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 «3PER» в жанрі Tower Defense на Unity"**

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

________________ Мурзін А.О
(прізвище та ініціали)

Керівник ________ Дібрівний О.А
(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

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
Мурзін Антон Олександрович

(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка гри «ЗPER» в жанрі Tower Defense на Unity

Керівник роботи Дібрівний Олесь Андрійович доктор філософії ,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 16 лютого 2022 року №22.

2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи:
   3.1 Технічна документація
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1 Розробка комп'ютерної гри в жанрі Tower Defense використовуючи мову С# за допомогою ігрового двигуна Unity та середу розробки Visual Studio
   4.2 Опис предметної області визначення алгоритмів та функцій проекту
   4.3 Аналіз існуючих ігор

4.4 Дослідження двигунів
4.5 Визначення функціоналу та алгоритмів проекту

5. Перелік графічного матеріалу:
5.1 Діаграма класів
5.2 Use case діаграма

6. Дата видачі завдання «11» квітня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір джерел інформації	04.04.2022	
2	Вимоги до встановленого додатку	04.04.2022-25.04.2022	
3	Оцінка якості тестування до систем	25.03.2022-06.05.2022	
4	Вступ,висновки,реферат + презентація	06.05.2022-13.05.2022	
5	Перевірка на плагіат + Предзахист	16.05.2022- 01.0620.22	
6	Захист роботи	02.06.2022-21.06.2022	
7	Випуск	30.06.2022	

Студент __________ Мурзін А.О
(підпис) (прізвище та ініціали)

Керівник роботи __________ Дібрівний О.А
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 54 с., 38 рис., 18 джерел.

РОЗРОБКА ІГОР, UNITY, TOWER DEFENSE, ГРА, ІГРОВИЙ ДВИГУН

Об'єкт дослідження — підвищення функціональності та продуктивності ігор на основі двигуна Unity.

Предмет дослідження – технології проектування та створення гри на основі двигуна Unity.

Мета роботи – підвищення ефективності розробки гри в жанрі Tower Defense.

Методи дослідження – метод дедукції та індукції, метод аналізу, ітераційний метод.

Для досягнення поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз існуючих додатків, таких як CityWars TD та TumbleWeed;
2. Проаналізувати технічні засобів для розробки додатку та вибір оптимальних рішень для розробки додатку
3. На підставі проведених досліджень розробити гру в жанрі Tower Defense
4. Протестувати додаток на відповідність поставленим завданням.

В роботі виконано аналіз існуючих застосунків для операційної системи Windows

Проаналізовано можливості ігрового двигуна Unity. Розроблено логіку додатку та зручність користування для користувачів.

Галузь використання – даний продукт може бути використаний у комерційних цілях.

Зміст

ВСТУП 4

1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ІГОР 6

1.1 Основні терміни та поняття 6

1.2 Історія розвитку ігор 13

1.3 Основні технології створення ігор 21

2 СИСТЕМИ СТВОРЕННЯ ІГОР 23

2.1 Апаратні вимоги до системи для розробки ігор 23

2.2 Основні переваги та недоліки систем розробки ігор 28

2.3 Основні переваги та недоліки Unity 30

3 ПРАКТИЧНІ АСПЕКТИ СТВОРЕННЯ ІГОР НА UNITY 32

3.1 Аналоги гри «3PER» в жанрі Tower Defense 32

3.2 Постановка ТЗ 33

3.3 Основні можливості системи unity 36

3.4 Розробка гри «3PER» в жанрі Tower Defense на Unity 37

3.5 Тестування гри 54

Висновки 59

Перелік використаної літератури 60

ВСТУП

Темою роботи є розробка гри «ЗPER» в жанрі Tower Defense на Unity.

Актуальність дослідження. Сьогодні ринок ігр переживає сильний підйом, так загальний обсяг ігрової промисловості за підсумками дев’яти місяців 2020 року склав 174,9 мільярда доларів (на 19,6% вище, ніж у 2019 році). Майже половина (49%) прийшла на ігри для смартфонів і планшетів - їх обсяг виріс на 25,6% і склав 86,3 млрд. дол. ПК-ігри зайняли 21,4% ринку з 37,4 мільярда доларів (+ 6,2%), а консольні - 29% ринку і 51,2 мільярда. -

Ці дані кажуть про те, що комп’ютерні ігри є популярними в середовищі різних соціальних груп, а також те, що ігри стали звичайною частиною багатьох людей.

При цьому слід зауважити, що найбільш і таким, що найбільше розвивається є ринок мобільних ігор, тому розробка гри «ЗPER» в жанрі Tower Defense на Unity є досить актуальною.

Ступінь вивчення проблеми: Сьогодні існує досить велика кількість різноманітних ігор різних жанрів для різних платформ, але найбільш динамічним є ринок мобільних ігор, а отже існує досить велика кількість інструментів розробки для мобільних пристроїв. Однією з поширених систем є система Unity. Ця система є міжплатформеною і дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, 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.

З приводу розробки додатків на Unity написано досить велику кількість посібників, статей та інших мтаеріалів, тому можна сказати, що ступінь вивчення проблеми розробки ігор на Unity є досить серйозною.

Об’єктом дослідження є розробка гри на Unity в жанрі Tower Defense.

Предметом роботи є гра «ЗPER» в жанрі Tower Defense на Unity.

Мета роботи - опис та систематизація інформації щодо розробки ігор на Unity на прикладі розробки гри «ЗPER» в жанрі Tower Defense.

Методи дослідження – метод теорії інформації, метод оптимального управління, обробка та аналіз інформації

Завданням роботи є розробка гри, що дозволяє виконувати найпростіші ігрові механіки.

Методика дослідження: Перш за все, необхідно вибрати правильну архітектуру та стек технологій із забезпеченням коректної працездатності продукту, за можливості підтримки його розробником з метою ефективного додавання нового функціоналу або виправлення виявлених недоліків.

Враховуючи вимоги до специфіки програмного продукту, найкращим рішенням є розробка гри на Unity для мобільних пристроїв на Android.

В результаті розробки гри можна виконувати найпростіші ігрові механіки.

Практичне значення одержаних результатів. Даний продукт може бути використаний у комерційних цілях.

.

1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ІГОР

1.1 Основні терміни та поняття

Грою називається інстинктивний спосіб отримання та розвитку навичок людьми та тваринами в момент відсутності безпосередньої загрози для життя [1].

Можна сказати, що грою є форма діяльності в умовних ситуаціях, спрямована на відтворення та засвоєння суспільного досвіду, фіксованого в соціально закріплених способах здійснення предметних дій, у предметах науки та культури.

Гра характеризується тенденцією до повторення та напрацювання життєво необхідних рефлексів.[1]

Також термін "гра" використовують для позначення набору предметів або програм, призначених для подібної діяльності. Створення типових для професії ситуацій та знаходження в них практичних рішень є стандартним для теорії управління (ділові ігри - моделювання виробничої ситуації з метою вироблення найбільш ефективних рішень та професійних навичок) та військової справи (військові ігри - вирішення практичних завдань на місцевості та за топографічними картами).

Гра може бути і є предметом вивчення різних наук, наприклад біології, фізіології тощо. буд. Поняття «гра» включає у собі величезний спектр уявлень, і різні автори по-своєму підходять до трактування цього визначення. Але як би різні автори не трактували термін «гра», вона завжди була однією з провідних форм розвитку психічних функцій людини та способом реального пізнання світу.

До основних видів ігор можна віднести:

- Настільні ігри
- Азартні ігри
- Рухливі ігри
- Спортивні ігри
- Рольові ігри

- Дитячі ігри
- Комп'ютерні ігри
- Психологічні ігри
- Трансформаційні ігри
- Челлендж.

Основними компонентами гри є:

- Уявна ситуація, роль і реалізують її ігрові дії, а також ролі, взяті на себе граючими
- Ігрові дії як реалізації цих ролей
- Ігрове вживання предметів, тобто заміщення реальних предметів ігровими, умовними
- Реальні відносини між гравцями [2].

Відповідно до Шмакова С. А., виділяються:

- Відсутність матеріальних результатів (насолода замість утилітарності)
- Зміст (те, що гра відображає)
- Сюжет гри
- Уявна ситуація (задум і вигадка гри)
- Правила гри (співвідношення всіх її компонентів)
- Ігрові дії
- Зовнішнє завдання гри
- Засоби гри
- Ризик
- Виграш [3].

Предметом даної роботи є комп'ютерні ігри, тобто комп'ютерні програми, що служить для організації ігрового процесу (геймплея), зв'язку з партнерами по грі, або сама виступає як партнер.

Нині, замість терміну комп'ютерна гра можна використовувати термін відеогра, тобто дані терміни можуть використовуватися як синоніми і бути взаємозамінними.

Комп'ютерні ігри можуть створюватися на основі фільмів та книг, а також існують і зворотні випадки. З 2011 року комп'ютерні ігри офіційно визнано у США окремим видом мистецтва.

Комп'ютерні ігри мали такий суттєвий вплив на суспільство, що в інформаційних технологіях відзначено стійку тенденцію до гейміфікації для неігрового прикладного програмного забезпечення.

До основних складових компютерних ігор відносять:

- Сеттинг - це середовище, в якому відбувається дія комп'ютерної гри; місце, час та умови дії
- Геймплей - компонент гри, який відповідає за інтерактивну взаємодію гри та гравця. Геймплей визначає, як гравець взаємодіє з ігровим світом, як ігровий світ реагує на дії гравця і як визначається набір дій, який пропонує гравцю гра
- Музика в комп'ютерних та/або відеоіграх – це будь-які мелодії, композиції або саундтреки відеоігор.

Основними жанрами комп’ютерних ігор є:

- Action
  - Платформери
  - Шутери
  - Файтинг
  - Beat-em up
  - Стелс-екшен
  - Виживання
  - Ритм-гра
- Action-Adventure
  - Survival-horror
- Пригоди (квести)
  - Текстовий квест
  - Графічний квест
  - Візуальна новела

    - Інтерактивне кіно
    - 3D у реальному часі
- Рольові ігри
    - Action-RPG
    - MMORPG
    - Roguelike
    - Тактичні RPG
    - RPG з відкритим світом
    - Партійні RPG від першої особи
    - JRPG
- Симулятори
    - Симулятори будівництва та менеджменту
    - Симуляція життя
    - Симулятори техніки
- Стратегії
    - 4X-Стратегії
    - Стратегії у реальному часі (RTS)
    - Тактика реального часу (RTT)
    - MOBA
    - Tower defense
    - Покрокові стратегії (TBS)
    - Покрокова тактика (TBT)
    - Варгейми
    - Глобальні стратегії
    - Настільні та карткові ігри
- Спортивні ігри
    - Гонки
    - Командні види спорту
    - Інші змагання

    - Спортивні єдиноборства
- Головоломки
    - Логічні ігри
    - Вікторини
- Інші жанри
    - 10.1 Інкрементальні ігри
    - 10.2 Казуальні ігри
    - 10.3 Ігри для вечірок
    - 10.4 Ігри з програмуванням
    - 10.5 Масовий мережевий мультиплеєр (MMO)
    - 10.6 Рекламні ігри
    - 10.7 Творчі ігри
    - 10.8 Навчальні ігри
    - 10.9 Фітнес-ігри.

Розробкою комп'ютерних ігор називається процес створення комп'ютерних ігор.

Розробкою комп'ютерних ігор може займатися як одна людина, і фірма (колектив розробників). Комерційні ігри створюються командами розробників, найнятими однією фірмою. Фірми можуть спеціалізуватись на виробництві ігор для персональних комп'ютерів, ігрових приставок або планшетних комп'ютерів. Розробка може фінансуватися іншою, більшою фірмою — видавцем. Фірма-видавець по закінченні розробки займається поширенням гри та бере на себе пов'язані з цим витрати. Протилежним підходом є така розробка, коли фірма самостійно (без участі видавців) розповсюджує копії ігор, наприклад, засобами цифрової дистрибуції.

Розробка найбільш великобюджетних ігор може коштувати десятки мільйонів доларів США, причому останні 2 десятиліття ці бюджети безупинно зростали, як і чисельність команд розробників і терміни розробки.

Ігровим двигуном називається комплекс програм, що виконують найбільш складні, ресурсомісткі та рутинні завдання, що раніше стояли перед усіма ігровими програмістами. Виведення графіки, пошук шляхів, конверсія моделей з 3д редакторів у гру, розрахунки фізики, розрахунки зіткнень, оптимізація використання ресурсів процесора та багато іншого, що раніше вимагало від ігрового програміста не кволого математичного апарату та знання мови низького рівня, що найменш навантажує процесор - тепер доступно «прямо із коробки».

Приблизно з 2010-х років відзначаються вибухоподібні зростання якості та зниження вартості ігрових двигунів, що дозволяють ігровому програмісту зосередиться на створенні ігрових механік, віддавши на відкуп реалізацію графіки, фізики, звуковий супровід, мережевий код та інші складні речі творцям ігор. Політика підприємств, що виробляють найбільш масові ігрові двигуни, націлена на максимальне ознайомлення майбутніх і просто ігрових програмістів з їх двигуном, створюючи сприятливу для компанії ситуацію на ринку праці.

Сьогодні завдяки ігровим двигунам новачок-програміст, без глибоких знань у математиці, мовах програмування, без особливих фінансових витрат здатний спробувати створити гру не реальну за мірками початку 2000-х років.

Ігровий движок - базове програмне забезпечення комп'ютерної гри. Поділ гри та ігрового двигуна часто розпливчасто, і не завжди студії проводять чітку межу між ними. Але в загальному випадку термін «ігровий движок» застосовується для того програмного забезпечення, яке придатне для повторного використання та розширення, і цим може бути розглянуто як основу для розробки безлічі різних ігор без істотних змін.

Існує досить велика кількість ігрових движків.

До найбільш популярних сучасних движків можна віднести:

- Unity - безкоштовний кросплатформовий ігровий двигун. Безкоштовна версія дещо обмежена порівняно з PRO-версією. Підтримує 3D, 2D графіку. Може використовуватися для створення як ігор, так і розрахованих на багато користувачів

- Unreal Engine 4/UDK - UE3 досі вважається найпопулярнішим ігровим двигуном верхнього рівня. Epic games випустила безкоштовну версію під назвою UDK (двійковий реліз двигуна), яка дозволяє використовувати двигун для створення некомерційних ігор та безкоштовних додатків. Комерційні також можуть бути за певних умов
- CryEngine 3 SDK - CryEngine 3 - двигун нового покоління, розроблений Crytek для створення ігор Crysis 2, Crysis 3 і Warface
- Blender Game Engine (BGE) - безкоштовний і відкритий ігровий двигун, який розповсюджується разом з пакетом з 3D-моделювання Blender. Вирізаний із пакета починаючи з версії 2.80
- Solar2D - безкоштовний кросплатформовий двигун для створення 2D-ігор та додатків від компанії Corona Labs. Підтримує iOS, Android, Windows, Mac OS, tvOS, Android TV та Fire OS. Підтримує підключення нативних бібліотек (iOS/Android)
- SpriteKit – безкоштовний ігровий 2D двигун від компанії Apple. Підтримує всі продукти Apple: iPhone, Macbook, AppleTV і т.д.

1.2 Історія розвитку ігор

Вважається, що історія комп'ютерних ігор почалася у 1940-х та 1950-х роках, коли в академічному середовищі розроблялися прості ігри та симуляції. Комп'ютерні ігри тривалий час не були популярними, і лише в 1970-х і 1980-х роках, коли з'явилися для широкої публіки аркадні автомати, ігрові консолі та домашні комп'ютери, комп'ютерні ігри стають частиною поп-культури.

Появі комерційних комп'ютерних ігор передував розвиток індустрії розважальних аркадних автоматів. Такі автомати випускалися з XIX століття, використовуючи дедалі складніші механізми, і з 1930-х і електрику; паралельно розвивалися і музичні автомати-«Джукбокс». Примітним і надзвичайно складним для свого часу аркадним автоматом став Nimatron - електромеханічний комп'ютер для гри в ньому, спроектований фізиком Едвардом Кондон і виставлений на Всесвітній виставці 1939-1940 років у Нью-Йорку. У 1947 році було запатентовано «Розважальний пристрій на основі електронно-променевої трубки» Томаса Голдсміта та Естла Манна, що вважається першим спеціально призначеним для гри пристроєм, що виводив зображення на екран, тобто «відеогрою».

На початку 1950-х років створювалися спеціалізовані комп'ютери на зразок Nimrod знову ж таки для гри в ньому і Bertie the Brain і OXO для гри в хрестики-нулики. Tennis for Two, розроблена фізиком Вільямом Хігінботам, імітувала гру в теніс з графічним інтерфейсом, використовуючи аналоговий комп'ютер і осцилограф як засіб виведення в реальному часі. У 1948-1950 роках Алан Т'юрінг і Девід Чампернаун розробили алгоритм шахової гри, проте комп'ютери на той час були недостатньо потужними, щоб реалізувати цей алгоритм. Британський журналіст Трістан Донован у книзі Replay: The History of Video Games описував 1950-і роки як «десятиліття фальстартів», одиничних пристроїв, створених у єдиному екземплярі для виставок та розібраних пізніше — творці цих пристроїв відкидали ідею комп'ютерних ігор як порожню витрату часу.

Протягом 1960-1970-х років в університетському та науковому середовищі США продовжували створюватися різні ігри - як навчальні програми, як вправи у програмуванні та просто для розваги студентів. Їхня кількість і складність зростали в міру того, як комп'ютери (мейнфрейми) ставали все більш доступними, з розвитком мов програмування та появою перших комп'ютерних мереж (ARPAnet), що дозволяли користувачам взаємодіяти один з одним та ділитися програмами. Система електронного навчання PLATO, особливо із запущеного у 1972 році покоління PLATO IV, стала особливо зручною для подібних проектів – на ній з'явилися власні версії Spacewar! і шахів, розраховані на багато користувачів стратегічні 4X-ігри на кшталт Empire і Star Trek (1972); адаптації настільних рольових ігор типу Dungeons & Dragons - dnd [en] (1974) та pedit5 [en] (1975). Oregon Trail (1971), написана для мейнфрейму HP 2100, передбачила симулятори виживання; Maze War (1973) та Spasim (1974) стали першими тривимірними комп'ютерними іграми [4]. Текстова гра Colossal Cave Adventure (1975) для мейнфрейму PDP-10, що поєднувала у собі симулятор спелеолога та елементи фентезі на кшталт Dungeons & Dragons, відкрила дорогу для квестів і interactive fiction. Якщо перші комп'ютерні рольові ігри, як dnd і pedit5, були відносно примітивними, їх спадкоємці ставали все складнішими - ігри на кшталт Moria[en] (1975), Oubliette (1977) і Avatar[en] (1979) використовували вже вид від першої особи, складні багаторівневі підземелля та безліч параметрів для налаштування персонажів.

У 1981 стартують продажі першого ПК від фірми IBM, а наступного року у користувачів з'являється можливість придбати 8-бітні комп'ютери Commodore 64 і ZX Spectrum, які вже могли похвалитися кольоровим зображенням. Зважаючи на конкурентоспроможну ціну, домашні ПК починають витісняти з ринку ігрові консолі другого покоління.

Рисунок 1 ZX Spectrum

6 червня 1984 року світ побачив легендарну гру «Тетріс», розроблену програмістом Олексієм Пажитновим.

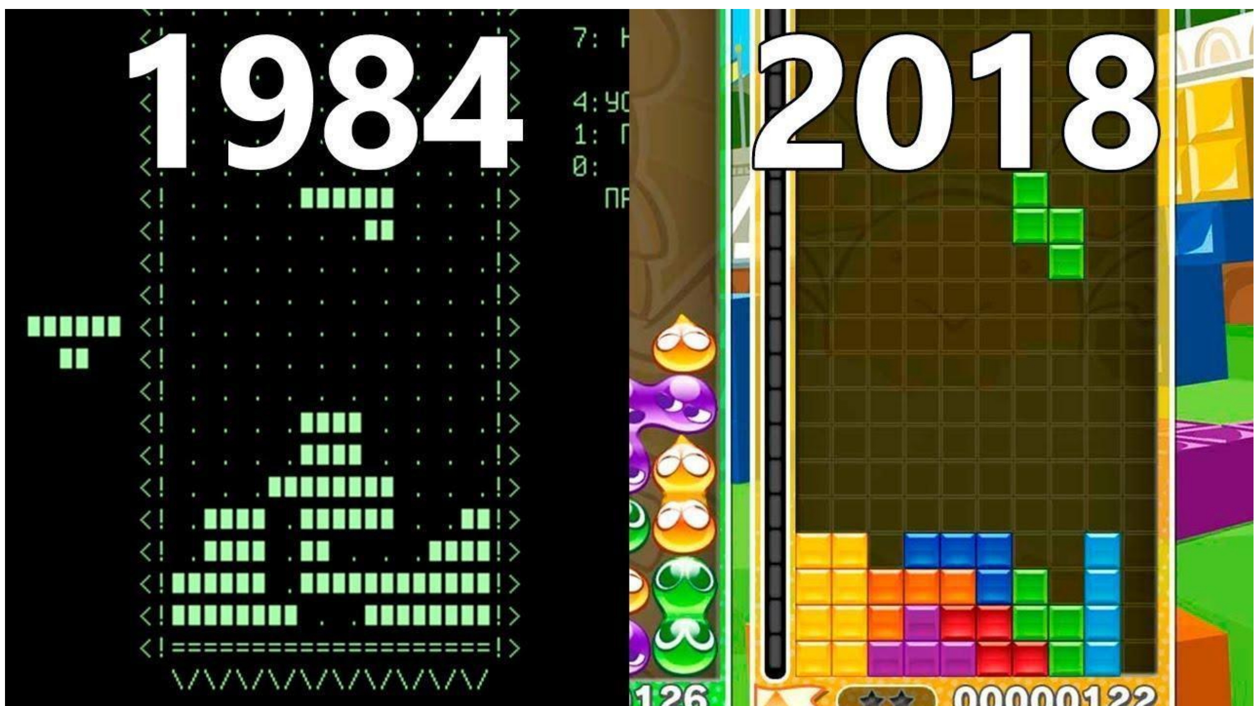


Рисунок 2 Тетріс тоді і зараз

1985 був надзвичайно багатий на ігри-шедеври, що вплинули на індустрію. "Super Mario Bros.", "Battle City", а також "Habitat" - онлайн-гра, що стала прообразом сучасної серії Sims.

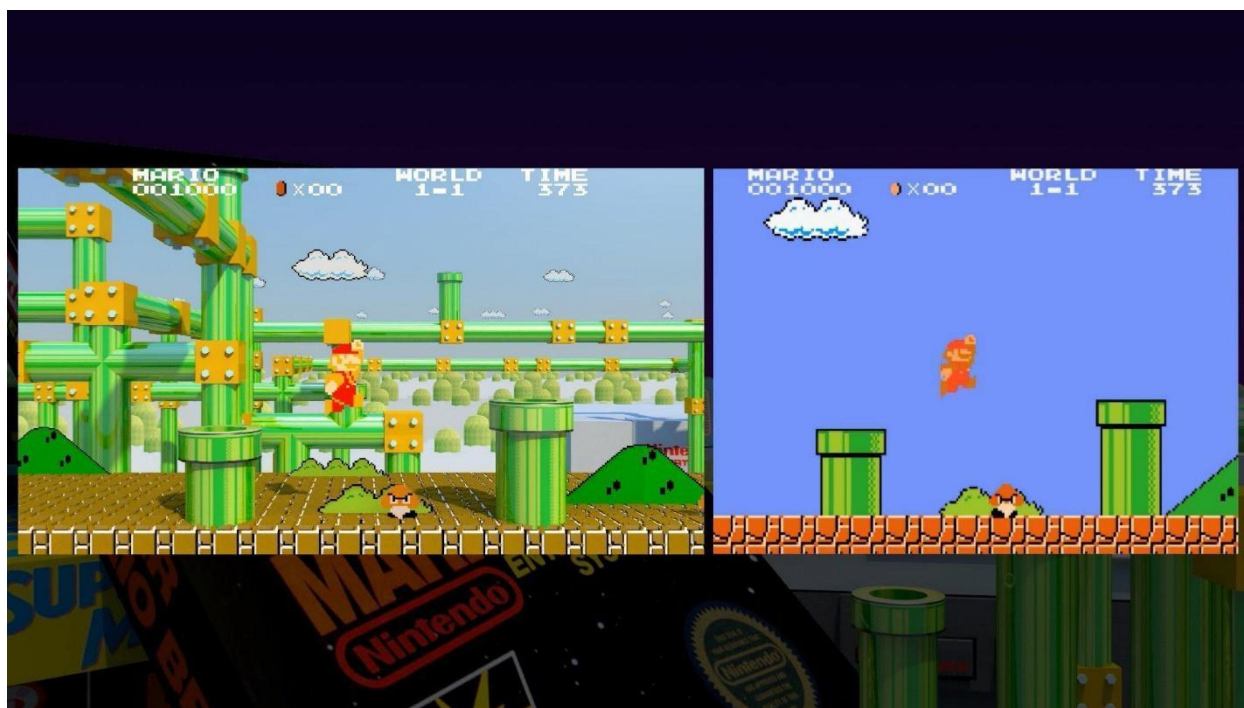


Рисунок 3 Ігри з серії Mario

У 1986 році виходить легендарна пригодницька гра The Legend of Zelda, яка вважається однією з трьох найкращих ігрових серій всіх часів. У цей час зароджується цікавий жанр – «японські рольові ігри» (jRPG), які кардинально від західних РПГ. Перша jRPG – Dragon Quest.

У 1987 році комп'ютерні відеокарти починають підтримувати новий стандарт VGA (256 кольорів), що зробило відеоігри барвистими та все більш схожими на сучасні. У цей же час була створена перша комп'ютерна звукова карта AdLib, що є ще одним якісним кроком на шляху еволюції ігор.

З'явилася гра "Maniac Mansion" - перша пригода, де для управління використовувалася комп'ютерна миша (інтерфейс point-and-click), а не текстові команди, як раніше.

Також у цей час була створена перша «Final Fantasy», що стала прабатьком серії однієї з найпопулярніших jRPG у світі.

У 1991 році було створено легендарну "Sonic the Hedgehog", що стала символом платформи Sega. Також з'явився новий жанр - "глобальні стратегії", піонером якого стала знаменита "Civilization".

Черговою значною віхою еволюції відеоігор став тривимірний шутер з видом від першої особи «Wolfenstein 3D», що вийшов у 1992. З'явився еталон файтингів «Mortal Kombat», а також перша стратегія в реальному часі (RTS) «Dune 2» та перша хоррор-гра Alone in the Dark».

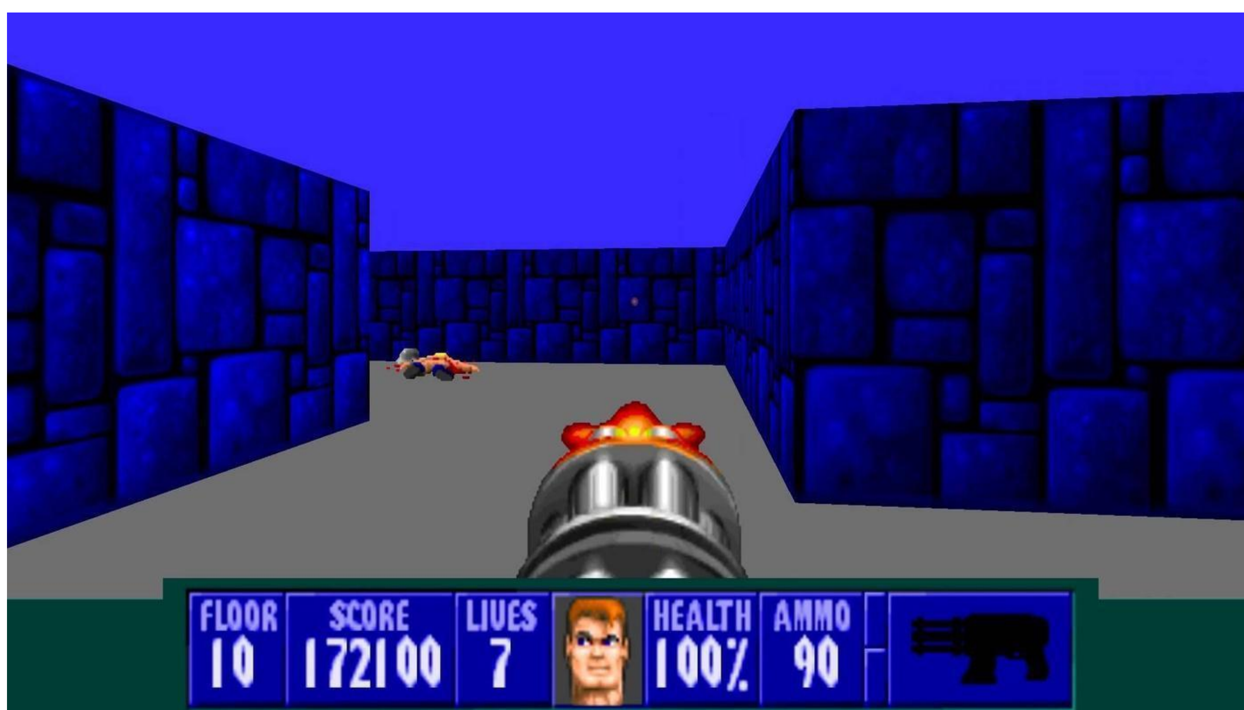


Рисунок 4 Wolfenstein 3D

У 1993 році справу "Wolfenstein 3D" продовжив культовий шутер "Doom", який став "законодавцем мод" серед ігор аналогічного жанру.

У 1994 році з'являються консолі п'ятого покоління, найкращою з яких стала Sony PlayStation. В цей же час Blizzard створює свій перший шедевр Warcraft: Orcs and Humans - RTS, що стала одним з лідерів жанру.

Рисунок 5 Warcraft

1995 теж був багатий на новинки: з'явилася популярна покрокова стратегія Heroes of Might and Magic, гоночна серія Need for Speed, на ПК вийшов легендарний квест Myst.

Рисунок 6 Heroes of Might and Magic

У грі "Highlander: The Last of the MacLeods" було вперше застосовано технологію захоплення руху, яка використовувалася раніше лише у фільмах.

1996 рік. Була випущена Voodoo I - перша відеокарта з підтримкою 3D-прискорення, що дало потужний поштовх до розвитку тривимірних ігор, таких як Duke Nukem 3D, Command & Conquer: Red Alert, Tom Raider, Resident Evil, Diablo.

1997 рік. Перша по-справжньому масова і успішна розрахована на багато користувачів онлайн-гра Ultima Online набирає популярність (рубіж в 100 000 гравців був досягнутий вже наступного року). Саме 1997 можна вважати відправною точкою в популяризації мережевих ігор. Тоді ж було випущено культову РПГ Fallout.

1998 рік. «Half-Life» - перша гра, в якій сюжет подавався безпосередньо всередині геймплею без відеовставок та тексту. "Starcraft" і Grand Theft Auto" (GTA) побачили світ теж саме цього року.

2002 рік. Виходить третина культової серії РПГ під назвою The Elder Scrolls III: Morrowind, що задає нові стандарти рольових ігор: величезний відкритий світ, пророблена система прокачування персонажа, цікава сюжетна лінія. Починає набирати популярність "Dota" - гра, що представляє модифікацію однієї з карток "Warcraft III".

2004 рік. У цей час були створені «Far Cry» – родоначальник легендарної серії шутерів від першої особи у відкритому світі з чудовою графікою та деякими унікальними геймплейними особливостями, а також наймасовіша MMORPG «World of Warcraft».

У 2005 році вийшла "Fahrenheit: Indigo Prophecy" - гра-шедевр, що представляє собою інтерактивний фільм з безліччю кінцівок та варіантів розвитку подій.

У 2007 році вийшли такі значущі ігри, як «Crysis» (краща на той час графіка), «S.T.A.L.K.E.R.» та «Bioshock».

2008 рік. Розвиток мобільних відеоігор був пов'язаний з появою двох найбільших магазинів App Store і Google Play.

2009 рік. Інді-гра Braid набуває популярності, що підштовхує незалежних розробників на створення інших подібних проектів, які починають змагатися з відеоіграми великих компаній.

2011 рік. Виходить легендарна інді-гра «Minecraft» у жанрі «пісочниця» (будівництво з виглядом від першої особи), а також «TES: Skyrim» – продовження однієї з найпопулярніших РПГ у світі.

2015 рік. Була випущена The Witcher 3: Wild Hunt - найкраща РПГ за мотивами слов'янського фентезі.

Рисунок 7 The Witcher 3: Wild Hunt

2016 рік. Blizzard випускає «Overwatch» - оригінальний розрахований на багато користувачів шутер з елементами РПГ і стратегії.

2017 рік. Зароджується новий ігровий жанр шутерів на виживання - "королівська битва", яскравим представником якого є надпопулярна гра "PlayerUnknown's Battlegrounds".

2016 рік. Суттєвого розвитку набуває ігрова індустрія віртуальної реальності, чому сприяв вихід на ринок таких VR-гарнітур як Oculus Rift, Sony Playstation VR, HTC Vive (2015 р).

За цей час були випущені такі значущі VR-проекти як The Elder Scrolls V: Skyrim VR, Doom: VFR, Fallout 4 VR. У цьому ж році світ був «захоплений» відомою AR-грою «Pokemon Go».

1.3 Основні технології створення ігор

Процес розробки гри зазвичай включає такі етапи:

- Підготовка («препродакшн» від англ. pre production);
- Уточнення геймдизайну;
- Виробництво;
- Підтримка.

Етапи можуть змінюватися в залежності від переваг фірми та особливостей проекту.

Проектування

1. Мета: - Ідея, - Жанр, - Сеттинг.

2. Засіб: - Програмний код, - Ігровий двигун.

Творчість

3. Ігрова механіка: - Об'єкти, - Керування, - фізичний двигун, - ІІ.

4. Рівні: - Розташування об'єктів (левелдизайн).

5. Графіка: - арти, - 2D, 3D моделі, - анімації,
- фони, - спецефекти, - оформлення екрану та меню.

6. Сюжет: — скрипти, події, — діалоги, оповідання, — відеовставки.

7. Звук: звукові ефекти, музика, озвучка.

Видання

8. Відшліфування: - Зведення матеріалу (а-версія), - Усунення помилок (b-версія).

9. Продаж: реклама, локалізація, система продажу.

10. Підтримка: - Випуск патчів, - Випуск доповнень.

Створення ігрового матеріалу (наповнення, контенту) - це суто творча частина процесу, а програмний код можна вважати інструментом. У такому випадку програмний код є каркасом (скелетом), на який будуть нанизуватися результати всіх наступних етапів розробки.

Цим етапом займаються програмісти.

Насамперед слід вибрати мову програмування, яка нам найбільше підходить. Після цього чекає важка і копітка робота з написання програмного коду, здатного оперувати двовимірними або тривимірними об'єктами у просторі, прив'язкою зображень та звуків. Для створення віртуального тривимірного простору доведеться використовувати складні геометричні формули для побудови проекції 3D-об'єктів на площину. По ходу розробки доведеться вивчити всі формати зображень та аудіофайлів, всілякі кодеки та кодування.

Благо, в наш час можна довго не поратися з написанням низькорівневої програмної частини, а відразу ж скористатися готовим програмним модулем (ігровим двигуном), де вже реалізовані базові функції, здатні пов'язати воєдино графіку, звук, об'єкти та їх рухи. Таким чином вибір мови програмування замінюється іншою дилемою - вибором готового ігрового двигуна.

Застосування ігрових двигунів ще не звільняє повністю від використання послуг програмістів, але зводить їх до мінімуму. Стандартний програмний модуль ще доведеться налаштувати під себе, додати щось своє, щоб ігровий проект вийшов більш унікальним.

2 СИСТЕМИ СТВОРЕННЯ ІГОР

2.1 Апаратні вимоги до системи для розробки ігор

Розробка ігор – досить широке поняття, яке включає мобільні іграшки, аркади і досить масивні проекти з віртуальною реальністю. Є і широкий вибір двигунів - Unity, Unreal Engine, CryEngine, Lumberyard або будь-який інший. Кожен з них підтримує кілька платформ, так що вибирати комп'ютер слід під платформу, а не під двигун.

Для тестування та розробки потрібен ігровий комп'ютер, потужність якого – орієнтовні системні вимоги + 1 покоління, щоб можна було тестувати без оптимізації. Відеокарта і процесор тут грають головну роль, у той же час кількість оперативної пам'яті та обсяг накопичувача не так важливий.

Так, сучасні рішення від NVIDIA будуть корисні не тільки при розробці комп'ютерних ігор, додаткові функції NVIDIA Studio будуть корисні при створенні рекламних роликів, а також при виборі набору технологій для реалізації основних елементів графіки, наприклад, трасування променів NVIDIA RT для створення тіні.

Базова карта NVIDIA RTX 2060 SUPER забезпечить прийнятний рівень продуктивності при створенні тривимірної графіки та тестуванні гри.

Комп'ютер на базі NVIDIA RTX 2070 SUPER вже може вважатися просунутим, ця карта з підтримкою інтелектуального трасування променів та віртуальної реальності дозволить створювати більш важкі ігри та навіть невеликі проекти, засновані на VR.

Таблица 1 Порівняльна таблиця характеристик відеокарт NVidia

	GEFORCE RTX 2060	GEFORCE RTX 2060 SUPER	GEFORCE RTX 2070	GEFORCE RTX 2070 SUPER	GEFORCE RTX 2080	GEFORCE RTX 2080 SUPER	GEFORCE RTX 2080 TI	TITAN RTX
Тактова частота, МГц: базова/у розгоні	1 365/1 680	1 470/1 650	1 410/1 620 (FE: 1 410/1 710)	1 605/1 770	1 515/1 710 (FE: 1 515/1 800)	1 650/1 815	1 350/1 545 (FE: 1 350/1 635)	1 350/1 770
Тензорні ядра	240	272	288	320	368	384	544	576
RT ядра	30	34	36	40	46	48	68	72
Частота пам'яті, МГц	1750	1750	1750	1750	1750	1750	1750	1750
Об'єм, Гб	6	8	8	8	8	8	11	24
Пікова продуктивність FP32, TFLOPS	6,5	7,1	7,5/7,9 (у версії FE)	9	10/10,6 (у версії FE)	11,2	13,4/14,2 (у версії FE)	19,9
Пропускна здатність відеопам'яті, Гб/с	336	448	448	448	448	496	616	672
TDP, Вт	160	175	175/185 (FE)	215	215/225 (FE)	250	250/260 (FE)	280

Для інді ігор підійде будь-яка картка, але для роботи з важкими тривимірними сценами або віртуальною реальністю, краще звернути увагу на топові рішення, наприклад, RTX 2080 SUPER або RTX 2080 Ti.

Щоб технологія була комфортною, а операції з компіляції не займали багато часу, потрібен потужний процесор. Добре підійдуть камінці від Intel, наприклад, для створення мобільної гри або аркади – Core i5, але для роботи з текстурами великої роздільної здатності та ємними тривимірними сценами, то краще вибрати Intel Core i7 9-ої або 10 серії.

Розробка проектів із віртуальною реальністю рекомендується на процесорах i9 з приставкою X, що є десктопним аналогом серверних рішень для створення надпродуктивних робочих станцій. Також такі процесори добре підійдуть для

підтримки серверної частини онлайн ігор, якщо планується створення мультиплеєра, який можна виділити половину потужності на потреби сервера і працювати на потужностях, що залишилися.

Найменше оперативної пам'яті - це 8 Gb, для звичайної роботи рекомендується 16 Gb, а якщо планується створення тривимірної гри, то 32. Частота не має великого значення, краще сконцентруватися на об'ємі.

Стабільність пам'яті безпосередньо залежить від продуктивності. На даний момент у комп'ютерах HyperPC найчастіше використовується пам'ять HyperX, яка зарекомендувала себе з позитивного боку.

Накопичувач для розробки ігор та будь-якої іншої діяльності у сфері IT просто потрібний SSD диск. Переважні SSD m.2, топові моделі з таким типом підключення можуть розвивати швидкість читання/запису до 3,5 Gb/s. Такий рівень продуктивності показує Samsung 970 EVO Plus, а також на рівні залишається модель Kingston A2000 зі швидкістю до 2,2 Gb/s.

Щоб розширити обсяг пам'яті, можна додати в систему HDD або встановити в локальну мережу, як окремий пристрій, що дасть можливість отримувати до нього доступ в будь-який час з будь-якого девайса. А мережевий адаптер Gigabyte Ethernet давно став уже обов'язковим атрибутом будь-якої ігрової материнки, тому швидкість доступу буде стабільно високою.

Unity є міжплатформеним середовищем розробки комп'ютерних ігор, що розроблене американською компанією Unity Technologies. Unity дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, 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.

Мінімальними системними вимогами Unity є:

- Процесор: багатоядерний Intel або AMD із підтримкою набору інструкцій SSE

- Відеокарта: з підтримкою DX10, DX11, або DX12
- Операційна система: Windows 7 (SP1+) / 10 64-розрядна.

Процесор є одним з найважливіших компонентів робочого комп'ютера для розробки в Unity. Для виконання інтенсивних завдань, яким потрібно багато часу, знадобиться процесор з великою кількістю ядер.

Для більшості користувачів чудовим вибором буде 12-ядерний AMD Ryzen 9 5900X. Цей процесор має 12 ядер та 24 потоки, що забезпечує відмінну продуктивність. Якщо дозволяє бюджет, можна придивитися до 16-ядерного Ryzen 5950X.

Тим, кому потрібна максимально можлива продуктивність для багатопотокових завдань, таких як робота з освітленням або компіляцією можна рекомендувати AMD Threadripper 3960X, що на 25% швидше, ніж Ryzen 5950X у таких завданнях, а Threadripper 3970X та 3990X будуть ще швидше. Слід звернути увагу, що ці процесори можуть бути трохи повільнішими в інших завданнях, тому рекомендується використовувати Threadripper тільки тим користувачам, які витрачають багато часу на створення освітлення або компіляцію.

Unity використовує відеокарту виключно для відображення графіки на екрані. Багато професійних програм в інших областях вже почали використовувати відеокарту для інших завдань, але в Unity це ще не реалізовано. Через це більш потужна відеокарта дасть вищий FPS на екрані монітора, але не покращить продуктивність у розробці.

Рекомендується використовувати наступні відеокарти для робочого комп'ютера, залежно від бюджету та того, чи планується розробка VR-контенту:

- NVIDIA GeForce RTX 3070 8 ГБ – ця відеокарта пропонує відмінну продуктивність і має достатню потужність для роботи з кількома дисплеями одночасно
- NVIDIA GeForce RTX 3090 24 ГБ на даний момент є однією з найкращих відеокарт для розробки ігор, VR та архітектурної візуалізації. Великий обсяг

відеопам'яті робить її придатною для робочої станції з трьома або навіть чотирма 4K-моніторами, а додаткова потужність відмінно підходить для ігор зі слабкою оптимізацією.

Хоча точний обсяг необхідної оперативної пам'яті залежатиме від конкретних проектів і від того, чи виконуються завдання, що вимагають великого обсягу оперативної пам'яті, наприклад, освітлення будівлі. Рекомендується:

32 ГБ оперативної пам'яті для більшості користувачів

64 ГБ + ОЗУ, якщо створюється освітлення.

При цьому слід мати на увазі, що це базові рекомендації і вони охоплюють лише обсяг оперативної пам'яті, необхідної для самого Unity. Якщо часто використовуються інші програми паралельно з Unity, то може знадобитися ще більше ОЗУ у системі, тому що кожній програмі потрібен окремий фрагмент ОЗУ.

З точки зору сховища (жорстких дисків) для Unity може бути не найважчим додатком для зберігання, але все ж таки важливо мати швидке і надійне сховище, щоб не відставати від решти системи.

2.2 Основні переваги та недоліки систем розробки ігор

Термін «ігровий двигун» з'явився в середині 1990-х у контексті комп'ютерних ігор жанру шутер від першої особи, схожих на популярну на той час Doom. Архітектура програмного забезпечення Doom була побудована таким чином, що являла собою розумний і добре виконаний поділ центральних компонентів гри (наприклад, підсистеми тривимірної графіки, розрахунку зіткнень об'єктів, звукові та інші) та графічних ресурсів, ігрових світів, що формують досвід гравця, ігрові правила та інше. Як наслідок, це набуло певної цінності за рахунок того, що почали створюватися ігри з мінімальними змінами, коли за наявності ігрового движка компанії створювали нову графіку, зброю, персонажів, правила гри тощо [5].

Поділ між грою та ігровим двигуном часто невизначено. Деякі двигуни мають розумний і ясний поділ, в той же час інші практично неможливо відокремити від гри. Наприклад, у грі двигун може «знати» про те, як малювати дугу, в той же час інший двигун може працювати з іншим рівнем абстракції, і в ньому дуга буде окремим випадком параметрів функцій, що викликаються. Однією з ознак ігрового двигуна є застосування архітектури управління даними. Це визначається тим, що якщо гра містить жорстко фіксовані дані, що впливають на логіку, правила гри, малювання об'єктів тощо, то складно застосовувати дане програмне забезпечення в різних іграх.

Більшість ігрових двигунів розроблено та налаштовано для того, щоб запустити певну гру на певній платформі. І навіть найбільш узагальнені багатоплатформні двигуни підходять для побудови ігор певного жанру, наприклад, шутерів першої особи або гонок. У цьому контексті можна сказати, що ігровий двигун стає не оптимальним при його застосуванні не для тієї гри або тієї платформи, для якої його розроблено. Цей ефект проявляється від того, що програмне забезпечення є набором компромісів, заснованих на тих припущеннях, якою має бути гра. Наприклад, проектування рендерингу всередині будівель призведе до того, що двигун, швидше за все, не буде таким же добрим для

відкритих просторів. У першому випадку двигун може використовувати BSP-дерево для відображення об'єктів, близьких до камери. У той самий час для відкритих просторів можуть використовуватися менш точні методи, і навіть найактивніше застосовуватися технології малювання з різною мірою деталізації, коли більш далекі об'єкти промальовуються менш чітко, оскільки займають меншу кількість пікселів.

Можна сказати, що ігровий двигун - це базове програмне забезпечення для будь-якої комп'ютерної гри. Поділ гри та ігрового двигуна дуже часто розпливчасто, і не завжди студії проводять чітку межу між ними. Але, загалом, термін «ігровий двигун» застосовується для програмного забезпечення, яке придатне для повторного використання. Навіть вже з цього визначення можна чітко виділити першу і найголовнішу перевага ігрового двигуна - можливість неодноразово його використовувати для безлічі ігрових проектів.

Основні преваги ігрових двигунів:

- Все вже зроблено зазделегідь, тому потрібно лише використати готові інструменти, щоб реалізувати наявні ідеї
- Є спільнота - інші розробники, які користуються цим же двигуном, можуть допомогти в разі потреби. Хтось пише статті або знімає ролики для YouTube, та й просто на StackOverflow можна поставити питання
- Більшість двигунів дозволяють в пару кліків портувати гру на іншу платформу. Іноді потрібно повозитися, наприклад, адаптувати керування під тачскрин або геймпад
- Є безкоштовні варіанти – скачав, і одразу в бій
- У багатьох двигунів є магазини з готовими скриптами, моделями, ефектами та іншими корисностями.

Основні мінуси використання двигунів:

- Іноді можуть трапитися баги, з якими нічого не можна зробити - тільки чекати, поки автори движка щось виправлять
- Менше свободи

- При використанні двигуна необхідно погодитись з ліцензією – іноді доводиться ділитися частиною прибутку
- Автори можуть кинути або переробити улюблений движок
- У складі пакету може бути багато того, що ніколи не знадобиться, а це збільшує розмір гри.

2.3 Основні переваги та недоліки Unity

Unity – міжплатформне середовище розробки комп'ютерних ігор [6], розроблене американською компанією Unity Technologies. Unity дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші[3]. Випуск Unity відбувся у 2005 році і з того часу триває постійний розвиток.

На Unity написані тисячі ігор, програм, візуалізації математичних моделей, які охоплюють безліч платформ і жанрів. У цьому Unity використовується як великими розробниками, і незалежними студіями.

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 утруднення при підключенні зовнішніх бібліотек.

Як правило, ігровий двигун надає безліч функціональних можливостей, що дозволяють їх задіяти в різних іграх, в які входять моделювання фізичних середовищ, карти нормалей, динамічні тіні та багато іншого. На відміну від багатьох ігрових движків, Unity має дві основні переваги:

- Наявність візуального середовища розробки
- Міжплатформенну підтримку [7].

Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегроване середовище, ланцюжок складання, що спрямоване на підвищення продуктивності розробників, зокрема етапів створення прототипів та тестування.

Під міжплатформною підтримкою надається як місця розгортання (установка персональному комп'ютері, на мобільному пристрої, консолі тощо.), а й наявність інструментарію розробки (інтегроване середовище можна використовувати під Windows і Mac OS).

Третьою перевагою Unity часто називають модульну систему компонентів Unity, за допомогою яких відбувається конструювання ігрових об'єктів, коли останні є комбінованими пакетами функціональних елементів. На відміну від механізмів успадкування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків, а не поміщення у вузли дерева успадкування. Такий підхід полегшує створення прототипів, що є актуальним при розробці ігр.

Як недолік системи часто наводять обмеження візуального редактора під час роботи з багатокомпонентними схемами.

Другим недоліком називається відсутність підтримки Unity посилань на зовнішні бібліотеки, роботу з якими програмістам доводиться налаштовувати самостійно, і це ускладнює командну роботу.

Ще один недолік пов'язаний із використанням шаблонів екземплярів (англ. prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним.

Слід відзначити також, що при роботі з освітленням двигун ламає тіні.

Також, WebGL-версія движка, в силу специфіки своєї архітектури (трансляція коду з C # C++ і далі в JavaScript), має ряд невирішених проблем з продуктивністю, споживанням пам'яті і працездатністю на мобільних пристроях.

3 ПРАКТИЧНІ АСПЕКТИ СТВОРЕННЯ ІГОР НА UNITY

3.1 Аналоги гри «ЗPER» в жанрі Tower Defense

Tower Defense є жанром комп'ютерних стратегічних ігор. Завдання гравця в іграх подібного жанру - розправитися з ворогами, що наступають, званими в деяких іграх «крипи» (від англ. creep - «повзуча тварюка») [8], до того, як вони перетнуть карту, за допомогою будівництва веж, атакуючих, коли вороги проходять поблизу. Противники та вежі зазвичай розрізняються за характеристиками та ціною. Коли вороги переможені, гравець заробляє гроші чи бали, які використовуються для покупки чи модернізації веж.

Підбір виду веж та їх розташування – невід'ємна стратегія гри. Зазвичай «повзучі тварюки» пробігають через подобу лабіринту, що дає гравцеві можливість стратегічного розміщення веж, але також є відомі версії гри, названі лінійними TD, де замість лабіринтів використовуються прямі шляхи. У деяких версіях гравець може сам вибудовувати лабіринт із веж та блоків.

Рисунок 8 Одна з версій гри Tower Defense

Існуєть несуттєві відмінності між різними іграми цього типу. Наприклад, у більшості версій, коли вежа модернізується, її радіус дії, рівень та потужність збільшуються одночасно. Однак у версії гри під назвою Onslaught Defence кожен параметр може бути покращено окремо. У деяких іграх ворог може боронитися. В окремих версіях маршрут просування ворога не обмежений будь-якими стінами. Хвилі наступу можуть починатися за командою гравця (тим даючи можливість спокійно підготуватися), або з певним проміжком часу. Можуть бути елементи економічної стратегії (наприклад, можна побудувати банк, який у геометричній прогресії збільшуватиме кошти). Противники в різних іграх можуть відрізнятися за своїми властивостями - наприклад, деякі з них можуть бути літаючими, і для їх поразки потрібні спеціальні протиповітряні вежі.

У грі GemCraft розділили вежі (і пастки) від їх активних здібностей по атаці (які забезпечуються дорогоцінним камінням). Камені можуть вийматися з веж і поєднуватися, посилюючись при цьому. Додатково з'явилися характеристики гравця, що розвиваються від карти до карти. Вже з'явилися інші ігри, які використовують цю знахідку.

Тобто можна сказати, що існує досить велика кількість різновидів Tower Defense.

3.2 Постановка ТЗ

Створення будь-якого проекту починається із розробки технічного завдання, що включає ключові вимоги до майбутнього продукту. Для проектування гри використовується документація, що враховує концепцію, специфікацію, ТЗ та план робіт. Окремо продумується дизайн, елементи геймплею, механіки та інтеграції.

У технічному завданні для мобільної гри докладно описуються всі параметри, які клієнт хотів би отримати від майбутнього продукту: від загальної стилістики до моделі монетизації та інструментів реалізації. Документ має містити

не лише суворі технічні дані, а й пояснювати бачення проекту замовником та цільовою аудиторією.

Універсального шаблону ТЗ для мобільної гри не існує, оскільки воно складається індивідуально для кожного продукту, але в документ зазвичай включаються такі пункти:

- Загальні відомості
- жанр гри (платформер, стратегія, РПГ та ін.);
- завдання та цілі проекту (команда AVADA MEDIA формує список завдань, де кожне завдання прив'язане до однієї або кількох цілей);
- вид гри 2D або 3D, ізометрія, ромбічна, вид зверху тощо;
- мультиплеєр, онлайн або синглплеєр;
- функціонал гри;
- цільові платформи: Android, IOS.
- Маркетингові рішення та прототипи

Після вивчення ринку у технічне завдання вносяться ідеї та особливості, що відрізняють гру від прототипів у даному жанрі, які зроблять її успішною на ринку.

- Формат екрану

Ще при розробці ТЗ необхідно визначитися з базовими характеристиками продукту, у тому числі із співвідношенням сторін (aspect ratio) та роздільною здатністю екрана, під який будуть створюватися всі типові елементи інтерфейсу, фони та текстури. Крім того, слід вказати, як зображення адаптуватиметься до екранів, що мають інше співвідношення сторін.

- Монетизація

Гра повинна приносити дохід тим чи іншим чином. Вона може продаватися за гроші або бути безкоштовною, але при цьому містити в собі вбудовану рекламу, що дає відрахування.

Play Market та App Store пропонують оптимальні можливості для реклами та продажу. Один із найпростіших варіантів — розміщення гри у платній секції та її подальше просування.

- Геймплей та екрани гри

У цьому розділі описуються всі ключові сцени, які будуть присутні у вашій грі, а також переходи між ними. Одна сцена є фоном і сукупністю всіх об'єктів на екрані, які повністю або частково замінюються при переході до нової сцени.

- Додаткові вимоги:

Магазини/маркети

Торгові майданчики, на яких гра продаватиметься або розповсюджуватиметься. Ця функція вимагає впровадження спеціального коду, який відповідає вимогам ігрових маркетів.

Модель із мікротранзакціями

Гравці можуть платити за додаткові переваги та зручності, наприклад, відключення реклами, придбання преміум-аккаунта з певними привілеями або ігрових предметів, що надають переваги тощо.

Соціальна складова

Гра може мати вихід у соціальні мережі, вести статистику досягнень, лідерборди та інші опції.

3.3 Основні можливості системи unity

Понад 50% усіх мобільних ігор розроблено саме на Unity. А в 2021 році, за версією звіту про глобальний ринок відеоігор від Newzoo [9], ця частка становитиме як мінімум 59%/

Спочатку Unity призначався для розробки на комп'ютерах Mac, пізніше з'явилося оновлення, що дозволяє працювати з Windows. У 2008 році Unity почав працювати з iOS, з Android - у 2010, а далі розробники змогли створювати шедеври для геймерських консолей Xbox і Playstation.

Unity є повноцінним ігровим двигуном, який передбачає, що весь процес девелопменту відбуватиметься в редакторі, що поставляється в комплекті. Багато популярних мобільних ігрових продуктів створені саме на цьому двигуні: Hearthstone: Heroes of Warcraft, Age of Magic, Royal Blood та інші.

Можна сказати, що однію з основних можливостей Unity є наявність магазину готових ассетів та плагінів. Це дозволяє розробляти проекти швидше та з меншими витратами.

У такому магазині можна отримати текстури, анімації, моделі та багато іншого, що дозволить зекономити час на розробку.

По-друге, програма має повноцінний графічний редактор, що дозволяє малювати карти, локації, розставляти персонажів. До прийнятного виду їх доводять у Photoshop. Під час створення Юніті 3д гри можна імпортувати 3D-моделі з більшості сторонніх редакторів, що полегшує процес роботи.

Тому Unity підходить розробникам, які ще не мають великої команди.

Варто зазначити, що на Unity створюються не тільки мобільні та комп'ютерні ігри, а й анімовані фільми, адже саме в Unity найчастіше створюють мультфільм та спецефекти компанії Disney та Warner Bros.

Можна сказати, що до основних можливостей системи Unity можна віднести:

- Зрозумілий редактор та інструментарій: за кілька днів основні речі може освоїти навіть той, хто вперше стикається з розробкою мобільного додатку.

А якщо питання залишаться, відповіді є на одному з багатьох ресурсів, форумів, а також в уроках на YouTube.

- Створення гри на Unity буде під силу навіть школяру.
- Сучасний рівень графіки, здатний конкурувати з дорожчими двигунами.
- Unity, безумовно, програє UnrealEngine за можливостями, але радує deferred освітленням, стандартним набором постпроцессингових ефектів, SSAO, прискореним опрацюванням лайтмапів.
- Ігровий двигун Unity надається умовно безкоштовно. Платити потрібно лише за розширення пакетів передплати. На ліцензії кілька разів на рік бувають знижки, як правило -20%.
- Велике ком'юніті розробників, безліч випущених ігор.
- Внутрішній Asset Store, де можна купити готові фрагменти коду, асети та звуки. Можливість створення фотореалістичної графіки.
- Розробка на Юніті дозволяє легко імпортувати між ОС Windows, Linux, OS X, Android, iOS, на консолі PlayStation, Xbox, Nintendo, на VR- та AR-пристрої.

3.4 Розробка гри «3PER» в жанрі Tower Defense на Unity

Tower Defense - це жанр гри, в якому гравцеві необхідно розправитися з ворогами (ботами), перш ніж вони перетнуть певну точку на карті. Робити це необхідно за допомогою будівництва веж у певних місцях на карті.

Створення гри «3PER» в жанрі Tower Defense на Unity починається з екрану вибору проекту. Тут потрібно натиснути New, щоби створити свій новий проект.

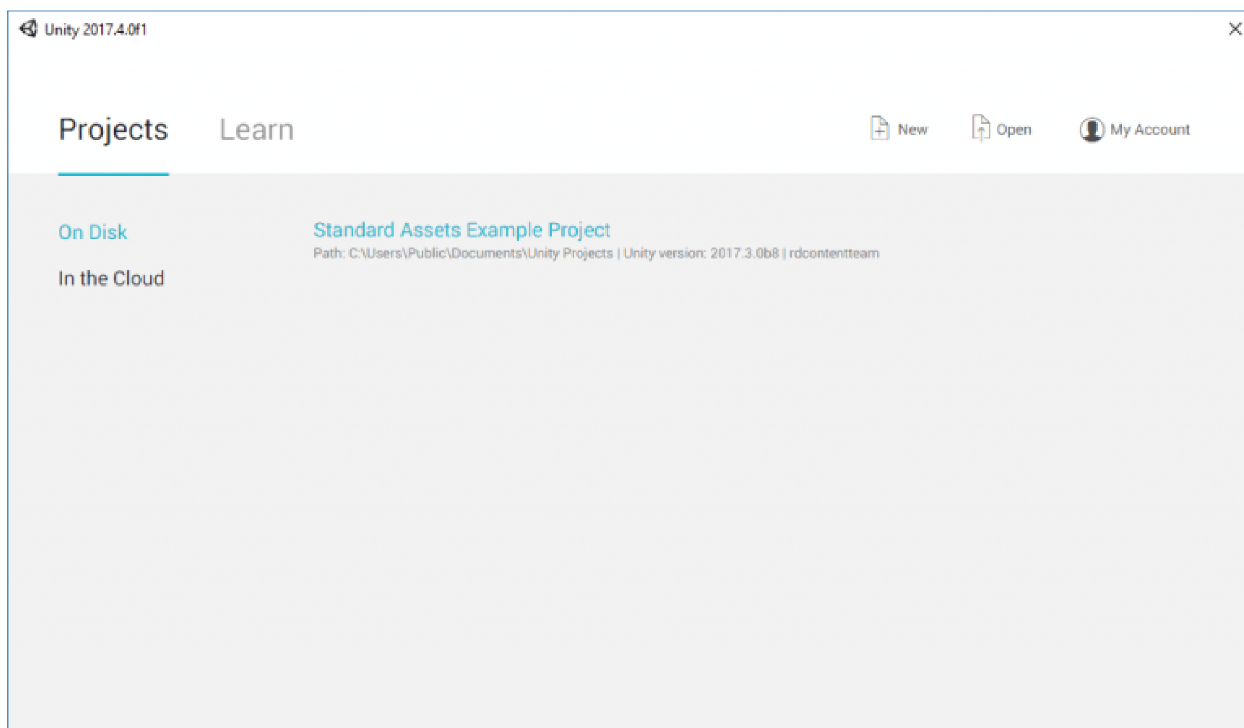


Рисунок 9 Створення нового проекту

Наступний крок представлено на рисунку.

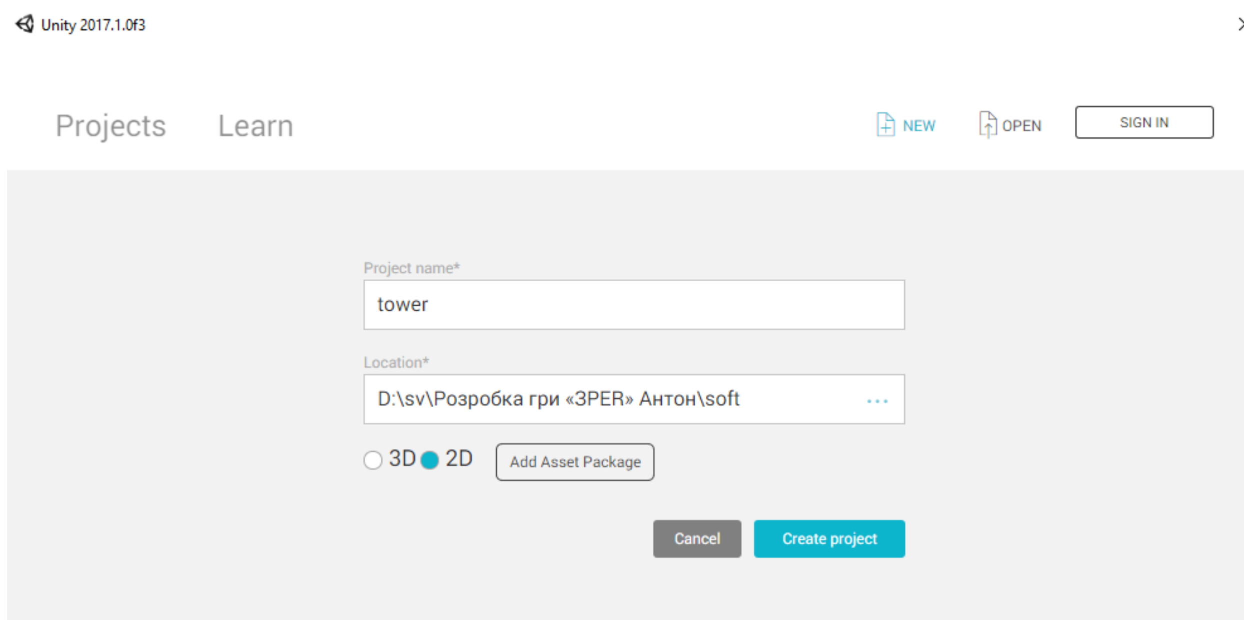


Рисунок 10 Створення нового проекту

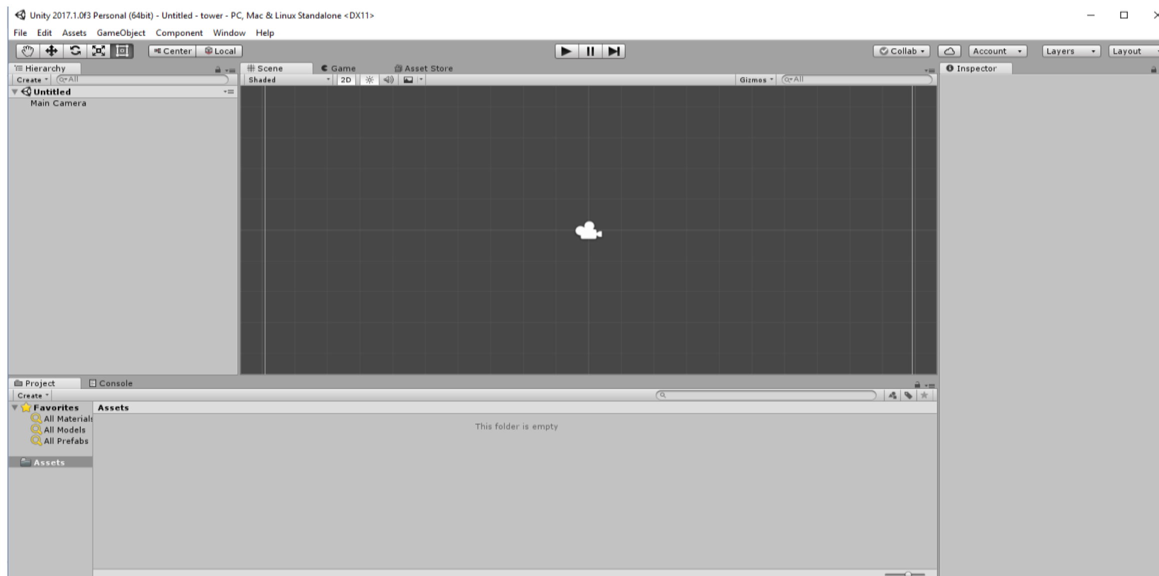


Рисунок 11 Головне вікно Unity

Для початку необхідно змінити платформу розробки на Android. Для цього в Unity слід відкрити меню File та вибрати Build Settings.

У вікні потрібно вибрати Android і потім натиснути Switch platform.

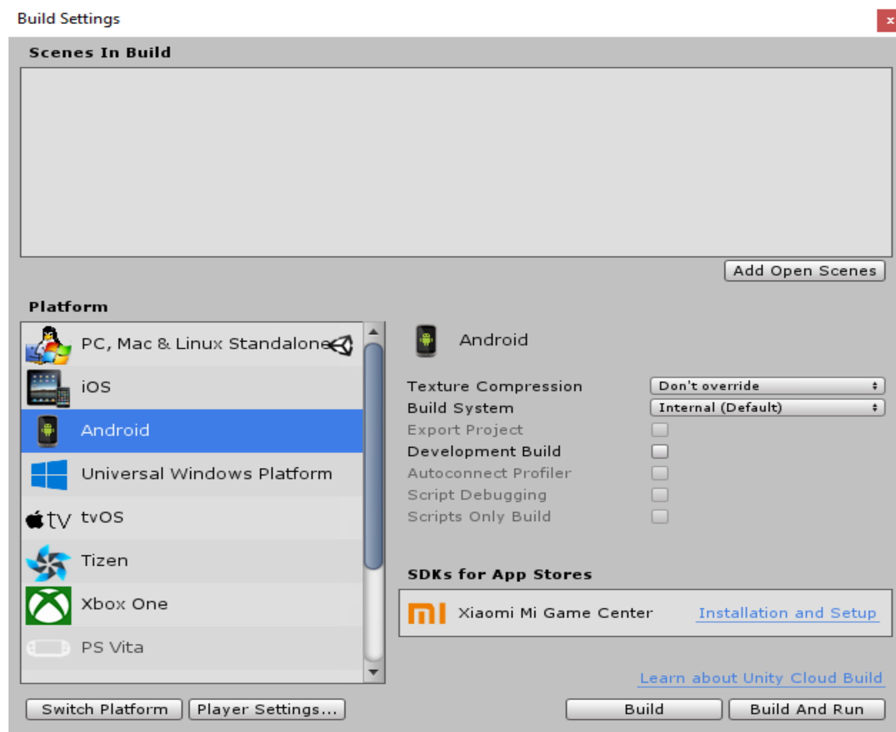


Рисунок 12 Налаштування проекту

Перемикання платформи повідомляє, що буде збиратися програма для Android. Це означає, що коли буде збиратися програма, Unity буде створювати APK-файл. Перемикання платформи також змушує Unity імпортувати всі ассети у проект заново.

Почнемо із підготовки сцени. У новому проекті у Unity слід додати кілька нових папок material, resources, scripts та scene. У цих папках будуть зберігатися матеріали, ігрові об'єкти, скрипти та ігрова сцена.

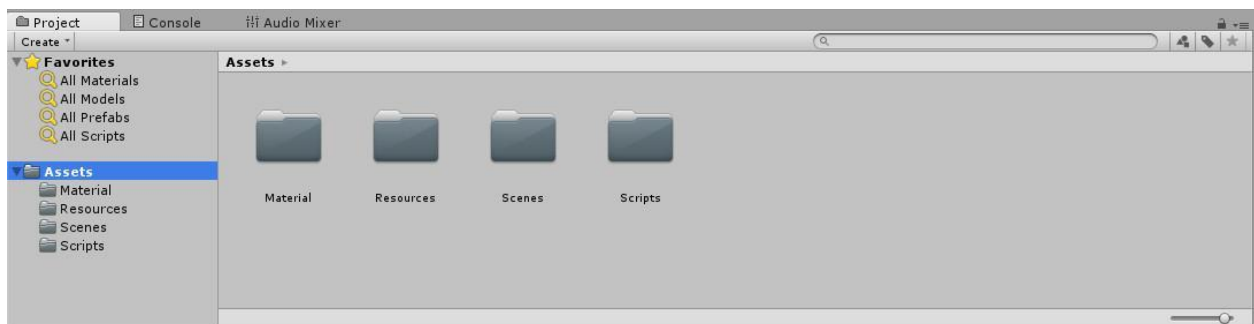


Рисунок 13 Створення матеріалів

Для початку необхідно створити ігрове поле, де проходитимуть усі основні дії.

Слід додати площину на сцену з позиціями по нулях та будь-яким матеріалом, перейменуємо її в pole.

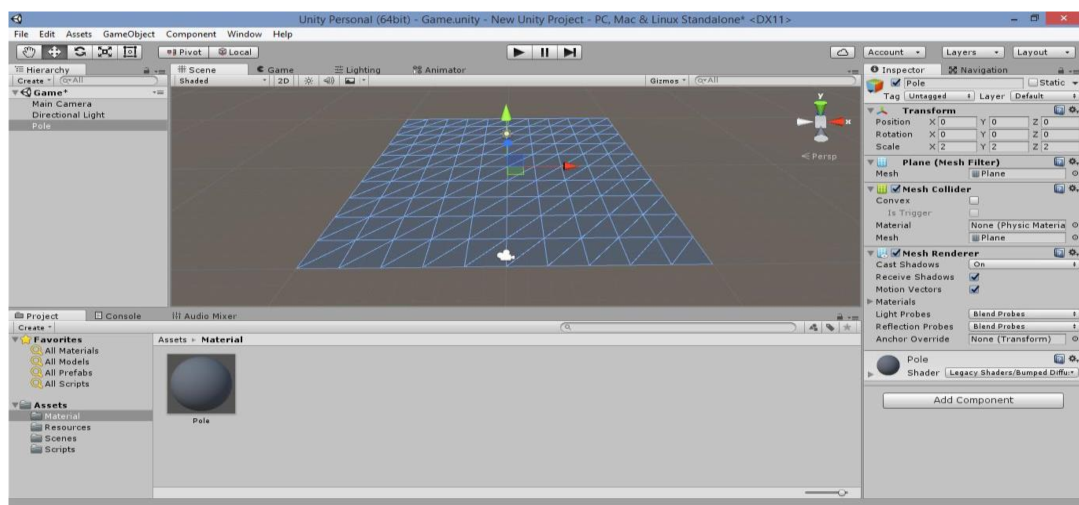


Рисунок 14 Створення ігрового поля

Для можливості розміщення веж на ігровому полі необхідно створити платформу.

Слід додати куб на сцену з розмірами 1, 0.1, 1 і відключити можливість відкидання тіней (Mesh render - Cast Shadows - off).

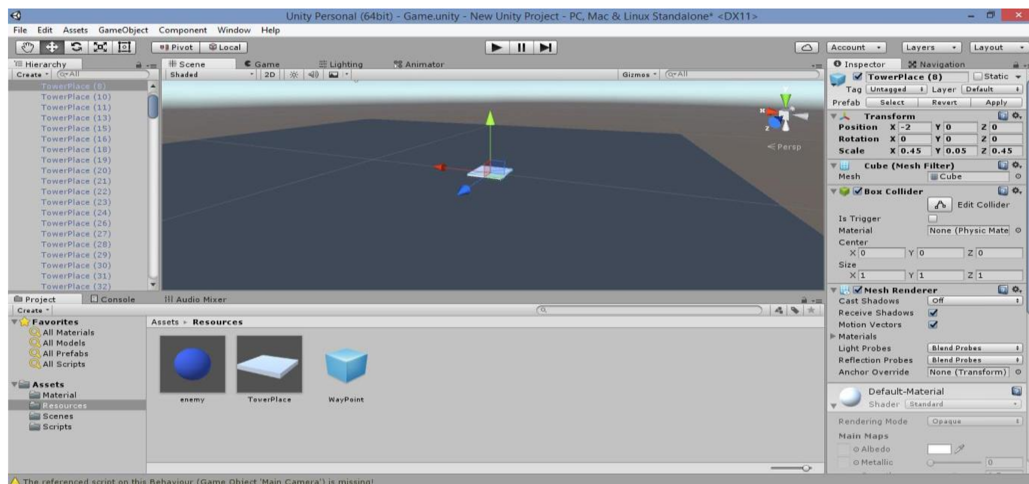


Рисунок 15 Створення обьектів та робота зі світлом

Слід перейменувати куб у TowerPlace і зберегти його на префаб.

Тепер можна продублювати платформу (ctr+D) так, щоб вони покривали все ігрове поле.

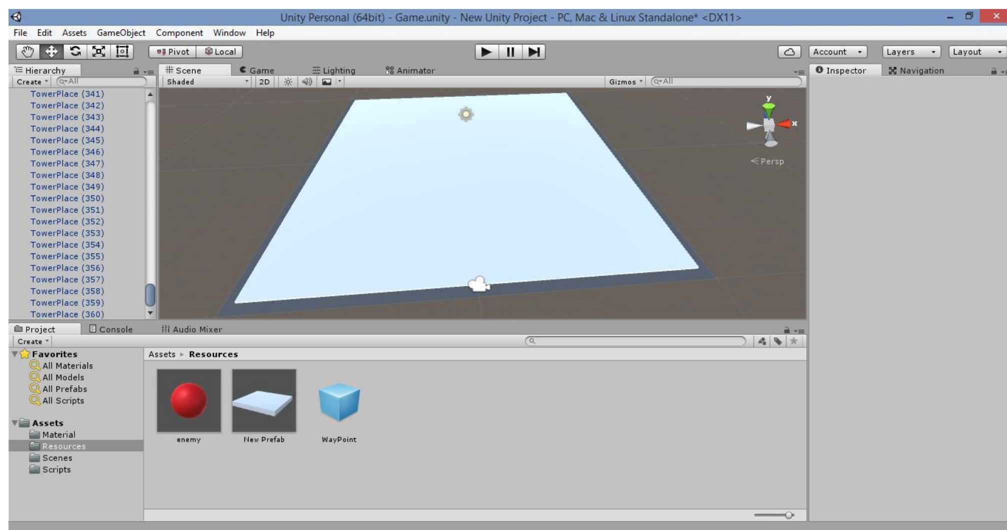


Рисунок 16 Створення ігрової зони

Перемістимо всі платформи в пустушку (TowerPlace) і у префабу змінимо трохи розміри для того, щоб була відстань між платформами.

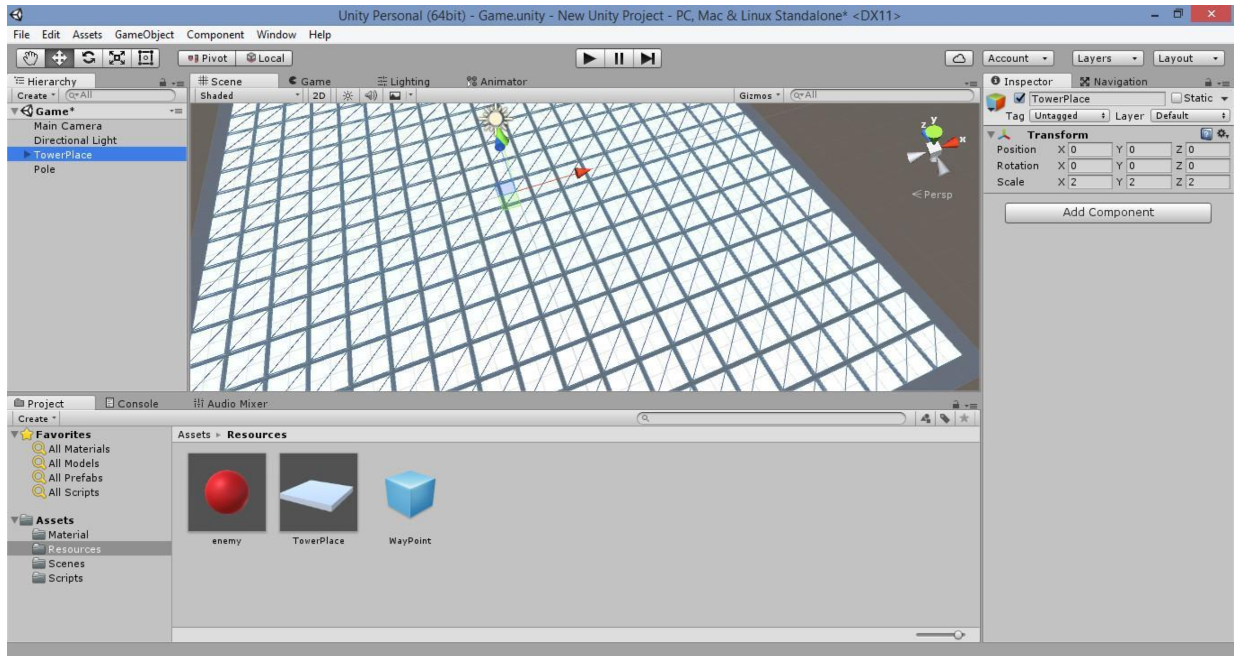


Рисунок 17 Поділ ігрової зони на плитки

Видалимо деякі об'єкти для того, щоб вийшла доріжка для руху ботів.

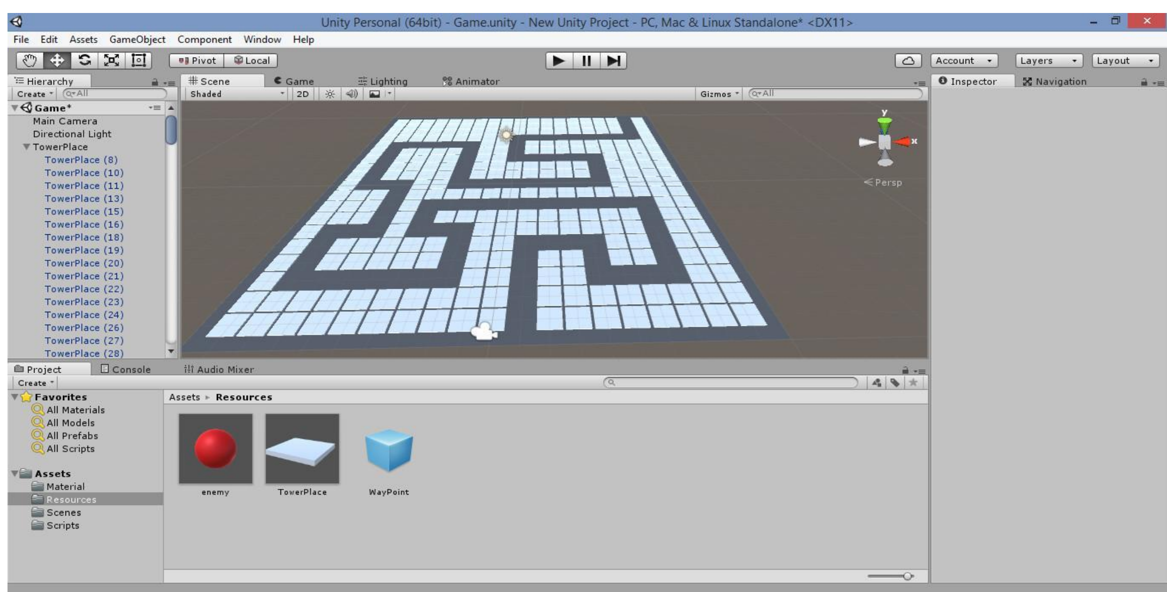


Рисунок 18 Створення дороги для ботів

Створимо порожній об'єкт на сцені з позиціями по нулях та назвою WayPoint. Змінимо його іконку, щоб він відображався на сцені. Піднімемо його по осі y, вісь x і z повинні залишатися зі значенням нуль.

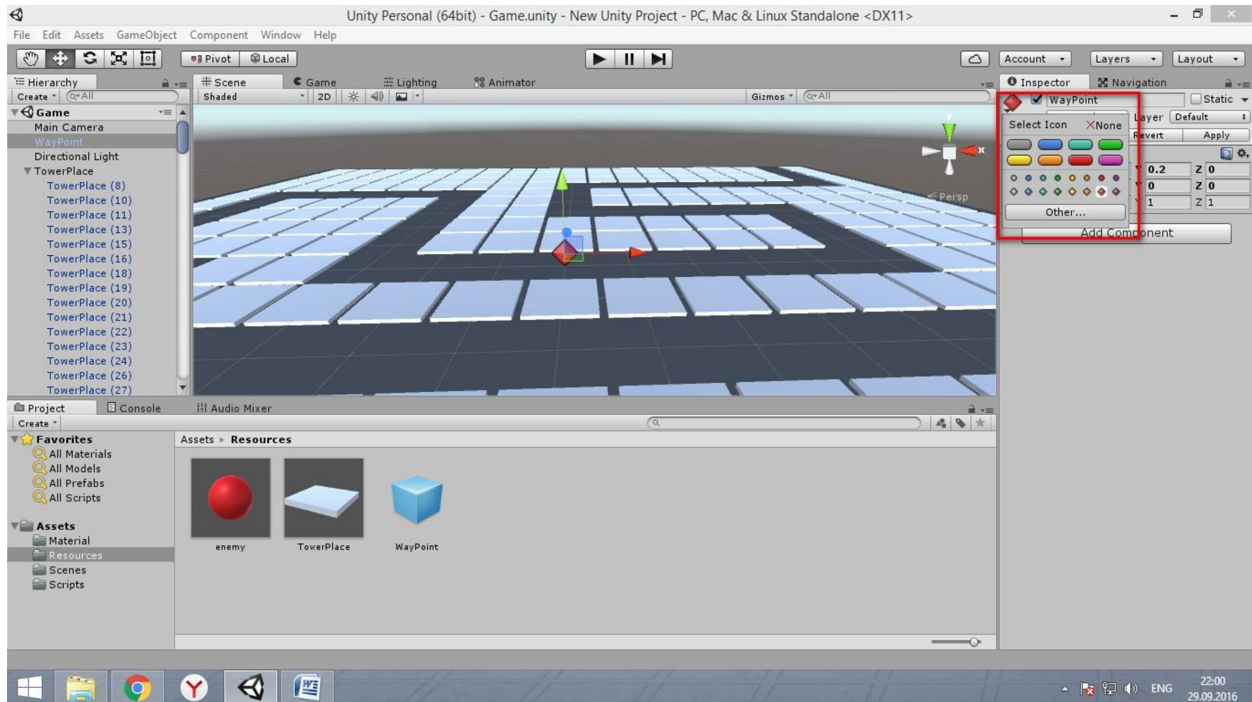


Рисунок 19 Створення чекпоінтів

Збережемо його в префаб і розмістимо наші вайпоінти на порожній доріжці, на краях повороту (дублювати CTR+D, переміщати із клавішею CTR). Перенесемо всі поїти в порожній об'єкт weapoints.

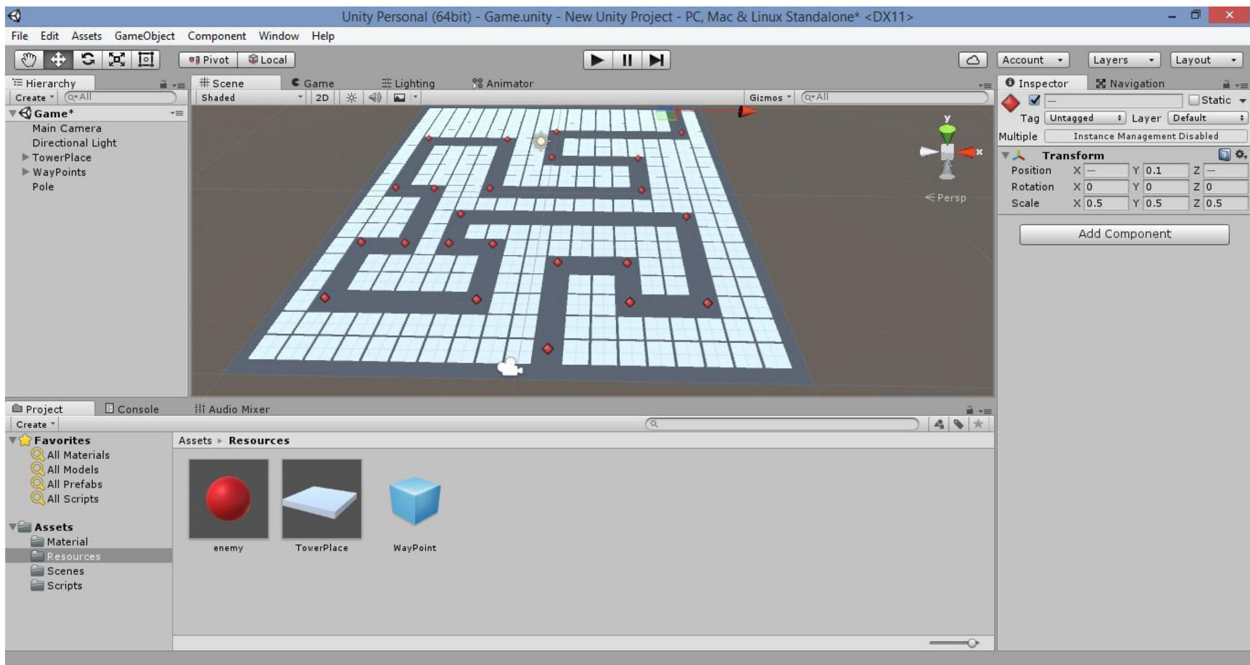


Рисунок 20 Розміщення чекпоінтів

Створимо куб, який створюватиме ботів на сцені. Ставимо його на початок шляху роботів і прибираємо у нього box collider, назвемо його Start.

Продублюємо цей куб і розмістимо його наприкінці шляху ботів (можна застосувати на куби різні матеріали), назвемо його End.

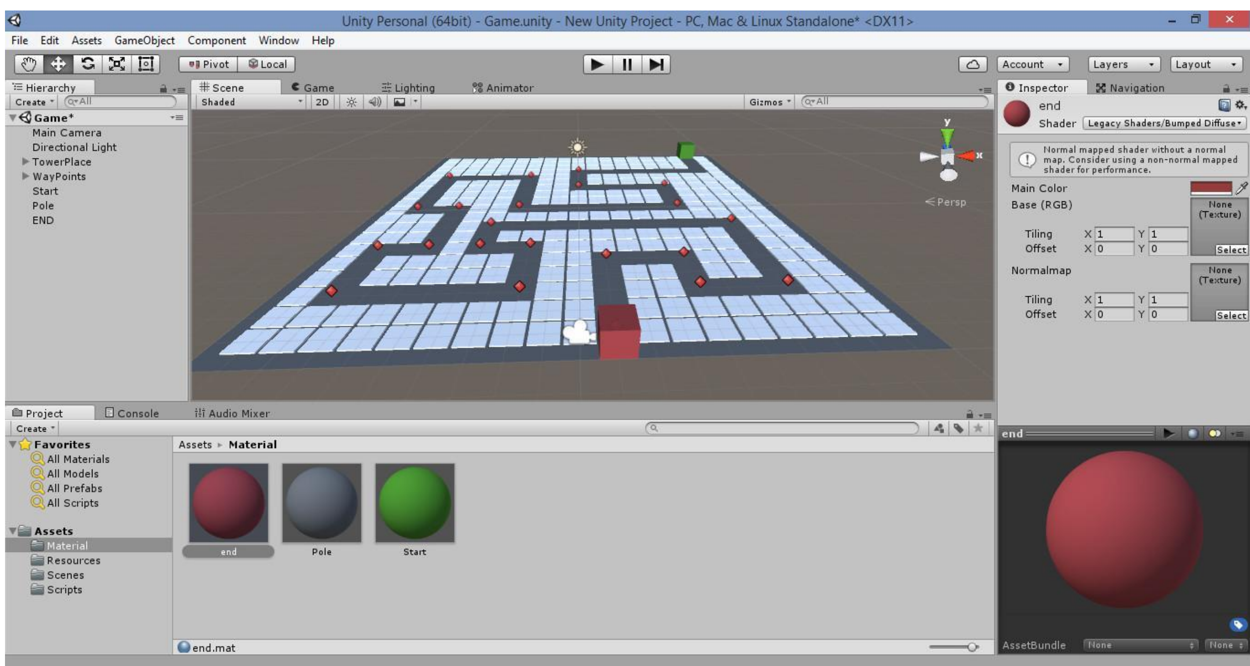


Рисунок 21 Створення початку кінця маршруту ботів

Створіть сферу на сцені та застосуйте до неї будь-який матеріал. Додати компонент rigidbody. Перемістіть сферу в префаб та видаліть зі сцени. Це буде наш супротивник, який рухатиметься сценою і якого має знищити вежа.

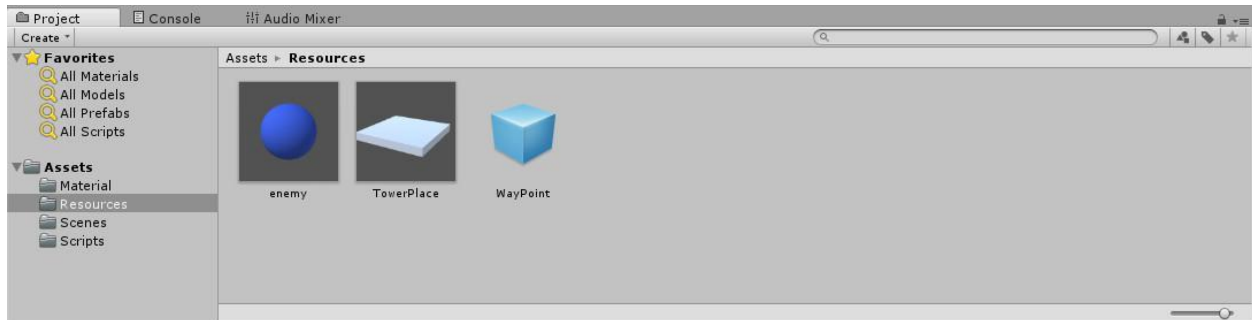


Рисунок 22 Створення колайдера супротивника

На цьому підготовка сцени закінчена, переходимо до скриптів.

У папці Scripts створюю новий c# скрипт spawner і переміщую його на об'єкт Start. Відкриваю скрипт і пишу такий код:

```
using UnityEngine;

using System.Collections;


public class spawner : MonoBehaviour

{
 public Transform EnemyOrefab;//образец объекта для клонирования
 public Transform spawnPoint;
 public float timeBetweerWaves = 5f;// время через которое произойдет создание нового объекта
 private float countdown = 3f;
 private int waveIndex = 0;
 void Update()
 {
  if (countdown <= 0f) {
  StartCoroutine(SpawnWave());
```

```
  countdown = timeBetweerWaves;
  }
 countdown -= Time.deltaTime;// с каждым кадром переменная будет уменьшатся
 }
 IEnumerator SpawnWave(){
 waveIndex++;
 for (int i = 0; i < 10; i++) { // после каждого появления объекта к нему будет добавляться еще один объект
 i = Random.Range(1, 10);
 SpawnEnemy ();
 yield return new WaitForSeconds (0.3f); // через определенное время
 }
}

void SpawnEnemy () {
 Instantiate (EnemyOrefab, spawnPoint.position, spawnPoint.rotation); // создание объекта на сцене
 }
}
```

Цей скрипт буде відповідати за появу супротивника на сцені, відстань між об'єктами і за кількість об'єктів, що з'явилися з початкової точки.

Створюю скрипт Enemy і переношу його на префаб противника (Enemy), відкриваю його та напишу такий код

```
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {

 public float speed = 10f; //скорость движения
 private Transform target;
 private int wavepointIndex =0;
```

```
void Start (){
target = weapoints.points [0]; // указываем наш массив с поинтами
}

void Update(){
Vector3 dir = target.position - transform.position; // движение объекта к каждому поинту
transform.Translate (dir.normalized * speed * Time.deltaTime, Space.World);
if (Vector3.Distance (transform.position, target.position) <= 0.3f) { // если дистанция до точки меньше 0.3
GetNextWaypoint ();
}
}
void GetNextWaypoint () {
if (wavepointIndex >= weapoints.points.Length - 1) { // и если значение точки меньше значения точки из скрипта weapoints (который мы создадим позже)
Destroy (gameObject); // то удаляем объект
return; // делаем повтор
}
wavepointIndex++;
target = weapoints.points [wavepointIndex];
}
}
```

Цей скрипт буде відповідати за рух супротивника по точках weapoints, швидкість руху та видалення зі сцени.

На об'єкті Start вказую в полі spawn object префаб Enemy та в полі Spawn point вказую сам Start.

Рисунок 23 Налаштування обьектів

Створюю ще один скрипт і назвемо його weapoints, переношу його відразу на об'єкт weapons на сцені (група наших поінтів). Відкрию скрипт і напишу код:

```
using UnityEngine;
 using System.Collections;

  public class weapoints : MonoBehaviour {

  public static Transform[] points; // здесь просто указывается
массив с нашими точками
```

```
void Awake(){
  points = new Transform[transform.childCount];
  for (int i = 0; i < points.Length; i++) {
  points[i]= transform.GetChild (i);
   }
  }
}
```

На цьому етапі вже можна запустити сцену, але слід не забувати перед цим додати сцену до налаштувань проекту File - Built settings і зберегти сцену.

Також слід налаштувати камеру як зручно для перегляду сцени

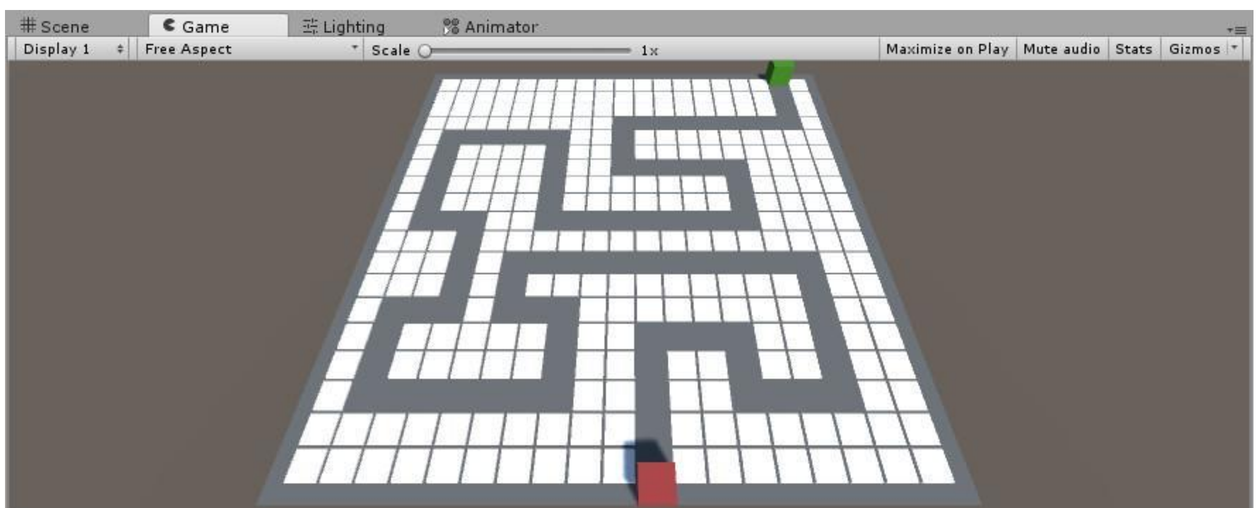


Рисунок 24 Тестовий варіант сцени

Можна запустити проект.

Для того, щоб вбудувати внутрішньоігрові покупки в Unity необхідно скористатися підтримкою внутрішньоігрових покупок без сторонніх плагінів. Для цього необхідно відкрити вкладку Services, вона знаходиться в меню Window>General>Services.

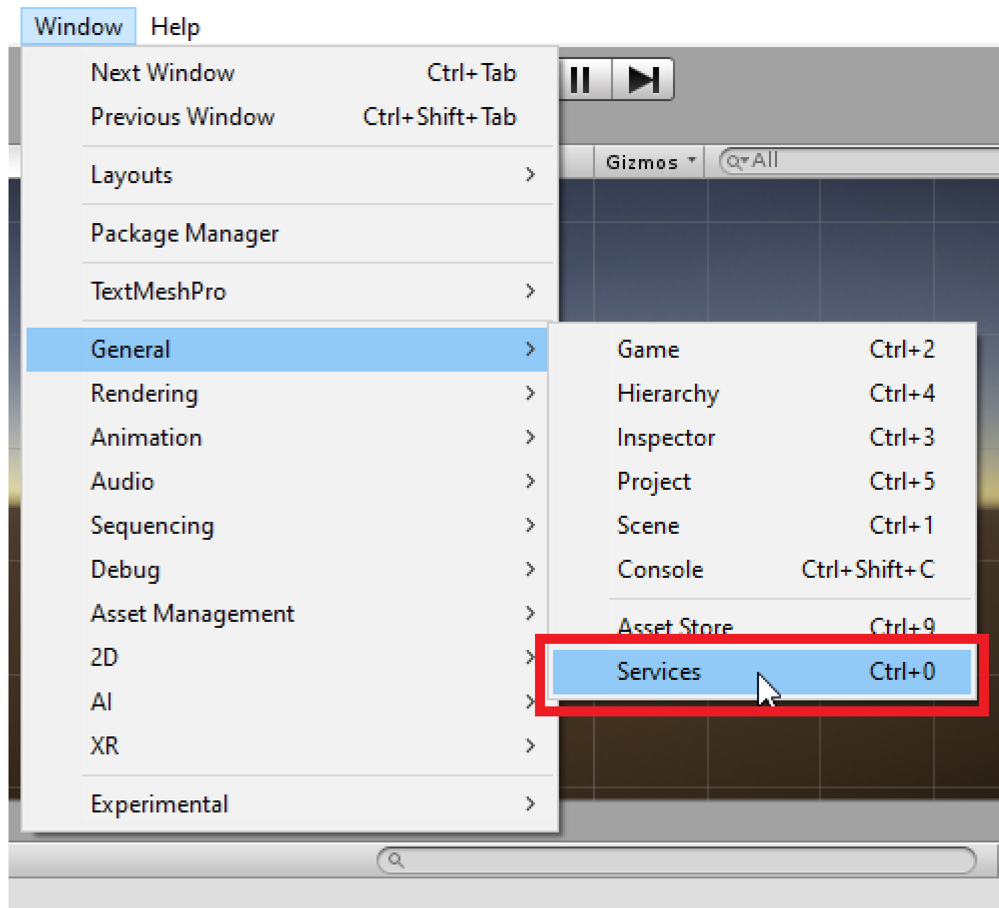


Рисунок 25 Додавання внутрішньоігрового магазину

У вікні натискаємо на розділ In-App Purchasing. Далі натискаємо перемикач або тиснемо на кнопку Enable.

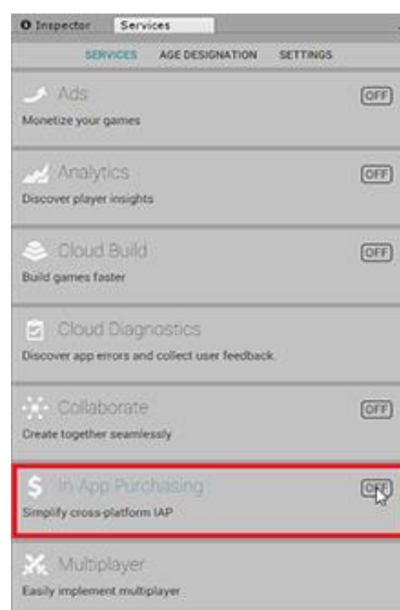


Рисунок 26 Додавання внутрішньоігрового магазину

Клацаємо по кнопці імпорт, щоб завантажити файли плагіна у проект.

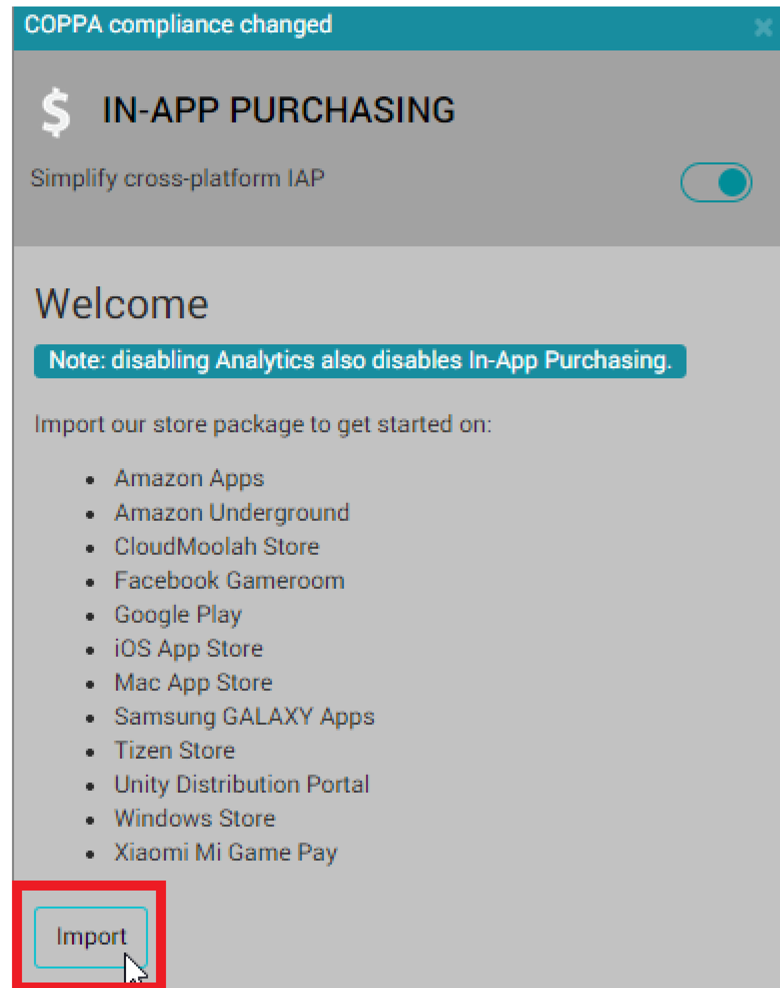


Рисунок 27 Додавання плагіну магазину

Для того, щоб покупки відображалися в Unity Analitics, необхідно вставити API ключ програми з Google Play. Отримати його можна в консолі розробника Google у вкладці Монетизація->Налаштування монетизації (якщо ви використовуєте стару версію консолі Інструменти розробки->Служби та API).

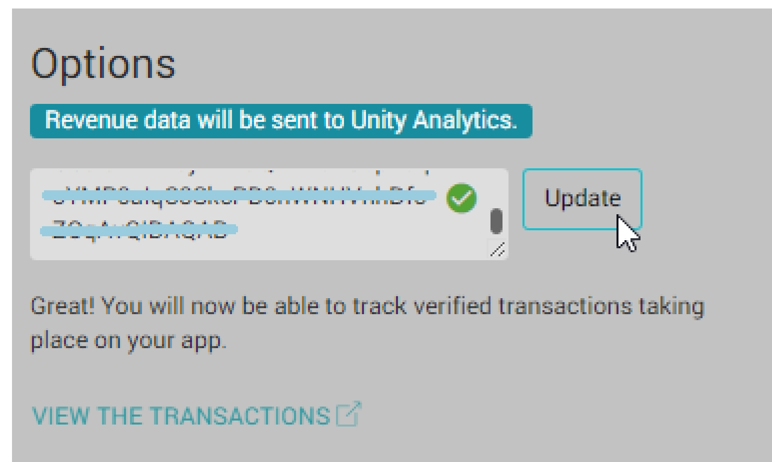


Рисунок 28 Підключення монетизації

Далі потрібно зайти до каталогу покупок Window->Unity IAP->IAP Catalog для створення ідентифікаторів покупок.

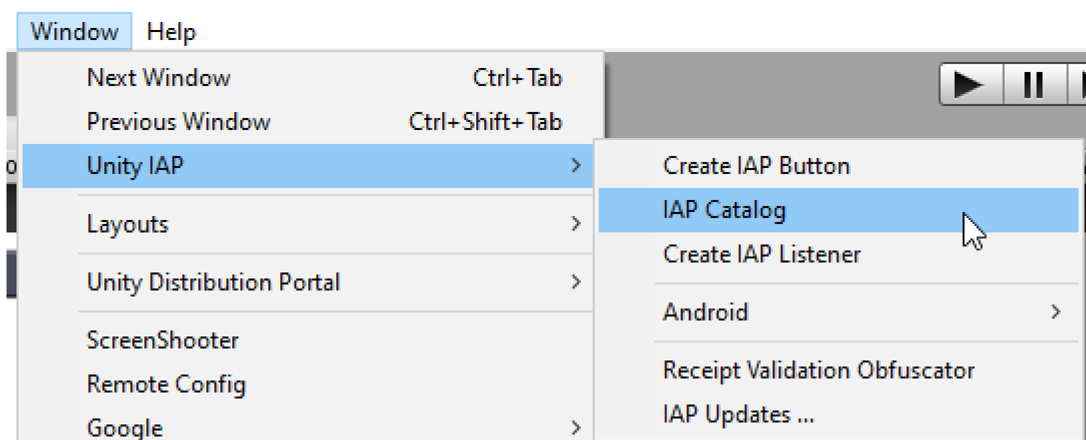


Рисунок 29 Створення ідентифікаторів покупок

У вікні IAP Catalog для створення покупки натискаємо кнопку Add Product та вводимо ID та вибираємо тип покупки. Це може бути consumable, non consumable або Subscription (витрачені, не витрачаються та передплата).

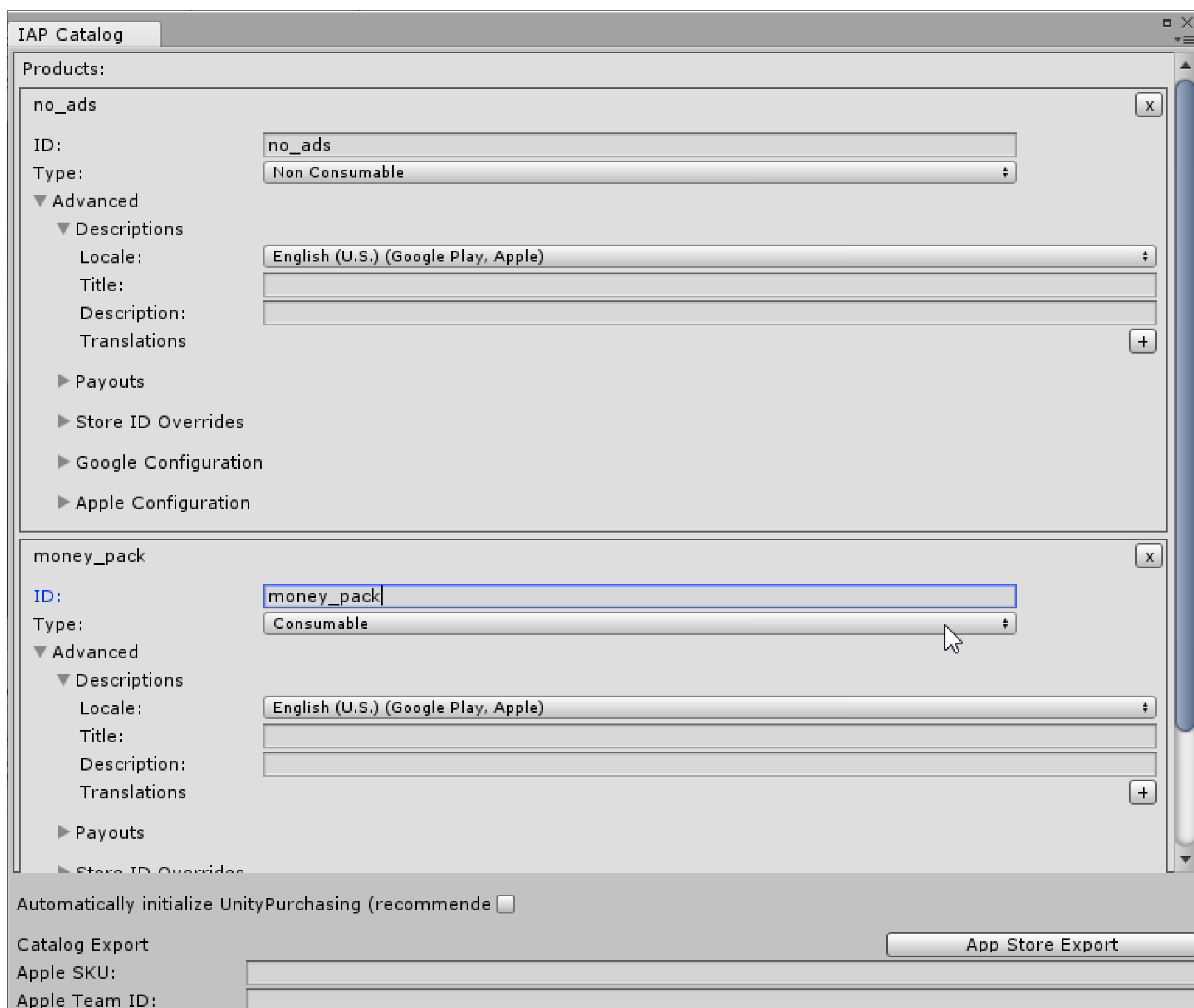


Рисунок 30 Створення ідентифікаторів покупок

Також не забудьте додати покупки з такими ж ідентифікаторами в консолі розробника AppStore і GooglePlay.

3.5 Тестування гри

Важливим етапом створення будь-якої програми є її тестування

Тестування це процес перевірки ПЗ на відповідність до тих функцій і завдань для якого воно створювалося.

Методи тестування зазвичай класифікують за такими ознаками:

- По об'єкту тестування
- За знанням внутрішньої будови системи
- за ступенем автоматизації та інші.

До тестування по об'єкту відносять такі методи як функціональне тестування, тестування продуктивності, юзабіліті-тестування.

У своїй роботі я використав саме тестування по об'єкту. для отримання розуміння чи виконує програма свої функції, чи зручно нею користуватися і наскільки потужною повинна бути система що дана програма могла працювати без нарікань.

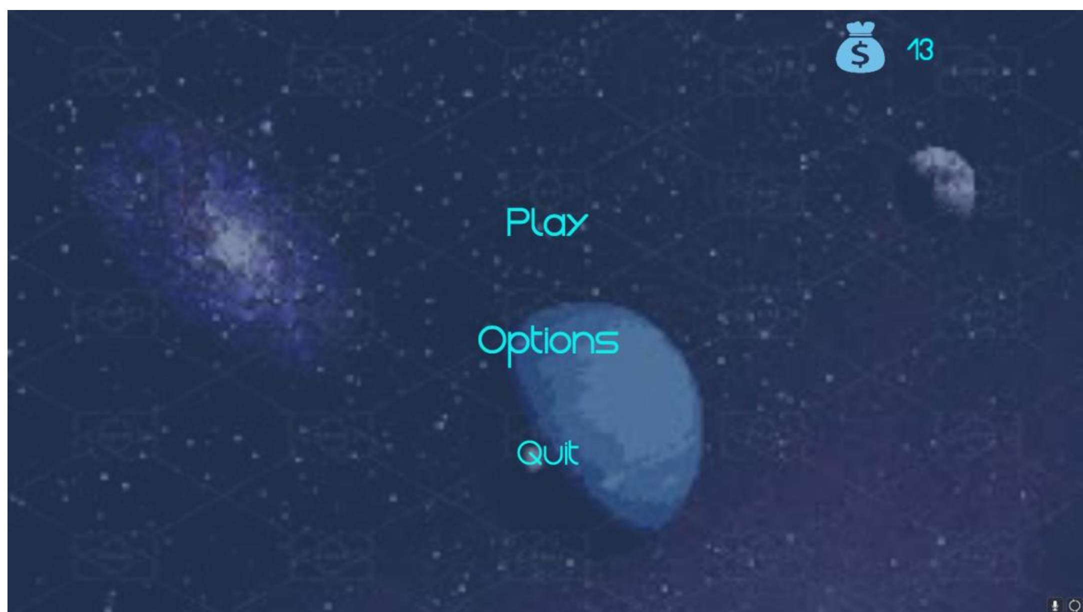


Рисунок 31 Головне меню

Після запуску гри нас зустрічає інтуїтивно зрозумілий інтерфейс. В якому ми можемо почати гру, зайти до опцій або вийти із гри.

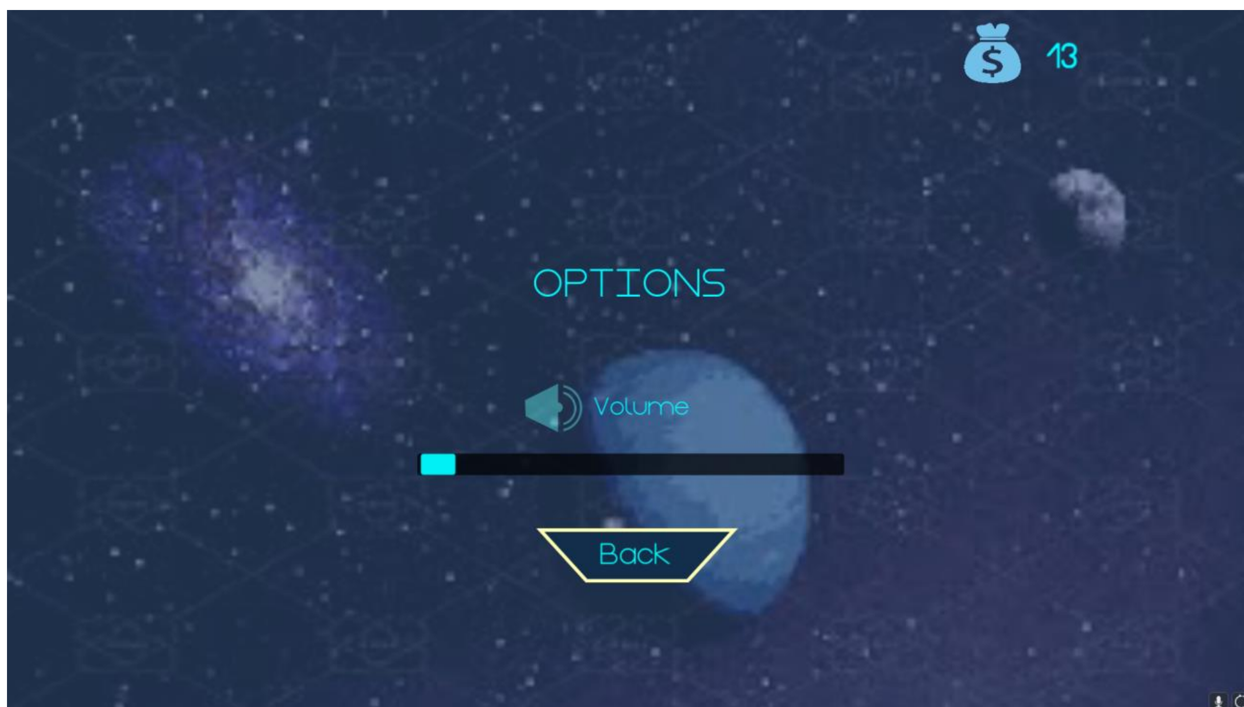


Рисунок 32 Меню опцій

У меню опцій розміщується повзунок гучності музики.

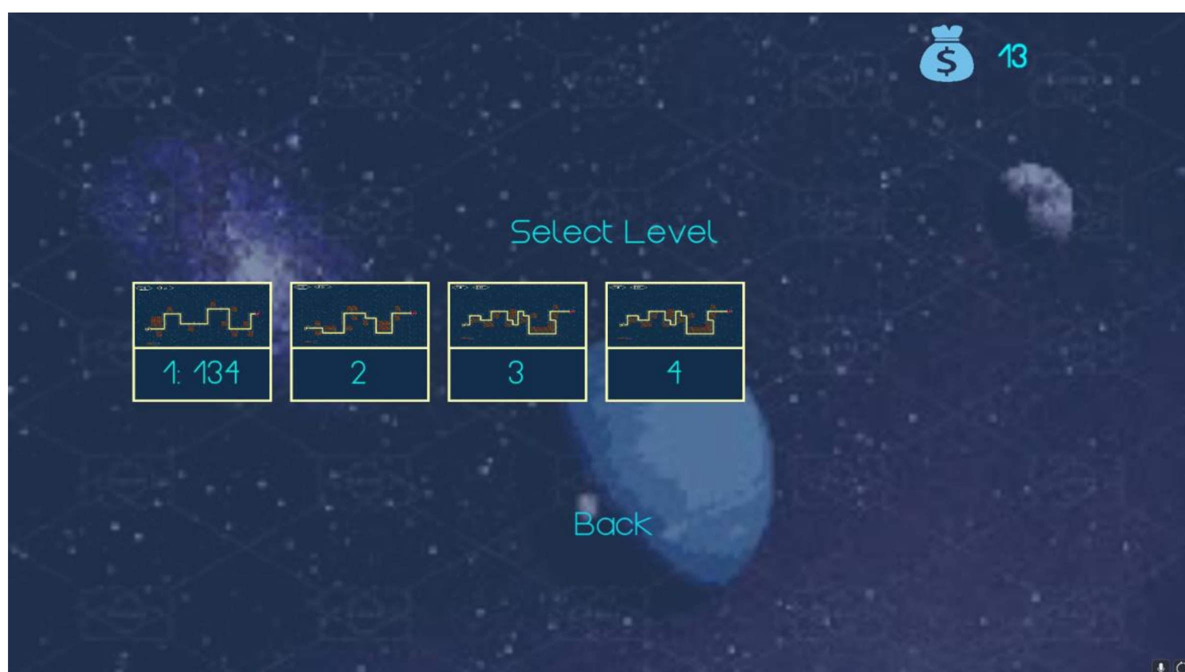


Рисунок 33 Меню вибору рівня

Після натискання кнопки грати ми потрапляємо до меню вибору рівня. При натисканні на рівень він запуститися

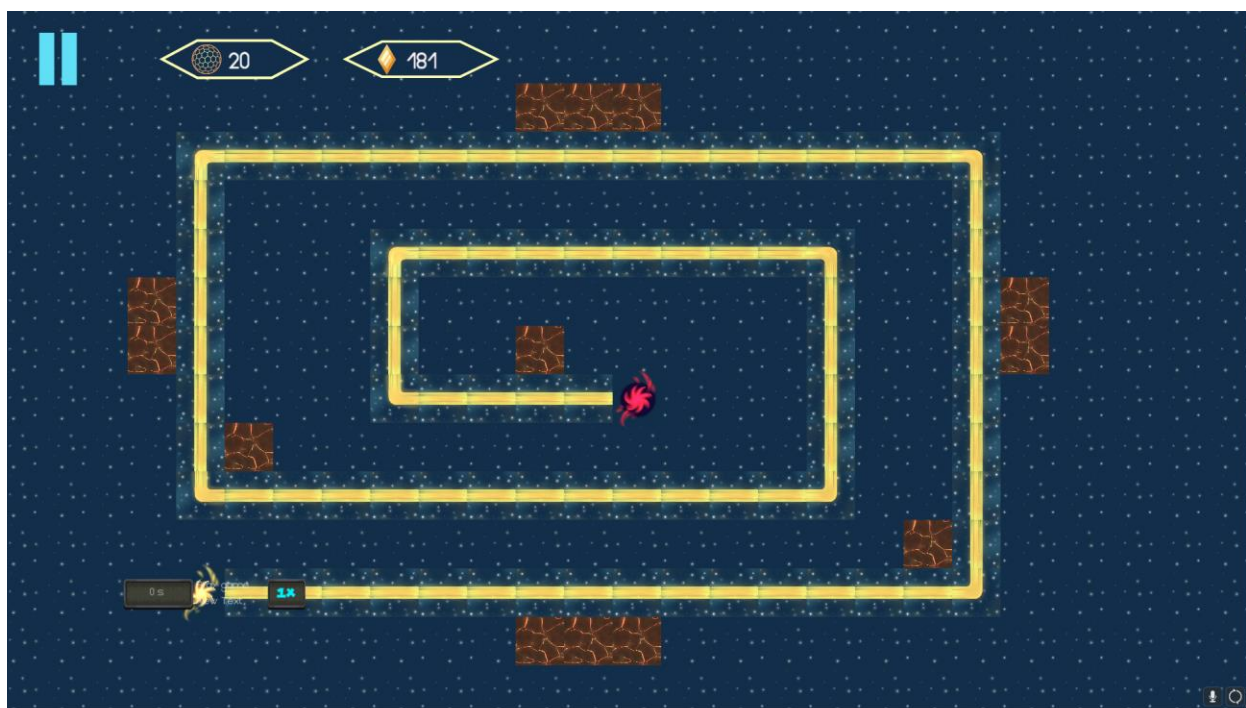


Рисунок 34 Перший рівень

При натисканні на одну з платформ справа з'явиться меню з вибором вежі та можливістю її покращення.

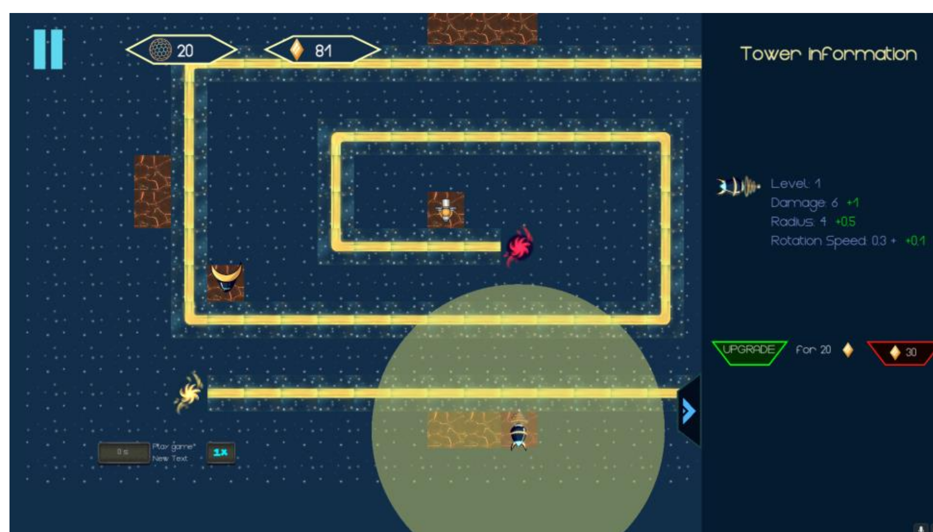


Рисунок 35 Вибір веж та їх покращення

Після закінчення етапу підготовки при натисканні кнопки старт починають з'являються вороги. Башти знищують їх.

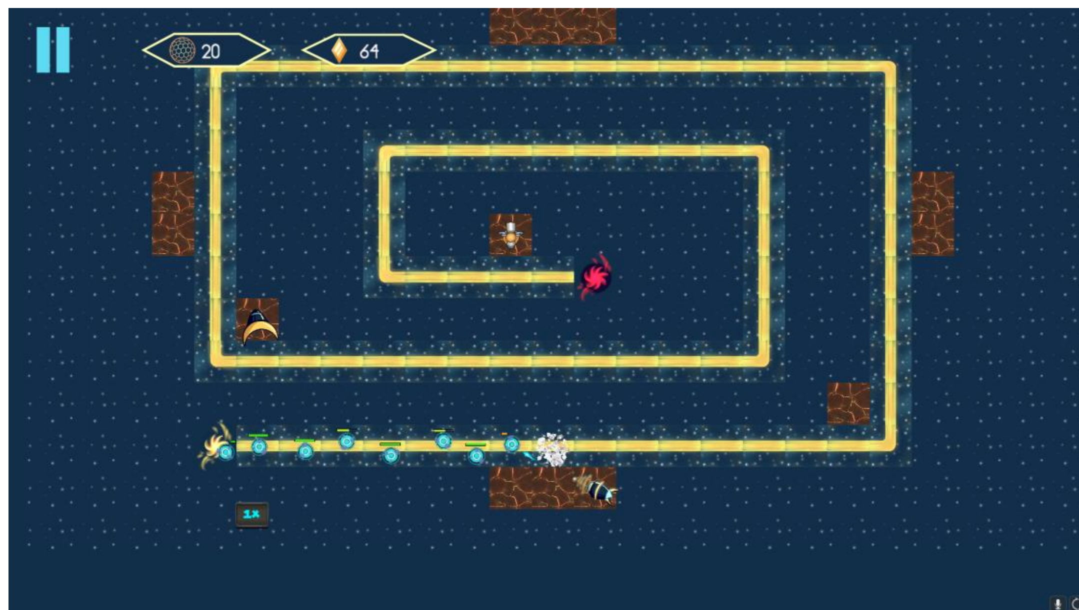


Рисунок 36 Робота кнопки початку хвилі здоров'я супротивників та стрільби веж

Рисунок 37 Зміна кількості життів гравця

Якщо противники все ж таки змогли дістатися до кінця маршруту, кожен ворог зменшує кількість життів гравця на 1.

Рисунок 38 Екран програшу

У разі зменшення кількості життів до 0 з'являється меню в якому можна вийти до екрана вибору рівня або почати з початку

Висновки

Робота присвячена розробці гри «3PER» в жанрі Tower Defense на Unity.

Проведено дослідження котрі обґрунтовують актуальність роботи та наукову новизну. А саме:

- Наведено теоретичні аспекти розробки ігор
- Надано основні терміни та поняття
- Наведено основні технології створення ігор
- Наведено основні апаратні вимоги до системи для розробки ігор
- Наведено основні переваги та недоліки систем розробки ігор
- Наведено основні переваги та недоліки Unity
- Наведено характеристику аналогів гри «3PER» в жанрі Tower Defense
- Поставлено ТЗ
- Наведено основні можливості системи unity
- Наведено характеристику розробки гри «3PER» в жанрі Tower Defense на Unity.

Враховуючи переваги та недоліки існуючих застосунків, було проаналізовано вимоги та спроектовано нову гру у жанрі Tower Defence. Спроектований додаток відповідає усім виявленим вимогам.

Проведено аналіз існуючих програмних засобів для розробки додатків для Android обрано оптимальні. Таким чином для розробки додатку було обрано систему Unity.

Розроблено програмний додаток дає можливість запускати його та виконувати найпростіші ігрові функції. У подальшому можливе доопрацювання додатку та його використання у комерційних цілях

Перелік використаної літератури

1. Шмаков С.А. РОЗДІЛ III ПРИРОДА І СТРУКТУРА ІГРИ// Ігри учнів - феномен культури. - М.: Нова школа, 1994. - 240 с.
2. Азаров Ю.П. Мистецтво любити дітей. - М., 1987. - С. 72.
3. Шмаков С.А. РОЗДІЛ III ПРИРОДА І СТРУКТУРА ІГРИ// Ігри учнів - феномен культури. - М.: Нова школа, 1994. - 240 с.
4. Donovan, Tristan. Replay, The History of Video Games: [англ.]. - Yellow Ant, 2010.
5. Jason, Григорій. Game Engine Architecture: [англ.]. - CRC Press, 2009. - 864 с.
6. Хокінг, Джозеф. Unity – у дії. Мультиплатформенна розробка на C#: [укр.]. – 2. – СПб: Пітер, 2016. – 336 с.
7. Торн, Алан. Мистецтво створення сценаріїв у Unity: [укр.]. - СПб: ДМК, 2016. - 362 с.
8. Bibby, Jay Flash Element TD (Tower Defense). Jay Is Games (1 листопада 2007).
9. https://newzoo.com
10. Паласіос, Хорхе Unity 5.x. Програмування штучного інтелекту в іграх/Хорхе Паласіос. – М.: ДМК Прес, 2016. – 849 с.
11. Ліновес Д. Віртуальна реальність в Unity - М., ДМК Прес, 2015. - 194 с.
12. Хокінг Д. Unity в дії. Спб, Пітер, 2020 - 352 с.
13. Гейг М. Розробка ігор на Unity 2018 за 24 години, М., Ексмо, 2020 р. – 464 с.
14. Бонд Д. Г. Unity та C#. Геймдев від ідеї до реалізації, СПб., Пітер, 2020 р. – 928 с.
15. Батфілд-Еддісон П. Unity для розробника. Мобільні мультиплатформні ігри, СПб., Пітер, 2018. – 304 с.
16. Денисов Д. Розробка гри на Unity. З нуля і до реалізації, Автор, 2021. – 325 с.
17. Ламмерс К. Шейдери та ефекти в Unity. Книга рецептів, ДМК-Прес, 2016 р. – 274 с.
18. Торн А. Основи анімації в Unity, ДМК-Прес, 2019 р. – 176 с.

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри «3PER» в жанрі Tower Defense на Unity

Виконав студент 4 курсу
групи ПД-44
Мурзін Антон Олександрович
Керівник роботи
Дібрівний Олесь Андрійович

Київ – 2022

АНАЛОГИ

Назва	Переваги	Недоліки
Kingdom Rush	• Гарний дизайн	• Непродуманий дизайн рівнів
	• Можливість гравця використати спеціальні здібності	• Затягнуті рівні
CITYWARS TOWER DEFENSE	• Незвичайна схема збирання ресурсів	• Однотипні супротивники
	• Безпосередня участь гравця на полі бою	• Не цікава схема бою

2

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - Підвищення ефективності розробки гри в жанрі Tower Defense
- **Об'єкт дослідження –** Підвищення функціональності та продуктивності ігор на основі двигуна Unity.
- **Предмет дослідження -** Технології проектування та створення гри на основі двигуна Unity

3

ТЕХНІЧНІ ЗАВДАННЯ

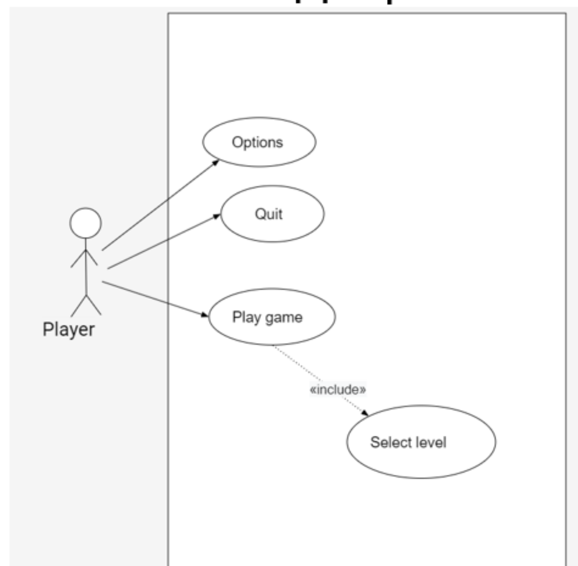
- Розробити гру в жанрі Tower Defence
- Підвищити функціональность та продуктивность ігор на основі двигуна Unity
- Розробити діаграму класів та діаграму use case

4

ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ РЕАЛІЗАЦІЇ

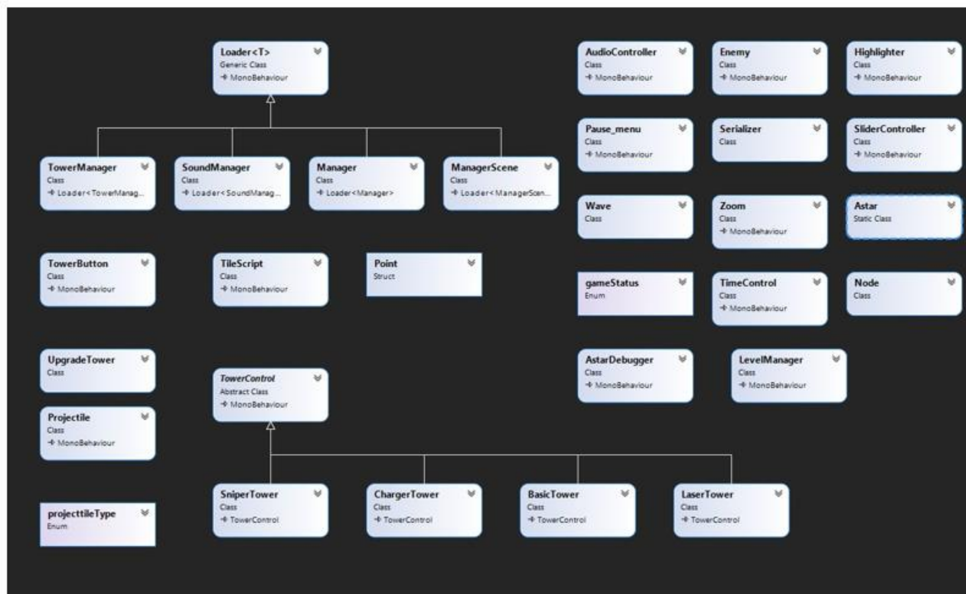
5

USE CASE Діаграмма



6

ДІАГРАМА КЛАСІВ



7

ГРАФІЧНИЙ ІНТЕРФЕЙС

8

ВИСНОВКИ

- В данній роботі було розроблено гру в жанри Tower Defence

- У подальшому можливе доопрацювання додатку та його використання у комерційних цілях.

- Підвищенно функціональность та продуктивность ігор на основі двигуна Unity за допомогою використання аддонів

9

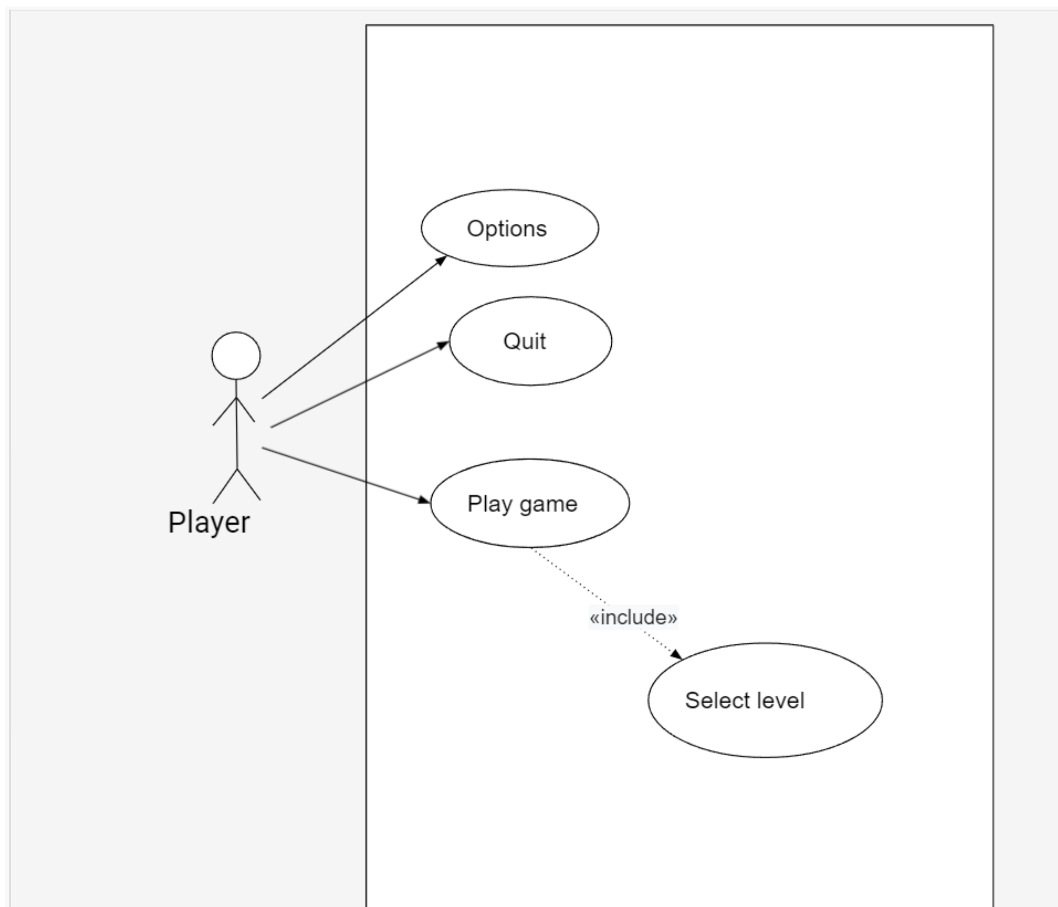
АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Мурзін А.О., ОСОБЛИВОСТІ РОЗРОБКИ ІГОР ЗА ДОПОМОГОЮ UNITY, Науково-технічна конференція «Застосування програмного забезпечення в ІКТ», К.: ДУТ, 2022
- Мурзін А.О., ОСОБЛИВОСТІ РОЗРОБКИ ІГОР ЗА ДОПОМОГОЮ UNITY, Науково-технічна конференція «Застосування програмного забезпечення в ІКТ», К.: ДУТ, 2022

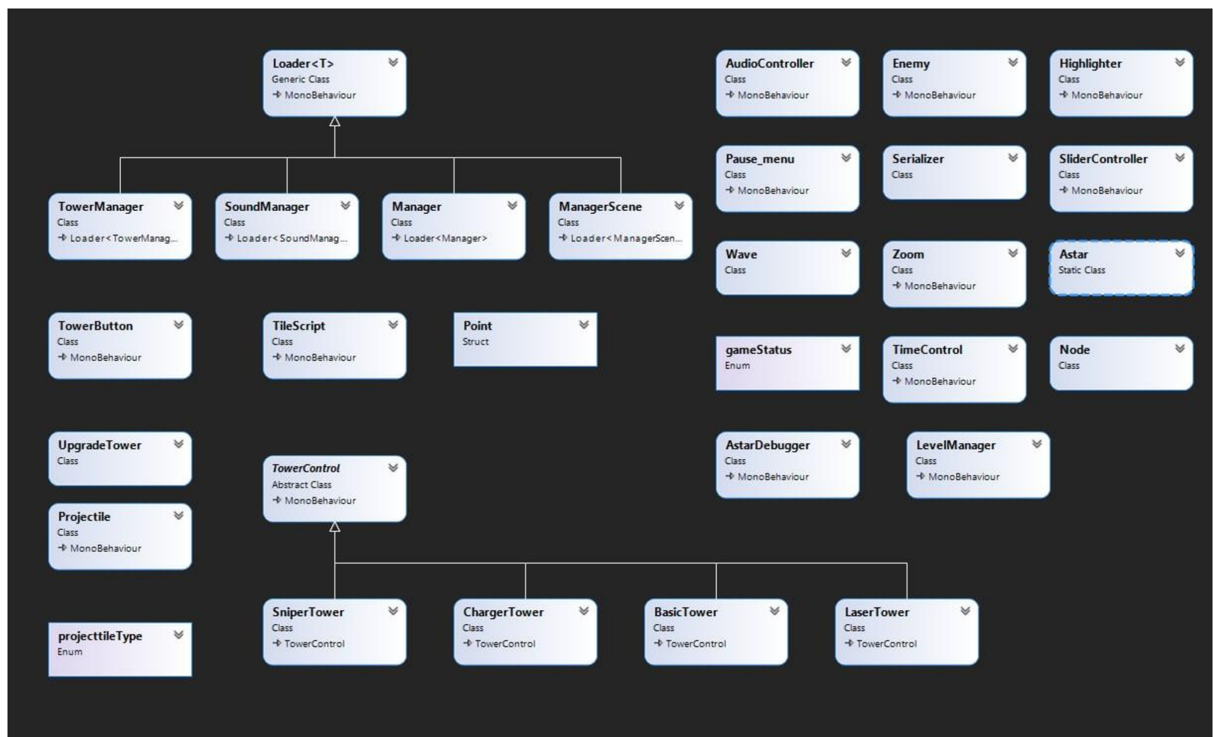
10

ДЯКУЮ ЗА УВАГУ!

Додаток Б



Use case діаграма



Діаграмма класів

Додаток В

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

public static class Astar
{
    private static Dictionary<Point,Node> nodes;

    private static void CreateNodes()
    {
        nodes = new Dictionary<Point,Node>();
        foreach (TileScript tile in ManagerScene.Instance.Tiles.Values)
        {
            nodes.Add(tile.GridPosition, new Node(tile));
        }
    }

    public static Stack<Node> GetPath(Point start,Point goal)
    {
        if (nodes == null)
        {
            CreateNodes();
        }
        HashSet<Node> openList = new HashSet<Node>();
        HashSet<Node> closedList = new HashSet<Node>();
        Stack<Node> finalPath=new Stack<Node>();
        Node currentNode = nodes[start];
        openList.Add(currentNode);
        while(openList.Count > 0)
        {
            for(int x=-1;x<=1;x++)
            {
                for(int y=-1;y<=1;y++)
                {
                    Point neighbourPos = new Point(currentNode.GridPosition.X - x,
currentNode.GridPosition.Y-y);
                    if(ManagerScene.Instance.InBounds(neighbourPos) &&
ManagerScene.Instance.Tiles[neighbourPos].WalkAble && neighbourPos!=currentNode.GridPosition)
                    {
                        int gCost = 0;
                        if(Math.Abs(x-y)==1)
                        {
                            gCost=10;
                        }
                        else
                        {
                            if(!ConnectedDiagonally(currentNode,nodes[neighbourPos]))
                            {
                                continue;
                            }
                            gCost=14;
                        }
                        Node neighbour = nodes[neighbourPos];

                        if (openList.Contains(neighbour))
                        {
                            if (currentNode.G +gCost<neighbour.G)
                            {
```

```
                                neighbour.CalcValues(currentNode,nodes[goal],gCost);
                            }
                        }
                        else if(!closedList.Contains(neighbour))
                        {
                            openList.Add(neighbour);
                            neighbour.CalcValues(currentNode,nodes[goal],gCost);

                        }
                    }
                }
            }
            openList.Remove(currentNode);
            closedList.Add(currentNode);

            if (openList.Count>0)
            {
                currentNode=openList.OrderBy(n=>n.F).First();
            }
            if(currentNode == nodes[goal])
            {
                while (currentNode.GridPosition != start)
                {
                    finalPath.Push(currentNode);
                    currentNode=currentNode.Parent;
                }

                break;
            }
        }

        // onlu debug
        //
GameObject.Find("Debugger").GetComponent<AstarDebugger>().DebugPath(openList,closedList,finalPath);
        return finalPath;
    }
    private static bool ConnectedDiagonally(Node currentNode,Node neighbour)
    {
        Point direction = neighbour.GridPosition - currentNode.GridPosition;
        Point first = new Point(currentNode.GridPosition.X -
direction.X,currentNode.GridPosition.Y+direction.Y);
        Point second = new Point(currentNode.GridPosition.X,currentNode.GridPosition.Y+direction.Y);
        Point three = new Point(currentNode.GridPosition.X,currentNode.GridPosition.Y-direction.Y);
        if(ManagerScene.Instance.InBounds(first) && !ManagerScene.Instance.Tiles[first].WalkAble)
        {
            return false;
        }
        if(ManagerScene.Instance.InBounds(second) && !ManagerScene.Instance.Tiles[second].WalkAble)
        {
            return false;
        }
        if(ManagerScene.Instance.InBounds(three) && !ManagerScene.Instance.Tiles[three].WalkAble)
        {
            return false;
        }
        return true;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AstarDebugger : MonoBehaviour
{
    [SerializeField]
    private TileScript start,goal;
    // Start is called before the first frame update
```

```
void Start()
{

}

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

//     if(Input.GetKeyDown(KeyCode.Space))
//     {
//         Astar.GetPath(start.GridPosition,goal.GridPosition);
//     }
// }
private void ClickTile()
{
    if (Input.GetMouseButtonDown(1))
    {
        Vector2 mousePoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);
                  RaycastHit2D hit = Physics2D.Raycast(mousePoint, Vector2.zero);
        if (hit.collider !=null)
        {
            TileScript tmp=hit.collider.GetComponent<TileScript>();
            if(tmp!=null)
            {
                if(start ==null)
                {
                    start=tmp;
                    start.SpriteRenderer.color=new Color32(255,132,0,255);
                }
                else if(goal ==null)
                {
                    goal=tmp;
                    goal.SpriteRenderer.color=new Color32(255,0,0,255);
                }
            }
        }
    }
}
public void DebugPath(HashSet<Node> openList,HashSet<Node> closedList,Stack<Node> path)
{
    foreach(Node node in openList)
    {
        if (node.TileRef !=start)
        {
            node.TileRef.SpriteRenderer.color=Color.cyan;
        }
        // PointToParent(node,node.TileRef.WorldPosition);
    }
    // foreach(Node node in closedList)
    // {
    //     if (node.TileRef !=goal)
    //     {
    //         node.TileRef.SpriteRenderer.color=Color.red;
    //     }
    //     // PointToParent(node,node.TileRef.WorldPosition);
    // }
    foreach (Node node in path)
    {
        if (node.TileRef !=goal && node.TileRef !=start)
        {
            node.TileRef.SpriteRenderer.color=Color.green;
        }
    }
}
private void PointToParent(Node node,Vector2 position)
```

```
    {

    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class Node
{
    public Point GridPosition{get;private set;}
    public TileScript TileRef {get;private set;}
    public Vector2 WorldPosition{get;set;}
    public Node Parent{get;private set;}
    public int G { get; set; }
    public int H { get; set; }
    public int F { get; set; }
    public Node(TileScript tileRef)
    {
        this.TileRef=tileRef;
        this.GridPosition=tileRef.GridPosition;
        this.WorldPosition=tileRef.WorldPosition;
    }
    public void CalcValues(Node parent,Node goal,int gCost)
    {
        this.Parent=parent;
        this.G=parent.G+gCost;
        this.H=(Math.Abs(GridPosition.X-goal.GridPosition.X)+Math.Abs(goal.GridPosition.Y -
GridPosition.Y)) *10;
        this.F=G+H;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Xml;
using System.Text;
using System.IO;

public class ManagerScene : Loader<ManagerScene>
{
    [SerializeField]
    private GameObject[] tilePrefabs;
    private XmlElement xRoot;
    public int levelIndex;
    [SerializeField]
    private Transform map;
    public Point spawnPoint;
    public Point finishPoint;
    public TileScript spawn;
    private Stack<Node> path;
    public Stack<Node> Path
    {
        get
        {
            if (path == null)
            {
                GeneratePath();
            }
            return new Stack<Node>(new Stack<Node>(path));
        }
    }
    private Point mapSize;
    public Dictionary<Point,TileScript> Tiles{get;set;}
    public float TileSize
```

```
    {
        get
        {
            return tilePrefabs[0].GetComponent<SpriteRenderer>().sprite.bounds.size.x;
        }
    }
    // Start is called before the first frame update
    void Start()
    {
        levelIndex = PlayerPrefs.GetInt("levelIndex");
        ReadXML();
        CreateLevel();
        SetLevelParam();
    }

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

    }
    private void CreateLevel()
    {
        Tiles = new Dictionary<Point,TileScript>();
        string[] mapData= ReadLevel();

        mapSize = new Point(mapData[0].ToCharArray().Length,mapData.Length);

        int mapX=mapData[0].ToCharArray().Length;
        int mapY=mapData.Length;
        Vector3 worldStart= Camera.main.ScreenToWorldPoint(new Vector3(0,Screen.height));
        for(int y=0;y<mapY;y++)
        {
            char[] newTiles=mapData[y].ToCharArray();
            for(int x=0;x<mapX;x++)
            {
                PlaceTile(newTiles[x].ToString(),x,y,worldStart);
            }
        }
        // Instantiate(tile);
    }
    public bool DefWalAble(int indexTile)
    {
        int[] array = { 0,7 };
        if (!Array.Exists(array, v => v == indexTile))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    private void PlaceTile(string tileType,int x,int y,Vector3 worldStart)
    {
        int tileIndex=int.Parse(tileType);
        bool WalkAble=DefWalAble(tileIndex);
        TileScript newTile=Instantiate(tilePrefabs[tileIndex]).GetComponent<TileScript>();
        newTile.Setup(new Point(x,y),new Vector3(worldStart.x+ (TileSize*x),worldStart.y -
(TileSize*y),0),map,WalkAble);
        if (tileIndex==8)
        {
            spawnPoint=new Point(x,y);
            spawn=newTile;
        }
        if(tileIndex == 9)
        {
            finishPoint=new Point(x,y);
```

```
        }
    }
    private string[] ReadLevel()
    {

        string data =  xRoot.SelectSingleNode("map").InnerText;
        data =
data.Replace("\t",string.Empty).Replace("\n",string.Empty).Replace("\r",string.Empty);
        return data.Split('-');
    }
    private void ReadXML()
    {

        TextAsset textAsset = (TextAsset) Resources.Load(@"levels/level"+levelIndex.ToString());
        string xmlData=textAsset.text;
        string msg_xml = Encoding.UTF8.GetString(Encoding.UTF8.GetPreamble());
        if (xmlData.StartsWith(msg_xml))
        {
            xmlData = xmlData.Remove(0, msg_xml.Length-1);
        }
        XmlDocument xmldoc = new XmlDocument ();
        xmldoc.LoadXml(xmlData);
        xRoot = xmldoc.DocumentElement;
    }
    private void SetLevelParam()
    {
        int
health=int.Parse(xRoot.SelectSingleNode("int[@name='health']").Attributes["value"].Value);
        Manager.Instance.TotalHealth=health;
        Manager.Instance.Health=health;
        int money=int.Parse(xRoot.SelectSingleNode("int[@name='money']").Attributes["value"].Value);
        Manager.Instance.TotalMoney=money;
    }
    public Wave[] SetWaves()
    {
        var waves=xRoot.SelectSingleNode("waves");
        List<Wave> levelWaves=new List<Wave>();
        foreach (XmlNode waveNode in waves.ChildNodes)
        {
            int indexEnemy =
int.Parse(waveNode.SelectSingleNode("int[@name='enemy']").Attributes["value"].Value);
            int
density=int.Parse(waveNode.SelectSingleNode("int[@name='density']").Attributes["value"].Value);
            int
totalEnemies=int.Parse(waveNode.SelectSingleNode("int[@name='total']").Attributes["value"].Value);
            Wave wave=new Wave(indexEnemy,0.5f,density,totalEnemies);
            levelWaves.Add(wave);
        }
        return levelWaves.ToArray();
    }
    public bool InBounds(Point position)
    {
        return position.X>=0 && position.Y >=0 && position.X < mapSize.X && position.Y < mapSize.Y;
    }
    public void GeneratePath()
    {
        path=Astar.GetPath(spawnPoint,finishPoint);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public struct Point
{
    public int X{get;set;}
    public int Y{get;set;}
```

```
    public Point(int x,int y)
    {
        this.X=x;
        this.Y=y;
    }
    public static bool operator ==(Point first, Point second)
    {
        return first.X == second.X && first.Y == second.Y;
    }
    public static bool operator !=(Point first, Point second)
    {
        return first.X != second.X || first.Y != second.Y;
    }
    public static Point operator -(Point first, Point second)
    {
        return new Point(first.X - second.X, first.Y - second.Y);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TileScript : MonoBehaviour
{
    public Point GridPosition{get;private set;}
    public bool IsEmpty{get;private set;}
    private Color32 fullColor=new Color32(255,118,118,255);
    private Color32 emptyColor=new Color32(96,255,90,255);
    private SpriteRenderer spriteRenderer;
    public SpriteRenderer SpriteRenderer{get;set;}
    public bool WalkAble{get;set;}
    public Vector2 WorldPosition
    {
        get
        {
            return new Vector2(transform.position.x,transform.position.y);
            // return new Vector2(transform.position.x-
(GetComponent<SpriteRenderer>().bounds.size.x/2),transform.position.y-
(GetComponent<SpriteRenderer>().bounds.size.y/2));
        }
    }
    // Start is called before the first frame update
    void Start()
    {
        SpriteRenderer=GetComponent<SpriteRenderer>();
    }

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

    }
    public void Setup(Point gridPos,Vector3 worldPos,Transform parent,bool walkAble)
    {
        WalkAble=walkAble;
        this.GridPosition=gridPos;
        transform.position=worldPos;
        transform.SetParent(parent);
        ManagerScene.Instance.Tiles.Add(gridPos,this);
    }
    public void ColorTile(Color newColor)
    {
        SpriteRenderer.color=newColor;
    }
}
```

```
public class Projectile : MonoBehaviour
{

    [SerializeField]
    int attackDamage;
        [SerializeField]
        float speed = 5f;
        [SerializeField]
        float explosionRange = 0f;
        [SerializeField]
    projecttileType pType;
        public int AttackDamage { get; set; }
        public projecttileType PType { get; }
        //Collider2D projectileCollider;
        Animator anim;
        bool isExploding = false;
        private Enemy target;
        public void Seek(Enemy _target)
        {
                target = _target;
        }
        private void Start()
        {
                anim = GetComponent<Animator>();
        }
        void Update()
        {

                if (target == null)
                {
                        Destroy(gameObject);
                        return;
                }
                var dir = target.transform.localPosition - transform.localPosition;
                float distanceThisFrame = speed * Time.deltaTime;
                var targetAngle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
                transform.localPosition = Vector2.MoveTowards(transform.localPosition,
                        target.transform.localPosition, distanceThisFrame);
                transform.rotation = Quaternion.AngleAxis(targetAngle, Vector3.forward);

        }
        private void OnTriggerEnter2D(Collider2D collision)
        {
                if (collision.tag == "Enemy")
                {
                        isExploding = true;
                        if (explosionRange == 0f)
                        {
                                Enemy newE = collision.gameObject.GetComponent<Enemy>();
                                newE.EnemyHit(AttackDamage);
                                Destroy(gameObject);
                        }
                        else
                        {
                                anim.Play("Boom", layer: 0);
                                Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position,
explosionRange);

                                Destroy(gameObject, anim.GetCurrentAnimatorStateInfo(0).length);
                                foreach (Collider2D collider in colliders)
                                {
                                        if (collider.tag == "Enemy")
                                        {
                                                Enemy newE = collision.gameObject.GetComponent<Enemy>();
                                                newE.EnemyHit(AttackDamage);
                                        }
                                }
                        }
```

```
            }
        }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BasicTower : TowerControl
{
    new void Start()
    {
        Upgrades=new UpgradeTower[]
            {
                new UpgradeTower(price: 20, damage: 1, attackRadius: .5f, rotationSpeed: 0.1f,
attackSpeed: 0.005f),
                new UpgradeTower(price: 30, damage: 1, attackRadius: .5f, rotationSpeed: 0.1f,
attackSpeed: 0.005f),
                new UpgradeTower(price: 40, damage: 1, attackRadius: .5f, rotationSpeed: 0.1f,
attackSpeed: 0.005f),
            new UpgradeTower(price: 40, damage: 1, attackRadius: .5f, rotationSpeed: 0.1f,
attackSpeed: 0.005f),
            };
    }
    public override string GetStats()
        {
            if(NextUpgrade!=null)
            {
                return string.Format("{0} \nRotation Speed: {1} + <color=#00ff00ff>
+{2}</color>", base.GetStats(),rotationSpeed,NextUpgrade.RotationSpeed);
            }
            return string.Format("{0} \nRotation Speed: {1}", base.GetStats(),rotationSpeed);
        }
    public override void Upgrade()
    {
            //over specific Upgrades
            timeBetweenAttacks -= NextUpgrade.AttackSpeed;
            base.Upgrade();
    }
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ChargerTower : TowerControl
{
        new void Start()
    {
            Upgrades =new UpgradeTower[]
            {
                new UpgradeTower(price: 20, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
explosionRange: 0.001f, attackSpeed: 0.005f),
                new UpgradeTower(price: 30, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
explosionRange: 0.001f, attackSpeed: 0.005f),
                new UpgradeTower(price: 40, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
explosionRange: 0.001f, attackSpeed: 0.005f),
            };
    }
    public override string GetStats()
        {
            if(NextUpgrade!=null)
            {
                return string.Format("{0} \nRotation Speed: {1} + <color=#00ff00ff>
+{2}</color>", base.GetStats(),rotationSpeed,NextUpgrade.RotationSpeed);
            }
            return string.Format("{0} \nRotation Speed: {1}", base.GetStats(),rotationSpeed);
        }
    public override void Upgrade()
```

```
    {
            //over specific Upgrades
            timeBetweenAttacks -= NextUpgrade.AttackSpeed;
            base.Upgrade();
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LaserTower : TowerControl
{
        public LineRenderer lineRenderer;
        BoxCollider2D lineCollider;
        const float lMultiplier = 20f;
        private float castCounter=0;
        RaycastHit hit;

        // Start is called before the first frame update
        new void Start()
        {
                Upgrades = new UpgradeTower[]
                {
                        new UpgradeTower(price: 20, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
chargeSpeed: 0.001f, castDuration: 0.005f),
                        new UpgradeTower(price: 20, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
chargeSpeed: 0.001f, castDuration: 0.005f),
                        new UpgradeTower(price: 20, damage: 1, attackRadius:.5f, rotationSpeed:0.1f,
chargeSpeed: 0.001f, castDuration: 0.005f),
                };
        }
        protected override void Init()
        {
                lineCollider = transform.Find("LaserBeam").GetComponent<BoxCollider2D>();
                base.Init();
        }
        // Update is called once per frame
        void Update()
        {
                if (isAttacking && castCounter > 0)
                {
                        castCounter -= Time.deltaTime;

                        // deal damage
                }
                //stop lasering
                else if (isAttacking && castCounter <= 0)
                {
                        isAttacking = false;
                        attackCounter = timeBetweenAttacks;
                        lineRenderer.positionCount = 0;
                        StartCoroutine(RotateTower());
                        lineCollider.transform.position = transform.position;
                        lineCollider.size = new Vector2(0f, 0f);
                }
                //if not casting
                else
                {
                        attackCounter -= Time.deltaTime;
                        if (targetEnemy == null || targetEnemy.IsDead)
                        {
                                Enemy nearestEnemy = GetNearestEnemy();
                                if (nearestEnemy != null)
                                {
                                        targetEnemy = nearestEnemy;
                                }
                                else
```

```
                    {
                        isAttacking = false;
                        hasTurned = false;
                    }
                }
                else
                {
                    StartCoroutine(RotateTower());
                    if (attackCounter <= 0 && hasTurned) //hasTurned
                    {
                        isAttacking = true;
                        castCounter = castDuration;
                        lineRenderer.positionCount = 2;
                        Vector3 difference = targetEnemy.transform.position -
transform.position;
                        lineRenderer.SetPosition(0, transform.position +
difference.normalized/3);
                        lineRenderer.SetPosition(1, targetEnemy.transform.position +
difference * lMultiplier);
                        float lineLength = Vector2.Distance(transform.position,
lineRenderer.GetPosition(1));
                        float lineWidth = lineRenderer.endWidth*1.5f;
                        lineCollider.transform.position = (transform.position +
lineRenderer.GetPosition(1)) / 2;
                        lineCollider.size = new Vector3(lineWidth, lineLength, 1f);
                        var dir = targetEnemy.transform.localPosition -
transform.localPosition;
                        var targetAngle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg-
90f;
                        lineCollider.transform.rotation =
Quaternion.AngleAxis(targetAngle, Vector3.forward);
                    }
                    else
                    {
                        isAttacking = false;
                        hasTurned = false;
                    }
                    if (Vector2.Distance(transform.localPosition,
targetEnemy.transform.localPosition) > attackRadius)
                    {
                        targetEnemy = null;
                        hasTurned = false;
                    }
                }
            }
        }
        private void OnCollisionEnter(Collision collision)
        {
            Debug.Log("hitTrigger");
            Debug.Log(collision.gameObject.GetComponents<Enemy>());
            foreach (Enemy newE in collision.gameObject.GetComponents<Enemy>())
            {
                newE.EnemyHit(Damage);
            }
        }
        private void OnCollisionStay(Collision collisionInfo)
        {
            Debug.Log("hit");
            foreach (Enemy newE in collisionInfo.gameObject.GetComponents<Enemy>())
            {
                newE.EnemyHit(Damage);
            }
        }
        public override string GetStats()
        {
            if (NextUpgrade != null)
            {
```

```
                    return string.Format("{0} \nRotation Speed: {1} + <color=#00ff00ff>
+{2}</color>", base.GetStats(), rotationSpeed, NextUpgrade.RotationSpeed);
                }
                return string.Format("{0} \nRotation Speed: {1}", base.GetStats(), rotationSpeed);
        }
        public override void Upgrade()
        {
                //over specific Upgrades
                timeBetweenAttacks -= NextUpgrade.ChargeSpeed;
                castDuration += NextUpgrade.CastDuration;
                base.Upgrade();
        }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SniperTower : TowerControl
{
        // Start is called before the first frame update
        public void Attack()
        {
                isAttacking = false;
                hasTurned = false;
                if (GetNearestEnemy() != null)
                {
                        //anim.Play("Shoot", layer: 0);

                        //
                        //ADD AUDIO
                        //Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Sniper);
                }
        }
}
using UnityEngine;

public class TowerButton : MonoBehaviour
{
    [SerializeField]
    TowerControl towerObject;
    [SerializeField]
    int towerPrice;
    public TowerControl TowerObject
    {
        get
        {
            return towerObject;
        }
    }
    public int TowerPrice
    {
        get
        {
            return towerPrice;
        }
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public abstract class TowerControl : MonoBehaviour
{
        public int Level{get;set;}=1;
        [SerializeField]
        public int Damage;
```

```
    [SerializeField]
    public float timeBetweenAttacks;
        //degrees per second
        [SerializeField]
        public float rotationSpeed;
        [SerializeField]
    public float attackRadius;
        [SerializeField]
        public float castDuration;
        [SerializeField]
    public Projectile projectile;
    [SerializeField]
    public int sellPrice { get; set; }

        public UpgradeTower[] Upgrades{get; protected set;}

        public UpgradeTower NextUpgrade
        {
                get
                {
                        if(Upgrades.Length > Level-1)
                        {
                                return Upgrades[Level-1];
                        }
                        return null;
                }
        }
    protected Enemy targetEnemy = null;
    protected float attackCounter;
        protected bool hasTurned = false;
    protected bool isAttacking = false;
    private SpriteRenderer rangeSpriteRenderer;

        public void Start()
        {
                //anim.Play("Boom", layer: 0);
                Upgrades =new UpgradeTower[]
                {
                        new UpgradeTower(20,1,.5f,0.1f, 0.005f),
                };
        }
    // Start is called before the first frame update
    protected virtual void Init(){
        rangeSpriteRenderer=this.transform.GetChild(0).GetComponent<SpriteRenderer>();
        rangeSpriteRenderer.transform.localScale=new
Vector3(this.attackRadius*2f,this.attackRadius*2f,1);
    }
    // Update is called once per frame
    void Update()
    {
        attackCounter -= Time.deltaTime;
        if(targetEnemy == null || targetEnemy.IsDead)
        {
            Enemy nearestEnemy = GetNearestEnemy();
            if (nearestEnemy != null)
            {
                targetEnemy = nearestEnemy;
            }
                        else
                        {
                                isAttacking = false;
                                hasTurned = false;
                        }
        }
        else
        {
                        StartCoroutine(RotateTower());
```

```
                    if (attackCounter <= 0 && hasTurned) //hasTurned
                    {
                        isAttacking = true;
                        attackCounter = timeBetweenAttacks;
                        hasTurned = false;
                        Attack();
                    }
            else
            {
                isAttacking = false;
                        hasTurned = false;
                    }
            if (Vector2.Distance(transform.localPosition, targetEnemy.transform.localPosition) >
attackRadius)
            {
                targetEnemy = null;
                        hasTurned = false;
                    }
        }
    }
        protected IEnumerator RotateTower()
        {
            while (!hasTurned && targetEnemy!=null)
            {
                var dir = targetEnemy.transform.localPosition - transform.localPosition;
                var targetAngle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
                transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.Euler(0,
0, targetAngle),
                    rotationSpeed * Time.deltaTime);
                if (Quaternion.Angle(transform.rotation, Quaternion.Euler(0, 0, targetAngle))
< 3f)
                {
                    hasTurned = true;
                }

                yield return null;
            }
        }
        public virtual void Attack()
    {
        isAttacking = false;
            hasTurned = false;
            if (GetNearestEnemy() != null)
            {
                Projectile newProjectile = Instantiate(projectile) as Projectile;
                newProjectile.Seek(targetEnemy);
                newProjectile.AttackDamage=Damage;
                newProjectile.transform.localPosition = transform.localPosition;
                if (newProjectile.PType == projecttileType.arrow)
                {
                    Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Arrow);
                }
                else if (newProjectile.PType == projecttileType.fireball)
                {
                    Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Fireball);
                }
                else if (newProjectile.PType == projecttileType.rock)
                {
                    Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Rock);
                }
            }
    }
    private float GetTargetDistance(Enemy thisEnemy)
    {
        if (thisEnemy == null)
        {
            thisEnemy = GetNearestEnemy();
```

```
            if(thisEnemy == null)
            {
                return float.PositiveInfinity;
            }
        }
        return Mathf.Abs(Vector2.Distance(transform.localPosition,
thisEnemy.transform.localPosition));
    }
        private List<Enemy> GetEnemiesInRange()
    {
        List<Enemy> enemiesInRange = new List<Enemy>();
        foreach (Enemy enemy in Manager.Instance.EnemyList)
        {
            if (Vector2.Distance(transform.localPosition, enemy.transform.localPosition) <=
attackRadius)
            {
                enemiesInRange.Add(enemy);
            }
        }
        return enemiesInRange;
    }
    protected Enemy GetNearestEnemy(bool inRange=true)
    {
        Enemy nearestEnemy = null;
        float smallestDistance = float.PositiveInfinity;
                List<Enemy> enemiesToLook = new List<Enemy>();
                if (inRange)
                        enemiesToLook = GetEnemiesInRange();
                else
                        enemiesToLook = Manager.Instance.EnemyList;
                foreach (Enemy enemy in enemiesToLook)
        {
            if (Vector2.Distance(transform.localPosition, enemy.transform.localPosition) <
smallestDistance)
            {
                smallestDistance = Vector2.Distance(transform.localPosition,
enemy.transform.localPosition);
                nearestEnemy = enemy;
            }
        }
        return nearestEnemy;
    }
    public void EnableRange(){
        if(rangeSpriteRenderer == null){
            Init();
        }
        rangeSpriteRenderer.enabled=true;
    }
    public void DisableRange(){
        rangeSpriteRenderer.enabled=false;
    }
        public virtual string GetStats()
        {
                if(NextUpgrade!=null)
                {
                        return string.Format("\nLevel: {0} \nDamage: {1} <color=#00ff00ff>
+{3}</color> \nRadius: {2} <color=#00ff00ff> +{4}</color>",
Level,Damage,attackRadius,NextUpgrade.Damage,NextUpgrade.AttackRadius);
                }
                return string.Format("\nLevel: {0} \nDamage: {1} \nRadius: {2}",
Level,Damage,attackRadius);
        }
        public virtual void Upgrade()
        {
                Manager.Instance.TotalMoney-=NextUpgrade.Price;
                sellPrice+=NextUpgrade.Price/2;
                Damage+=NextUpgrade.Damage;
```

```
            attackRadius+=NextUpgrade.AttackRadius;
            rotationSpeed+=NextUpgrade.RotationSpeed;
            Init();
            Level+=1;
        }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class UpgradeTower
{
    public int Price{get; private set;}
    public int Damage{get; private set;}
        public float AttackSpeed { get; private set; }
        public float AttackRadius{get; private set;}
    public float RotationSpeed{get; private set;}
        public float ExplosionRange { get; private set; }
        public float ChargeSpeed { get; private set; }
        public float CastDuration { get; private set; }
        public UpgradeTower(int price, int damage, float attackRadius, float rotationSpeed, float
attackSpeed)
    {
        Price=price;
        Damage=damage;
        AttackRadius=attackRadius;
        RotationSpeed=rotationSpeed;
                AttackSpeed = attackSpeed;

        }
        public UpgradeTower(int price, int damage, float attackRadius, float rotationSpeed, float
explosionRange, float attackSpeed)
        {
                Price = price;
                Damage = damage;
                AttackSpeed = attackSpeed;
                AttackRadius = attackRadius;
                RotationSpeed = rotationSpeed;
                ExplosionRange = explosionRange;
        }
        public UpgradeTower(float chargeSpeed, int price, int damage, float attackRadius, float
rotationSpeed,  float castDuration)
     {
                Price=price;
        Damage=damage;
                ChargeSpeed = chargeSpeed;
                AttackRadius = attackRadius;
                RotationSpeed = rotationSpeed;
                CastDuration = castDuration;
     }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AudioController : MonoBehaviour
{
    public AudioSource audio_;

    private void Start()
    {
        if (!PlayerPrefs.HasKey("volume")) audio_.volume = 1;
    }

    private void Update()
    {
        audio_.volume = PlayerPrefs.GetFloat("volume");
```

```
    }
}
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.Collections.Generic;

public class Enemy : MonoBehaviour
{

        [SerializeField]
        float speed = 1;
        [SerializeField]
        float navigation;
        [SerializeField]
        public int startingHealth;
        [SerializeField]
        int revertAmount;

        GameObject exit;
        Transform enemy;
        Collider2D enemyCollider;
        Animator anim;
        int target = 0;
        float navigationTime = 0f;
        bool hasCome = false;

        public float health;
        public Slider healthSlider;
        public Gradient healthGradient;
        public Image fill;

        public float x_offset;
        public float y_offset;
        public bool IsDead { get; private set; } = false;

        private Stack<Node> path;
        public Point GridPosition{get;set;}
        private Vector3 destination;
        // Start is called before the first frame update
        void Start()
        {
                exit = GameObject.FindWithTag("Finish");
                enemy = GetComponent<Transform>();
                enemyCollider = GetComponent<Collider2D>();
                anim = GetComponent<Animator>();
                healthSlider.maxValue = health;
                healthSlider.value = health;
                fill.color = healthGradient.Evaluate(1f);
                //Manager.Instance.RegisterEnemy(this);
                SetPath(ManagerScene.Instance.Path);
        }

        // Update is called once per frame
        void Update()
        {
                Move();
        }
        private void Move()
        {
                navigationTime += Time.deltaTime;

                transform.position=Vector2.MoveTowards(transform.position, destination, speed *
Time.deltaTime);
                if(transform.position==destination)
                {
                        if(path!=null && path.Count>0)
```

```
                        {
                                GridPosition=path.Peek().GridPosition;
                                destination=path.Pop().WorldPosition;
                                destination.x += x_offset;
                                destination.y += y_offset;
                        }
                }


        }

        private void OnTriggerEnter2D(Collider2D collision)
        {
                if (collision.tag == "Finish")
                {
            Manager.Instance.Health -= 1;
            Manager.Instance.UnregisterEnemy(this);
                        Manager.Instance.IsWaveOver();
                }
        }
        private void OnTriggerStay2D(Collider2D collisionInfo)
        {
                foreach (BoxCollider2D col in
collisionInfo.gameObject.GetComponents<BoxCollider2D>())
                {
                        if (col.gameObject.tag == "LaserBeam")
                        {
                                LaserTower lt = col.gameObject.GetComponentInParent<LaserTower>();
                                EnemyHit(lt.Damage*Time.deltaTime);
                        }
                }
        }
        public void EnemyHit(float hitpoints)
        {
                if (health - hitpoints > 0)
                {
                        //change color
                        health -= hitpoints;
                        healthSlider.value = health;
                        fill.color = healthGradient.Evaluate(healthSlider.normalizedValue);
                        //hurt
                        Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Hit);
                        anim.Play("Hurt");
                }
                else
                {
                        health = 0;
                        healthSlider.value = health;
                        fill.color = healthGradient.Evaluate(healthSlider.normalizedValue);
                        anim.SetTrigger("Explode");
                        //dying
                        Die();
                }
        }
        public void Die()
        {
                IsDead = true;
                enemyCollider.enabled = false;
                Manager.Instance.TotalKilled += 1;
                Manager.Instance.Score += 1;
                Manager.Instance.AddMoney(revertAmount);
                Manager.Instance.EnemyList.Remove(this);
                Destroy(enemy.gameObject,anim.GetCurrentAnimatorStateInfo(0).length);
                Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.Death);
                Manager.Instance.IsWaveOver();
        }
```

```
        private void SetPath(Stack<Node> newPath)
        {
                if(newPath !=null)
                {
                        this.path=newPath;
                        GridPosition=path.Peek().GridPosition;
                        destination=path.Pop().WorldPosition;
                }
        }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// 1
[RequireComponent(typeof(SpriteRenderer))]
[RequireComponent(typeof(Collider))]
public class Highlighter : MonoBehaviour
{
    // 2
    // reference to SpriteRenderer component
    private SpriteRenderer SpriteRenderer;

    [SerializeField]
    private Material originalMaterial;

    [SerializeField]
    private Material highlightedMaterial;

    void Start()
    {
        // 3
        // cache a reference to the SpriteRenderer
        SpriteRenderer = GetComponent<SpriteRenderer>();

        // 4
        // use non-highlighted material by default
        EnableHighlight(false);
    }

    // toggle betweeen the original and highlighted materials
    public void EnableHighlight(bool onOff)
    {
        // 5
        if (SpriteRenderer != null && originalMaterial != null &&
            highlightedMaterial != null)
        {
            // 6
            SpriteRenderer.material = onOff ? highlightedMaterial : originalMaterial;
        }
    }
    private void OnMouseOver()
    {
        EnableHighlight(true);
    }

    private void OnMouseExit()
    {
        EnableHighlight(false);
    }

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

```
using System.IO;
using Assets.Scripts;

public class LevelManager : MonoBehaviour
{
    int levelUnLock;
    public Button[] buttons;
    public float scoreMain { get; set;} = 0;
    public float score { get; set; } = 0;
    [SerializeField]
    Text MoneyLabel;
    [SerializeField]
    GameObject SelectLvl;
    [SerializeField]
    Button levelBtnPrefab;
    [SerializeField]
    Transform levelList;


    void Start()
    {
        //PlayerPrefs.DeleteAll();
        MoneyLabel = GameObject.Find("MoneyLabel").GetComponent<Text>();
        if (PlayerPrefs.HasKey("MoneyScore"))
        {
            scoreMain = PlayerPrefs.GetFloat("MoneyScore");
            if (scoreMain != 0)
            {
                MoneyLabel.text = MoneyLabel.text.Remove(MoneyLabel.text.Length - 1) +
scoreMain.ToString();
            }
        }
        SetLevelList();
    }
    public void loadLevel(int levelIndex)
    {
        Debug.Log(levelIndex);
        PlayerPrefs.SetInt("levelIndex", levelIndex);
        PlayerPrefs.Save();
        SceneManager.LoadScene(1);
    }
    public void SetLevelList()
    {
        Object[] worlds = Resources.LoadAll("levels", typeof(TextAsset));
        Sprite[] levelImages = Resources.LoadAll<Sprite>("ImagesLevel");
        for(int j=0;j<worlds.Length;j++)
        {
            int param=j+1;
            Button levelBtn=Instantiate(levelBtnPrefab);
            levelBtn.gameObject.transform.GetChild(1).GetComponent<Image>().sprite=levelImages[j];
            levelBtn.gameObject.transform.GetChild(0).GetComponent<Text>().text=(j+1).ToString();
            levelBtn.gameObject.transform.SetParent(levelList);
            levelBtn.gameObject.transform.localScale=new Vector3(1.0f,1.0f,1.0f);
            levelBtn.onClick.AddListener(() => loadLevel(param));
        }
    }
    public void LoadScore()
    {
        var listLvl = new List<GameObject>(GameObject.FindGameObjectsWithTag("Level")); ;
        if (PlayerPrefs.HasKey("MoneyScore"))
        {
            for (int i = 0; i < listLvl.Count; i++)
            {

                var textElem = listLvl[i].GetComponentInChildren<Text>();
                if (PlayerPrefs.HasKey("Score_" + (i+1).ToString()))
                {
```

```
                    var scoreStr = PlayerPrefs.GetInt("Score_" + (i + 1).ToString()).ToString();
                    if (textElem.text.Length == 1)
                    {
                        textElem.text = textElem.text + ": " + scoreStr;
                    }

                }
            }
        }
    }

    public void QuitGame()
    {
        Application.Quit();
    }
}
using UnityEngine;

public class Loader <T>: MonoBehaviour where T: MonoBehaviour
{
    private static T instance;
    public static T Instance
    {
        get
        {
            if (instance == null)
            {
                instance = FindObjectOfType<T>();
            }
            else if (instance != FindObjectOfType<T>())
            {
                Destroy(FindObjectOfType<T>());
            }
            DontDestroyOnLoad(FindObjectOfType<T>());
            return instance;
        }
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using Assets.Scripts;
using System.IO;
using System.Linq;

public enum gameStatus
{
  next,play,gameover,win
}



public class Manager : Loader<Manager>
{
  [SerializeField]
  Text totalMoneyLabel;
```

```
[SerializeField]
Text currentWave;
        [SerializeField]
        Text healthLabel;
        [SerializeField]
Text playBtnLabel;
[SerializeField]
Text ScoreLabel;
[SerializeField]
Button playBtn;

[SerializeField]
Enemy[] enemies;

[SerializeField]
GameObject WavesInfoPanel;

int waveNumber = 0;
int totalMoney = 280;

public float money = 0;

gameStatus currentState = gameStatus.play;
        public List<Enemy> EnemyList = new List<Enemy>();
        public int TotalHealth { get; set;} = 20;
        public int health;
        public int TotalKilled { get; set; } = 0;
public int Score { get; set; } = 0;
public int MainScore { get; set; } = 0;
public int LevelScore { get; set; } = 0;
public bool gameOver = false;

public Wave[] Waves{get;set;}
public Wave CurrentWave
{
  get
  {
    if(Waves.Length>=waveNumber)
    {
      return Waves[waveNumber];
    }
    return null;
  }
}
[SerializeField]
```

```
Transform posEnemyList;
[SerializeField]
private GameObject waveStatPrefab;
[SerializeField]
private Transform WaveList;
[SerializeField]
private GameObject gameOverMenu;
[SerializeField]
private GameObject WinMenu;


private void SaveParams()
{
    // PlayerPrefs.SetInt("Level" + ManagerScene.Instance.levelIndex.ToString(), 1);
    if (LevelScore < Score)
    {
        PlayerPrefs.SetInt("Score_" + ManagerScene.Instance.levelIndex.ToString(), Score);
    }
    PlayerPrefs.SetFloat("MoneyScore", money);
    PlayerPrefs.Save();

}

private void LoadParams()
{
    money = PlayerPrefs.GetFloat("MoneyScore");
    if (PlayerPrefs.HasKey("Score_" + ManagerScene.Instance.levelIndex.ToString()))
    {
        LevelScore = PlayerPrefs.GetInt("Score_" + ManagerScene.Instance.levelIndex.ToString());
    };

}

public int Health
{
    get
    {
        return health;
    }
    set
    {
        health = value;
        healthLabel.text = health.ToString();
    }
}
```

```
public int TotalMoney
{
    get
    {
        return totalMoney;
    }
    set
    {
        totalMoney = value;
        totalMoneyLabel.text = totalMoney.ToString();
    }
}
            public AudioSource AudioSrc { get; private set; }
// Start is called before the first frame update
void Start()
{
    LoadParams();
    ManagerScene.Instance.GeneratePath();

    Waves=ManagerScene.Instance.SetWaves();
    SetPossibleEnemies();
                        Health = TotalHealth;
    totalMoneyLabel.text=TotalMoney.ToString();
    healthLabel.text=TotalHealth.ToString();
    playBtn.gameObject.SetActive(false);
                        AudioSrc = GetComponent<AudioSource>();
    ShowMenu();
}
private void Update()
{
    HandleEscape();
}

            private static int SortByName(GameObject o1, GameObject o2)
            {
                        return o1.name.CompareTo(o2.name);
            }

            IEnumerator Spawn()
{
    if (CurrentWave.EnemiesPerSpawn > 0 && EnemyList.Count < CurrentWave.TotalEnemies)
    {
        for (int i = 0; i < CurrentWave.EnemiesPerSpawn; i++)
        {
            if (EnemyList.Count + TotalKilled < CurrentWave.TotalEnemies)
```

```
            {
                var spawnPoint=ManagerScene.Instance.spawn;
                var center = spawnPoint.transform.position;
                var size  = spawnPoint.GetComponent<BoxCollider2D>().size;
                Enemy newEnemy = Instantiate(enemies[CurrentWave.IndexEnemy]) as Enemy;
                //need balancing
                newEnemy.health = (int)(newEnemy.startingHealth * (1 + (float)waveNumber/10));
                newEnemy.x_offset = Random.Range(-size.x/2, size.x/2);
                newEnemy.y_offset = Random.Range(-size.y / 2, size.y/2);
                Vector2 pos = new Vector3(center.x + newEnemy.x_offset, center.y + newEnemy.y_offset);

                newEnemy.transform.position = pos;
                RegisterEnemy(newEnemy);
            }
        }
        yield return new WaitForSeconds(CurrentWave.SpawnDelay);
        StartCoroutine(Spawn());
    }
}

public void RegisterEnemy(Enemy enemy)
{
    EnemyList.Add(enemy);
}
public void UnregisterEnemy(Enemy enemy)
{
    EnemyList.Remove(enemy);
    Destroy(enemy.gameObject);
}
public void DestroyEnemies()
{
    foreach (Enemy enemy in EnemyList)
    {
        Destroy(enemy.gameObject);
    }
    EnemyList.Clear();
}
public void AddMoney(int amount)
{
    TotalMoney += amount;
}
public void SubtractMoney(int amount)
{
    TotalMoney -= amount;
}
```

```
public void IsWaveOver()
{
                    healthLabel.text = Health.ToString();
    if ((TotalHealth-Health+TotalKilled)>=CurrentWave.TotalEnemies)
    {
        // if (waveNumber <= enemies.Length)
        // {
        //     enemiesToSpawn = waveNumber;
        // }
        SetCurrentGameState();
        ShowMenu();
    }
}
public void SetCurrentGameState()
{
    if (Health<=0)
    {
        this.Health = 0;
        playBtn.interactable=false;
        currentState = gameStatus.gameover;
        GameOver();
    }
    else if(waveNumber==0 && (TotalHealth-Health + TotalKilled) == 0)
    {
        currentState = gameStatus.play;
    }
    else if (waveNumber >= Waves.Length-1 && EnemyList.Count == 0)
    {
        currentState = gameStatus.win;
        playBtn.interactable=false;
        WinGame();
    }
    else
    {
        currentState = gameStatus.next;
    }
}
public void PlayBtnPressed()
{
                    switch (currentState)
    {
        case gameStatus.next:
            waveNumber += 1;
            break;
        default:
```

```
            Health = TotalHealth;
            totalMoney = TotalMoney;
                                    totalMoneyLabel.text = TotalMoney.ToString();
            healthLabel.text = TotalHealth.ToString();
                                    AudioSrc.PlayOneShot(SoundManager.Instance.Newgame);

                                    break;
    }
    TotalKilled = 0;
    currentWave.text = "Wave " + (waveNumber + 1);
    SetWaveList();
    StartCoroutine(Spawn());
    playBtn.gameObject.SetActive(false);

}
public void ShowMenu()
{
    switch (currentState)
    {
        case gameStatus.gameover:
            playBtn.interactable=false;
            playBtnLabel.text = "Play Again?";
            AudioSrc.PlayOneShot(SoundManager.Instance.Gameover);
            break;

        case gameStatus.next:
            playBtnLabel.text = "Next Wave?";
            break;

        case gameStatus.play:
            playBtnLabel.text = "Play game";
            break;

        case gameStatus.win:
            playBtnLabel.text = "Play game";
            break;

    }
    playBtn.gameObject.SetActive(true);
}
private void HandleEscape()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        // TowerManager.Instance.DisableDrag();
```

```
            TowerManager.Instance.towerBtnPressed = null;
        }
    }
    public void WinGame()
    {
        if(currentState == gameStatus.win)
        {
            WinMenu.SetActive(true);
            Time.timeScale = 0f;
            // Временная заглушка для подсчета итоговых очков
            Score=Score+Health+TotalMoney;
            money = money + Score / 10;
            ScoreLabel.text="Score: "+ Score.ToString();

            SaveParams();
            //Serializer.SaveXml(MainScore, path);
        }
    }
    public void GameOver()
    {
        if (currentState == gameStatus.gameover)
        {

            gameOverMenu.SetActive(true);
            Time.timeScale = 0f;
            money = money + Score / 10;
            SaveParams();
            //Serializer.SaveXml(MainScore, path);
        }
    }

    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    public void Quit(int _sceneNumber)
    {
        SceneManager.LoadScene(_sceneNumber);
    }
    public void ShowWavesInfo()
    {
        // TowerManager.Instance.SelectTile();
        TowerManager.Instance.towerPanel.gameObject.SetActive(true);
        TowerManager.Instance.DisableChilds();
```

```
        WavesInfoPanel.SetActive(true);
    }
    public void SetWaveList()
    {
        for(int i=WaveList.transform.childCount-1;i>=0;i--)
        {
            Destroy(WaveList.transform.GetChild(i).gameObject);
        }
        for(int i=waveNumber;i<waveNumber+5;i++)
        {
            if(i<Waves.Length-1)
            {
                GameObject waveStat=Instantiate(waveStatPrefab);
                waveStat.transform.SetParent(WaveList);
                waveStat.transform.localScale=new Vector3(1.0f,1.0f,1.0f);
                Text stats=waveStat.GetComponentInChildren<Text>();
                Image enemyImage=waveStat.transform.GetChild(1).GetComponent<Image>();
                enemyImage.sprite=enemies[Waves[i].IndexEnemy].GetComponent<SpriteRenderer>().sprite;
                stats.text=string.Format("Density: {0}\t Total: {1}", Waves[i].EnemiesPerSpawn,Waves[i].TotalEnemies);
            }

        }
    }
    public void SetPossibleEnemies()
    {
        List<int> listPosEnemies=new List<int>();
        foreach (Wave item in Waves)
        {
            listPosEnemies.Add(item.IndexEnemy);
        }
        HashSet<int> uniqEnemies=new HashSet<int>(listPosEnemies);
        int[] posEnemies=uniqEnemies.ToArray();
        for(int i=0;i<posEnemies.Length;i++)
        {
            GameObject _gameObject = new GameObject(string.Format("enemy{0}",i));
            _gameObject.AddComponent<SpriteRenderer>();
            SpriteRenderer imageEnemy=enemies[i].GetComponent<SpriteRenderer>();
            _gameObject.GetComponent<SpriteRenderer>().sprite = imageEnemy.sprite;
            _gameObject.transform.SetParent(posEnemyList);
            float startpos=(posEnemies.Length-1)*(-40.0f)/2.0f;
            _gameObject.transform.localPosition=new Vector3(startpos+(i*40.0f),0,0);
            _gameObject.GetComponent<SpriteRenderer>().sortingOrder=3;
        }
    }
}
```

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

public class Pause_menu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;

    // Update is called once per frame
    public void PauseBtn()
    {
        if (GameIsPaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void LoadMenu()
    {
        SceneManager.LoadScene("MainMenu");
        Time.timeScale = 1f;
    }
}
using System;
using System.Xml.Serialization;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Assets.Scripts
{
    public class Serializer
    {
        static public void SaveXml(int score, string datapath)
        {

            XmlSerializer serializer = new XmlSerializer(typeof(int));

            FileStream fs = new FileStream(datapath, FileMode.Create);
            serializer.Serialize(fs, score);
            fs.Close();

        }

        static public int DeXml(string datapath)
        {
```

```
            XmlSerializer serializer = new XmlSerializer(typeof(int));

            FileStream fs = new FileStream(datapath, FileMode.Open);
            int score = (int)serializer.Deserialize(fs);
            fs.Close();

            return score;
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SliderController : MonoBehaviour
{
    public Slider slider;
    public float oldVolume;

    private void Start()
    {
        oldVolume = slider.value;
        if (!PlayerPrefs.HasKey("volume")) slider.value = 1;
        else slider.value = PlayerPrefs.GetFloat("volume");
    }

    private void Update()
    {
        if(oldVolume != slider.value)
        {
            PlayerPrefs.SetFloat("volume", slider.value);
            PlayerPrefs.Save();
            oldVolume = slider.value;
        }
    }
}
using UnityEngine;

public class SoundManager : Loader <SoundManager>
{
    [SerializeField]
    AudioClip arrow;
    [SerializeField]
    AudioClip death;
    [SerializeField]
    AudioClip fireball;
    [SerializeField]
    AudioClip gameover;
    [SerializeField]
    AudioClip hit;
    [SerializeField]
    AudioClip level;
    [SerializeField]
    AudioClip newgame;
    [SerializeField]
    AudioClip rock;
    [SerializeField]
    AudioClip towerBuilt;

    public AudioClip Arrow
    {
        get
        {
            return arrow;
        }
```

```
    }
    public AudioClip Death
    {
        get
        {
            return death;
        }
    }
    public AudioClip Fireball
    {
        get
        {
            return fireball;
        }
    }
    public AudioClip Gameover
    {
        get
        {
            return gameover;
        }
    }
    public AudioClip Hit
    {
        get
        {
            return hit;
        }
    }
    public AudioClip Level
    {
        get
        {
            return level;
        }
    }
    public AudioClip Newgame
    {
        get
        {
            return newgame;
        }
    }
    public AudioClip Rock
    {
        get
        {
            return rock;
        }
    }
    public AudioClip TowerBuilt
    {
        get
        {
            return towerBuilt;
        }
    }
}
using UnityEngine;
using UnityEngine.UI;

public class TimeControl : MonoBehaviour
{
    public Text speed;
    public void Speed()
    {
        if (speed.text == "1x")
```

```
        {
            Time.timeScale = 1.5f;
            speed.text = "1,5x";
        }
        else if (speed.text == "1,5x")
        {
            Time.timeScale = 2f;
            speed.text = "2x";
        }
        else if (speed.text == "2x")
        {
            Time.timeScale = 1f;
            speed.text = "1x";
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine.EventSystems;
using UnityEngine;
using UnityEngine.UI;

public class TowerManager : Loader<TowerManager>
{
    public TowerButton towerBtnPressed{get; set;}
    SpriteRenderer spriteRenderer;
    [SerializeField]
    public Transform towerPanel;
    [SerializeField]
    public GameObject towerInfo;
    [SerializeField]
    public GameObject choiceTower;

    [SerializeField]
    Text infoLabel;
    [SerializeField]
    Text upgradePriceLabel;
    [SerializeField]
    Text sellPriceLabel;
    [SerializeField]
    Image towerImage;
    [SerializeField]
    public Button upgradeBtn;

    private List<TowerControl> TowerList = new List<TowerControl>();
    private List<Collider2D> BuildList = new List<Collider2D>();
    private Collider2D buildTile;
    private Collider2D buildTileOld = null;
    private RaycastHit2D hitTile;
    private TowerControl selectTower;
    // Start is called before the first frame update
    void Start()
    {
        towerPanel.gameObject.SetActive(false);
        buildTile = GetComponent<Collider2D>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        // if (Input.GetMouseButtonDown(0) && Input.touchCount > 0)
        {
            Vector2 mousePoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);
                      RaycastHit2D hit = Physics2D.Raycast(mousePoint, Vector2.zero);
            if (hit.collider && (hit.collider.tag == "TowerSide"  || hit.collider.tag ==
"TowerFull") && !EventSystem.current.IsPointerOverGameObject())
```

```
            // if (hit.collider && (hit.collider.tag == "TowerSide"  || hit.collider.tag ==
"TowerFull") && !EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))
            {
                //buildTile.gameObject.GetComponent<SpriteRenderer>().color = Color.red;
                towerPanel.gameObject.SetActive(true);
                buildTile = hit.collider;
                SelectTile();
                hitTile = hit;
                if(hit.collider.tag == "TowerSide")
                {

                    if(selectTower != null){
                        selectTower.DisableRange();
                    }
                    DisableChilds();
                    choiceTower.SetActive(true);
                }
                else if(hit.collider.tag == "TowerFull")
                {
                    DisableChilds();
                    towerInfo.SetActive(true);
                    foreach(TowerControl tower in TowerList)
                    {
                        if(tower.transform.position == hit.transform.position){
                            if (selectTower != null)
                            {
                                selectTower.DisableRange();
                            }
                            selectTower=tower;
                            break;
                        }
                    }
                    if(selectTower!=null){
                        ViewTowerInfo();
                    }
                }
            }
            else if(hit.collider && hit.collider.tag == "Respawn" &&
!EventSystem.current.IsPointerOverGameObject())
            // else if(hit.collider && hit.collider.tag == "Respawn" &&
!EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))
            {
                buildTile = hit.collider;
                SelectTile();
                if(selectTower != null)
                {
                    selectTower.DisableRange();
                }
                Manager.Instance.ShowWavesInfo();
            }
            else if(!EventSystem.current.IsPointerOverGameObject())
            // else if(!EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))
            {
                ClocePanel();
            }
          }
        if(selectTower != null)
        {
            if(selectTower.NextUpgrade != null)
            {
                if (selectTower.NextUpgrade.Price <= Manager.Instance.TotalMoney)
                {
                    upgradeBtn.interactable=true;
                }
                else{
                    upgradeBtn.interactable=false;
                }
```

```
            }

        }
    }
    public void SelectTile()
    {
        if (buildTileOld != null && buildTile != buildTileOld)
        {
            buildTileOld.GetComponent<SpriteRenderer>().color = Color.white;
        }
        buildTile.gameObject.GetComponent<SpriteRenderer>().color = new Color(0.9843137f,
0.4823529f, 1, 1); ;
        buildTileOld = buildTile;
    }
    public void DisableChilds() // Вызывешь где-нибудь, например, через по нажатию UI кнопки
    {
        for (int i = 0; i < towerPanel.childCount; i++)
        {
            towerPanel.GetChild(i).gameObject.SetActive(false); // Выключаем или включаем каждого
полученного ребёнка по порядку.
        }
        towerPanel.Find("ClosePanel").gameObject.SetActive(true);

    }
    public void ViewTowerInfo()
    {
        infoLabel.text=selectTower.GetStats();
        if (selectTower.NextUpgrade ==null)
        {
            upgradeBtn.interactable=false;
            upgradePriceLabel.text="Max lvl!";
            // upgradeBtn.GetChild(1).SetActive(false);
        }
        else
        {
            upgradeBtn.interactable=true;
            upgradePriceLabel.text="for "+selectTower.NextUpgrade.Price;
        }
        sellPriceLabel.text=selectTower.sellPrice.ToString();
        SpriteRenderer m_SpriteRenderer = selectTower.GetComponent<SpriteRenderer>();
        Sprite sprite=m_SpriteRenderer.sprite;
        towerImage.sprite=sprite;
        selectTower.EnableRange();
    }
    public void ClocePanel()
    {
        if (selectTower != null)
        {
            selectTower.DisableRange();
        }
        towerPanel.gameObject.SetActive(false);
        if (buildTileOld != null)
        {
            buildTileOld.GetComponent<SpriteRenderer>().color = Color.white;
        }
        //backBtn.SetActive(false);
        //DestrTower();
    }
    public void TowerUpgrade()
    {
        if(selectTower != null)
        {
            if(selectTower.Level <= selectTower.Upgrades.Length && Manager.Instance.TotalMoney >=
selectTower.NextUpgrade.Price)
            {
                selectTower.Upgrade();
            }
```

```
            ViewTowerInfo();
        }
    }
    public void DestrTower()
    {
        Manager.Instance.TotalMoney += selectTower.sellPrice;
        TowerList.Remove(this.selectTower);
        Collider2D selectBuild = null;
        foreach (Collider2D buildTag in BuildList)
        {
            if (buildTag.transform.position == selectTower.transform.position)
            {
                selectBuild = buildTag;
                buildTag.tag = "TowerSide";
                break;
            }
        }
        BuildList.Remove(selectBuild);
        Destroy(selectTower.gameObject);
        selectTower = null;
        DisableChilds();
        choiceTower.SetActive(true);
    }
    public void RegisterBuildSite(Collider2D buildTag)
    {
        BuildList.Add(buildTag);
    }
    // покупка башни: вычет денег из общего счета игрока
    public void BuyTower(int price)
    {
        Manager.Instance.SubtractMoney(price);
    }
    public void RegisterTower(TowerControl tower)
    {
        TowerList.Add(tower);
    }
    public void RenameTagBuildSite()
    {
        foreach(Collider2D buildTag in BuildList)
        {
            buildTag.tag = "TowerSide";
        }
        BuildList.Clear();
    }
    public void DestroyAllTowers()
    {
        foreach(TowerControl tower in TowerList)
        {
            Destroy(tower.gameObject);
        }
        TowerList.Clear();
    }
    public void PlaceTower(RaycastHit2D hit)
    {
        if (towerBtnPressed !=null)
        {
            TowerControl newTower = Instantiate(towerBtnPressed.TowerObject);
            newTower.Start();
            newTower.sellPrice = towerBtnPressed.TowerPrice;
            newTower.transform.position = hit.transform.position;
            BuyTower(towerBtnPressed.TowerPrice);
            Manager.Instance.AudioSrc.PlayOneShot(SoundManager.Instance.TowerBuilt);
            RegisterTower(newTower);
            if(selectTower != null){
                selectTower.DisableRange();
            }
            selectTower=newTower;
```

```
            choiceTower.SetActive(false);
            towerInfo.SetActive(true);
        }
    }
    public void SelectTower(TowerButton towerSelected)
    {
        if (towerSelected.TowerPrice <= Manager.Instance.TotalMoney)
        {
            towerBtnPressed = towerSelected;
            // EnableDrag(towerBtnPressed.DragSprite);
            buildTile.tag = "TowerFull";
            RegisterBuildSite(buildTile);
            PlaceTower(hitTile);
            ViewTowerInfo();
        }
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Wave
{
    public int IndexEnemy{get;set;}
    public float SpawnDelay{get;set;}
    public int EnemiesPerSpawn{get;set;}
    public int TotalEnemies{get;set;}

    public Wave(int indexEnemy,float spawnDelay,int enemiesPerSpawn,int totalEnemies)
    {
        IndexEnemy=indexEnemy;
        SpawnDelay=spawnDelay;
        EnemiesPerSpawn=enemiesPerSpawn;
        TotalEnemies=totalEnemies;
    }
}
using UnityEngine;

public class Zoom : MonoBehaviour
{
    Vector3 touchStart;
    public float maxZoom;
    public float minZoom;
    public float sensitivity;
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            touchStart = GetComponent<Camera>().ScreenToWorldPoint(Input.mousePosition);
        }
        if (Input.touchCount == 2)
        {
            Touch touchZero = Input.GetTouch(0);
            Touch touchOne = Input.GetTouch(1);

            Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

            float prevMagnitude = (touchZeroPrevPos - touchOnePrevPos).magnitude;
            float currentMagnitude = (touchZero.position - touchOne.position).magnitude;

            float difference = currentMagnitude - prevMagnitude;
            ZoomCamera(difference * 0.01f);
        }
        else if (Input.GetMouseButton(0))
```

```
        {
            Vector3 direction = touchStart - 
GetComponent<Camera>().ScreenToWorldPoint(Input.mousePosition);
            GetComponent<Camera>().transform.position += direction;
        }
        ZoomCamera(Input.GetAxis("Mouse ScrollWheel"));
    }
    void ZoomCamera(float increment)
    {
        GetComponent<Camera>().orthographicSize = 
Mathf.Clamp(GetComponent<Camera>().orthographicSize - increment * sensitivity, minZoom, maxZoom);
    }
}
```