- 448 с.

17.Heimburg, E. Localizing MMORPGS, Perspectives on Localization Heimburg, E. – Amsterdam. : John Benjamins Publishing Company, 2006. – 241 p.

18.Jakobson R. On linguistic aspects of translation / Jakobson R. – London. : Gregg Press, 1959. – 341 p.

19.Klinberg G. Children's Fiction in the Hands of the Translators / Klinberg G. – Munchen. : CWK Gleerup, 1986. – 481 p.

20.Mangiron C. Game localisation: unleashing imagination with «restricted» translation [Electronic resource]. – Mode of access : www.jostrans.org/issue06/art_ohagan.php

21.Maxwell-Chandler H. The Game Localization Handbook / Maxwell-Chandler H. – NY. : Charles River Media, 2005. – 348 p.

22.Mayoral R. Concept of constrained translation. non-linguistic perspectives of translation / Mayoral R. – Boston. : Meta, 1988. – 591 p.

23.O'Hagan M. Game Localization: Translating for the global digital entertainment industry / O'Hagan M. – Amsterdam. : John Benjamins Press, 2014. – 256 p.

24.O'Hagan M. Video games as a new domain for translation research / O'Hagan M. – Amsterdam. : John Benjamins Press, 2013. – 243 p.

25.Palumbo G. Key Terms in Translation Studies / Palumbo G. – London. : Continuum, 2009. – 317 p.

26.Paquin, R. Translator, Adapter, Screenwriter. Translating for the audiovisual / Paquin R. – London. : Translation Journal,1998. – 132 p.

27.Remael A. Audiovisual translation / Remael A. – Amsterdam. : John Benjamins Publishing Company, 2010. – 18 p. 76

28.Sainsbury L. The Postmodern Carnival of Children's Literature: Necessary Playgrounds and Subversive Space in the Protean Body of Children's Literature / Sainsbury L. – Surrey. : University Press of Surrey, 1998. – 234 p.

29.Sánchez, J. L. G. Playability: analysing user experience in video games. Behaviour & Information Technology [Electronic Resource]. – Mode of access : http://dx.doi.org/10.1080/0144929X.2012.710648

30.Schäler, R. Linguistic resources and localisation / Schäler, R. – Limerick. : John Benjamins Press, 2008. – 615 p.

31.Tan E. S. The game experience / Tan E. S. – Amsterdam. : Elsevier, 2008 – 415 p.

32.Thayer A. Localization of digital games: The process of blending for the global games market. Technical Communication / Thayer A. – Oxford. : Lancer Books, 2004. – 319 p.

33.Tonkin H. The Translator as Mediator of Culture / Tonkin H. – Amsterdam. : John Benjamins Publishing Company, 2010. – 318 p.

34.Top 100 countries by game revenues [Electronic resource]. – Mode of access: https://newzoo.com/insights/rankings/top-100-countries-by-game-revenues/

35.Translation. An Advanced Resource Book / Munday J. – NY. : Tuttle publishing, 2004. – 341 p.

ДОДАТКИ

Додаток А(CameraScript.cs)

```
using UnityEngine;
using System.Collections;

public class CameraScript : MonoBehaviour
{
    Camera camera;
    TowerPlace currTP;

    void Start()
    {
        camera = GetComponent<Camera>();
    }

    void Update()
    {
        // Випускаємо промінь з камери туди, де зараз знаходиться курсор миші.
Ray ray = camera.ScreenPointToRay(Input.mousePosition);

        // Змінна для зберігання результату перетину променя
        RaycastHit hit;

        // Якщо промінь перетнувся з якимось об'єктом і цей об'єкт має тег
"TowerPlace"
        if (Physics.Raycast(ray, out hit) && hit.collider.gameObject.tag ==
"TowerPlace")
        {
            bool isMouseDown = Input.GetMouseButtonDown(0);
            bool isMouseUp = Input.GetMouseButtonUp(0);
            TowerPlace                          tp                          =
hit.collider.gameObject.GetComponent<TowerPlace>();

            if (isMouseDown || isMouseUp)
            {
                if (isMouseDown)
                {
                    tp.MouseDown();
                }
                else if (isMouseUp)
                {
                    tp.MouseUp();
                }
            }
            else
            {
                if (currTP != tp)
                {
                    if (currTP != null)
                        currTP.MouseLeave();

                    tp.MouseEnter();
                    currTP = tp;
                }
            }
        }
        else
        {
            if (currTP != null)
                currTP.MouseLeave();
```

```
                currTP = null;
            }
        }
    }
```

Додаток Б(Tower.cs)

```
using UnityEngine;
using System.Collections;
using System;

public class Tower : MonoBehaviour
{
    public float FindRadius = 2f;
    public float TimeShoot = 1f;
    public GameObject bullet;

    Enemy enemy;
    Transform towerHead;
    float timerShoot = 0f;

    void Start()
    {
        towerHead = transform.Find("Head");
    }

    void Update()
    {
        if (enemy == null)
        {
            // Шукаємо супротивника
            FindEnemy();
        }
        else
        {
            // Повертаємо гармату у бік супротивника і робимо постріл за
пострілом
            towerHead.LookAt(enemy.transform);

            // Виробляємо постріл
            Shoot();

            float dist = Vector3.Distance(enemy.transform.position,
transform.position);
            if (dist > FindRadius)
                enemy = null;
        }
    }

    // Постріл
    private void Shoot()
    {
        timerShoot -= Time.deltaTime;

        if (timerShoot <= 0)
        {
            timerShoot = TimeShoot;
            GameObject obj = (GameObject)Instantiate(bullet,
towerHead.transform.position, towerHead.transform.rotation);
            Bullet b = obj.GetComponent<Bullet>();
```

```
                b.Enemy = enemy;
            }
        }

        // Пошук супротивника
        private void FindEnemy()
        {
            var enemies = GameObject.FindObjectsOfType<Enemy>();
            float min = FindRadius;
            Enemy minEnemy = null;

            foreach (var e in enemies)
            {
                float dist = Vector3.Distance(e.transform.position,
transform.position);

                if (dist <= min)
                {
                    min = dist;
                    minEnemy = e;
                }
            }

            enemy = minEnemy;
        }
    }
```

Додаток В(Bulet.cs)

```
using UnityEngine;
using System.Collections;

public class Bullet : MonoBehaviour
{
    public float Speed = 0.5f;
    public float TimeLife = 1f;
    public Enemy Enemy;
    public float Damage = 25f;

    float timerLife = 0f;

    void Start()
    {
        timerLife = TimeLife;
    }

    void Update()
    {
        timerLife -= Time.deltaTime;

        Vector3 dir = Enemy.transform.position - transform.position;
        float _speed = Speed * Time.deltaTime;

        if (timerLife <= 0)
        {
            // Час життя кулі закінчився
            timerLife = TimeLife;
            Destroy(gameObject);
        }
        else if (dir.magnitude <= _speed)
        {
            // Куля влучає в ціль
            Enemy.SetDamage(Damage);
```

```
            Destroy(gameObject);
            return;
        }

        transform.Translate(new Vector3(0, 0, _speed));
    }
}
```

Додаток Г(Enemy.cs)

```
using UnityEngine;

public class Enemy : MonoBehaviour
{
    public float speed = 2f;
    public float MaxLife = 100f;

    private Transform waypoints;
    private Transform waypoint;
    private int waypointIndex = -1;
    private float life;

 void Start()
    {
        waypoints = GameObject.Find("WayPoints").transform;
        NextWaypoint();

        life = MaxLife;
    }

 void Update()
    {
        Vector3 dir = waypoint.transform.position - transform.position;
        dir.y = 0;

        float _speed = Time.deltaTime * speed;
        transform.Translate(dir.normalized * _speed);

        if (dir.magnitude <= _speed)
            NextWaypoint();
    }

    void NextWaypoint()
    {
        waypointIndex++;

        if (waypointIndex >= waypoints.childCount)
        {
            Destroy(gameObject);
            return;
        }

        waypoint = waypoints.GetChild(waypointIndex);
    }

    public void SetDamage(float value)
    {
        life -= value;

        if (life <= 0)
            Destroy(gameObject);
    }
}
```

Додаток Д(Spawner.cs)

```
using UnityEngine;
using System.Collections;

public class Spawner : MonoBehaviour
{
    public GameObject spawnObject;
    public float spawnTime = 1f;

    private float timer = 0;

 void Start()
    {

 }

 void Update()
    {
        timer -= Time.deltaTime;

        if (timer <= 0)
        {
            Instantiate(spawnObject, transform.position, transform.rotation);

            timer = spawnTime;
        }
    }
}
```

Додаток Е(TowerPlace.cs)

```
using UnityEngine;
using System.Collections;

public class TowerPlace : MonoBehaviour
{
    // Чи можна будувати?
    bool isCanBuild = true;

    public GameObject tower;

 void Start()
    {

 }

 void Update()
    {

 }

    // Клавіша миші натиснута, курсор знаходиться над об'єктом
    public void MouseDown()
    {
        if (isCanBuild)
        {
            isCanBuild = false;
            Instantiate(tower, transform.position, transform.rotation);
            GetComponent<Renderer>().material.color = new Color(1, 1, 1);
        }
```

```
        }

        // Клавішу миші відпущено, курсор знаходиться над об'єктом
        public void MouseUp()
        {

        }

        // Якщо курсор миші опинився над об'єктом
        public void MouseEnter()
        {
            if (isCanBuild)
                GetComponent<Renderer>().material.color = new Color(1, 0, 0);
        }

        // Якщо курсор миші більше не знаходиться над об'єктом
        public void MouseLeave()
        {
            if (isCanBuild)
                GetComponent<Renderer>().material.color = new Color(1, 1, 1);
        }
    }
```

Додаток Є(Menu.cs)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
    public void OnPlayButtonClick() {
        SceneManager.LoadScene("Scene1");
    }
    public void OnExitButtonClick() {
        Application.Quit();
    }
}
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«Розробка гри «OutDef» у жанрі tower defense на Unity C#»

Виконав студент 5 курсу
Групи ППЗ-51
Мягких Микита Олександрович
Керівник роботи
К.т.н., доцент Негоденко О. В.

Київ – 2022

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- *Об'єкт дослідження* – процес розробки гри у жанрі «Tower defense»
- *Предмет дослідження* – технології розробки ігрового програмного забезпечення для пристроїв на базі Android платформи.
- *Мета роботи* – вдосконалення ігрових модулів додатку за рахунок використання рушія Unity.

Актуальність теми. Перші відеоігри з’являлися на комп’ютери та приставки. Та й зараз при слові «відеогра» на думку спадає саме комп’ютерна гра. Але останнім часом мобільні ігри набирають шалену популярність. Це пов’язано з тим, що в останні роки спостерігається зростання попиту на мобільні пристрої, через доступність всіх можливостей, включаючи вихід в Інтернет, не тільки через стаціонарні ПК, а й через планшети, смартфони і звичайні телефони. Потужний комп'ютер для роботи потрібен далеко не всім, а ось мобільний пристрій, який займає мало місця, завжди з вами і все може – зовсім інша справа.

АНАЛОГИ

Kingdom Rush	**OutDef**
Переваги: • *Сюжет* • *Графічна оболонка* • *Безкоштовність* • *Простий інтерфейс*	Переваги: • Безкоштовність • Простий інтерфейс • Легкість ігрових механік • Можливість гнучкої реалізації схем розташування
Недоліки: • *Складні механіки для початківців* • *Відсутність української мови* • *Однотипність рівнів* • *Присутній дестабілізуючий донат*	Недоліки: • Однотипність геймплею

3

ТЕХНІЧНІ ЗАВДАННЯ

1. Створити базові класи/методи для реалізації механіки жанру «Tower defense».
2. Створити графічні 3Д моделі для гри, оптимізувати їх та додати до них анімацію.
3. Розробити основну сцену гри, з'єднати об'єкти сцени з реалізованими методами та протестувати роботу їх з різним розміщенням веж.
4. Спроектувати мапу гри, створити ключові точки ворогів, розробити меню та зв'язати його з основною сценою.
5. Остаточне тестування програмного продукту.

4

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Unity – основний ігровий рушій в розробці

C# - мова програмування, за допомогою якої створюються скріпти в ігровому рушію

Visual Studio – середовище для написання коду, використовується як основне в ігровому рушію

5

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

Рис 3.1 – Алгоритм камери

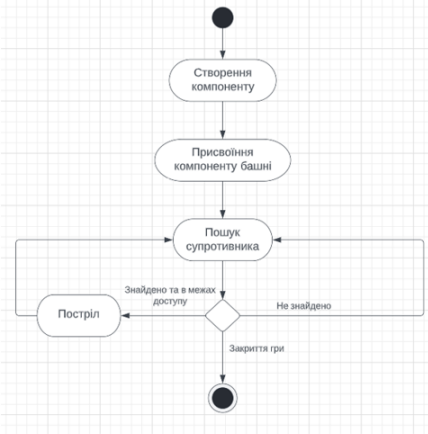

Рис 3.2 – Алгоритм вежі

Рис 3.3 – Алгоритм пошуку найближчого супротивника

МЕТОДИ ТА КЛАСИ ПРОГРАМИ

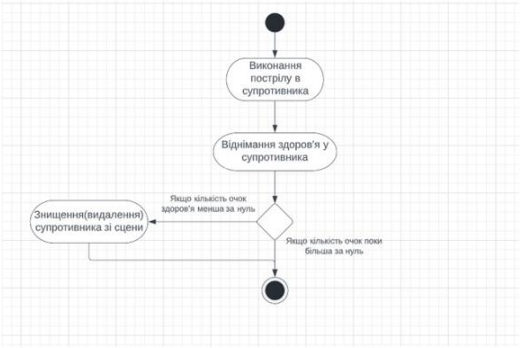

Рис 3.5 – Алгоритм смерті супротивника від пострілу вежі

Рис 3.4 – Алгоритм створення ворога

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технолоніях», м.Київ: ДУТ, 20 квітня 2022 року.

8

ВИСНОВКИ

1. Проведено дослідження котрі обґрунтовують актуальність роботи та наукову новизну. А саме рівень зацікавленості(популярності) ігрового жанру «Tower defense», конкурентоспроможність ринку, проаналізовано переваги та недоліки ігрових застосунків цього жанру.
2. Враховуючи переваги та недоліки існуючих ігрових додатків, було проаналізовано вимоги та спроектовано гру та її структурні елементи. Спроектований додаток відповідає усім виявленим вимогам.
3. Проведено аналіз існуючих програмних засобів для розробки додатків для платформ Android та Windows. Таким чином для розробки додатку було обрано мову програмування C# та середовище розробки Unity, що працює також і з середовищем Visual Studio.
4. Для перевірки правильності роботи додатку та відповідності усім вимогам було створено тест кейси.
5. Розроблено ігровий додаток у жанрі «Tower defense» з урахуванням усіх виявлених вимог. Розроблений додаток може використовувати кожен користувач, що встановить ігровий застосунок на свою платформу. За допомогою розробленого додатку користувачі можуть користуватись усіма механіками вище згаданого жанру, поєднуючи їх роботу в ігровому світі.

Робота присвячена розробці ігрового додатку у жанрі «Tower defense», що підвищить комерційну конкуренцію та приведе до покращення майбутніх ігрових механік, ігрових світів цього жанру.

ДЯКУЮ ЗА УВАГУ!