МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА ПРИКЛАДНОЇ ІНФОРМАТИКИ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

___________ __Гамаюн В.П.
(підпис) (ПІБ)

"_____"____________2021р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "БАКАЛАВР"

Тема: Розробка мобільного додатку для ОС Android______________

Виконавець: __________________________ _Каташ Кирил Андрійович
(підпис) (ПІБ)

Керівник: ______________________________ Гамаюн Володимир Петрович
(підпис) (ПІБ)

Нормоконтролер: ____________________ Боровик Володимир Миколайович
(підпис) (ПІБ)

Київ 2021

ВСТУП

Ринок мобільних девайсів та програм – один з найбільш великих сегментів світової економіки. Неможливо представити сучасний світ без використання мобільних девайсів, або технологій пов’язаних з ними. Щороку випускається безліч додатків для мобільних девайсів і одним з найбільш популярних видів застосунків – мобільні ігри.

Мобільні ігри є не лише «забавкою» вони можуть надавати безліч рішень для повсякденних завдань, використовуватися як платформи для тестування реальних ситуацій та надавати інструментарій для створення моделей будь-якої складності.

Також важливим є той факт – що завдяки мобільним іграм розвивається безліч важливих для людства технологій та програмних рішень, таких як: оптимізація пам’яті (як для зберігання даних, так і оперативної), робота з графікою, удосконалення обчислювальних пристроїв та зменшення їх розмірів.

Саме тому я вважаю актуальною тему демонстрації можливостей мобільних пристроїв з використанням однієї з найбільш популярних тем для створення додатків – мобільної гри.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність дослідження

Ринок мобільних додатків є одним з найбільш швидко розвинених у світі. Його загальний об’єм складає близько 160 мільярдів доларів і ця цифра зростає кожного року на 5-7% відсотків. Мобільні ігри займають домінуючий сегмент не тільки у сфері мобільних додатків, але і у сфері розваг у цілому. Витрати на мобільні ігри у таких країнах як Японія, Китай, Північна Корея та США складають близько 110 доларів на рік з кожного гравця.

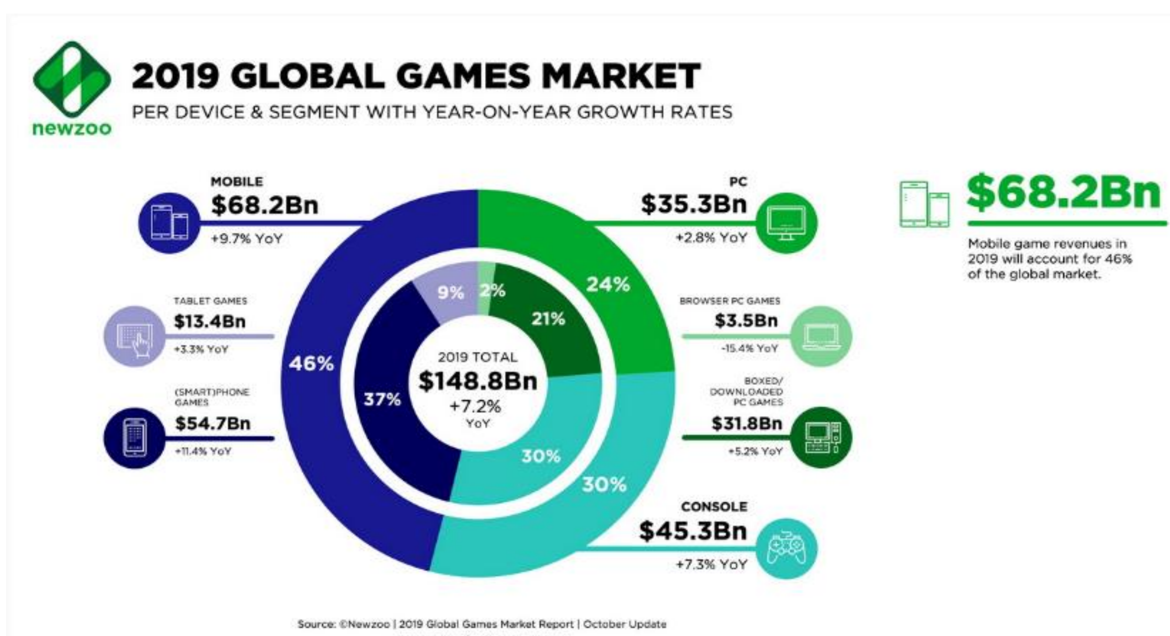


Рис. 1.1 Сегмент мобільних ігор

Головними факторами у розробках мобільних застосунків та ігор є дотримання балансу між кінцевим функціоналом продукту та дизайном користувача, який забезпечить «безболісне» користування продуктом, поставить низький поріг входу та забезпечить швидкий та зручний доступ до функціоналу.

Впровадження нових технологій, таких як апаратне прискорення, віртуалізація, використання нейронних мереж для спрощення обчислень, або для програмування поведінки штучного інтелекту – все це пов’язано саме з розвитком мобільних технологій.

Також ми «повинні дякувати» мобільним розробкам, оскільки завдяки ним ми бачимо тенденцію до спрощення інтерфейсів програм, утиліт, покращення у сфері взаємодії користувача та «лицьової» сторони програми, оскільки світ відходить від вже застарілої, але перевіреної часом, системи «вікон» до системи одного активного вікна, яке управляється інтуітивно зрозумілими жестами, командами, та іншими методами управління.

Актуальність даної теми полягає також у тому, що Android додатки займають перше місце у світі по кількості, оскільки дана ОС є системою з відкритим кодом, та достатньої кількістю інформації у вільному доступі, яка може бути корисна навіть людям з повністю відсутніми навичками програмування.

Дана робота буде використовувати сучасний потужний ігровий рушій (з англ. Game engine) Unreal Engine 4, який застосовується при розробці багатьох сучасних мобільних (та не тільки) додатків. Його функціонал дозволить забезпечити приємний інтерфейс, продемонструвати роботу з об'єктами, їх взаємодію з навколишнім середовищем, освітленням та іншими складовими.

А що найголовніше - продемонструє всю потужність та наглядне використання найсучасніших технологій.

1.2 Постановка завдання

Для виконання поставленого завдання, а саме – розробки Android додатку-гри за допомогою ігрового рушія Unreal Engine 4 нам потрібно визначитися з типом гри.

Для розробки я вибрав популярний жанр ігор, а саме гру-бігун (з англ. runner) по типу відомої гри Subway Surf.

Сам додаток буде складатися з таких елементів: персонаж гри (макет людини який нескінченно біжить у просторі), процедурно генерованого світу (тобто кожен раз після запуску гри буде створюватися абсолютно випадковим шляхом), перешкод, які будуть заважати персонажу, лічильників рахунків, інтерфейсу, різних застосунків, які наш персонаж буде у змозі використовувати.

Для проектування такої роботи ми використаємо такі додатки як Unreal Engine 4, Adobe Photoshop (для створення текстур та матеріалів), та Android SDK для публікації та перенесення кінцевого проекту на девайс з встановленою OS Android.

У процесі створення додатку будуть використані поняття об'єктно-орієнтованого програмування, функціонального програмування, побудування моделей для подальшого їх використання, а також оформлення інтерфейсу користувача. Буде побудована система для реєстрації тактильного управління користувача для забезпечення зручного управління. Всі ці елементи будуть розглянуті нижче у даній роботі і детально вивчені.

У кінці повинен вийти повністю робочий Android-додаток, який забезпечує виконання всіх поставлених задач для даного жанру гри, а також повністю функціонує на девайсі з системою Android.

1.3 Аналіз методів розв'язання поставленої задачі

Для розв'язання поставленої задачі необхідно провести аналіз складових проекту. При поєднанні даних складових повинна вийти суцільна

Додаток-гра даного жанру складається з таких основних елементів:

1) Ігрове поле (процедурно-генерований світ).
2) Персонаж гри.
3) Перешкоди на шляху гравця.
4) Елементи, з якими може взаємодіяти персонаж гри.
5) Меню гравця, яке відображає його результати
6) Меню для управління грою.

Приклад вигляду основної частини проекту представлено нижче:

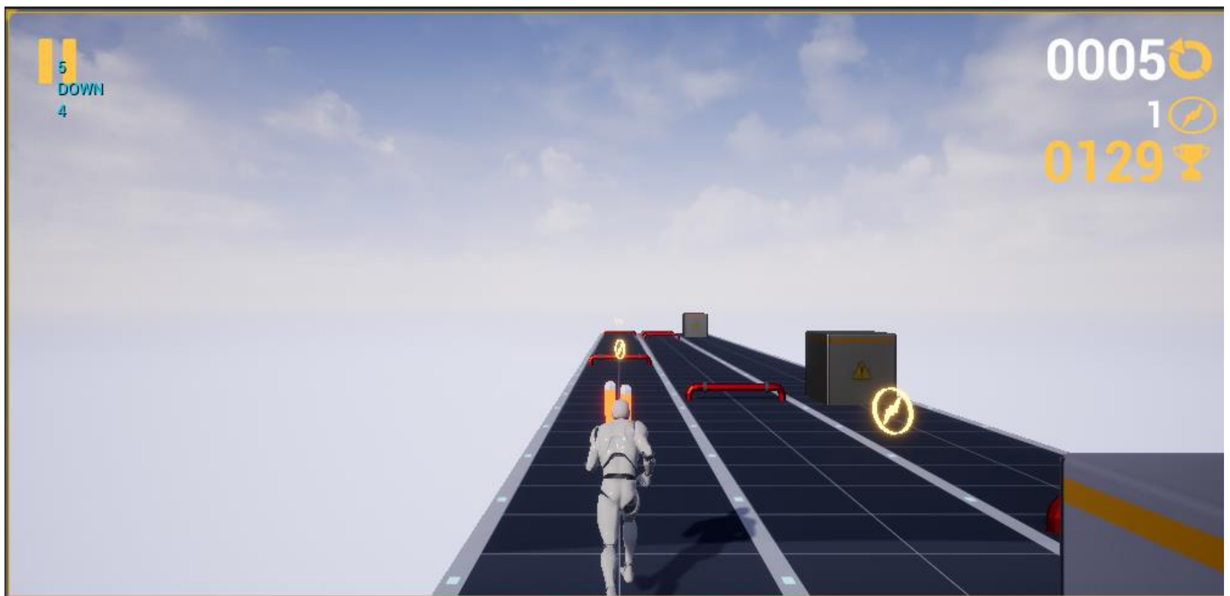


Рис. 1.2 Приклад вигляду основної частини додатку

По-перше необхідно описати логіку поведінки нашого ігрового світу: гравітацію, перешкоди, точки початку генерації рівня та створення нашого персонажу. Дані функції бере на себе використовуваний ігровий рушій: Unreal Engine 4. Його функціонал містить у собі логіку вже описану: гравітацію, зовнішнє середовище, початкові матеріали для створення персонажу гри, та опис його основних елементів поведінки: рухи, коллізію (зчитування зіткнення з навколишнім середовищем) та макет.

Всі ці елементи є базовими і включені в стандартний пакет ігрового рушія, що спрощує нам роботу з логікою гри.

По-друге необхідно створити всі матеріали необхідні для оформлення гри – моделі, текстури до моделей, налаштувати їх розміри та перевірити на

належне використання. Для даної роботи буде використаний редактор Adobe Photoshop або його аналог Adobe Illustrator, для розробки 3д моделей буде використано редактор Blender. У цих програмах будуть згенеровані основні елементи, такі як: перешкоди, частини ігрового світу та предмети, з якими гравець взаємодіє.

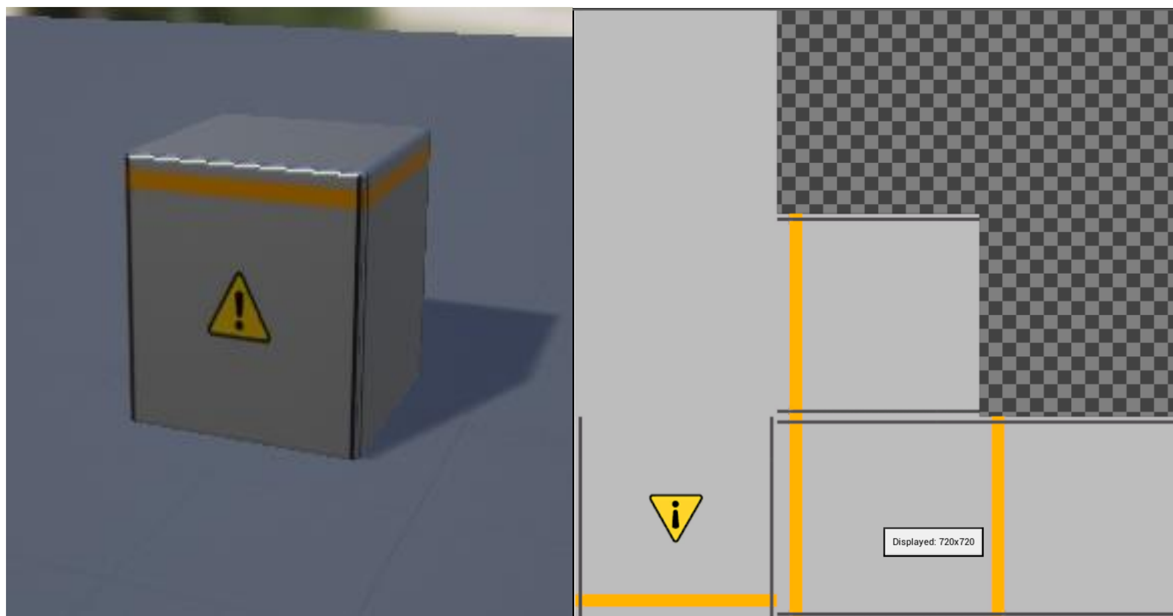


Рис. 1.3 Приклад текстур та моделей

Також потрібно правильно оформити логіку гри – нескінченний біг нашого персонажу зі зростаючою швидкістю, випадково згенеровану генерацію перешкод, перевірку на зіткнення, анімації персонажа, генерація додаткових елементів та взаємодія з ними, а також опис команд введених за допомогою жестів (тактильна взаємодія).

Після цього необхідно портувати даний проект на систему Android з дотриманням усього функціоналу, забезпеченням правильної роботи та корректним відображенням інтерфейсу користувача.

Для забезпечення всього функціоналу буде використана візуальна мова програмування для Unreal Engine – Blueprint. Дана мова програмування є візуальною версією коду мови програмування високого рівня C++. Вона дозволяє працювати з такими елементами коду як: класси, функції, актори (об’єкти, які можуть бути розміщені у ігровому світі), об’єкти з якими

можлива взаємодія – (з англ. pawn) та персонажі (з англ. characters) – об’єкти які можуть рухатися, взаємодіяти з іншими акторами, або об’єктами.

Вся логіка даної мови програмування будується на блоках, кожен з яких виконує свою певну функцію та реалізує певні, звичайні для програмування операції: опис об’єктів, класів, функцій, їх виклики, створення змінних різних типів, взаємодія різних частин коду один з одним.

Загалом існує 6 типів блоків:

1) Event (подія) - деяка подія, після якої починає виконуватися наступний код. Приклади подій: event tick (викликається кожен кадр), event - beginPlay (викликається при запуску гри), event - onActorBeginOverlap (викликається, коли хітбокс будь-якого об'єкта перетнувся з хітбоксом іншого об'єкта). Події - це власне функції, які викликаються самим двигуном при певних умовах.
2) Функції – звичайні функції, як і у інших частина програмування. Складають більшу частину коду на Blueprint. Існують внутрішньоігрові функції, які можна використовувати, наприклад: Get / Set variable - отримує змінну або встановлює в неї вказане значення, AddActorWorldOffset - зрушує об'єкт в світових координатах на вказану відстань. PrintString - друкує на екрані вказаний текст. Текст видно тільки в редакторі, в уже компільованою грі його не буде. Delay - призупиняє виконання скрипта на вказану кількість часу і так далі. Також функції можуть бути створені самим програмістом, після чого викликані в коді. Функції можуть приймати або повертати значення, в такому випадку у блоку функції з'являться додаткові контакти, зліва і справа, для під'єднання вступних значень і отримання висновку.
3) Блок Branch (з англ. Branch - гілка) - аналог умовного оператора if-else. Даний оператор буде перевіряти виконання умови за аналогією з схожими операторами у інших мовах програмування.

4) Блоки Switch on ... (з англ. Switch - перемикач) - аналог оператора switch case. Тобто блок, який в залежності від заданих параметрів буде видавати відповідний результат.
5) Блоки циклів for, for with break (цикл for з умовою виходу), while. Блоки, які забезпечують циклічне виконання певних частин коду.
6) Блок Construction Scipt (англ. Construction - будівництво) - спеціальна функція, що викликається при генерації об'єкта або зміні його характеристик. Може бути викликана навіть до початку гри в редакторі.

Всі показані блоки будуть використані у розробці даного проекту і описані нижче, для наглядного прикладу прикладений рисунок:

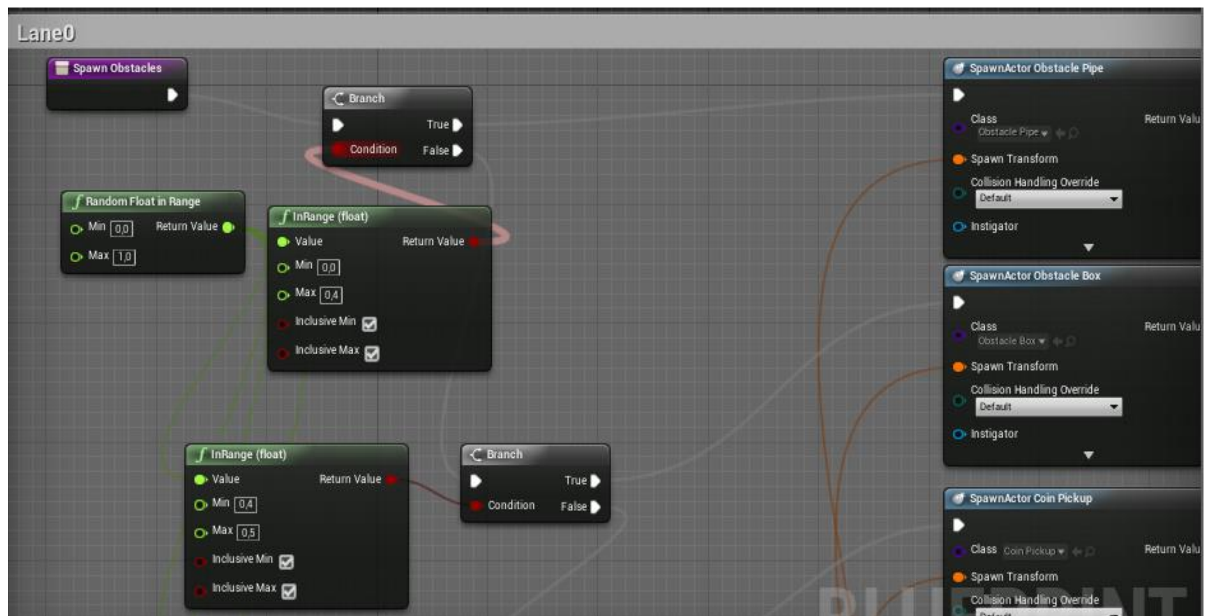


Рис. 1.4 Приклад коду Blueprint

Зазначу, що Blueprint - строго типізована мова (як і C ++), тому всі змінні повинні мати свій тип даних. Підтримуються також масиви, словники, а також дана мова візуального програмування - об'єктно-орієнтована, тому підтримує всі принципи ООП: абстракція, інкапсуляція, успадкування і поліморфізм.

1.4 Порівняльний аналіз схожих додатків та методів їх розробки

Для розробки додатків на операційну систему Android зазвичай використовують такий програмний комплекс як Android Studio, а для розробки саме мобільних ігор можуть використовуватися такий ігровий рушій як Unity, або будь-яка мова програмування високого рівня.

Android Studio – це інтегроване середовище розробки створене з єдиною метою максимально просто і швидко розробляти програми для девайсів з однойменною операційною системою. Функціонал даної програми вражає: він спрощує роботу з SDK (комплект для розробки програмного забезпечення), надає зручний доступ до основних складових будь-якої програми (кнопки, перемикачі, текстові блоки), ємулює роботу девайсу з встановленною на ньому Android системою, а також у максимально короткі строки забезпечує збирання виконуючого файлу з розширенням (.apk).

Дане середовище підтримує дві високорівневі мови програмування : Java та C++, хоча в будь-який час може бути переналаштоване на інші мови програмування.

Хоча даний інструмент і вважається одним із основних у розробці будь-якого додатку, але у випадку розробки саме гри він не є ефективним, оскільки має ряд недоліків, таких як: висока складність розробки (необхідність вручну дописувати логіку програми, створювати персонажів і т.д), низька швидкість компіляції (оскільки проект такого масштабу буде використовувати багато оперативної пам’яті) та відсутність функціоналу тестування саме функцій гри.

Інтерфейс Android Studio виглядає наступним чином:

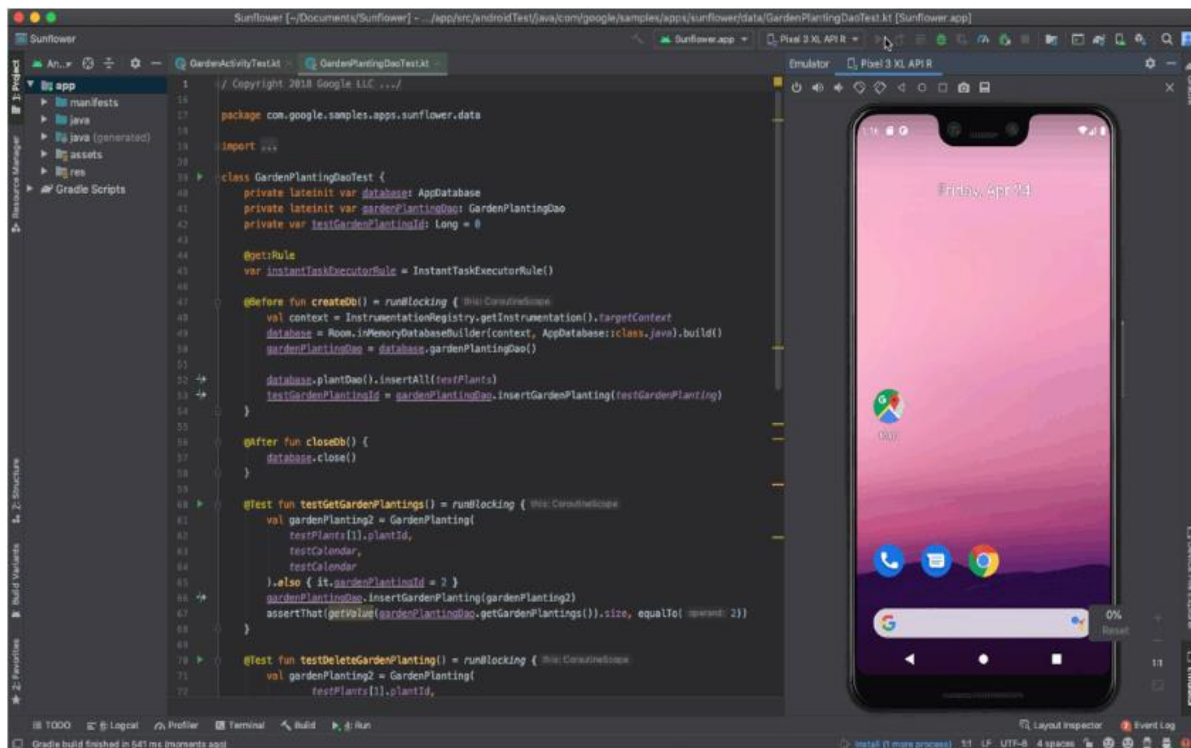


Рис. 1.5 Інтерфейс Android Studio

Дані недоліки є занадто значними для використання даного застосунку у своїй роботі.

Наступним інструментом, який міг бути використаний у процесі побудови даного проекту є ігровий рушій Unity.

Unity – це міжплатформенне середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies. Unity дозволяє створювати додатки, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші. Випуск Unity відбувся в 2005 році і з того часу йде постійний розвиток.

Основними перевагами Unity є наявність візуальної середовища розробки, міжплатформенної підтримки і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами і труднощі при підключенні зовнішніх бібліотек .

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

Даний інструмент також надає дуже зручний інтерфейс для налаштування гри, зокрема підтримує зручну систему Drag&Drop (з англ. – «візьми та переклади), редактор рівнів, систему роботи як з двовимірними так і з трьохвимірними об’єктами, роботу з анімаціями, текстурами, звуком, відео-файлами.

Також Unity автоматично знижує навантаження на оперативну пам’ять за допомогою використання системи Level of Detail, яка автоматично знижує деталізацію моделей при віддаленні від гравця.

Серед сильних і слабких сторін можливо видиліти такі:

Як правило, ігровий движок надає безліч функціональних можливостей, що дозволяють їх задіяти в різних іграх, в які входять моделювання фізичних середовищ, карти нормалей, динамічні тіні і багато іншого. На відміну від багатьох ігрових движків, у Unity є дві основні переваги: наявність візуального середовища розробки та міжплатформенна підтримка. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегроване середовище, лінію складання, що направлено на підвищення продуктивності розробників, зокрема, етапів створення прототипів і тестування. Під міжплатформенною підтримкою надається не тільки місця розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі і т. Д.), але і наявність інструментарію розробки (інтегроване середовище може використовуватися під Windows і Mac OS).

Останньою перевагою називається модульна система компонентів Unity, за допомогою якої відбувається конструювання ігрових об'єктів, коли останні є комбінованими пакети функціональних елементів. На відміну від механізмів успадкування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків, а не об’єднання в вузли дерева успадкування. Такий

підхід полегшує створення прототипів, що надзвичайно актуально при розробці ігор.

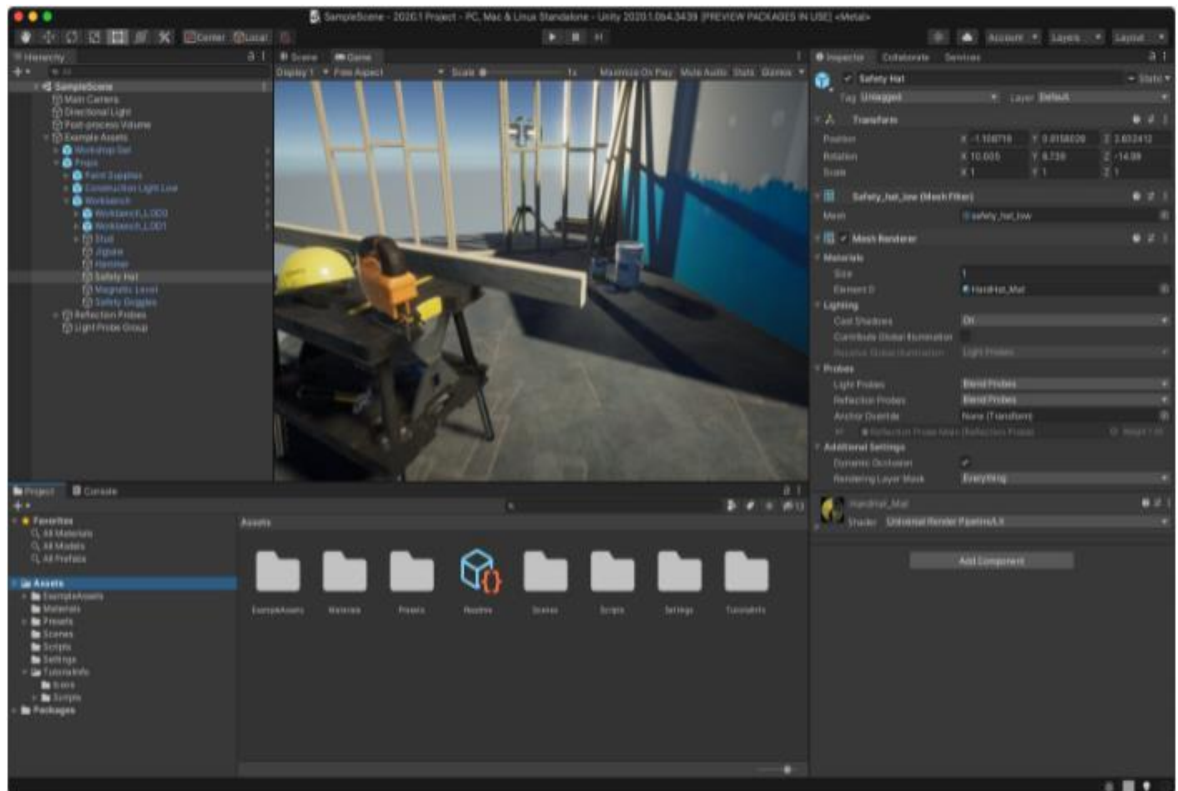


Рис. 1.6 Середовище розробки Unity

Як недоліки наводяться обмеження візуального редактора при роботі з багатокомпонентними схемами, коли в складних сценах візуальна робота ускладнюється. Другим недоліком називається відсутність підтримки Unity посилань на зовнішні бібліотеки, роботу з якими програмістам доводиться налаштовувати самостійно, і це також ускладнює командну роботу. Ще один недолік пов'язаний з використанням шаблонів примірників (з англ. Prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним. Також, WebGL-версія движка, в силу специфіки своєї архітектури (трансляція коду з C # в C ++ і далі в JavaScript), має ряд невирішених проблем з продуктивністю, споживанням пам'яті і працездатністю на мобільних пристроях.

Зазначимо, що дане середовище розробки може бути використане для побудови аналогічного проекту, але Unreal Engine 4 надає більший

інструментарій, та використовує кращу графіку, що забезпечує вцілому більш приємний користувацький досвід.

Також, для розробки будь-якого додатку може бути використана будь-яка мова програмування високого рівня, така як Python, Java, C++ та інші. Даний вид розробки не застосовувася, оскільки є неймовірно ресурсно-затратним, потребує розробки багатьох аспектів проекту і навіть побудови власного ігрового рушія.

Висновки до розділу

У даному розділі ми дослідили методи проетування даного дипломного проекту, визначили основні підходи та технологіїі до розробки а також набір необхідного програмного забезпечення і визначили жанр додатку-гри. Аналогом нашого проекту, є популярна гра – Subway Surfers. Це гра-платформер – одна з найпопулярніших ігр у світі з понад мільярдом завантажень, вона передбачає інтуітивно зрозумілий інтерфейс, процедурно-генерований світ та взаємодію персонажа з навколишнім середовищем. Загалом, даний проект унаслідує багато притаманних рис характеру даного жанру, в зв’язку з чим може бути проведена аналогіями між проектом та даною грою.

Рис. 1.7 Геймплей Subway Surfers

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ-ГРИ ДЛЯ ОС ANDROID

2.1 Специфікація вимог до мобільного додатку-гри

Як будь-який програмний продукт додаток-гра повинна мати певні вимоги до своєї розробки. Кінцевий продукт повинен задовольняти вимогам поставленим у даному розділі. Вимоги наведені нижче будуть наведені за всіма необхідними стандартами :

1)*Генерація ігрового світу. Створення ігрового світу для подальшої взаємодії з ним.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – звичайний.

Вимоги до якості:

- зручність використання (інтуітивний інтерфейс для користувача, захищеність від помилок при взаємодії) – гравець має швидко і зручно взаємодіяти з навколишнім середовищем гри.

- ефективність – ігрове поле має ефективно виконувати поставлені перед ним завдання, не викликати помилок.

2)*Робота системи управління. Система, яка надає користувачу інтерфейс для управління на мобільному девайсі.* Стан вимоги – відібрано. Пріорітет – критичний. Рівень ризику – значимий.

Вимоги до якості:

- захищеність (конфіденційність, захищеність від несанкціонованого доступу, змін) – система не повинна давати збоїв та бути правильно налаштована користувач не буде мати можливості налаштувати її (для забезпечення захисту від помилок).

- зручність використання (вивчаємість, керованість, естетика інтерфейсу користувача) – система управління має бути зручною та інтуітивно зрозумілою.

3)*Створення ігрового поля. Ігрове поле автоматично генерується та взаємодіє з персонажем гри.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – значимий.

Вимоги до якості:

- надійність (завершеність, придатність та відмовостійкість) – ігрове поле повинно генеруватися завжди та без помилок.

- ефективність виконання (використовуваність ресурсів, здатність до місткості) – ігрове поле повинно ефективно використовувати ресурси як мобільного девайсу так і девайсу для розробки.

4)*Створення системи підрахунку рахунку. Система яка буде автоматично підраховувати загальний рахунок.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – значимий.

Вимоги до якості:

- супроводжуваність (можливість багаторазового використання та аналізованість) – система повинно чітко працювати і безвідмовно.

- захищеність (конфіденційність та цілістність) – доступ до рахунку повинен бути лише у розробника гри.

5)*Система руху персонажу гри. Персонаж мобільного додатку повинен коректно реагувати на всі введені команди користувача.* Стан вимоги – відібрано. Пріорітет – критичний. Рівень ризику – значимий.

Вимоги до якості:

- зручність використання (керованість, захищеність від помилок користувача) – система повинна бути зручною та зрозумілою.

-надійність (відновлюваність та відмовостойкість) – помилки у роботі системи мають бути відсутні навіть при багаторазовому перезапуску гри.

6)*Забезпечення кроссплатформеності. Додаток повинен бути розроблений під ОС Android, але тестування повинно проводитися на Windows.* Стан вимоги – відібрано. Пріорітет – критичний. Рівень ризику – значимий.

Вимоги до якості:

- функціональність (точність та завершеність) – додаток повинен однаково добре функціонувати як при тестуванні так і у кінцевому релізі.

- переносимість (адаптованість та зручність установки) – система повинна однаково добре працювати на будь-якому пристрої з данними операційними системами.

7)*Система меню. Меню гри яке буде управляти основними функціями.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – значимий.

Вимоги до якості:

- зручність використання (досяжність, захищеність від помилок користувача) – користувач повинен ефективно і зручно використовувати дану систему.

8)*Інтерфейс користувача.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – звичайний.

Вимоги до якості:

- зручність використання (керованість, естетика інтерфейсу користувача) – інтерфейс користувача повинен надавати вичерпну інформацію про гру, рахунок та іншу необхідну інформацію.

9)*Система генерації предметів для взаємодії. Автоматично генерує предмети з якими може взаємодіяти наш гравець.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – звичайний.

Вимоги до якості:

- функціональність (точність та завершеність)– забезпечення постійної генерації предметів.

- надійність (відмовостійкість, завершеність).

10)*Система тестування. Надає зручний інструментарій для тестування нових функцій та окремих частин додатку.* Стан вимоги – відібрано. Пріорітет – критичний. Рівень ризику – звичайний.

Вимоги до якості:

- надійність (відновлюваність, відмовостійкість) – система повинна забезпечувати можливість постійного проведення тестування та усунення помилок.

- зручність використання (інтерфейс користувача, керованість) – система тестування має мати зручний інтерфейс який у короткі сроки дозволить протестувати та усунути можливі помилки, нові функції та перевірити функціонал всього додатку.

11)*Система взаємодії персонажа гри з навколишнім середовищем та іншими елементами. Повинна реєструвати тригери для взаємодії персонажу з навколишнім середовищем, забезпечувати виконання анімацій та коректну роботу.* Стан вимоги – відібрано. Пріорітет – важливий. Рівень ризику – звичайний.

Вимоги до якості:

- ефективність виконання (швидкодія та адаптивність) – у разі фіксації моменту взаємодії повинен виконувати необхідні анімації незалежно від інших зовнішніх факторів.
- надійність (придатність до використання, безвідмовність) – система повинна виконувати свої функції безвідмовно.

Дані вимоги при їх виконанні повинні забезпечити наш додаток безвідмовним функціоналом та усунути всі можливі помилки. Зазначимо що це лише основні вимоги, які повинні бути виконані для забезпечення мінімальних достатніх функцій.

2.2 Розробка архітектури додатку на ОС Android

Наш додаток буде складатися з таких складових: система ігрового світу, система управління, система тестування, анімації та все пов’язане з персонажем гри, система генерації ігрового поля, створення перешкод та інших предметів з якими може взаємодіяти наш ігровий персонаж.

Для наглядного прикладу ми використаємо Use-case діаграми (діаграми варіантів використання), у яких буде описаний повний функціонал нашого застосунку:

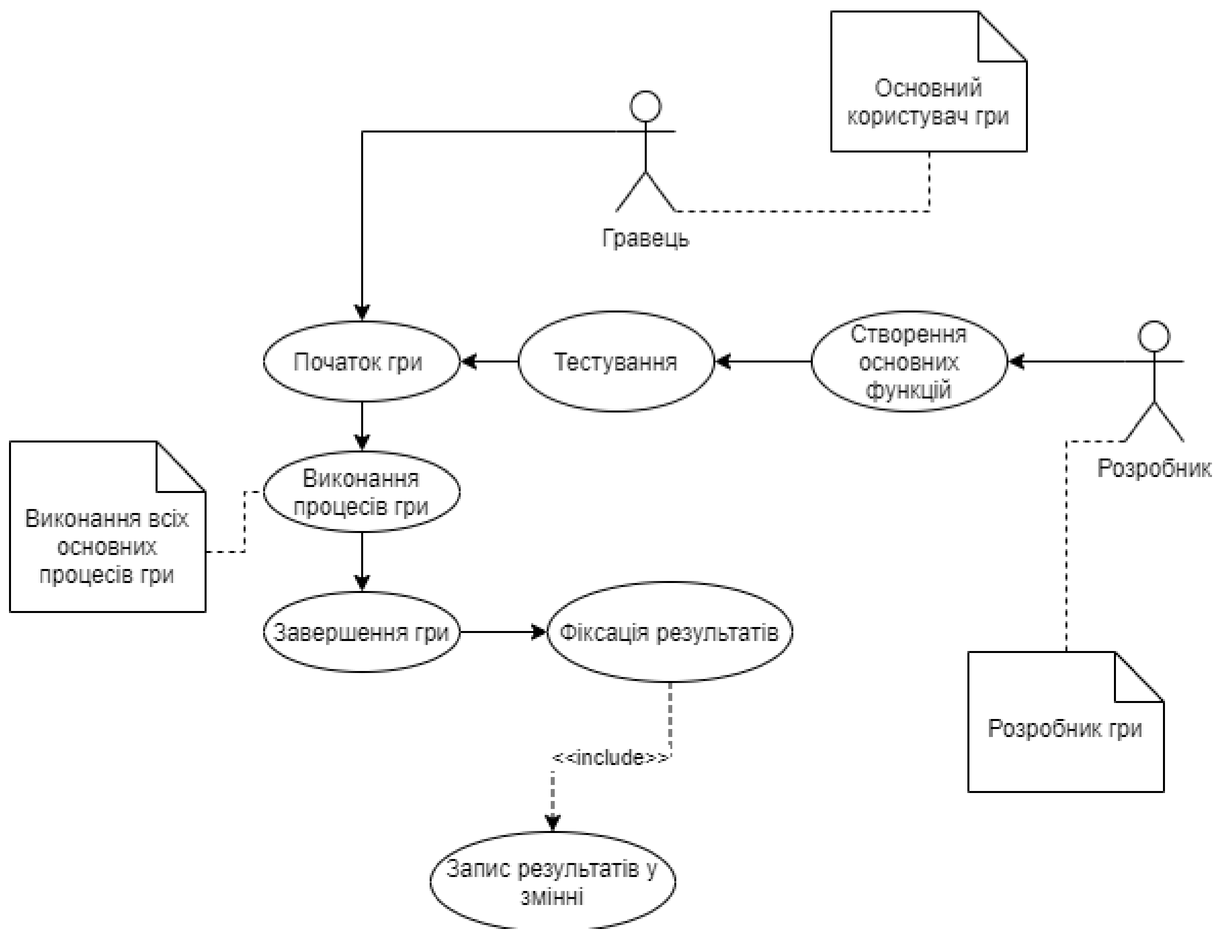


Рис. 2.1 Основна Use Case діаграма

На даній діаграмі відображені основні процеси які відбуваються під час розробки додатку та його використання гравцем (користувачем). Дані варіанти використання потребують декомпозиції для пояснення повного функціоналу.

У наступній діаграмі буде проведена декомпозиція варіанту використання «Початок гри» та наглядно продемонстровані всі процеси.

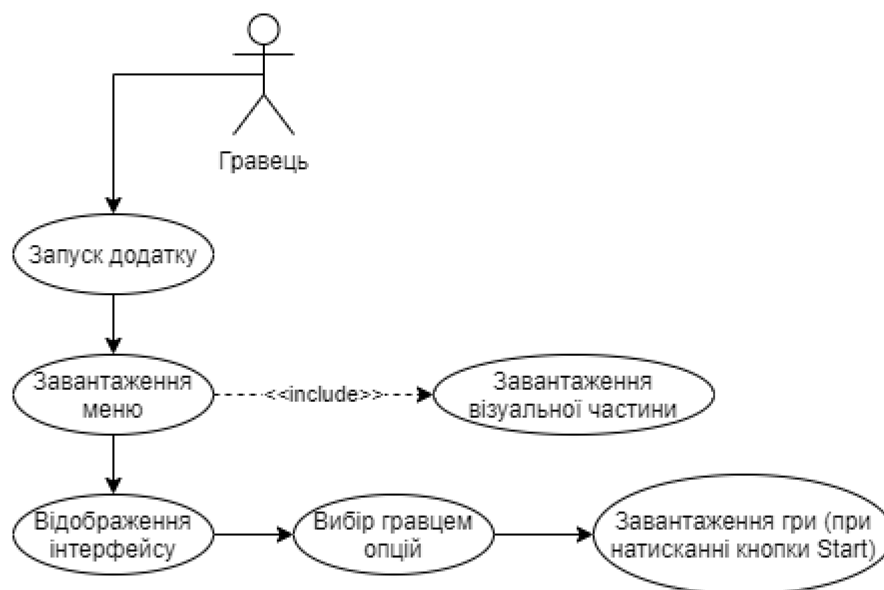


Рис. 2.2 Декомпозиція варіанту використання «Початок гри»

Дана декомпозиція демонструє всі процеси які відбуваються від початку завантаження користувачем додатку і до запуску безпосередньо гри, дані функції будуть описані нижче при безпосередній розробці.

Далі буде продемонстрована декомпозиція варіанту використання «Виконання процесів гри».

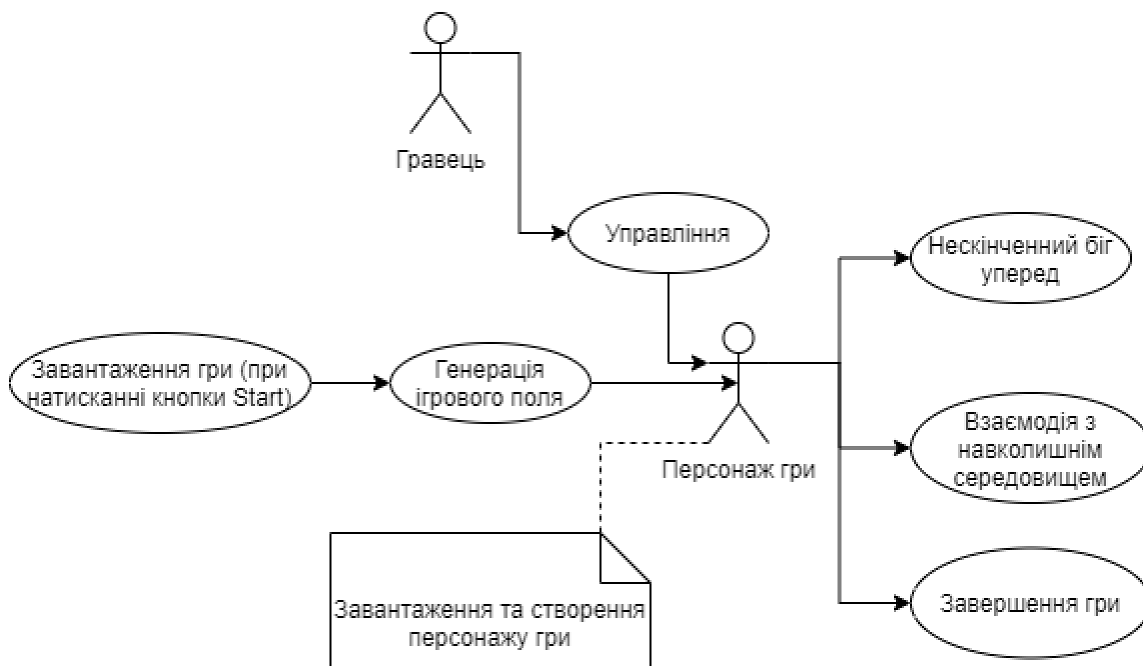


Рис. 2.3 Декомпозиція варіанту використання «Виконання процесів гри»

На даній діаграмі продемонстровані процеси, які відбуваються під час виконання основних функцій гри. Тобто, після завантаження даного рівня гравець буде мати змогу керувати персонажем, який нескінченно рухається уперед, має можливості взаємодіяти з навколишнім середовищем (підбирати предмети, переміщуватися у просторі, стрибати) та після зіткнення виконується функція завершення гри.

Останньою декомпозицією будуть продемонстровані функції варіанту використання «Завершення гри».

Рис. 2.4. Декомпозиція варіанту використання «Завершення гри»

Дана діаграма демонструє процеси, які відбуваються після зіткнення гравця з перешкодою. По-перше до персонажу гри застосовується «анімація смерті» з подальшим видаленням персонажу з ігрового поля, а по друге всі основні ігрові процеси зупиняються. Далі на екран девайсу виводиться «екран завершення гри» у якому гравець може зробити вибір, чи повернутися йому до головного меню, чи перезапустити гру.

Висновки до розділу

У даному розділі ми визначили основні функції додатку, спроектували його зовнішній вигляд та майбутні рішення, які будуть використані в процесі розробки. Даний проект буде складатися з таких основних папок: RunnerFiles ThirdPersonBp. У них будуть міститися всі необхідні матеріали для роботи додатку.

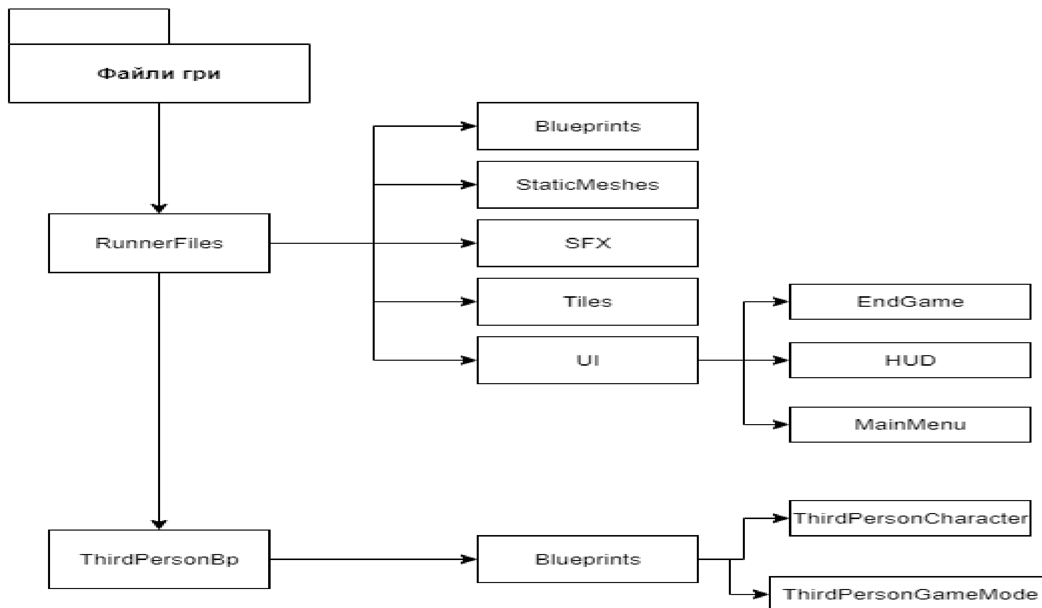


Рис. 2.5. Структура проекту

У папці RunnerFiles містяться усі необхідні матеріали для роботи логіки гри. А у папці ThirdPersonBp – логіка поведінки ігрового персонажу, та логіка світу. У папці BluePrints – основні елементи мови Blueprint, яка забезпечує функціонал гри і буде розглянута нижче. StaticMeshes – 3д моделі, які використовує додаток, SFX – звуки, а UI – інтерфейс користувача. EndGame, HUD, MainMenu містять файли необхідні для відображення всіх екранів.

На даному етапі проект повністю спроектований та підготовлений до програмної реалізації.

РОЗДІЛ 3. РОЗРОБКА ДОДАТКУ ТА ЙОГО СТРУКТУРИ

3.1 Схема роботи додатку-гри на ОС Android

Для створення гри-додатку нам необхідні (як було вище зазначено) такі складові. Ігровий світ, який передбачає усю необхідну логіку та функціонал гри, а також забезпечує виконання основних функцій в тому числі і фізичні явища, анімації, тощо. Процедурно-згенероване ігрове поле, яке буде автоматично створюватись у ігровому світі та взаємодіяти з ігровим персонажем, перешкодами. Ігровий персонаж, який матиме змогу пересуватися у ігровому світі, взаємодіяти з ігровим полем та іншими об’єктами та матиме власні анімації. Перешкоди та об’єкти з якими буде взаємодіяти наш ігровий персонаж. Також, необхідна логіка взаємодії персонажу гри з певними предметами, яка буде складатися з певних функцій, анімацій та безпосередньо впливати на ігровий світ.

Для оформлення гри потрібно створити ігровий інтерфейс, з яким користувач буде взаємодіяти, починати гру, ставити її на паузу, меню управління та всі необхідні функції. Будуть використані вже створені 3-Д моделі та малюнки для надання належного вигляду.

Розробка управління для Android девайсу є також ключовим моментом, повинна бути створена підтримка і зчитування жестів, якими буде керуватися логіка розробленого додатку-гри. Для переносу гри з персонального комп’ютеру на мобільний девайс будуть використані вбудовані можливості ігрового середовища розробки Unreal Engine 4 та пакети SDK для операційної системи Android, які забезпечать коректний та безвідмовний перенос функціоналу та підтримку роботи на зазначеній вище операційній системі.

Схема роботи мобільного додатку є дуже простою. Заздалегіть скомпільована гра запускається на девайсі з встановленою системою Android, йде завантаження необхідних файлів та користувач потрапляє у головне меню. Дане меню є інтуітивно зрозумілим, та надає можливості для запуску

гри. При натисканні кнопки Play (з англ – «Грати») відбувається запуск нашого ігрового світу, відбувається генерація ігрового поля та персонажу. Персонаж починає рух, а користувач має можливість керувати ним.

Запускається процес процедурної генерації ігрового поля. Тобто - метод алгоритмічного створення даних із використанням комбінацій алгоритмів, який передбачає випадковість. У комп'ютерній графіці він зазвичай використовується для створення текстур та 3D-моделей. У відеоіграх він використовується для автоматичного генерування великої кількості вмісту, а у конкретному випадку для генерації перешкод, предметів взаємодії і так далі. Залежно від реалізації, переваги процедурної генерації можуть включати менші розміри файлів, більше вмісту та випадковість для певного ігрового процесу, що у даному проекті є ключовим аспектом.

Персонаж рухається під управлінням користувача (гравця), долає перешкоди, взаємодіє з різними предметами, тим часом автоматично відбувається підрахунок пройденої дистанції.

Після зіткнення ігрового персонажу з перешкодою, застосовується анімація смерті та показується «екран смерті», який надає можливість користувачу або вийти в меню, або розпочати гру з початку.

Зазначу, що дана схема роботи є загальною та надає інформацію лише про найбільш значущі аспекти роботи додатку, детальніше буде розписано у наступних розділах.

3.2 Розробка додатку-гри на ОС Android

3.2.1 Початковий етап розробки

Як зазначалося вище для розробки мобільного додатку для ОС Android буде використано ігровий рушій – Unreal Engine 4, версії 4.23.1. Дана версіє не є останньою, але містить більш зручний (на думку автора роботи) інструментарій для роботи з анімаціями.

Для створення додатку встановимо такі налаштування: створимо новий проект, з видом гри від третього лиця, тобто камера розташована позаду ігрового персонажу, що є доцільним для вибраного жанру, та налаштуваннями під мобільні девайси.

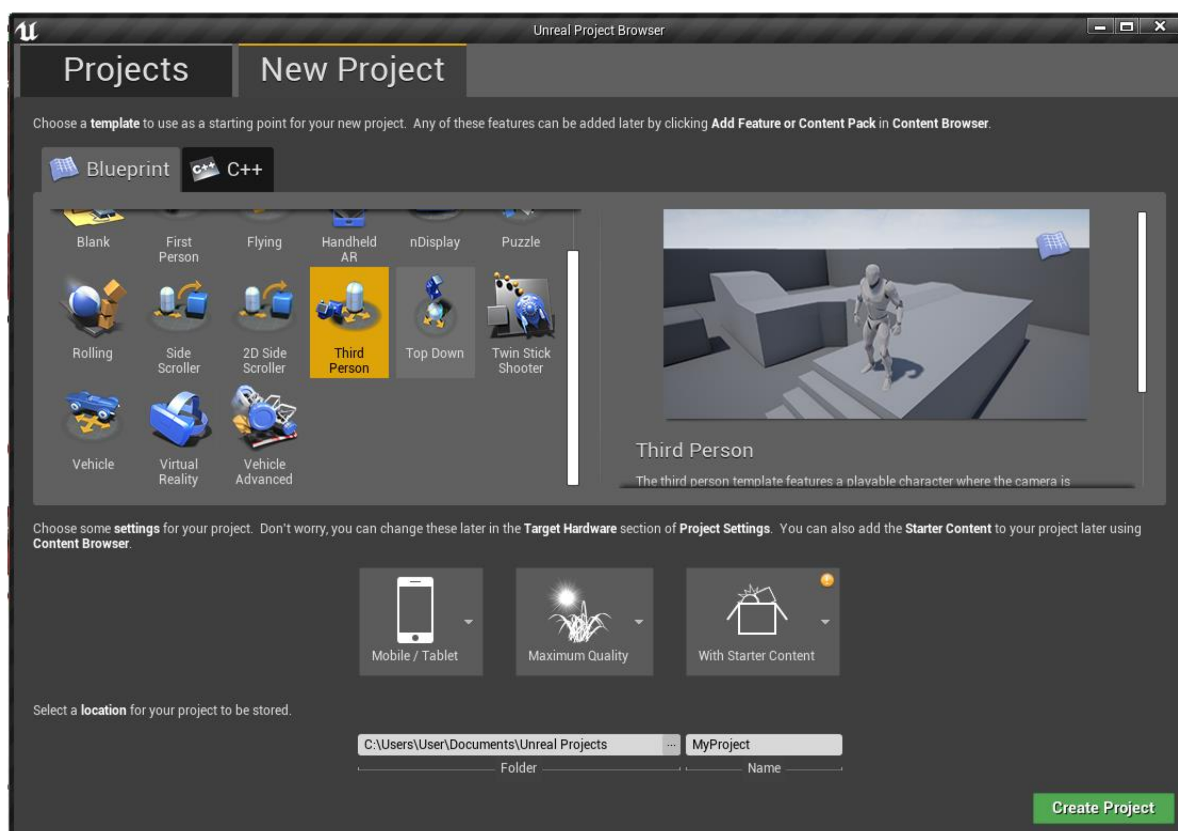


Рис. 3.1 Створення нового проекту

Даний проект буде містити всі базові необхідні нам налаштування, які будуть використані під час розробки веб-додатку. У процесі подальшої розробки ці аспекти дозволять нам не витрачати час і велику кількість ресурсів та сконцентруватися на механізмах роботи додатку і виконанні поставлених завдань.

У щойно створеному проекті створюється тестовий рівень, який за замовчуванням є стандартним та 2 Blueprint файли: ThirdPersonCharacter та ThirdPersonGameMode. Дані налаштування містять у собі основні функції гри від третього лиця, такі як: переміщення ігрового персонажу, стрибки, взаємодія персонажу з твердими об’єктами, та «правила ігрового світу». Також вони містять усі необхідні змінні та функції для обрахунку положення персонажу та об’єктів у просторі, їх геометричних розмірів, їх властивостей, кольору, тощо. На тестовому рівні це буде виглядати таким чином:

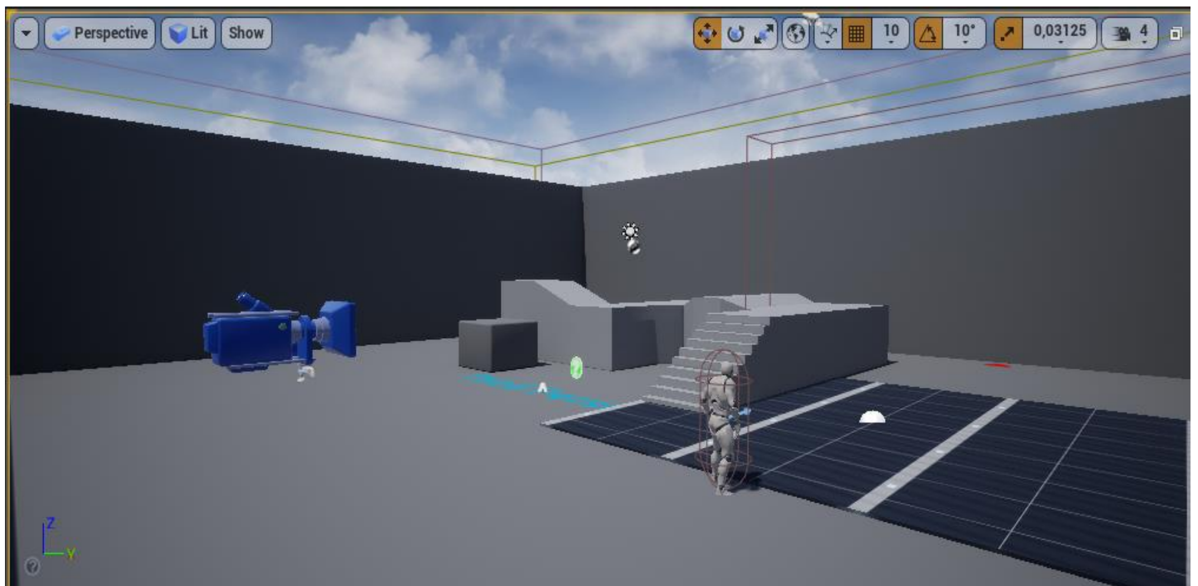


Рис. 3.2 Тестовий рівень додатку

Як ми можемо бачити камера знаходиться на місці, яке не задовольняє умови нашого ігрового світу. Для зміни «точки зору» ми заходимо у файл – ThirdPersonCharacter, у вкладку – Viewport, яка відповідає саме за положення камери та персонажу у просторі та надає приблизний вигляд цих об’єктів. Після цього за допомогою інструменту «Переміщення» пересуваємо камеру далі по осі X, та по осі Z, для того щоб підняти її над персонажем. Такі налаштування забезпечать користувачу необхідний кут огляду, та є одним з основних аспектів нашого додатку.

3.2.2 Рухи персонажу та основний елемент ігрового світу

Основний елемент будь-якої гри це анімація рухів, у даному додатку будуть застосовані анімації встановлені за замовчуванням, тобто у Blueprint ThirdPersonCharacter.

У нашому випадку нам необхідні лише рухи уперед, тобто персонаж постійно має пересуватись прямо. Для цього проведемо невеличкі налаштування у даній схемі. Знаходимо поле з коментарем – Movement, а у ньому знаходимо функцію AddMovementInput, яка відповідає за переміщення нашого ігрового персонажу та підключаємо її до події EventTick. Дана подія буде відбуватися кожен кадр рівня гри, де присутній даний персонаж. Тобто таким чином кожен кадр в якому присутній персонаж буде просувати його уперед (активувати анімацію руху).

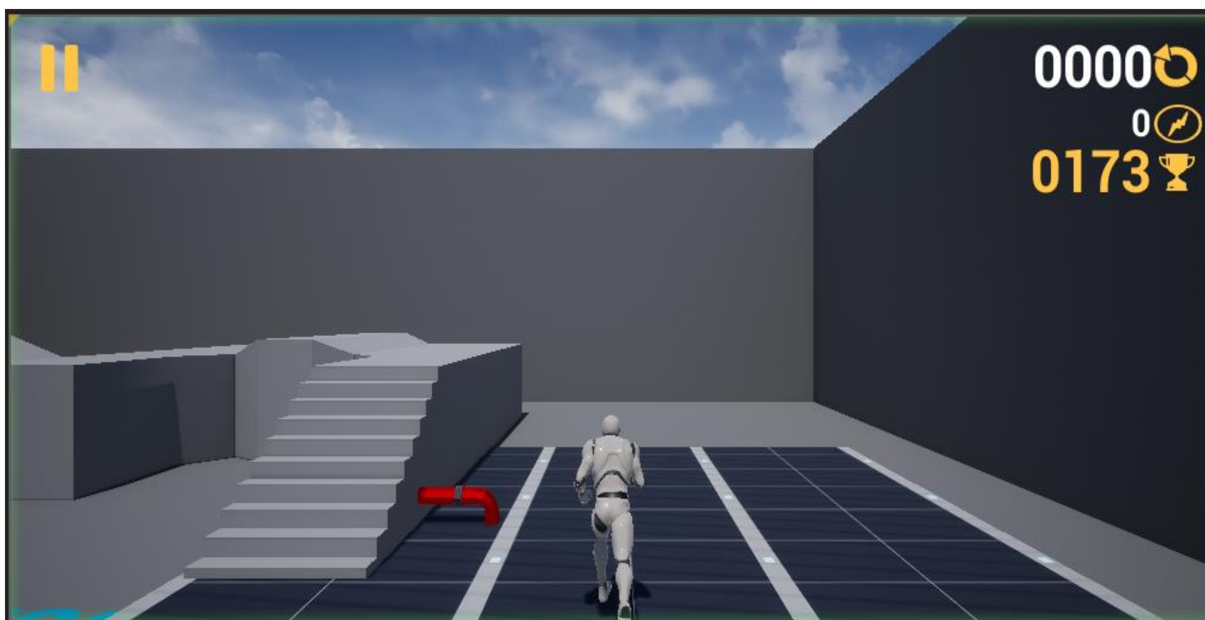


Рис. 3.3. Рух персонажу

Зазначимо, що EventTick може бути лише один на кожен BluePrint, для попередження виникнення помилок. Персонаж все ще буде мати можливість рухатись у інших напрямах у довільному порядку, а це йде у розріз з необхідним функціоналом гри. Для забезпечення функціоналу зміни «ліній руху» нам необхідно створити елемент ігрового поля по якому буде нескінченно рухатись уперед ігровий персонаж.

Таким елементом буде «плитка», яка матиме прямокутну форму та буде автоматично створюватись у ігровому світі. Для її створення нам необхідні такі деталі як: матеріали та текстури, та імпортувати їх у наш проект (звичайним перетягуванням). Всі використані матеріали та текстури були попередньо створені у таких засобах програмного забезпечення як: Blender та Photoshop. Вони мають необхідні розширення та можуть бути легко використані у даній роботі.

Далі створюємо об'єкт прямокутної форми, надаємо йому форму плитки та застосовуємо до нього текстури та матеріали і розташовуємо горизонтально стосовно просторової сітки, обов'язково з центром у точці (0,0,0). Далі додаємо такий об'єкт як стрілка з назвою SpawnPoint, а також 3 стрілки з назвами: Lane0, Lane1, Lane3. SpawnPoint повинна бути розташована прямо на краю нашої плитки, для забезпечення функції автоматичного створення ігрового поля.

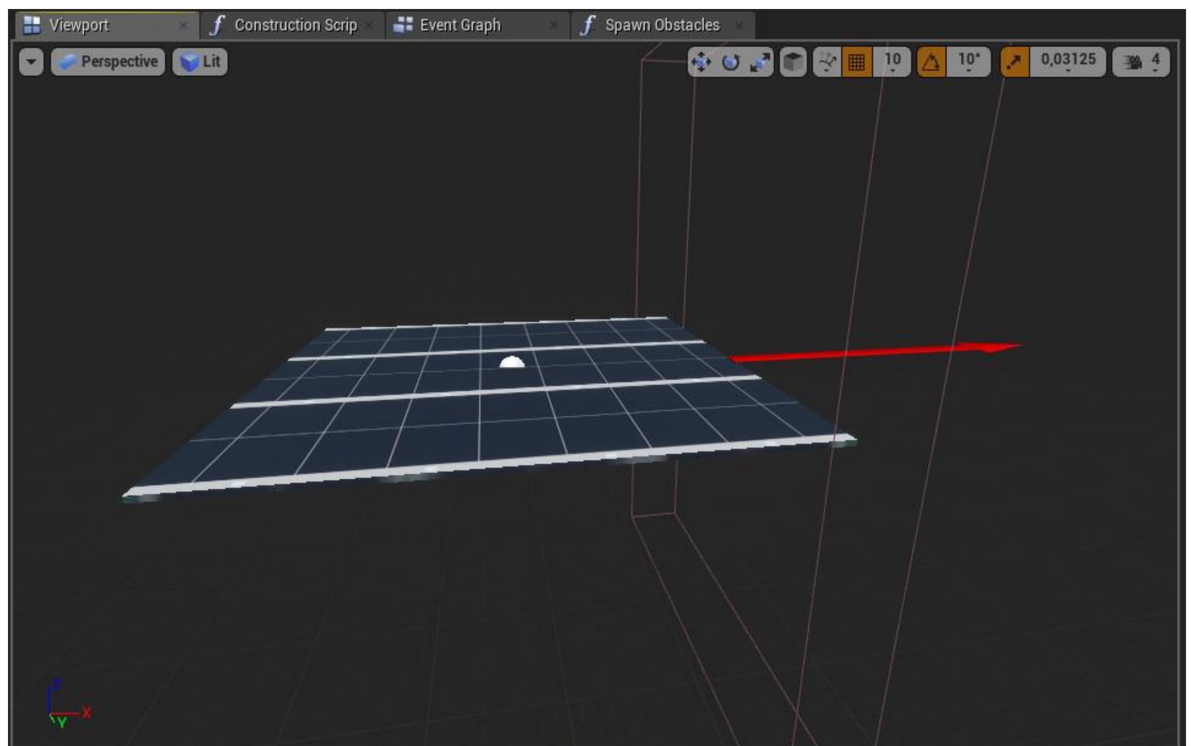


Рис 3.4 Основний елемент ігрового поля – плитка

Для цього у ThirdPersonGameMode створюємо функцію SpawnTile. Ця функція буде зчитувати позицію стрілки SpawnPoint та генерувати на її місці нову плитку за допомогою вбудованої функції SpawnActor та циклу, який

буде повторятися 10 разів. Після чого, через півтори секунди наша плитка буде зникати. Це зроблено для того, щоб не використовувати ресурси девайсу на якому буде запущений додаток, оскільки частина поля, яку персонаж пробіг буде позаду нього і фактично непотрібною.

Таким чином буде реалізований алгоритм нескінченної генерації ігрового поля.

Для реалізації переміщення персонажу по лініям «зміна ліній напряму руху» ми використаємо об’єкти-стрілки Lane0, Lane1, Lane2. Кожна з цих стрілок буде відповідати за умовну лінію по якій персонаж гри буде пересуватися уперед. Лінії пронумеровані від 0 до 2 відповідно зліва направо. Лінії повинні бути створені та поміщені на умовні частини нашої плитки, які були намальовані за допомогою відповідного матеріалу.

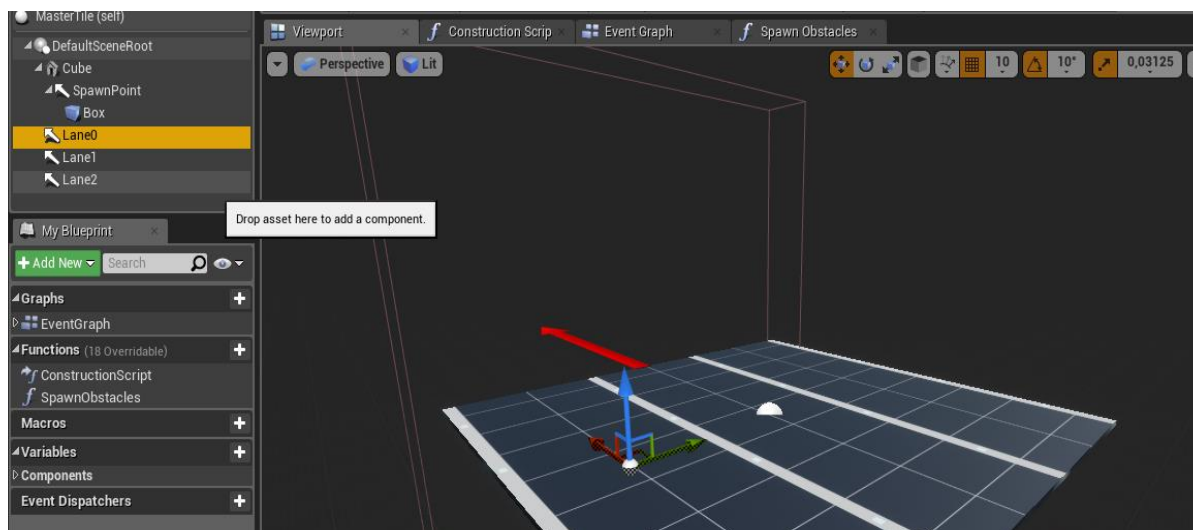


Рис. 3.5 Лінії напряму

Приблизні позиції ліній – (-250, 0, 0), (0,0.0) та (250,0,0) в трьох вимірах, одна лінія по центру, інші 2 зсунуті на 250 пікселів направо чи наліво.

Далі у Blueprint ThirdPersonCharacter ми створюємо 2 змінні Lane та NewLane та масив чисел Lane У.

За допомогою функції Clamp, яка перевіряє чи входять наші лінії у діапазон від 0 до 2 ми встановлюємо логіку нашої зміни ліній: в залежності від лінії ми можемо переміщатися або вліво, або у обидві сторони вбік, або направо. За домомогою LerpTimeline – вбудованої функції, яка дозволяє нам програти анімацію за певний проміжок час ми встановлюємо анімацію

переміщення у сторону на 1 секунду. Це додає плавності та коректності переходу, зміна виглядає дійсно переміщенням, а не «телепортацією». Після чого порівнюючи розташування персонажу на наших лініях пересуваємо його відповідно до введених команд і функції SetActorLocation яка встановлює положення персонажу, поки що з клавіатури за допомогою подій – Left та Right.

Для доцільного використання функції SetActorLocation нам також потрібно отримати позицію змінних X та Z. Це реалізується за допомогою функції GetWorldLocation, яка повертає значення змінних позиції у трьохвимірному просторі. Таким чином ми будемо мати генерацію плит та основні елементи руху у нашому додатку.

Рис. 3.6 Генероване ігрове поле та зміна напрямку бігу

Зазначу, що для коректної генерації ігрового поля потрібно створити новий рівень, який буде повністю порожнім, а у налаштуваннях нашого Blueprint для основної плити використати подію OnBeginPlay (подія яка відбувається при запуску рівня) та підключити її до функції створення поля. Це дозволить нам при початку гри у цьому світі почати процес генерації нескінченного ігрового поля та забезпечить нам захищеність від помилок.

3.2.3 Рахунок та прискорення

Як було показано вище, наша гра містить такий елемент як рахунок, а також повинна з часом ускладнюватись завдяки збільшенню швидкості руху ігрового персонажа.

Реалізуються ці функції наступним чином: у файлах Blueprint – MasterTile (основна плитка), ThirdPersonGameMode та ThirdPersonCharacter створюються змінні цілого типу – CurrentPoints та PointsMultiplier. Ці змінні будуть використані у файлі MasterTile.

Навколо плитки на її кінці створюється «прозорий» об’єкт через який може пройти наскрізь персонаж гри. Цей прозорий об’єкт є тригером який і буде виконувати функції прискорення та збільшення ігрового рахунку.

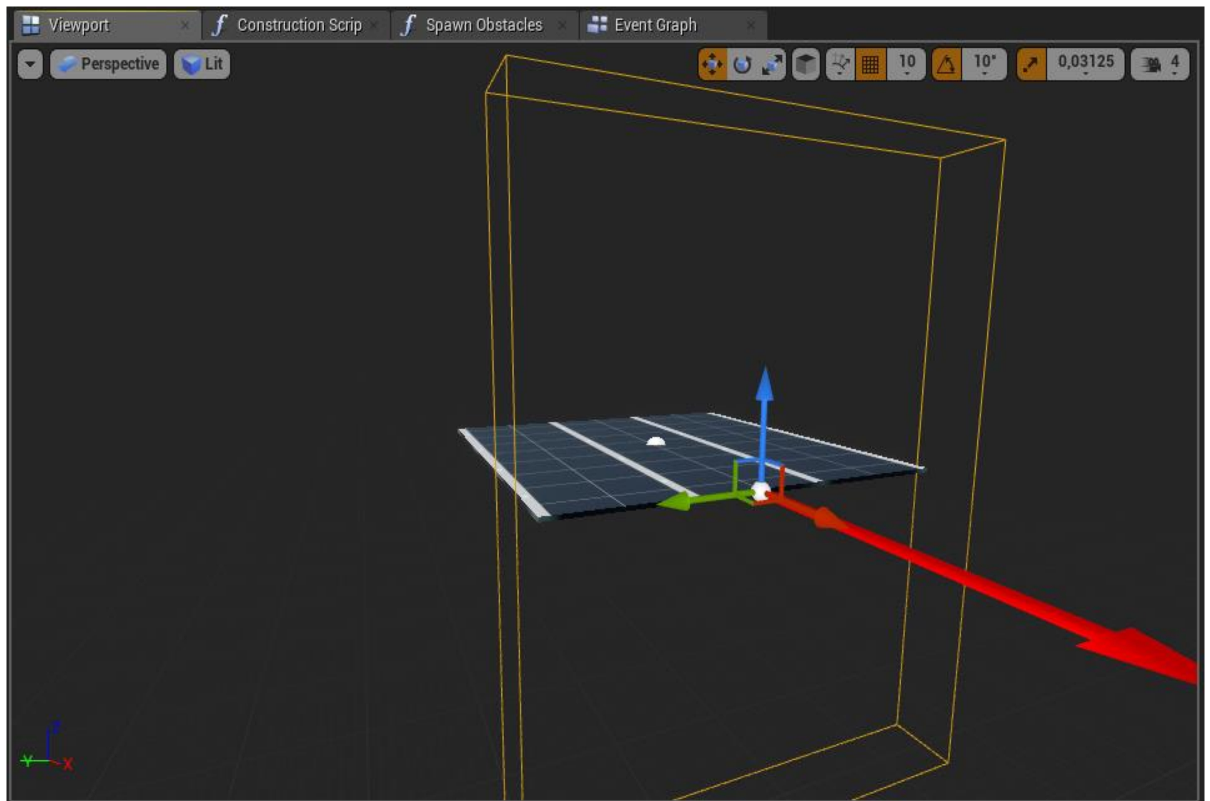


Рис. 3.7 Тригер

Даний об’єкт повинен мати подію OnComponentBeginOverLap (Box), та функцію CastToThirdPersonCharacter, що означає що при контакті з персонажем гри він буде виконувати певну дію, а саме викликати за допомогою функції CastToThirdPersonGameMode з функцією get GameMode, що повертає значення ігрового світу ми зчитуємо значення двох наших

змінних – CurrentPoints та PointMultiplier. Для PointMultiplier за допомогою операції множення ми встановлюємо коефіціент множення нашого рахунку. Далі обидва значення додаємо та за допомогою функції Set встановлюємо значення нашого рахунку.

При кожному проходженні через даний тригер до рахунку гравця буде додаватися та сума, яку встановлено у налаштуваннях гри.

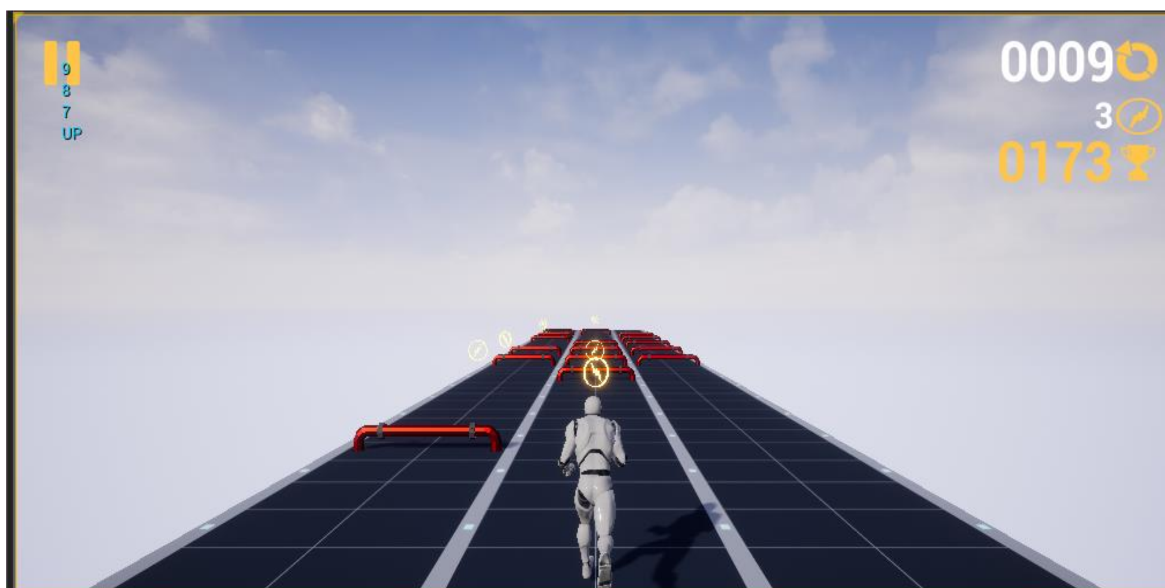


Рис. 3.8 Рахунок гри

Для збільшення швидкості ми знову використаємо функцію CastToThirdPersonCharacter та отримаємо об’єкт CharacterMovement, який містить інформацію про швидкість та основні параметри руху. Щоразу проходячи даний тригер ми домножуємо швидкість на 1.01, що збільшує її плавно але стабільно. Далі перевіряємо функцією Clamp на входження в діапазон від 600 (мінімально встановлена швидкість руху персонажу) до 1200 (максимальна швидкість яка задається розробником).

Після проходження тригеру максимальна швидкість персонажу буде постійно збільшуватись, поки, через деякий час, не досягне свого максимуму. Після проходження через даний тригер він також усувається функцією DestroyActor, оскільки свою функцію він виконав, а ресурси девайсу потрібно використовувати доцільно.

3.2.4 Створення перешкод для гравця

Перешкоди, які будуть випадковим чином з’являтися на шляху нашого ігрового персонажа та будуть становити для нього «смертельну» загрозу – один із основних елементів ігрового процесу.

В популярних ігрових додатках схожого жанру використовується 2 типу перешкод: перешкоди які можливо перестрибнути (або оминути іншим чином) та перешкоди які становлять найбільшу загрозу для гравця – вертикальні.

Обидва типи цих перешкод буде реалізовано з дотриманням стилістики гри і оформлено у приємній для користувача графічній стилістиці.

Перший тип перешкод, а саме ті, які можливо перестрибнути будуть мати форму водопровідної труби. Для створення даного об’єкту потрібно заздалегіть створити static mesh (з англ. – статичний об’єкт) у будь-якому 3-д редакторі та надати йому форми водопровідної труби. У даному проекті використовувався 3-д редактор Blender, який має ряд переваг таких як: безкоштовний спосіб дистрибьюції програмного забезпечення та низький поріг входу.

Також у самому ігровому рушії Unreal Engine 4 ми додаємо матеріал, з кольоровими налаштуваннями (1,0,0) у кольоровій системі RGB та додаємо металічного кольору використовуючи перемикач metallic. Такі налаштування нададуть даному об’єкту вигляду водопровідної труби та будуть яскраво виглядати.

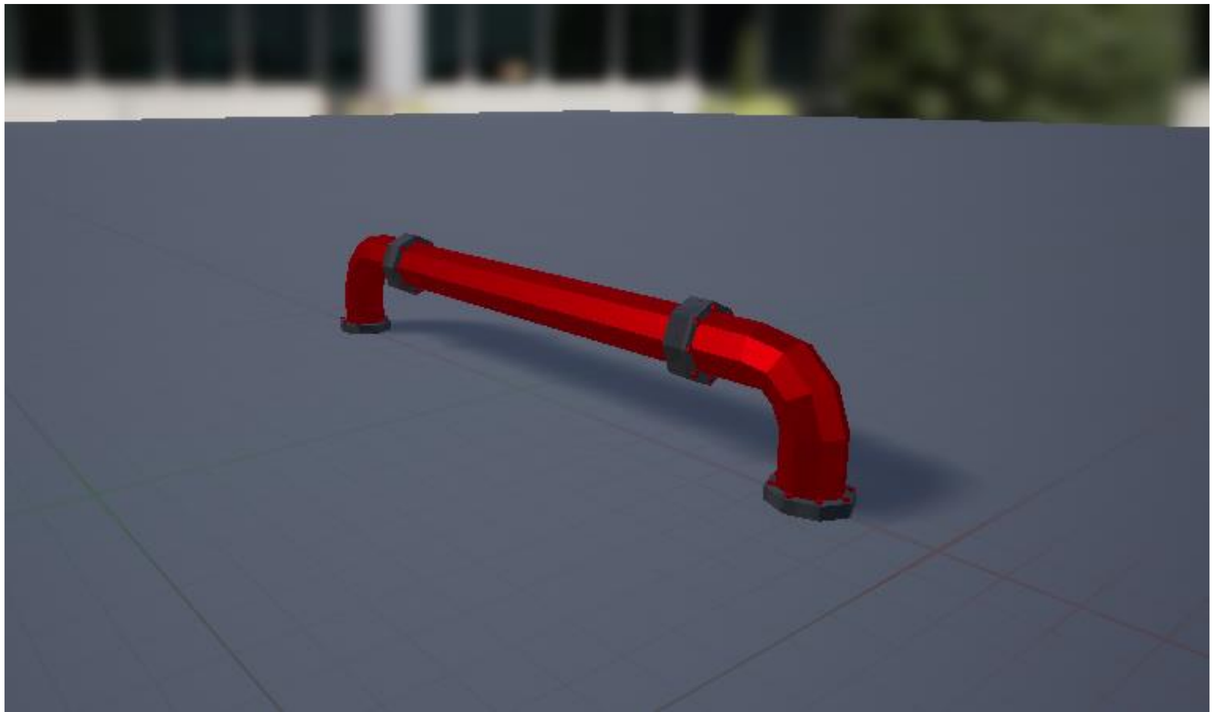


Рис. 3.9 Об’єкт - перешкода

Даний об’єкт повинен мати налаштування згідно з яких ігровий персонаж не буде мати можливості пройти крізь нього, тож ми встановимо налаштування Collision Presets на BlockAll. У даному проекті це забезпечить уникнення можливості проходження ігрового персонажу крізь об’єкт, тобто при зіткненні з даним об’єктом персонаж буде зупинятися за неможливістю продовжити рух далі.

Далі для взаємодії з персонажем та забезпечення функціоналу даного об’єкту нам необхідно створити Blueprint з назвою ObstaclePipe. Даний файл буде надавати нашому об’єкту функцію, яка при дотику ігрового персонажу до даної перешкоди буде активувати анімацію смерті та завершати гру.

Для цього у створеному Blueprint додаємо об’єкт за допомогою Static Mesh та вибираємо зі списку запропонованих дану водопровідну трубу. Вона з’являється у нашій вкладці Vievport. Масштабуємо об’єкт та розвертаємо його для забезпечення правильного механізму розташування об’єкту на нашому ігровому полі. На даний момент розробки нам необхідно перевірити як саме буде з’являтися перешкода у нашому ігровому світі, а для цього потрібно написати функцію, яка забезпечить генерацію даної перешкоди.

Переходимо у Blueprint нашого основного елементу ігрового поля, а саме: Master Tile. Переходимо на вкладку Functions та натискаємо кнопку «Add Function», надаємо функції ім'я – SpawnObstacles.

Дана функція буде відповідати за появу об'єктів на нашому ігровому полі, а саме генерацію об'єктів на місцях, які були заздалегіть позначені Lane0, Lane1 та Lane2.

За появу об'єктів у ігровому світі відповідає класс SpawnActor. Даний клас є вбудованим у ігровий рушій та надає зручний функціонал для створення об'єктів у ігровому світі. У нашому випадку нам необхідно протестувати процес генерації перешкоди, отож підключаємо тіло функції SpawnObstacles напряму до блоку функції SpawnObstacles.

Далі нам необхідно задати параметри для правильної генерації об'єкту, тож ми викликаємо функцію GetWorldTransform, яка замінює об'єкт у ігровому світі на інший. Як агрумент у цю функцію передаємо об'єкт Lane0, тобто наступний об'єкт буде з'являтися на цьому місці. Після чого підключаємо функцію до поля SpawnTransform.

Для постійного виклику функції у Blueprint нашої «плитки» переходимо на вкладку EventGraph, яка відповідає за події та подію Event BeginPlay підключаємо до функції Spawn Obstacles.

Рис. 3.10 Виклик функції створення перешкод

Дана подія буде відбуватися щоразу як елемент ігрового поля буде з'являтися у ігровому світі. Тобто під час появи «плитки» автоматично буде викликана функція яка буде на лінії Lane0 генерувати перешкоду – водопровідну трубу.

Для появи інших перешкод та об’єктів а також забезпечення випадкової генерації перешкод, а не постійної буде дописана функція SpawnObstacles. На даний момент вона виконує основну функції – генерує перешкоду зліва від ігрового персонажу.

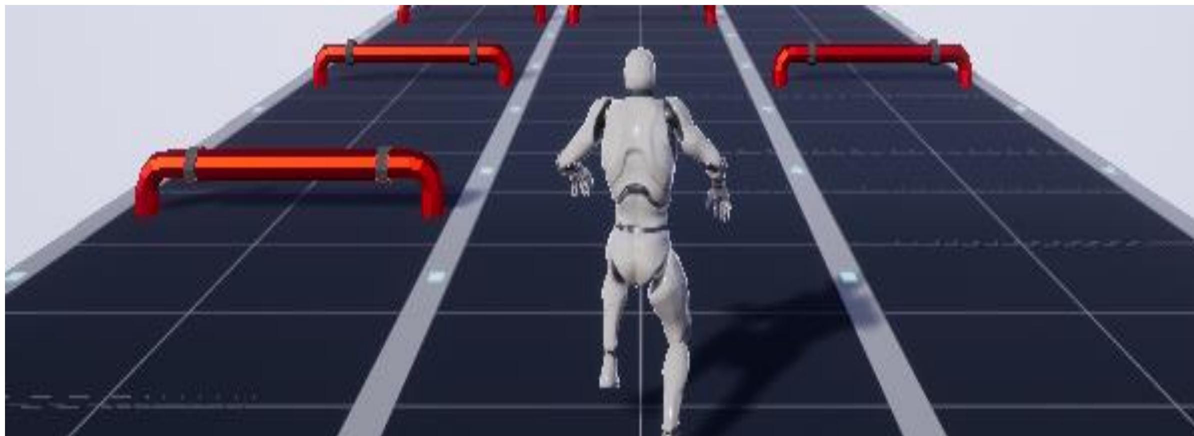


Рис. 3.11 Ігрова перешкода на ігровому полі

Створення іншої перешкоди – високого об’єкту через який наш ігровий персонаж не у змозі перестрибнути або перейти іншим способом також відбувається схожим чином.

Для початку у 3-д редакторі Blender створюється об’єкт прямокутного типу, який у даному випадку виглядає як звичайна коробка. Для даного об’єкта необхідно розробити текстуру у редакторі Adobe Photoshop, для коректного відображення об’єкту у ігровому світі, а також для нанесення певних зображень на сторони даного об’єкту.

Тож створюємо текстуру Crate_Diffuse, яка є прикладом розгортки даного об’єкту і має «облягати» його зі всіх сторін. Далі створюємо матеріал куди ми завантажуємо дану текстуру використовуючи TextureSample та підключаємо її до перемикача BaseMaterial. Для надання більш металевого кольору задаємо перемикачу Metallic константу – 1. Це надасть характерного металічного блиску та загалом гарно відобразиться на стилістиці ігрового світу.

Після налаштування всіх графічних моментів, які стосується даного об’єкту, ми переходимо у папку з наявними файлами Blueprints та створюємо новий файл з назвою ObstacleBox.

Ідентично до прикладу вище ми створюємо Static Mesh з об'єктом ObstacleBox. Виконуємо функції масштабування та розвертаємо його для коректного відображення у ігровому світі. Встановлюємо параметр Collision Preset на BlockAll для попередження проходження ігрового персонажу крізь об'єкт.

На даному етапі ми створили два типи перешкод, які повинні відображатися у нашому ігровому світі, отож потрібно перевірити коректність їх відображення.

Переходимо у Blueprint MasterTile та заходимо у вкладку з функцією SpawnObstacles. Для генерації різних об'єктів, а також для їх випадкового виникнення на різних лініях використаємо функцію Random Int in Range. Дана функція буде повертати випадкове ціле значення на відрізку який буде заданий у її тілі. Мінімальне значення встановлюємо 0, максимальне 2, оскільки ігрове поле на даний момент містить 2 перешкоди.

Через класс SpawnActor з заданим значенням ObstacleBox буде відбуватися генерація новоствореної перешкоди. Класи SpawnActor які відповідають за створення водовпровідних труб та об'єктів-коробок повинні бути паралельно підключені до функції Random Int in Range. Отож, коли число буде дорівнювати 0 не буде створюватись нічого, коли 1 – буде створюватись труба, а коли 2 – об'єкт коробка.

Для генерації на всіх трьох умовних лініях руху використаємо 3 таких блоки, з різними аргументами Get World Transform. Для лівої частини це буде Lane0, центральної – Lane1, крайньої правої – Lane2. Тепер генерація буде випадковою і на ігровому полі буде відображатися таким чином:

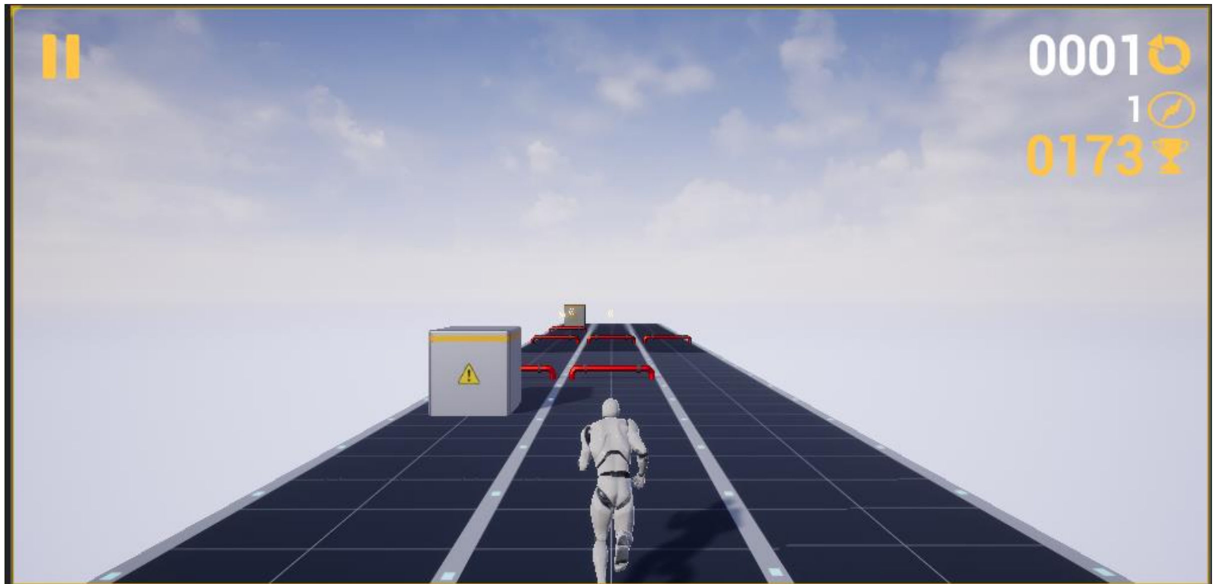


Рис 3.12 Генерація основних перешкод

Генерація як і раніше здійснюється викликом події Event BeginPlay з подальшим викликом функції Spawn Obstacles.

Наразі дані об'єкти – перешкоди виконують функцію лише зупинки гравця, але ніяк не запускають анімацію смерті чи зупинки гри. Це пов'язано з тим, що у нас ще немає функції, яка при виклику би «вбивала» ігрового персонажу.

Для створення даної функції перейдемо у Blueprint ThirdPersonCharacter. Оскільки дана функція має впливати саме на ігрового персонажа, нам необхідно описати її саме у тілі даного файлу.

Створюємо нову функцію DeathFunction, її функціонал дуже простий: при виклику даної функції, персонаж має відображати анімацію смерті, а процес гри – зупинятися.

Викликаємо функцію SetSimulatePhysics, яка у ігровому рушії відповідає за анімації акторів, та передаємо у неї атрибут Mesh, таким чином дана функція буде застосовуватись до скелету нашого ігрового персонажу, далі заходимо у тіло функції та ставимо галочку навпроти SetRagDoll.

При виклику цієї функції наш персонаж буде мати поведніку «ганчіркової ляльки», яка буде падати вниз згідно з гравітацією ігрового світу, а також викликаємо функцію DisableMovement з аргументом CharacterMovement, що зупинить анімацію руху ігрового персонажу.

Повертаємося у Blueprints Obstacle Box та ObstaclePipe відповідно, у вкладці Viewport натискаємо на необхідні нам об’єкти (водопровідну трубу чи коробку) та встановлюємо на вкладці Details Collision Preset на Overlap All.

Це налаштування забезпечить нам дію нашого тригера саме у той час, як ігровий персонаж доторкнеться до тіла об’єкту. Якщо не змінити даний параметр то тригер буде задіяний лише при проходженні скрізь даний об’єкт.

Переходимо у вкладку Event Graph та з події Event ActorBeginOverlap викликаємо функцію застосування Cast to ThirdPersonCharacter, а після цього викликаємо функцію DeathFunction.

Тепер дотик до перешкод буде викликати анімацію смерті «ігрового персонажу» та зупиняти гру.

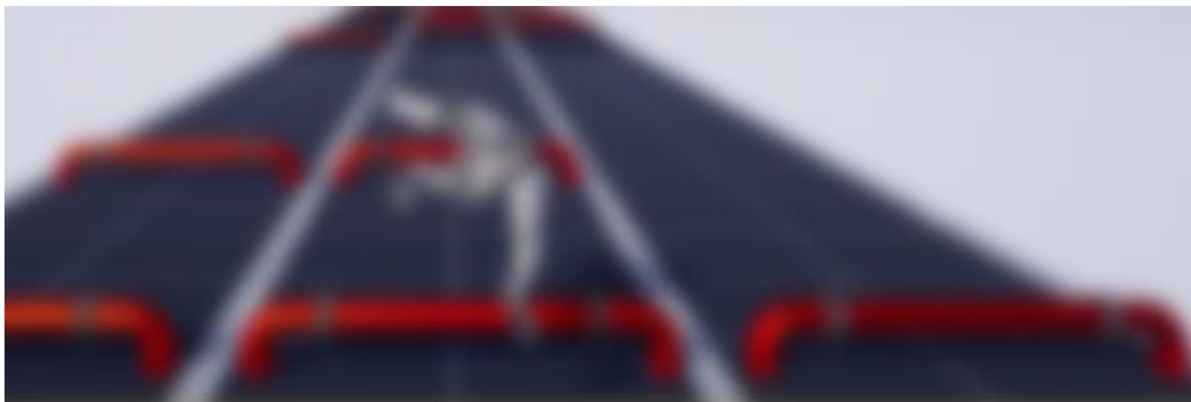


Рис. 3.13 Анімація смерті

3.2.5 Створення основної ігрової валюти

Одним з основних показників успішності гравця є кількість підібраних монет – об'єктів, які генеруються у ігровому світі, але не несуть ніякої шкоди для гравця, а лише збільшують його рахунок монет.

Як і перешкоди такі об'єкти також складаються з матеріалу та 3-д моделі, для створення 3-д моделі використаємо редактор Blender, у якому задаємо зовнішній вигляд об'єкту.

Монета буде мати вигляд кола у якому зображена блискавка, а матеріал для неї буде містити такі налаштування: базовий колір ми встановлюємо у положення: жовтий, а для надання візуального ефекту світла використаємо ще один жовтий колір, який буде помножений на константу 15, та включений у перемикач EmissiveColor.

Рис 3.14 Зовнішній вигляд монети

Для надання монеті функціоналу використаємо файл Blueprint. Як і вище генеруємо файл Coin_Pickup, який буде містити весь необхідний функціонал для підбирання монет.

У вкладці Viewport розміщуємо наш об'єкт CoinPickup та піднімаємо його для надання йому ефекту левітації у повітрі. Для того, щоб наш об'єкт мав привабливий вигляд використаємо анімацію RotatingMovement. Дана анімація заздалегіть передбачена у ігровому рушію Unreal Engine 4 та надає ефекту повільного обертання.

Для того щоб персонаж міг вільно проходити скрізь об'єкт використаємо налаштування OverLapOnlyPawn.

Функціонал монети реалізується на вкладці Event Graph:

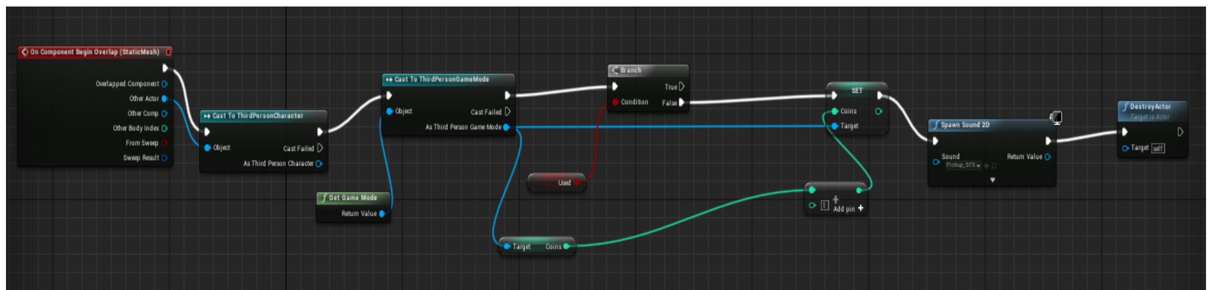


Рис. 3.15 Робота функціоналу монети

Як можемо бачити з даного скріншоту при виклику події Begin Overlap, тобто коли гравець перетинає даний об'єкт викликається функція Cast to ThirdPersonCharacter. Далі викликається функція Cast to ThirdPersonGamemode з аргументом Get GameMode. Це забезпечує доступ до змінної Coins у тілі даного класу.

Йде перевірка умови на використання монети і якщо монета була підібрана змінна Coins збільшується на 1 за допомогою функції Set. Блок Spawn Sound2d програє заздалегіть записаний звук під час підбирання монети, а функція DestroyActor прибирає монету у той час, коли ігровий персонаж її підібрав.

Для створення об'єкта монети у ігровому полі ми використаємо функцію Spawn Objects. Додаємо ще один блок SpawnActor з встановленим значенням Coin_Pickup на кожну з наших умовних ліній, а також до блоку функції

Random Int in Range додаємо число 3, яке і буде відповідати за генерацію нашої монети.

Переходимо до ігрового світу та перевіряємо функціонал:

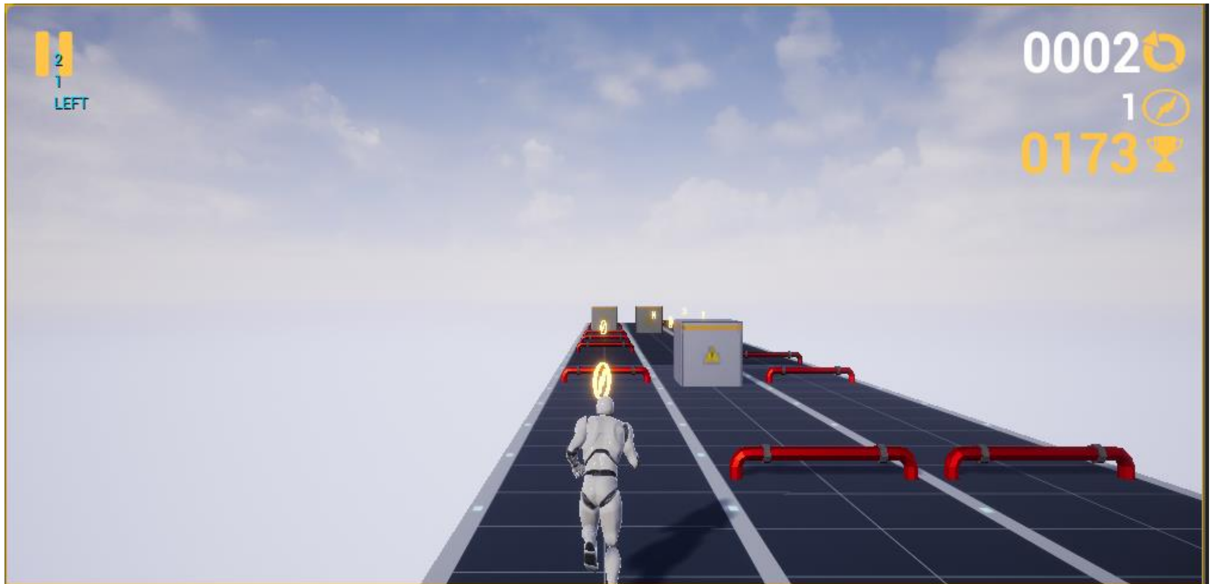


Рис. 3.16 Відображення монет

Як можливо помітити функціонал є повністю робочим та задовольняє всі умови, які були задані на процесі проектування гри. Тобто ігровий персонаж має можливість взаємодіяти з монетами, вони з’являються у ігровому світі та коректно виконують усі необхідні вимоги.

3.2.6 Об'єкти взаємодії

Одним з типів предметів з якими ігровий персонаж буде мати можливість взаємодіяти є предмети-бонуси. Дані ігрові об'єкти надають персонажу певні тимчасові переваги, які полегшують ігровий процес.

Використання даних об'єктів є типовим для ігор даного жанру оскільки робить ігровий процес більш різноманітним та складним. У даному проекті будуть використані 2 типи предметів-бонусів: магніт, який буде притягувати монети до ігрового персонажу та чоботи, які будуть дозволяти ігровому персонажу стрибати набагато вище. Приклад таких бонусів продемонстрований на рисунку:

Рис 3.17 Приклад предмету-бонусу

Зазначимо, що вказані предмети є рідкістними, та повинні появлятися лише у окремих випадках для уникнення порушень балансу ігрового процесу. Вони також повинні бути яскравими, для того щоб користувач міг заздалегіть їх побачити та використати.

Першим з двох предметів-бонусів буде розроблений магніт, оскільки його функціонал є більш складним та потребує взаємодії не тільки з ігровим персонажем, але і з іншим ігровим об'єктом – монетами. Механізм роботи повинен уникати помилок, наприклад помилки з подвійним зарахуванням монети гравцю, а також з правильним і вчасним вимкненням функціоналу даного об'єкта.

Як і вище, для створення нового об’єкту у ігровому світі спочатку необхідно створити 3-д модель у редакторі Blender. У даному випадку модель буде складатися з двох елементів (що буде забезпечувати зовнішній вигляд магніту) та матеріалу, який буде розроблений у середовищі Unreal Engine 4.

Матеріал буде складатися з двох елементів: чистого червоного кольору з кольорової схеми RGB, та Emissive Color, який буде встановлено на червоний колір з інтенсивністю – 15.

Рис 3.18 Зовнішній вигляд магніту

Для створення функціоналу магніту, нам необхідно постійно перевіряти умову його активності, тож, оскільки даний бонус буде використаний до ігрового персонажу, нам необхідно створити змінну MagnetActive булевого типу у Blueprint ThirdPersonCharacter. За замовчуванням він повинен набувати значення – 0, тобто бути вимкненим при запуску гри.

Далі переходимо у файл Coin_Pickup, який відповідає за отримання монет ігровим персонажем, переходимо у вкладку Viewport та створюємо об'єкт взаємодії типу сфера (Sphere Collision). Дана сфера повинна бути дуже великих розмірів, щоб охоплювати всю площу ігрового поля, а також функціонувати навіть при ввімкненому бонусі чобіт.

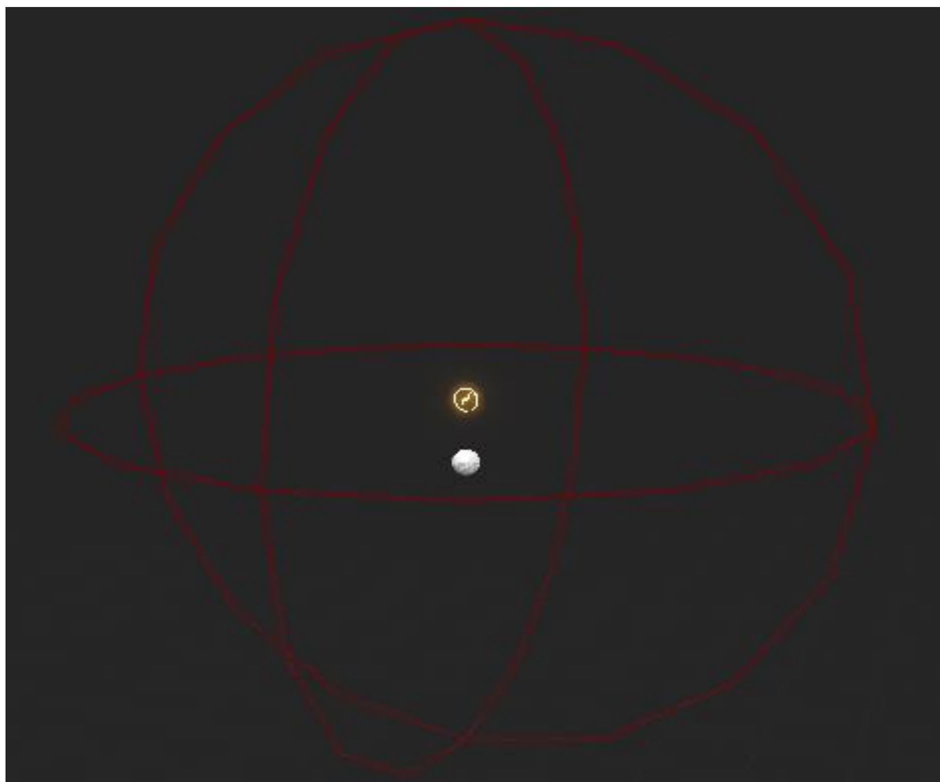


Рис 3.19 Сфера-колізія

Встановлюємо параметр Collision Presets на OverlapAllDynamic, що дозволить нам взаємодіяти з динамічними об'єктами ігрового світу та проходити наскрізь ігрове поле та перешкоди.

Створюємо подію OnBeginOverlap, тобто при контакті зі сферою у нас будуть відбуватися певні події, а саме: монета повинна притягуватись до гравця та програватися анімація отримання монети персонажем, при цьому кількість монет має збільшуватись на 1, як і при контакті зі звичайною монетою.

Отож під'єднуємо подію до виклику функції Cast to ThirdPersonCharacter, для того, щоб отримати доступ до змінної MagnetActive. Перевіряємо чи є активним наш магніт за допомогою умовного оператору, та за умови його активності виконуємо дві дії, за допомогою послідовності (оператор

sequence). Перша дія викликає функцію Cast To ThirdPersonGameMode, та змінює значення змінної Coins додаючи один. Після чого значення змінної Used змінюється на один.

Інша гілка застосовує такий атрибут як Timeline – подію, яка відбувається протягом певного часу, у даному випадку – 2 секунд. Використовуючи функції GetActorTransform та GetPlayerLocation для того щоб помістити даний об'єкт (монету) над головою ігрового персонажу, а також функцію GetWorldDeltaSeconds, яка правильним чином буде активувати анімацію за рахунок відслідковування ігрового часу і Vinterp To для плавного переміщення об'єкту ми встановлюємо положення методом SetActorLocation. Через 2 секунди після переміщення об'єкту він буде знищений методом ActorDestroy.

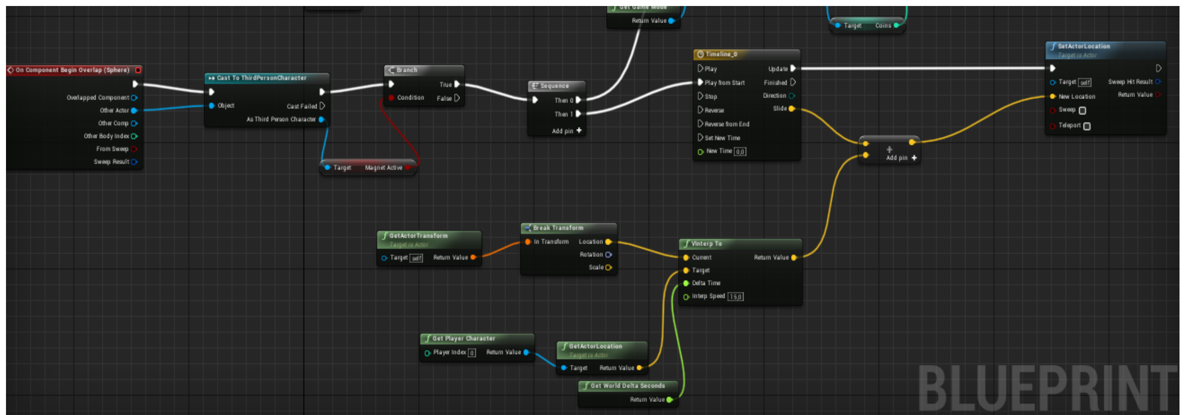


Рис 3.20 Blueprint Coin_pickup

Для запуску даного функціоналу створимо новий BluePrint Magnet_Pickup у якому розташуємо наш магніт та додамо обертальний рух за допомогою RotatingMovement. Виставимо Collision Presets на OverlapOnlyPawn для проходження крізь ігрового персонажа. Для події ActorBeginOverlap розміщуємо Cast To ThirdPersonCharacter та змінюємо значення булевої функції Magnet Active на 1, після чого робимо тіло нашого об'єкта невидимим за допомогою Set Visibility з аргументом Static Mesh (тіло об'єкта).

Встановлюємо таймер затримки на 10 секунд, та знову з викликом функції Cast To ThirdPersonCharacter змінюємо значення булевої функції Magnet Active на 0, вимикаючи магніт.

Таким чином під час отримання даного бонусу, ігровий персонаж починає «притягувати» до себе всі монети, які входять у сферу, активуючи анімацію їх отримання. Після десятисекундного інтервалу, бонус автоматично вимикається та ігровий процес продовжується у звичайному порядку:

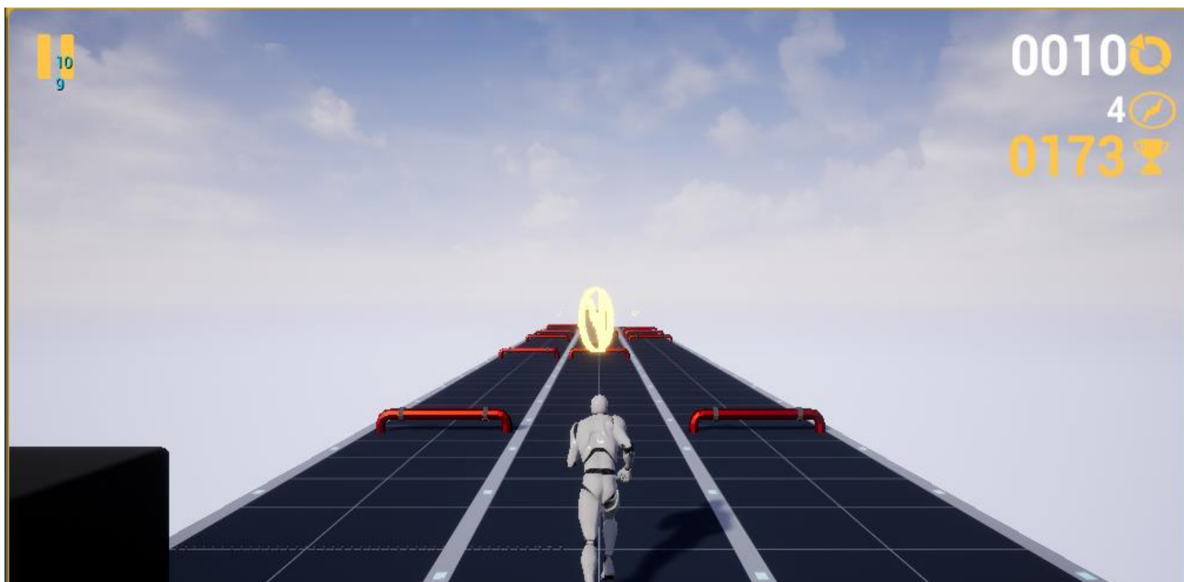


Рис. 3.21 Робота бонусу-магніта

Для генерації даного елемента у ігровому світі заходимо у файл MasterTile, де ми додаємо ще одну гілку з фукнцією SpawnActor та вказуємо даний об'єкт. Розміщуємо його на всіх трьох можливих лініях за допомогою стрілок Lane0, Lane1, Lane2. Зазначимо, що бонуси мають з'являтися на ігровому полі набагато рідше від перешкод та монет, у подальшому буде розроблений випадковий алгоритм виникнення ігрових об'єктів з можливістю налаштування певних шансів.

Як можливо побачити з наведеного вище рисунку магніт виконує всі необхідні функції – з'являється на ігровому полі, активує бонус, виконує необхідні анімації, зникає після 10 секунд та корректно обраховує кількість монет, які отримує ігровий персонаж.

Іншим ігровим предметом бонусом, як було зазначено вище, будуть чоботи, при взаємодії з якими ігровий персонаж буде стрибати у 2 рази вище.

Даний предмет повинен з'являтися на ігровому полі не занадто часто, але достатньо для того, щоб ігровий процес видавався різноманітним та захопливим.

Як і інші об'єкти ігрового світу об'єкт-бонус чоботи має складатися з «скелету» (Static Mesh) та матеріалу. У даному випадку створюємо макет нашого об'єкту у 3-д редакторі Blender та створюємо необхідний матеріал: у поле Emissive Color додаємо жовтий колір, для досягнення яскравого ефекту. Отриманий таким чином об'єкт буде мати вигляд:

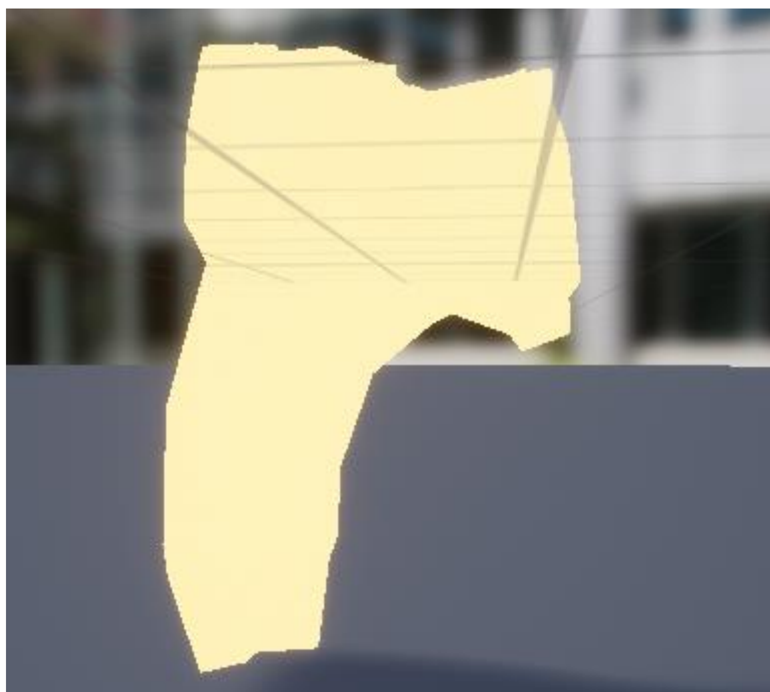


Рис 3.21 Макет об'єкту чобота

Тепер дана модель повністю готова до використання у нашому проекті. Для розробки функціоналу створюємо новий Blueprint Boots_Pickup, який буде містити логіку роботи нашого бонусу.

У вкладці Viewport додаємо наш об'єкт та розташовуємо його правильним чином: по-перше, для того щоб здавалося враження, що об'єкт знаходиться у повітрі розташовуємо його над нашою стартовою площиною та розвертаємо у просторі.

Налаштовуємо фізику поведінки нашого об'єкту, а саме Collision Presets у положення OverlapOnlyPawn. Це дозволить нашому персонажу проходити крізь об'єкт та застосовувати його.

Для активації функціоналу створюємо подію ActorBeginOverlap, яка забезпечить виконання подальшого коду при зіткненні з об’єктом. Викликаємо функцію Cast To ThirdPersonCharacter, для того щоб мати доступ до класу CharacterMovement. Даний клас описує всі можливі правила руху нашого ігрового персонажу. Викликаємо функцію Set для змінної JumpZVelocity (висота стрибку по лінії Z, тобто вертикалі) та задаємо їй значення 1200, після чого за допомогою функції SetVisibility робимо чоботи невидимими.

Встановлюємо затримку на 10 секунд оператором Delay, після чого знову викликаємо Cast To ThirdPersonCharacter, щоб уникнути помилок і аналогічно повертаємо значення JumpZVelocity на 600. Після чого функцією DestroyActor прибираємо предмет з ігрового світу.

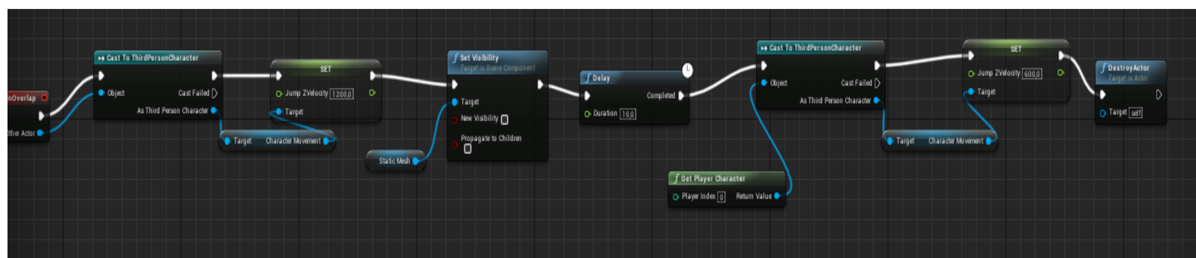


Рис 3.22 Логіка роботи Boots_Pickup

Даний код при виконанні забезпечить чітку та безвідмовну роботу нашого об’єкта у ігровому світі, а також зробить процес гри більш цікавим та «затягуючим».

Для генерації об’єкту бонусу, як і раніше потрібно перейти у Blueprint Master Tile, який відповідає за створення перешкод, монет та інших об’єктів. Додаємо до вже існуючого алгоритму генерації код: у перемикачі створюємо 4 варіант – згенерувати бонус-чоботи. І за допомогою SpawnActor з класом Boots Pickup та встановленим GetWorldTransform приводимо механізм створення об’єкта у дію.

Для забезпечення рівномірного розподілу по всім трьом уявним лініям Lane0, Lane1 та Lane2 відповідно до кожного блоку нашого алгоритму додаємо SpawnActor з класом Boots Pickup. Компілюємо та від цього часу даний об’єкт буде з’являтися на кожній лінії.

У ігровому світі предмет буде працювати таким чином:

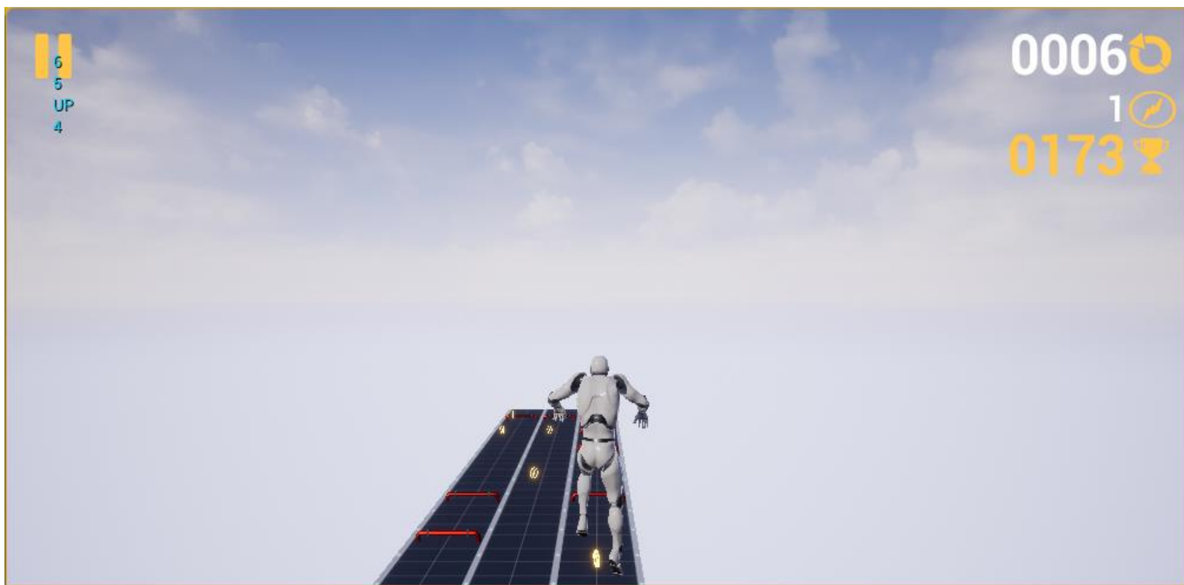


Рис 3.23 Механізм роботи об’єкта магніту

Як можемо бачити вище, бонус дійсно дає можливість нашому ігровому персонажу стрибати на 600 пікселів вище.

3.2.7 Графічний інтерфейс користувача

У розробці будь-якої гри основним елементом є графічний інтерфейс користувача, який забезпечує взаємодію користувача додатку з функціями гри та виконує інші управлінські функції.

При розробці мобільного додатку-гри інтерфейс користувача є особливо важливим, оскільки сама з цієї складової додатка відбувається «знайомство» користувача з будь-якою розробкою.

У даному випадку буде реалізовано три елементи графічного інтерфейсу: ігровий інтерфейс, який буде відображати основні елементи: рахунок, кількість зібраних монет та показник рекорду, а також надавати можливість поставити гру у режим паузи, меню паузи та головне меню.

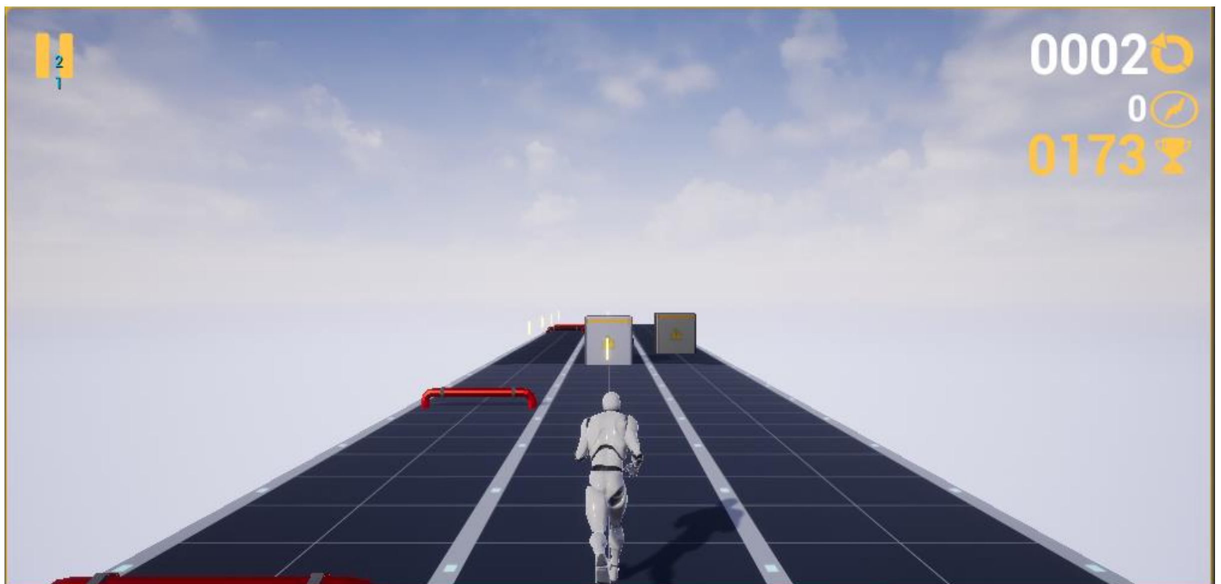


Рис 3.24 Ігровий інтерфейс користувача

Для створення даного інтерфейсу ми використаємо вбудовану можливість ігрового рушія Unreal Engine 4 під назвою Widget Blueprint. Даний файл містить всі основні елементи та віджети, які розробник може помістити у розроблену гру.

Як можливо бачити з прикладу вище ігровий інтерфейс містить 3 поля з верхнього правого боку: рахунок, кількість монет та найвищий рахунок, а з правого верхнього боку – кнопку паузи, яка дозволить зупинити ігровий процес.

Для розробки даного інтерфейсу створюємо файл RunnerHud. На порожньому полі починаємо розташовувати наші елементи. Основний ігровий інтерфейс буде також доповнений піктограмами, які забезпечать правильний зовнішній вигляд. Дані піктограми – об'єкти типу Image, які заздалегіть були розроблені у Adobe Photoshop, розташовуємо їх зверху справа та закріплюємо за допомогою Anchor до правого верхнього боку. Починаємо створювати текстові поля: поле для ігрового рахунку, поле для кількості монет та поле для відображення рекорду.

У верхній лівій частині створюємо кнопку на яку ми накладаємо заздалегіть спроектоване зображення Pause для нормального вигляду, а для положення Hovered (тобто курсор розташований над кнопкою) – додаємо зображення Pause_Hovered, вирівнюємо всі частини зображення, масштабуємо та отримаємо такий зовнішній вигляд:

Рис 3.25 Макет ігрового інтерфейсу користувача

Зазначимо, що нам необхідно встановити функціонал для наших полів та кнопки, тож вибираємо перше текстове поле: поле ігрового результату та у вкладці Content натискаємо на Create Binding. Автоматично створюється функція Get Text 0.

Дана функція відповідає за відображення ігрового рахунку, тож необхідно отримати доступ до нього. Викликаємо Cast To ThirdPersonGameMode, який містить змінну CurrentPoints та підключаємо до Return Node, який є аргументом, що поверне функцію. Автоматично створиться блок що перетворює цілочисленне значення на текстове.

Тепер при запуску додатку та початку гри, даний елемент буде відображати ігровий рахунок у реальному часі.

Наступним елементом, який буде відображати кількість ігрових монет є текстовий блок, розташований нижче від ігрового рахунку. З правої частини даного поля додаємо малюнок-піктограму з зображенням монети. Це надасть інтерфейсу зручного та графічно привабливого забарвлення.

Для реалізації функціоналу текстового поля у вкладці Content натискаємо на Create Binding. Автоматично створюється функція: Get Text 1. Тіло функції буде мати наступний вигляд:

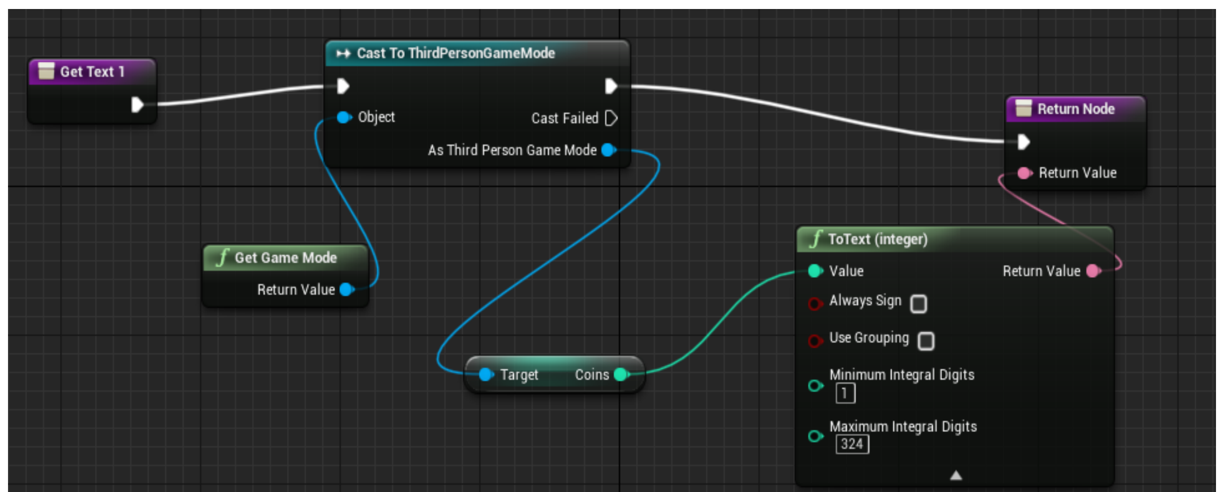


Рис 3.26 Тіло функції Get Text

Як можемо бачити, при виклику даної функції, вона отримує значення змінної Coins, яка знаходиться у Blueprint ThirdPersonGameMode та записує дане значення у наше поле, при цьому зміна кількості відбувається повністю автоматично.

Наступним елементом нашого графічного інтерфейсу користувача буде текстове поле, яке буде містити найбільший ігровий рахунок, тобто рекорд. Зазначимо, що рекорд має залишатися незмінним після кожного

перезавантаження рівня, або гри, тож ми не можемо напряму записувати значення у текстове поле з будь-якого нашого класу.

Для реалізації механізму збереження гри нам необхідно створити файл збереження гри. Це реалізується за допомогою Blueprint з єдиною функцією SaveGame. У ньому ми створюємо змінну HighScoreValue і саме ця змінна буде відповідати за наш найбільший рахунок.

Оскільки рекордний ігровий рахунок повинен фіксуватися на момент «смерті» ігрового персонажу ми переходимо у ThirdPersonCharacter у функцію Death Function, яка відповідає за цей процес.

Після того, як відбувається анімації смерті ми викликаємо ThirdPersonGameMode та перевіряємо чи існує наш слот збереження даних за допомогою умовного оператору Branch. У разі, якщо дані збережено і слот існує ми порівнюємо два значення: HighScore Value та Current Points і записуємо у слот збереження більше значення.

Таким чином дана функція буде фіксувати остаточне значення поточного ігрового рахунку та записувати його у змінну, якщо воно виявиться більшим.

У іншому випадку, якщо відсутній файл з даними, створюється новий файл зі збереженими даними у Endless Save Game. Знову порівнюються значення HighScore Value та Current Points і за допомогою функції Save Game To Slot відбувається збереження значення змінної.

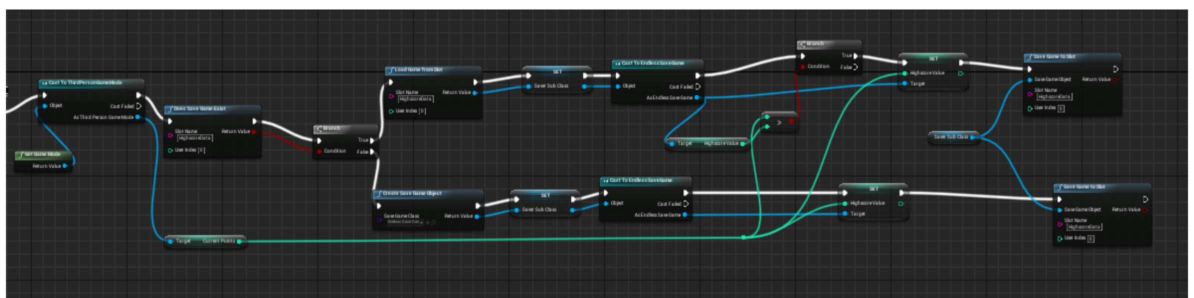


Рис 3.27 Тіло функції DeathFunction

Повертаємося до нашого інтерфейсу користувача та у вкладці Content натискаємо на Create Binding. Автоматично створюється функція: Get Text 2. У ній ми використаємо функцію Load Game From Slot та передамо значення

Highscore Data. При виклику даної функції файл збереження буде виводити на екран гравця найбільше значення, тобто ігровий рекорд.

Останнім елементом даного інтерфейсу є кнопка паузи гри. При натисканні на яку ігровий процес зупиняється та запускається інший інтерфейс.

Для створення даного елементу використовуємо блок Button та як було зазначено вище два спрайти: Pause та Pause_Hovered, які надають даному елементу анімації.

Для створення функціоналу переходимо у меню подій Events та створюємо подію OnClicked. У ігровому рушії Unreal Engine 4 є функція, яка відповідає за зупинку ігрового процесу з назвою: Set Game Paused, після чого викликаємо інший віджет, який буде описано нижче, під назвою Paused та додаємо його до екрану гравця використовуючи Add To Viewport.

Фунціонал кнопки зображений на малюнку:

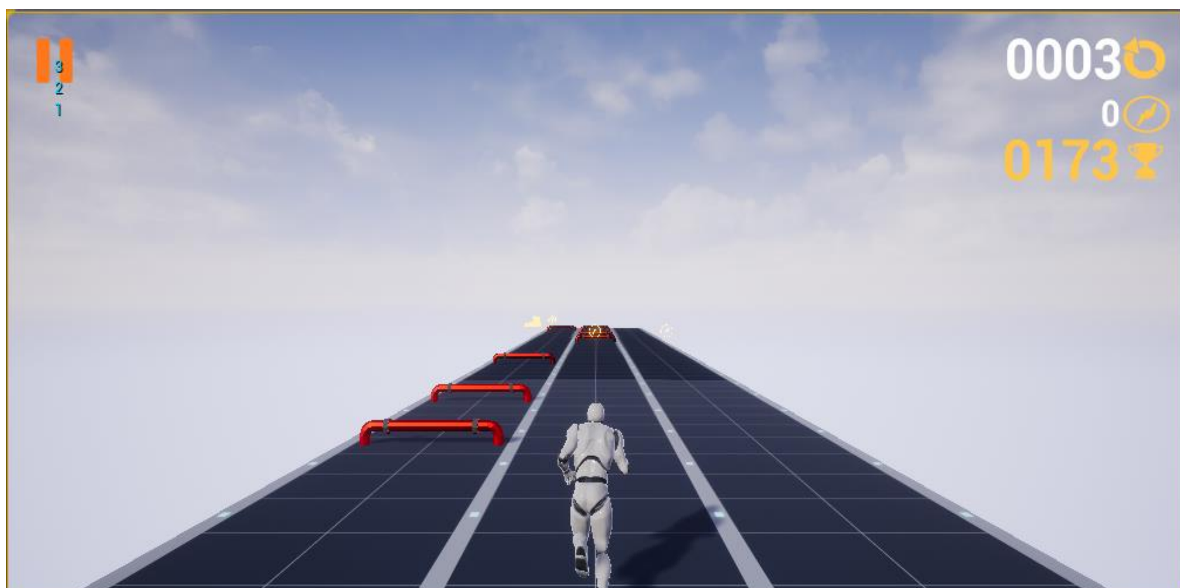


Рис 3.28 Функціонал кнопки паузи

Після натискання на кнопку ігровий процес зупиняється (на даний момент назавжди, оскільки не існує функціоналу продовження гри). Даний функціонал буде доданий до іншого графічного інтерфейсу, який, як було зазначено вище буде викликатися після натискання кнопки Пауза. Для його створення використовуємо Widget Blueprint з назвою Paused.

Даний елемент інтерфейсу не є занадто складним і повинен мати 2 функції: повернення до головного меню, та продовження процесу гри. Також він буде містити текстове поле з написом: GAME PAUSED, що є типовим для ігр даного жанру.

Після виклику функції також буде відбуватися штучне «замилення» (blur ефект) екрану гравця для створення ефекту паузи. Даний інтерфейс буде мати приблизно такий вигляд:

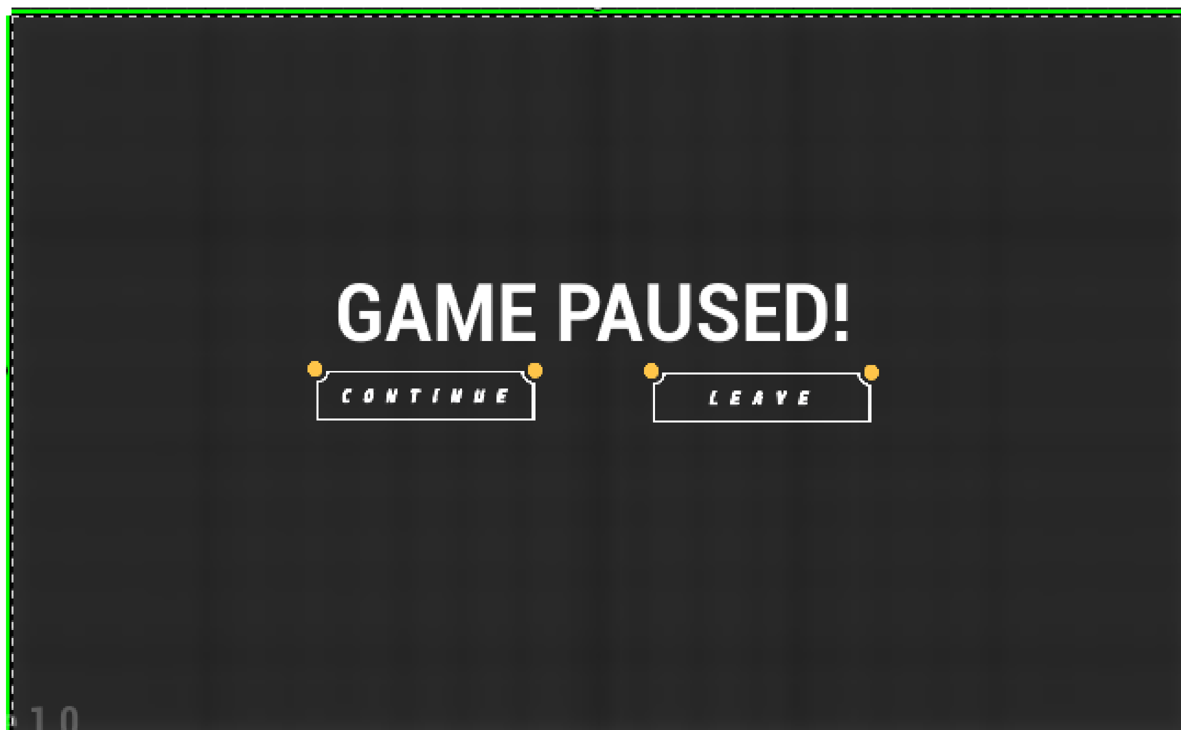


Рис 3.29 Екран паузи

Зупинимося на кожному елементі даного інтерфейсу користувача окремо. Перший елемент – текстове поле з надписом про зупинку ігрового процесу реалізується за допомогою блоку Text. Далі встановлюємо розмір шрифту та розміщуємо блок посередині, наносимо необхідний напис за закріплюємо по центру за допомогою перемикача Anchors. Це необхідно для того, щоб при масштабуванні надпис завжди залишався по центру.

Два елементи типу Button які виконують функції відновлення ігрового процесу та виходу до головного меню гри також розташовуємо по центру з ввімкнутим Anchors «по центру».

Кнопка Continue має запускати ігровий процес ті прибирати даний віджет з поля зору гравця. Для її створення використаємо графічний редактор Adobe Photoshop. Зображення для даного елементу нам необхідно у трьох варіантах: звичайне положення кнопки : Continue_button, положення коли над кнопкою висить курсор Continue hovered та положення після натискання. Додаємо необхідні елементи та створюємо подію On Clicked.

Після натискання на кнопку викликається функція Set Game Paused з вимкненим перемикачем Paused, а далі за допомогою функції Remove From Parent віджет видаляється з поля зору користувача.

Функціонал іншої кнопки створюється аналогічним чином: є декілька можливих положень LeaveButton та LeaveHovered, це забезпечує гарний графічний інтерфейс користувача. За допомогою події On Clicked ми викликаємо функцію Open Level з іменем LevelMain. Даний рівень буде створений нижче та виконує функції головного меню.

Останнім елементом є анімація «замилення» заднього фону, для цього встановлюємо на інших елементах ZOrder в положення 1. Так наші елементи будуть залишатися на поверхні. Додаємо анімацію, та у редакторі створюємо дві точки з інтервалом секунда, ці точки повинні бути підв'язані до змінної BlurStrength, та мати значення 0 та 15 відповідно. Після чого у події Event Construct додаємо функкцію Play Animation. Отже, за секунду після виклику нашого віджету задній фон буде «замилено».

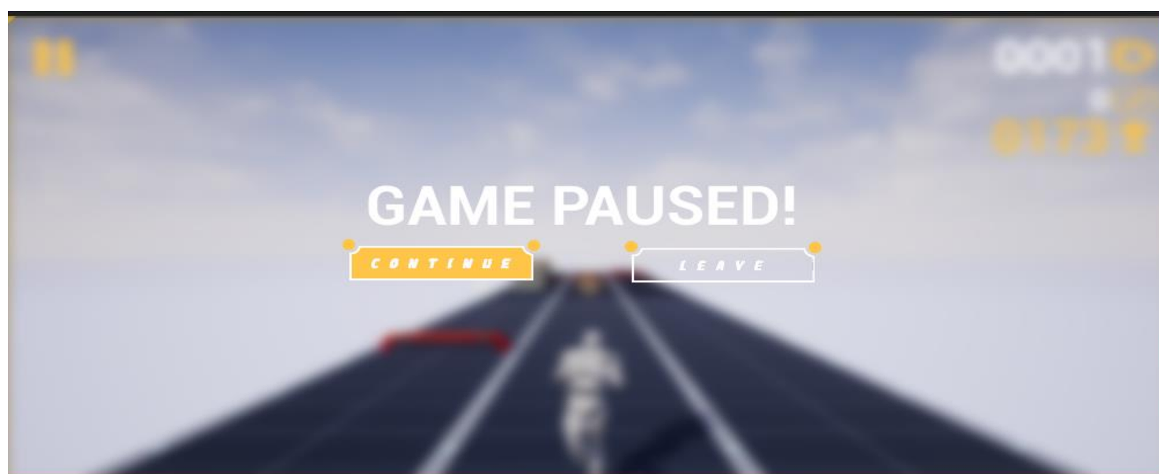


Рис 3.30 Ігровий інтерфейс паузи

Як бачимо всі елементи ігрового інтерфейсу користувача працюють коректно та безвідмовно, а також забезпечують додаток важливими механізмами роботи.

Наступним елементом ігрового інтерфейсу користувача буде, як було зазначено вище екран, який випливає після смерті ігрового персонажа та дозволяє повернутися у головне меню, або запустити ігровий процес з початку.

Даний елемент інтерфейсу буде мати назву Game Over Screen. Для його створення використаємо вже відомий нам Widget Blueprint з однойменною назвою.

Макет даного віджету зображений на малюнку:

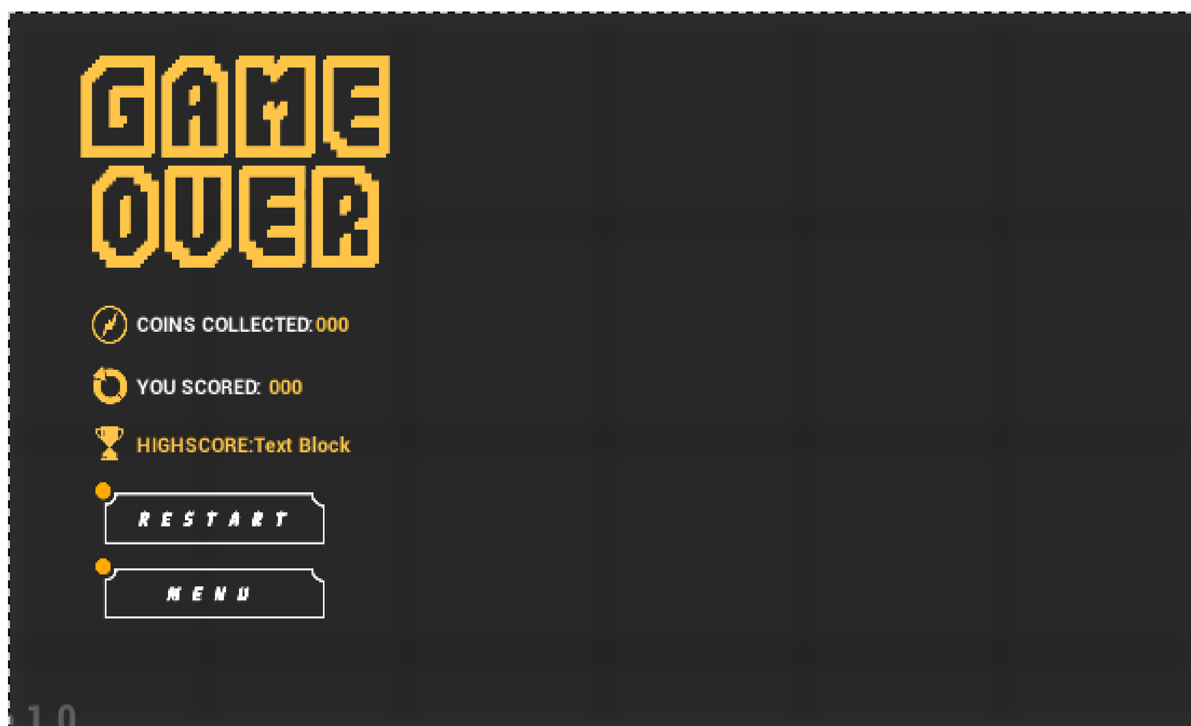


Рис 3.31 Екран закінчення гри

Як можемо бачити даний інтерфейс складається з таких елементів: зображення з надписом Game Over, піктограми, які були раніше використані у ігровому інтерфейсі та надписи з оновлюваними даними, а також двох кнопок, які виконують функціонал описаний вище. Також даний інтерфейс буде містити анімацію «замилення» зображення.

Почнемо створення нового інтерфейсу з розміщення надпису «Гра закінчена», робиться це за допомогою вже відомого інструменту Image. У

даний інструмент додається заздалегіть розроблене зображення GameOver. Дане зображення масштабується та переміщується на своє місце. Для закріплення зображення задамо значення Anchors на верхній лівий кут. Це дозволить даному елементу бути закріпленим не зважаючи на розширення на інших девайсах.

Наступним елементом буде комбінація з двох текстових строк та піктограми-зображення, яка буде закріплена поруч. Створюємо дані елементи відповідно за допомогою Image та Text Block. У блок зображення додаємо Coin. Перший текстовий блок заповнюємо відповідним надписом. Третій же елемент повинен відображати кількість монет, тому створюємо функцію Get Text 0, яка буде зчитувати кількість монет.

Викликаємо Cast To ThirdPersonGameMode та отримуємо значення змінної Coins, після чого переводимо її з числових даних у строкові та повертаємо. Відтепер дане поле буде відображати число монет, які персонаж підібрав у процесі гри.

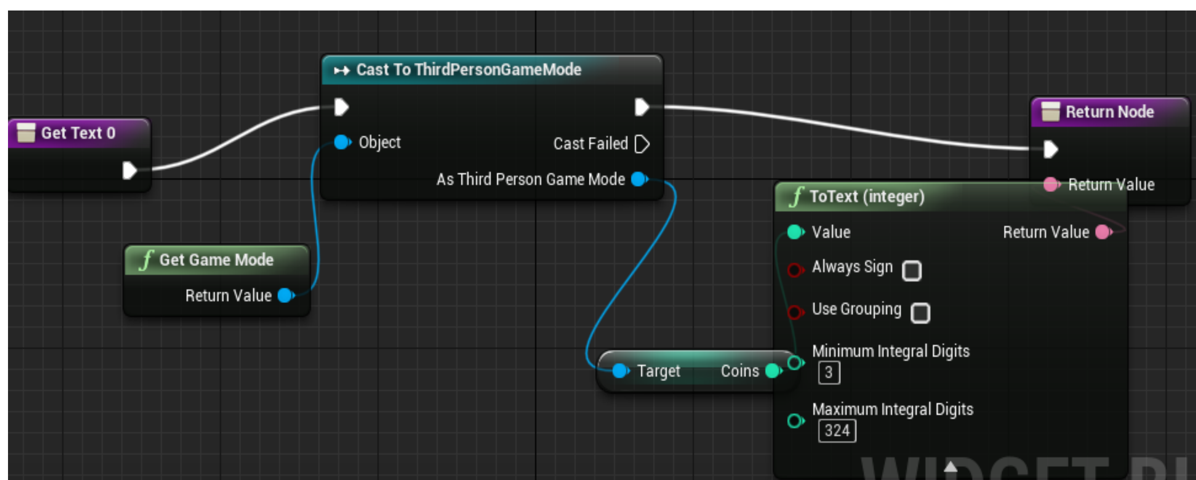


Рис 3.31 Функціонал текстових блоків

Другий елемент даного інтерфейсу – готовий. Переходимо до наступної комбінації полів а саме ігрового рахунку, який гравцю вдалося досягнути. Розробка даного елементу проходить аналогічним шляхом: створення блоку Image з елементом ScoreIcon, статичного текстового поля та поля з функцією Get Text 1. Єдиною відмінністю буде зчитування змінної Current Points з

Blueprint Third Person Game Mode. Після чого всі елементи ми кріпимо за допомогою налаштування Anchors до лівого боку екрану.

Наступний елемент: найбільший рахунок, тобто рекорд розроблюється аналогічним чином, але з деякими важливими змінними. По перше: застосовується зображення Highscore Icon. По-друге до текстових полів застосовується жовтий колір, який ми отримуємо за допомогою інструмента «Піпетка».

Функція для текстового блоку, який буде містити найвищий рахунок створюється наступним чином: ми завантажуємо файл збереження Highscore Data за допомогою функції Load Game From Slot та з файлу EndlessSaveGame отримуємо значення рекорду методом Cast To. Це забезпечить виведення саме найбільшого ігрового рахунку, не зважаючи на кількість перезавантажень гри.

Наступними складовими інтерфесу «екрану смерті» я кнопки, які створюються елементами Button. До першої кнопки, яка виконує функцію перезапуску гри ми додаємо наступні елементи: зображення, яке відповідає за звичайний вигляд елементу RestartButton, для положення коли курсор знаходиться над нашою кнопкою задається зображення з назвою RestartHovered. Даний елемент також кріпиться за допомогою Anchors до лівої середини екрану.

Створення функціоналу для даного елементу є доволі простим: додаємо подію On Clicked а далі за допомогою функції ExecuteConsoleCommand виконуємо restartlevel. Дана функція з аргументом запустить основний рівень нашої гри з самого початку. Вона є вбудованою у стандартну бібліотеку ігрового рушія Unreal Engine 4. Після цього функціонал першої кнопки буде повністю завершений.

Переходимо до створення наступної кнопки, яка відповідає за повернення до головного меню. Інтерфейс головного меню буде описаний нижче. Аналогічно до попереднього елементу додаємо зображення MenuButton для звичайного вигляду та MenuHovered для виділенного.

Функціонал створюється аналогічно: для події OnClick викликаємо функцію Open Level та передаємо туди значення LevelMain. Тож при натисканні на дану кнопку ми будемо переходити до рівню з головним меню. Механізм роботи головного меню буде описаний нижче.

Головне меню – один із основних елементів будь-якого додатку, або відеогри. З його допомогою користувач може запускати ігровий процес, виходити з гри та реалізовувати управління додатком у цілому. У даному проекті головне меню буде виконувати лише 2 функції: запуск гри та вихід з неї. Макет даного меню показаний нижче:

Рис 3.32 Ігрове меню

Як ми можемо бачити меню має простий мінімалістичний дизайн, що є ознакою мобільних додатків та містить невелику кількість елементів, а саме: 2 кнопки та задній фон.

Для створення даного елементу генерується новий рівень з назвою LevelMain у даному рівні не буде нічого, окрім нашого ігрового меню, тож використовуємо «чистий аркуш». З’являється чорний екран, оскільки у нашому рівні немає нічого, крім порожнього простору.

Переходимо до файлів Blueprint та створюємо файл Main Menu, який буде відповідати за функції меню. У даний файл поміщуємо заздалегіть створену картинку: BackGround розмірами 1280 на 720 пікселів. Ця картинка буде нашим «заднім фоном», що забезпечить нам основну частину зовнішнього

вигляду. Її необхідно закріпити за допомогою Anchors на всю площину екрану.

Після того як ми створюємо елементи кнопки, необхідно переконатися що параметр ZOrder для заднього фону дорівнює нулю, а для кнопок рівняється одиниці. Це забезпечить положення кнопок на поверхні заднього фону. Функціонал першої кнопки: Start повинен запускати ігровий процес, тож після додавання на нього зображень StartButton та StartBotton Pressed ми створюємо подію OnClick.

Дана подіє буде виконувати єдину функцію OpenLevel з ім’ям Level1, який, як зазначено вище є рівнем з основним ігровим процесом. За допомогою цієї функції, після натискання на дану кнопку буде запускатися основний ігровий процес.

Інший об’єкт типу Button також буде викликати подію OnClick, але використовувати буде функцію Execute Console Command з параметром Exit. Даний параметр буде закривати наш додаток при виклику функції. Звісно, для даного елементу необхідні графічні налаштування тож додаємо оформлення у вигляді StartButton та StartButtonPressed.

Макет даних елементів буде виглядати таким чином:

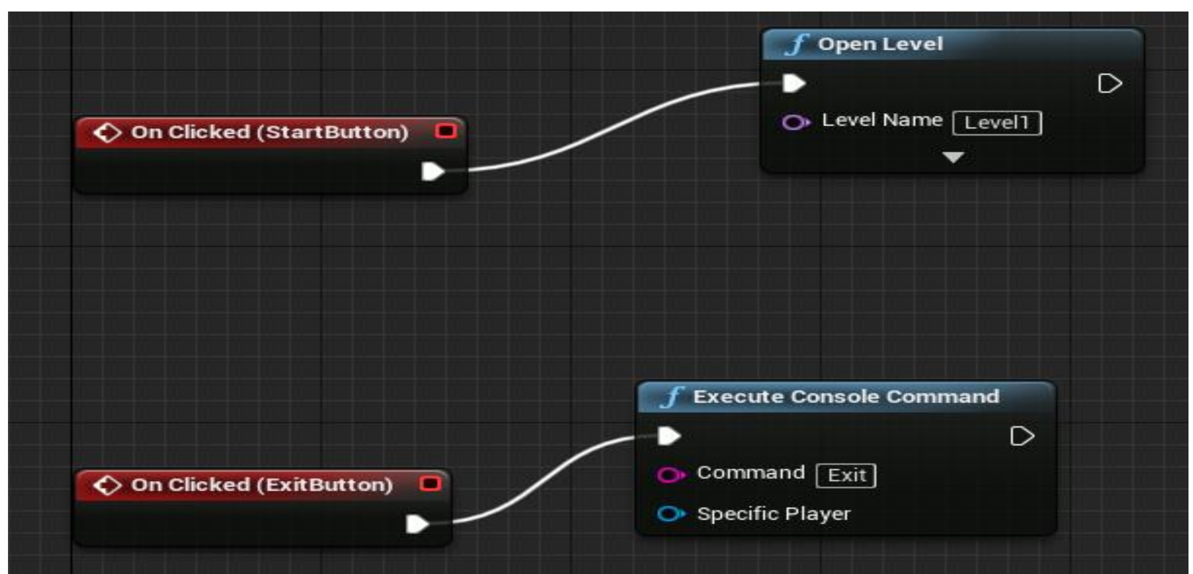


Рис 3.33 Функціонал кнопок

3.2.8 Тестування і виправлення помилок

Після створення і налаштування головного меню наш основний функціонал готовий для використання і портування на Android девайси. За виключенням графічних налаштувань, які будуть оформленні нижче та управління, яке реалізується у останній момент основні функції перебувають у готовому варіанті.

Проте при тестуванні ігрових механік, можно було виявити ряд суттєвих недоліків, які негативно впливають на загальний ігровий процес та параметри деяких функцій.

Основними недоліками на даному етапі є: миттєва смерть ігрового персонажа при запуску гри за рахунок того, що перша частина ігрового поля, а саме перша ігрова «плитка» може містити перешкоду, яка «вбиває» ігрового персонажа, а також рідкість генерації перешкод та об’єктів для взаємодії у ігровому світі.

Остання проблема викликає одразу два недоліки: занадто часту генерацію бонусів, що знижує їх цінність та змінює ігровий процес і створення значної кількості перешкод, які гравець не в змозі уникнути.

Почнемо з вирішення першої помилки, для цього переходимо у наші Blueprints, дублюємо наш файл MastetTile та називаємо його firslTile, прибираємо у події Event BeginPlay функцію Spawn Obstacles, після чого переходимо у ThirdPersonGameMode і додаємо до події BeginPlay Spawn FirstTile перед основним тілом функції:

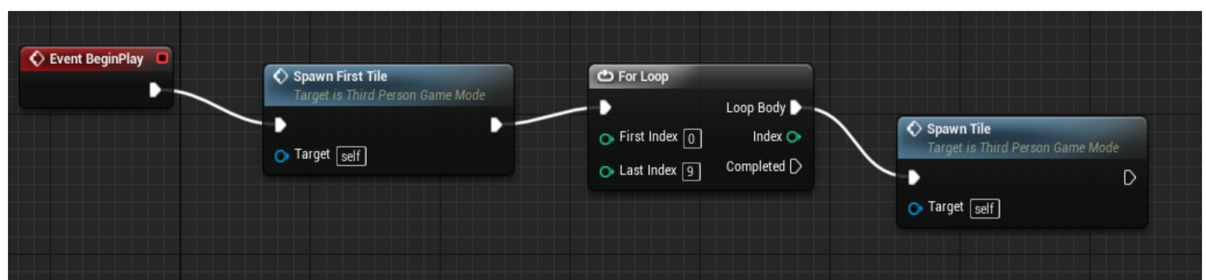


Рис 3.34 Видалення помилки «миттєвої смерті»

Це забезпечить ігровому персонажу «безпечну зону» довжиною у одну плитку на початку ігрового процесу та забезпечить гравця часом, необхідним для того щоб зорієнтуватися.

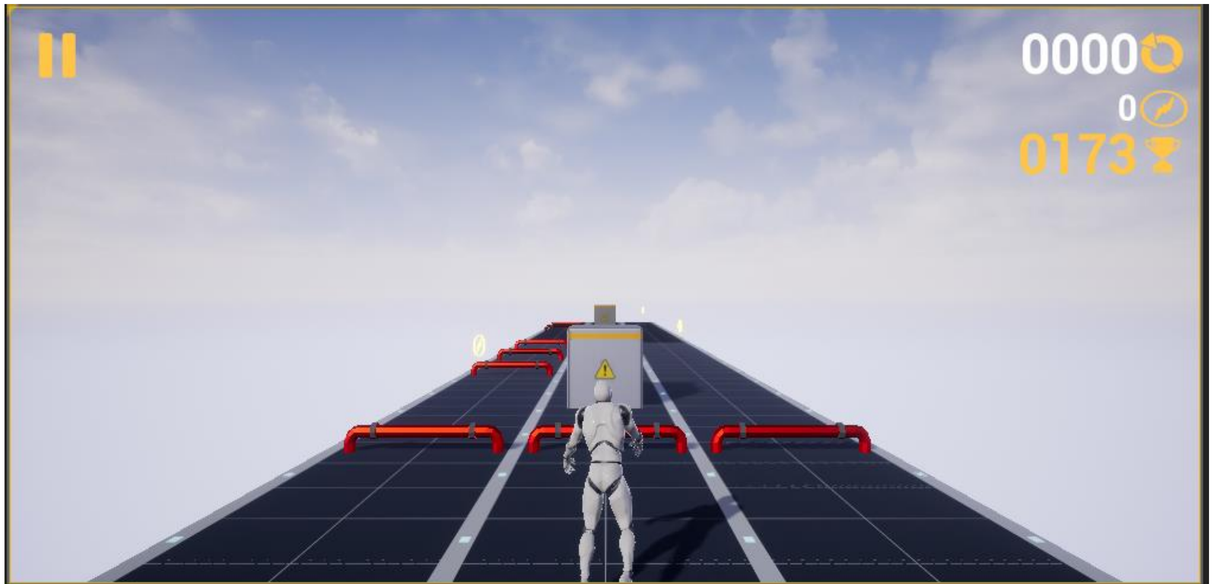


Рис 3.35 «Безпечна зона» ігрового персонажа

На усунення другої помилки – з частотою генерації об’єктів у ігровому світі знадобиться переробити алгоритм «випадкового» створення їх у ігровому світі. Наразі даний алгоритм містить у собі функцію, яка видає випадкове цілочисленне значення в діапазоні від 0 до 5. В залежності від отриманого числа на ігровій платформі гарантовано з’являється один з елементів. Легко зрозуміти що у всіх предметів буде один і той же шанс випадення - 16.6%.

Для того, щоб уникнути даного явища, кожному предмету необхідно призначити його шанс випадення. Для реалізації даної функції буде застосовано ряд умов, а саме: перевірку умов генерації перешкоди та 2 функції RandomFloat in Range та InRange (float).

Алгоритм буде працювати наступним чином: при виклику функції SpawnObstacles активувати функцію Random Float in Range на відрізку від 0 до 1, а для кожної перешкоди ми створимо функцію in Range (Float) зі своїми відрізками, що фактично і буде дорівнювати «шансу створення». У нашому випадку шанси випадення будуть розподілені наступним шляхом: для перешкоди типу ObstaclePipe це буде відрізок від 0 до 0,4 що гарантує шанс створення 40% на будь-якій з ліній, для перешкоди ObstacleBox – шанс буде нижчий – від 0,4 до 0,5 (10 відсотків).

Для об’єктів монет та бонусів шанси повинні бути відповідні: для монет – проміжок від 0,5 до 0,95 – 45 відсотків, оскільки основним ігровим елементом є саме збір даних монет, а для магніту і чобіт, які збільшують висоту стрибка відповідно: 0,95 – 0,975 та 0,975 до 1. Так, частота генерації об’єктів бонусів є доволі низькою, усього 5 відсотків на кожну з ліній, але беручи до уваги кількість ліній – три, а також те, що ігровий персонаж пробігає один ігровий елемент за час, що приблизно дорівнює двом секундам дана частота є оправданою.

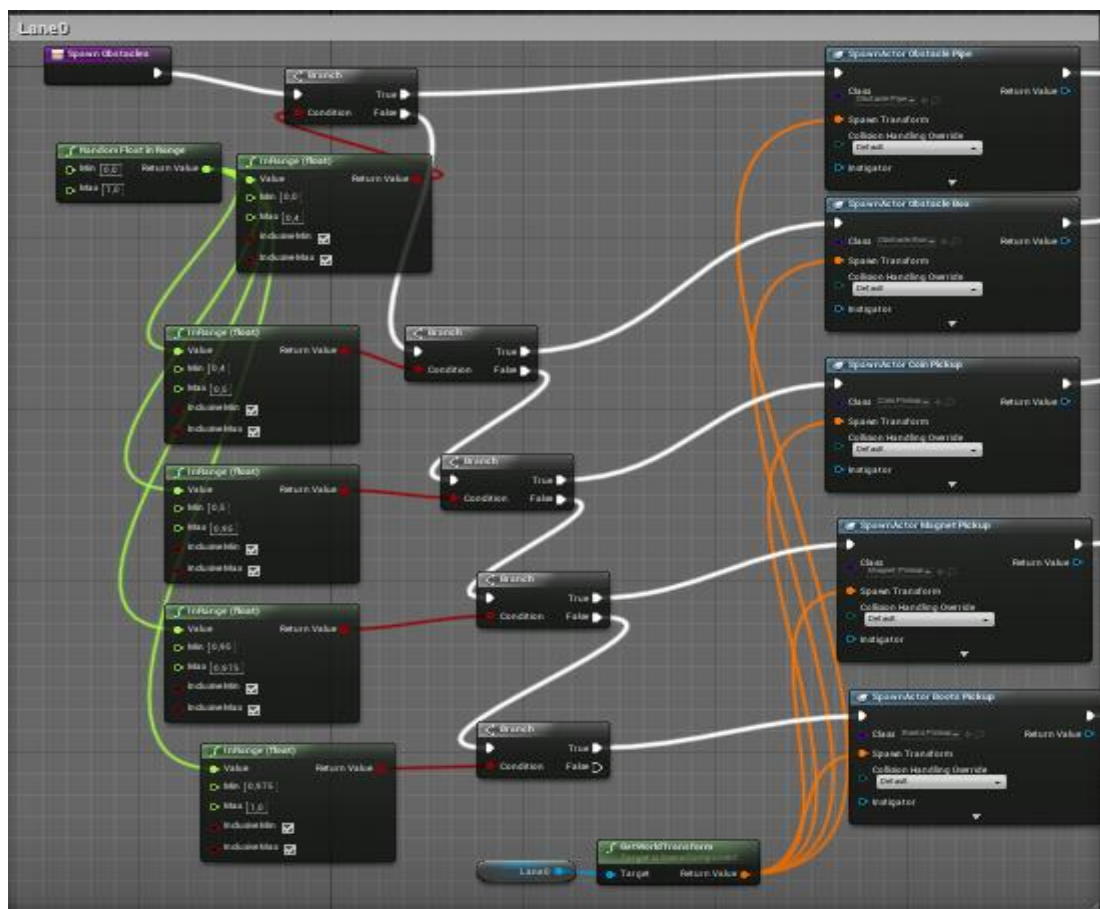


Рис 3.36 Алгоритм створення перешкод

Дані налаштування забезпечують коректність роботи генерації ігрових об’єктів а також запобігають створенню безвихідних ситуацій (генерація трьох великих перешкод одночасно), а також забезпечують ігровий процес стійким алгоритмом, який не надає переваги користувачу.

3.2.9 Управління на ОС Android

На даний момент наш додаток є повністю функціональним, він підтримує необхідні функції, має коректний ігровий процес, підтримує елементи меню та інтерфесів.

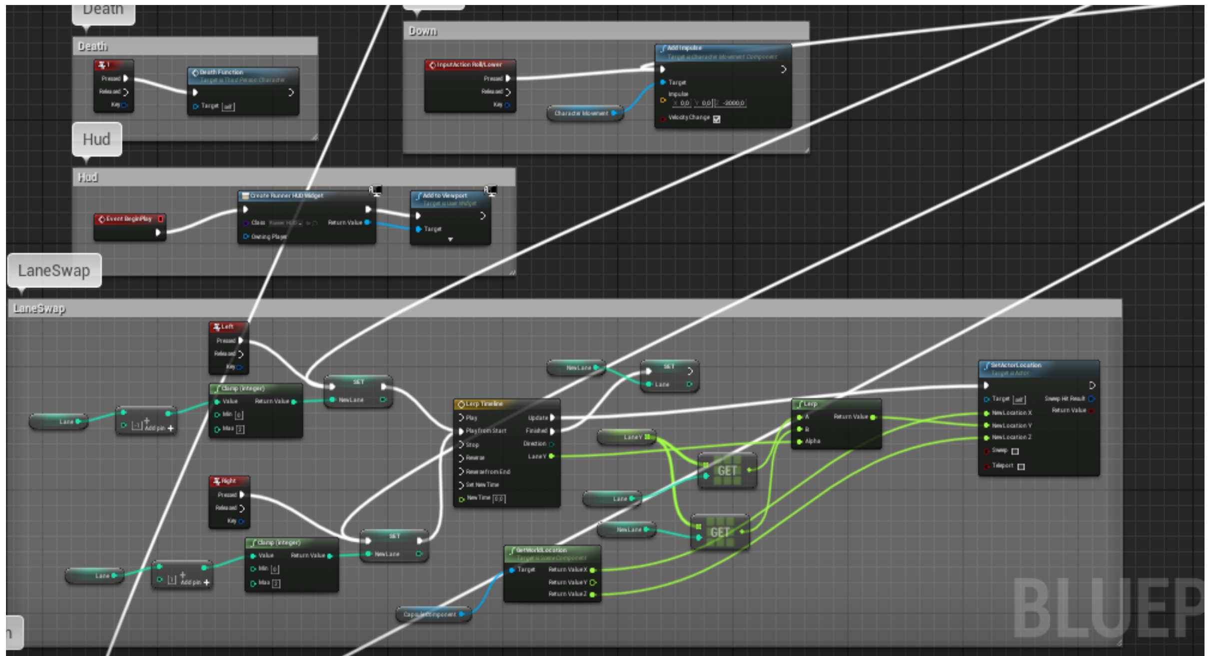


Рис 3.37 Наявний інтерфейс управління

Але, як продемонстровано вище, на даному етапі розробки додаток використовує клавіші, які користувач повинен натискати на персональному комп’ютері. Оскільки додаток-гра розроблений для ОС Android він має містити елементи управління для даної системи, а саме: систему жестів, яка буде виконувати всі функції переміщення.

Загальний вигляд системи буде таким: користувач за допомогою жестів, а саме проведення пальцем (свайпу) в потрібному напрямку буде змінювати лінії, стрибати, та швидко опускатися униз.

Зазначимо, що для кожного «жесту» буде відповідна дія: свайп – угору буде активувати стрибок, відповідно свайп донизу буде переносити ігрового персонажа вниз, а свайпи вліво і вправо будуть переносити його у відповідні сторони.

Основний алгоритм зміни ліній та рухів вже побудовано, нам необхідно додати лише механізм розпізнавання жестів.

Даний механізм буде побудований на двовимірних векторах. Вектор - у найпростішому випадку математичний об'єкт, який характеризуться величиною і напрямком. Наприклад, у геометрії и в природничих науках вектор є спрямований відрізок прямо в евклідовому просторі (или на площині).

Приклади: радіус-вектор, швидкість, момент сили. Якщо в просторі задана система координат, то вектор однозначно задається набором своїх координат. Тому в математиці, інформатиці и інших науках упорядкований набір чисел часто теж називають вектором. У більш загального сенсі вектор у математиці розглядається як елемент деякого (лінійного) простору.

Оскільки простір екрану є двовимірним ми будемо використовувати відповідні вектори з назвами: TouchStart2dVector та TouchEnd2dVector. Ці структури будуть зчитувати 2 точки: першу точку, до якої наш користувач прикладе палець, та другу точку, яка буде кінцем нашої умовної лінії свайпу. В залежності від числових даних, які будуть отримані цими векторами, буде відбуватися певна дія.

Умовний механізм роботи показаний нижче:

Рис 3.38 Механізм роботи свайпу

Тож перейдемо до логіки роботи даного алгоритму. Переходимо до Blueprint ThirdPersonCharacter, оскільки саме він відповідає за рухи нашого

ігрового персонажу. Викликаємо подію InputActionFinger1, яка відповідає за зчитування поведінки пальця у Unreal Engine 4. Викликаємо поступово 2 функції, які нададуть доступ до рухів персонажу: Get Player Controler та GetInputTouchState. Після чого виконуємо команду MakeVector 2D, тобто відкидаємо третю змінну з трьохвимірного простору.

Паралельно до цієї частини коду підводимо частини коду: Set IsPressed – True, для того, щоб розуміти, що користувач натиснув на екран та функцію Set, яка передає значення у Start2dVector.

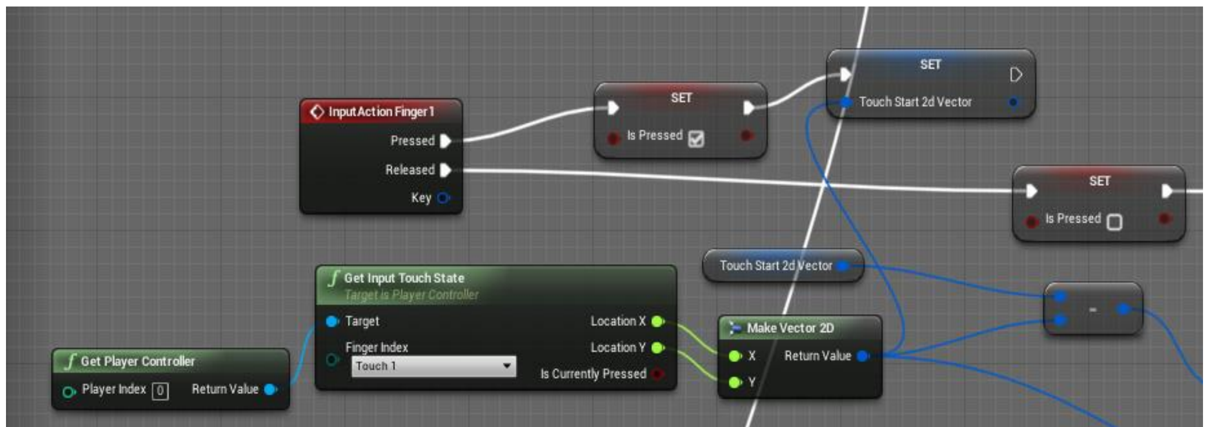


Рис 3.39 Логіка зчитування позиції вектору

Тобто, при натисканні на екран встановлюються початкові позиції натискання і фіксуються у відповідному векторі.

Далі змінна IsPressed повертається у початкове, значення, тобто дорівнює – False. Для запобігання випадкових натискань встановимо перевірку на довжину вектора у 100 пікселів. Якщо, наш вектор буде меньшої довжини, відповідні ігрові анімації не будуть програватися.

Після того, як користувач ігрового додатку відпускає сенсорний екран, тобто припиняє рух пальця по ньому фіксується точка закінчення руху у змінну End 2d Vector.

На даний момент часу у нас є два положення пальця гравця, їх координати записані у відповідні змінні, як же визначити у якому напрямку користувач виконав свайп. Згадаємо сітку координат, яка складається з двох вісей: абсиц і ординат, або X та У відповідно.

Як можемо бачити, вісі будуть мати різні значення: додатні чи від'ємні в залежності від їх положення у просторі.

Нам необхідно попарно порівнювати значення Х та У, отримані у результаті зчитування руху гравця.

Для цього створюємо 3 умовних оператори Branch. У функцію Break Vector 2d, заводимо різницю наших векторів Start 2d Vector та End 2d Vector. Та отримуємо 2 значення вісей координат Х та У.

Тепер нам необхідно попарно порівняти ці значення, що дасть нам 4 варіанти розвитку подій:

1) Ми порівнюємо значення по модулю Х та У, якщо змінна Х виявляється більшою за змінну У та змінна Х без модуля більша за 0 ми виконуємо рух вліво.
2) Якщо значення по модулю Х більше за значення У по модулю і змінна Х менша за нуль, виконуємо рух Вправо.
3) Якщо змінна Х менша за модулем за змінну У та змінна У більша за нуль ми виконуємо анімацію стрибка.
4) Останнім варіантом наших введених даних, я те що змінна Х менша за модулем за змінну У, але У менше за 0, що у нашому випадку відповідає за рух донизу.

Як можемо бачити у нас передбачені всі можливі варіанти взаємодії гравця з ігровим світом. Для завершення необхідно під'єднати всі ці варіанти до відповідних анімацій.

Відповідно для свайпу уліво ми використовуємо раніше створену анімацію Lane Swap, та під'єднуємо до блоку який відповідає за рух на ліву лінію. Відповідні дії ми робимо і для свайпу вправо.

Для руху угору і униз у нашому додатку застосовані інші елементи, а саме: для руху угору, тобто стрибку, ми знаходимо подію Input Action Jump та під'єднуємо до функції, яка знаходиться поруч – Jump. Для руху униз, ми знаходимо блок коду Input Action Low Lover та функцію Add Impulse, яка буде рухати «фігурку» ігрового персонажу донизу.

Після налаштування відповідних елементів, нам необхідно надати нашому проекту можливості зчитувати рухи не тільки з сенсорного екрану, а і рухи курсору, що будуть імітувати рух руки гравця.

Для цього переходимо у вкладку Edit – Project Settings, знаходимо вкладку Input, яка відповідає за введення даних та виставляємо наступні параметри:

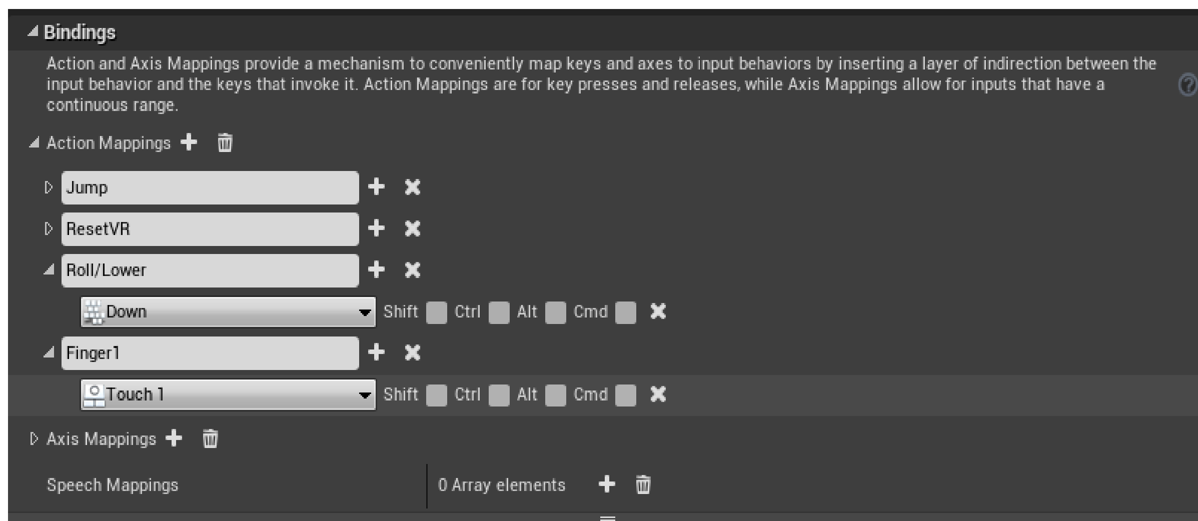


Рис 3.40 Параметри проекту

Встановлюємо галочку Use Mouse for Touch у полі Mouse Properties, що дозволить нам тестувати дані нововведення за допомогою курсору миші.

Після цих налаштувань управління для ОС Android повинно працювати коректно.

3.2.10 Порт додатку гри на ОС Android

Метою розробки даного проекту на ігровому рушії Unreal Engine 4 було запустити кінцевий продукт на девайсі з ОС Android. Як вказано вище, дана операційна система є найбільш розповсюдженою у всьому світі та встановлена більш ніж на двох мільярдах пристроїв.

Даний проект є універсальним та може бути запущений на будь-якому смартфоні, планшеті і т.д. який використовує це програмне забезпечення. Досягається це завдяки таким наборам для розробки як SDK та NDK.

Android SDK - універсальний засіб розробки мобільних додатків для операційної системи Android. Відмінною рисою від звичайних редакторів для написання кодів є наявність широких функціональних можливостей, що дозволяють запускати тестування і налагодження вихідних кодів, оцінювати роботу програми в режимі сумісності з різними версіями ОС Android і спостерігати результат в реальному часі. Засіб підтримує велику кількість мобільних пристроїв, серед яких виділяють: мобільні телефони, планшетні комп'ютери, розумні окуляри, сучасні автомобілі з бортовими комп'ютерами на, телевізори з розширеним функціоналом, особливі види наручних годинників і багато інших мобільні гаджети, габаритні технічні пристосування.

Android SDK був випущений в жовтні 2009 року. Середовище розробки є кросс-платформенним, значна частина комплекту написана на мові програмування Java. До складу SDK включені різні засоби розробки, в тому числі відладчик, набір бібліотек, телефонний емулятор на базі движка QEMU, набір документації, прикладів додатків і посібники. Середовище Android SDK може бути запущена на комп'ютерах, що використовують ОС Linux, Mac OS X 10.5.8 і новіше, Windows 7 і новіше. Однак, станом на березень 2015 року система SDK не може бути безпосередньо запущена на пристроях під управлінням ОС Android.

У 2015 році вийшов комплект розробки Android Studio (розроблений Google із застосуванням технологій IDE IntelliJ), що став основним. У наприкінці 2015 року ADT став вважатися застарілим, тоді як Android Studio став основною системою розробки додатків для Android. Крім інтеграцій з IDE є використання сторонніх текстових редакторів для створення Java і XML файлів і використання утиліт командного рядка (потрібно Java Development Kit і Apache Ant) для створення проектів, їх компіляції та відлагодження. Також доступні утиліти управління підключеними Android пристроями для перезавантаження і установки додатків: fastboot і adb (Android Debug Bridge).

До складу SDK можливе включення фрагментів застарілих версій платформи Android для випадків, коли розробники готові займатися продовженням розвитку своїх додатків для застарілих телефонних апаратів і планшетів. Частина коштів розробки поставляється у вигляді окремо завантажуваних доповнень.

Додатки Android в закінченому вигляді представляють собою пакети формату .apk і після установки зберігаються в каталозі / data / app. Усередині пакет APK містить кодові файли .dex ,файли ресурсів.

Android NDK - це допоміжний інструмент для Android SDK, який дозволяє створювати критично важливі для додатків частини у власному коді. Він надає заголовки і бібліотеки, які дозволяють створювати операції, обробляти введення користувачів, використовувати апаратні датчики, отримувати доступ до ресурсів додатків і т. Д. При програмуванні на C або C ++. Якщо ви пишете власний код, ваші програми, як і раніше будуть упаковані в файл .apk, і вони все одно будуть працювати всередині віртуальної машини на пристрої. Основна модель додатків для Android не змінюється.

NDK може підвищити продуктивність додатків. Це зазвичай справедливо для додатків, пов'язаних з процесором. Багато мультимедійних додатків і відеоігор використовують власний код для задач, що вимагають великих ресурсів. Покращення продуктивності може виходити з трьох джерел. По-

перше, власний код компілюється в двійковий код і запускається безпосередньо в ОС, а Java-код транслюється в байт-код Java і інтерпретується за допомогою віртуальної машини Dalvik Virtual Machine (VM). В Android 2.2 або вище компілятор Just-In-Time (JIT) додається в Dalvik VM для аналізу і оптимізації байт-коду Java під час роботи програми (наприклад, JIT може скомпілювати частину байтового коду для двійкового коду перед його виконанням). Але в багатьох випадках власний код і раніше працює швидше, ніж Java-код.

Java-код запускається Dalvik VM на Android. Dalvik VM спеціально розроблений для систем з обмеженими апаратними ресурсами (обсяг пам'яті, швидкість процесора).

Ще одне джерело підвищення продуктивності в NDK є те, що власний код дозволяє розробникам використовувати деякі функції процесора, недоступні в Android SDK, такі як технологія NEON, технологія множинних даних з однієї інструкцією (SIMD), що дозволяє обробляти декілька елементів даних паралельно. Одним конкретним прикладом таких задач є перетворення кольору для відеофрагменту або фотографії. Припустимо, ми повинні перетворити фотографію 1920 × 1280 пікселів з колірного простору RGB в колірний простір УCbCr. Наївний підхід - застосувати формулу перетворення до кожного пікселя (тобто понад два мільйони пікселів). За допомогою NEON ми можемо обробляти кілька пікселів одночасно, щоб скоротити час обробки.

Третій аспект полягає в тому, що ми можемо оптимізувати критичний код на рівні збірки, що є звичайною практикою в розробці настільних програм.

З недоліків можливо виділити те що: NDK не може безпосередньо звертатися до безлічі API-інтерфейсів, доступних в Android SDK, а розробка в NDK завжди буде вводити додаткову складність в вашу програму.

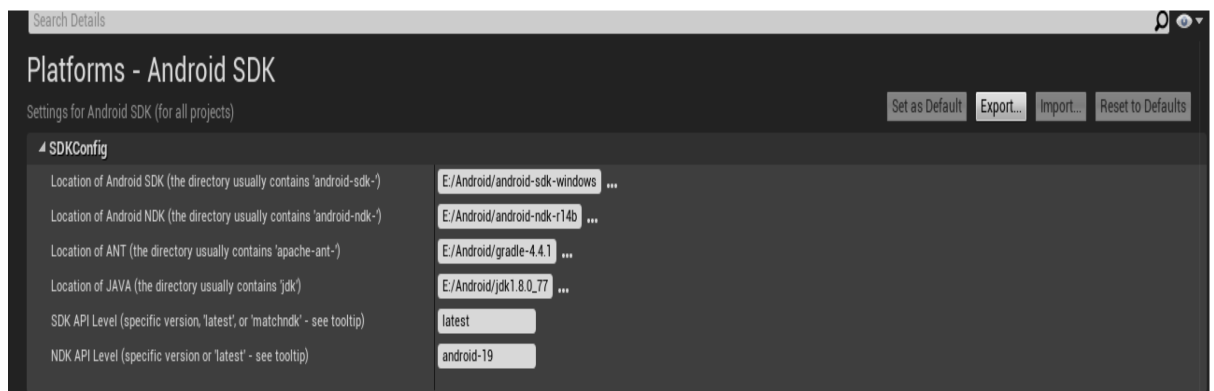
Після того, як було вказано на різницю між цими «пакетами» розробника переходимо безпосередньо до налаштування проекту. Для початку нам

необхідно визначитися з версіями SDK та NDK, які ми будемо використовувати для портування додатку.

Заходимо у налаштування нашого проекту через вкладку Project Settings. Знаходимо вкладку Android та натискаємо «прийняти можливість використання налаштувань Android у даному проекті». Даний параметр автоматично перекомпілює всі необхідні налаштування проекту: текстури, файли Blueprint, об’єкти під звичайний додаток Android.

Зараз наш проект структурно перебудований під вид Android додатку, але при його «пакуванні» та публікації він не отримає файли налаштувань SDK та NDK і не запуститься на ОС Android.

Для того щоб додати ці елементи необхідно завантажити їх з офіційного сайту дистрибутивів Android. Використаємо найпопулярніші версії NDK: r14-b, Grade – grade-4.4.1. Та виставляємо налаштування як на рисунку нижче:



3.41 Настройки Android

Після цього проходить процес переналаштування даного проекту для Android девайсу. Останнім кроком для публікації буде пакування проекту у файл з розширенням .app.

APP - виконавчий файл, створений за допомогою програмного модуля Symbian OS - операційної системи для мобільних пристроїв та інших гаджетів. Формат APP відноситься до категорії файлів Symbian OS Application і може бути скомпільовано за допомогою інструментів Symbian OS.

APP - це унікальний формат до складу якого можуть входити виконавчі програми, ресурси і процеси, які використовуються додатками, ярлики, програмні плагіни і оболонки.

Він не затребуваний в колі звичайних користувачів, проте без нього неможлива коректна робота окремих модулів операційної системи і багатьох виконавчих файлів.

Для того щоб спакувати наш проект переходимо у вкладку File та вибираємо такі налаштування:

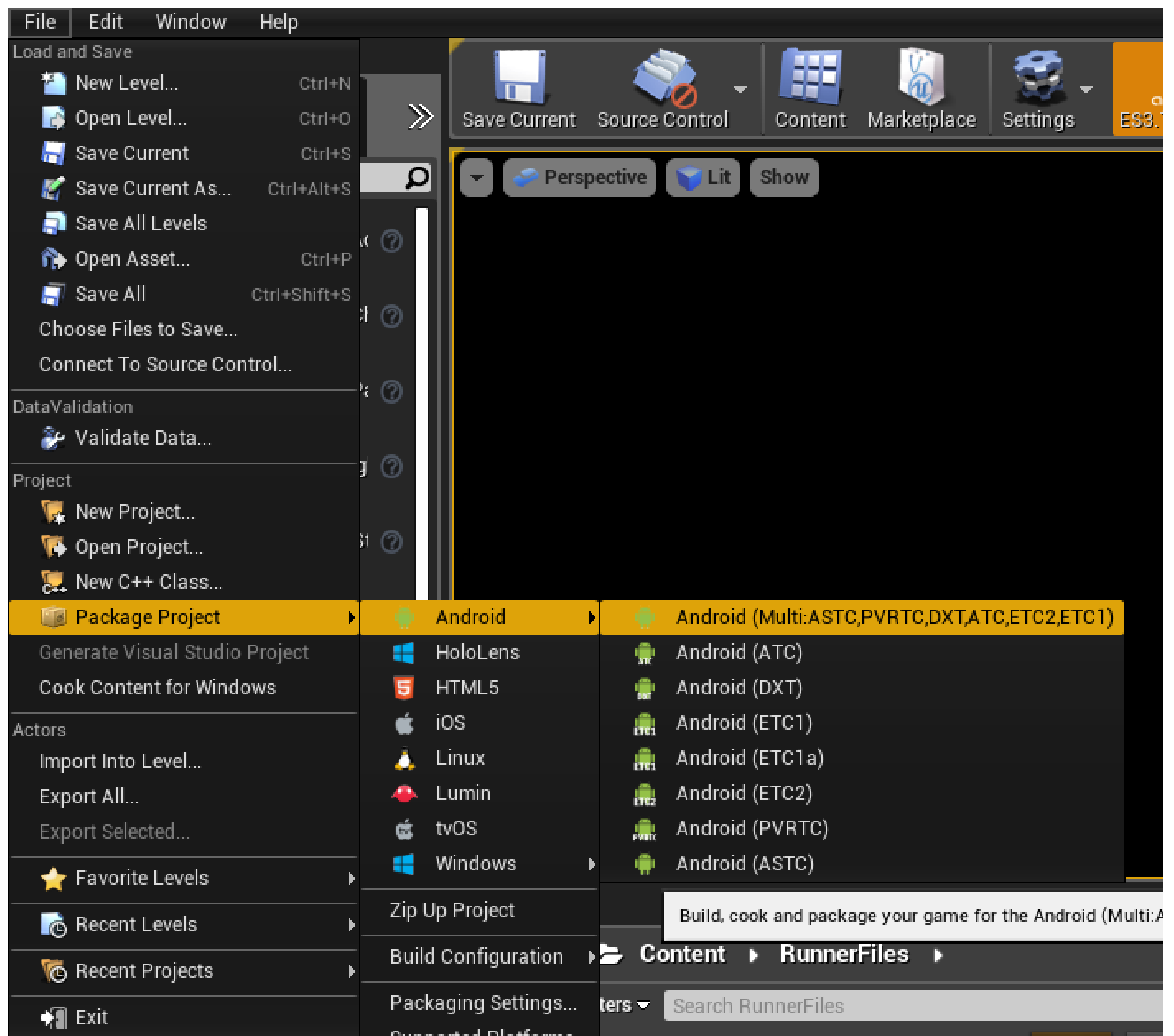


Рис. 3.41. Пакування проекту на Android

Запускається процес компіляції та пакування файлу. Під час цього процесу необхідно ввімкнути тестовий девайс за допомогою USB порту до ПК, на якому був розроблений даний проект.

Зазначимо, що процес створення файлу .app може зайняти значний час, оскільки всі текстури, частини коду, об’єкти та класи будуть автоматично адаптовані до Android девайсу. Графічні зображення будуть понижені у якості (для забезпечення оперативної роботи додатка), а інші налаштування будуть перекомпільовані для того, щоб операційна система могла взаємодіяти з даним файлом.

Запускаємо наш проект на тестовому девайсі:

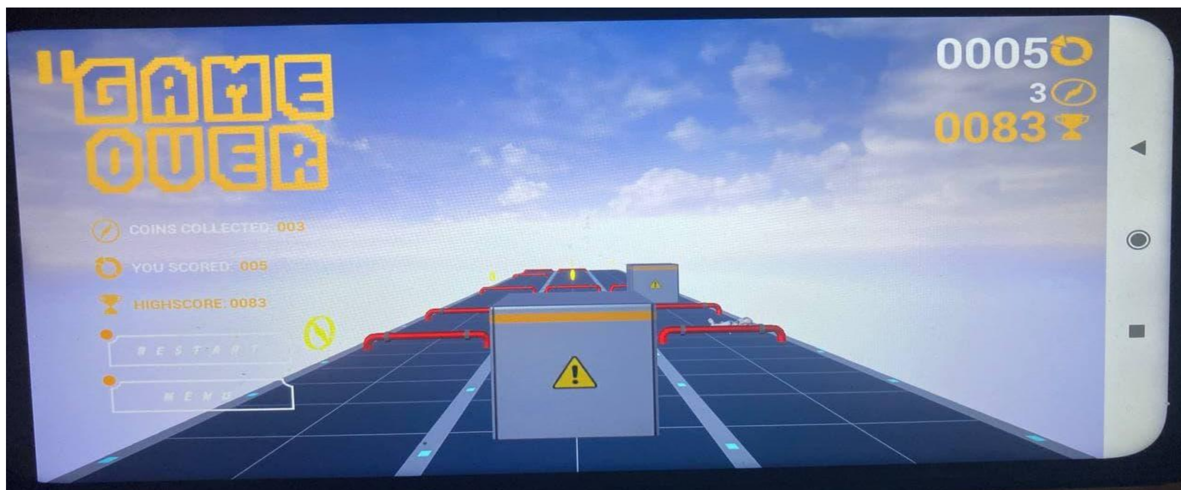


Рис. 3.42 Готовий проект

Висновки до розділу

У третьому розділі дипломного проекту ми детально розглянули процес розробки мобільного додатку для операційної системи Android. Дослідили створення функцій, анімацій, елементів меню та об’єктів ігрового світу, а також фізику взаємодії ігрового персонажу та налаштування управління для користувача. Як продемонстровано вище, даний проект є повністю готовим, скомпільованим та відповідає всім необхідним стандартам, які були задані у першій частині нашої роботи. Тож на даний момент проект є повністю закінченним.

ВИСНОВКИ

Виконуючи завдання дипломного проекту, а саме: Розробка мобільного додатку-гри для ОС Android ми ознайомилися з принципами побудови додатків для даної операційної системи у ігровому рушії Unreal Engine 4.

В процесі проектування додатку було виділено основні вимоги та функції додатку, які були успішно реалізовані у остаточній версії. При виконанні завдання були використані такі технології як: Blueprint (мова програмування Unreal Engine 4), графічні редактори для трьохвимірного створення об'єктів – Blender, для створення текстур та матеріалів Adobe Photoshop, пакети розробників для операційної системи Android (SDK та NDK), тестування програмного забезпечення за допомогою вбудованих можливостей Unreal Engine 4 та за допомогою тестування самого розробника.

У ході розроблення додатку були також виділені основні елементи ігр даного жанру: створення найвищого рахунку, взаємодія з об'єктами та елементами ігрового світу. Створений скелет персонажу та зовнішній вигляд ігрового світу, елементи меню.

Були використані основні принципи об'єктно-орієнтованого програмування: інкапсуляція, наслідування та поліморфізм. Кінцевий проект може бути перекладний на дві високорівневі мови програмування, а саме: Java та C++ .

А найголовніше: було досліджено один з найновіших напрямів створення програмних додатків та забезпечення – візуальний, у якому код кінцевого продукту створюється за допомогою поєднання блоків об'єктів, змінних, класів та функцій та побудови логіки роботи. Як наслідок програміст витрачає набагато менше часу на розробку елементів системи та відповідне їх тестування.