МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ
з навчальної практики
«ТЕХНОЛОГІЧНА ПРАКТИКА (навчальна)»
для студентів 2-го курсу
рівня вищої освіти «бакалавр» та «молодший бакалавр»

спеціальності 122 «Комп'ютерні науки»

Затверджено
на засіданні групи забезпечення
спеціальності 122 Комп'ютерні науки
від «30» березня 2023 року
Протокол № 13

Голова групи _____ Кузніченко С.Д.

Затверджено на засіданні
кафедри Інформаційних технологій
протокол № 6 від «16» березня 2023р.

Завідувач кафедри _____ Казакова Н.Ф.

Одеса – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ
з навчальної практики
«ТЕХНОЛОГІЧНА ПРАКТИКА (навчальна)»
для студентів 2-го курсу
рівня вищої освіти «бакалавр» та «молодший бакалавр»

спеціальності 122 «Комп'ютерні науки»

Одеса – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ
з навчальної практики
«ТЕХНОЛОГІЧНА ПРАКТИКА (навчальна)»
для студентів 2-го курсу
рівня вищої освіти «бакалавр» та «молодший бакалавр»

спеціальності 122 «Комп'ютерні науки»

Затверджено
на засіданні групи
забезпечення спеціальності
Протокол № 13
від «30» березня 2023р.

Одеса – 2023

Методичні вказівки до навчальної практики „***Технологічна практика (навчальна)***”, для студентів ІІ року навчання за спеціальністю 122 «Комп’ютерні науки», рівень вищої освіти бакалавр / Гадяцький І.А. Молчанова А.Ю. – Одеса, ОДЕКУ, 2023.

ЗМІСТ

ВСТУП

Практична підготовка студентів має на меті набуття студентами професійних навичок та вмінь для прийняття відповідних рішень під час самостійної роботи.

Навчальна практика «Технологічна практика (навчальна)» проводиться на базі учбових лабораторій кафедри інформаційних технологій із застосуванням відповідного програмного та апаратного забезпечення. Термін проведення практики дорівнює 120 годин, тобто п’ятнадцять днів по шість учбових годин. Крім того, на практику відводиться 30 годин самостійної роботи.

Навчальна практика Призначена для студентів, що навчаються за спеціальністю 122 «Комп’ютерні науки». Практика базується на знаннях студентів з дисциплін: «Алгоритмізація та програмування», «Операційні системи», «Алгоритми та структури даних».

Мета, завдання та задачі практики

Мета навчальної практики – оволодіння базовими вміннями створення 2D гри Flappy Bird в середовищі Unity, використовуючи різноманітні інструменти та технології, такі як скрипти, фізика, анімація, звуки та графіка.

У результаті виконання програми практики студент повинен:

знати:

- принципи будування об’єктів;
- їх властивості (інкапсуляцію, спадковість та поліморфізм);
- елементи програмування мовою об’єктного програмування: C#;
- структуру інтегрованого середовища розробки програм Unity.

уміти:

- розуміти принципи роботи ігрового рушія Unity та створення 2D гри з використанням цього інструменту;
- розуміти базові принципи геймплею та дизайну гри;
- розуміти використання скриптів та роботи з компонентами Unity для розробки ігрових механік;
- розуміти використання анімації, звуків та графіки для створення затягуючого геймплею;
- вміти відлагоджувати проблеми та вирішувати їх на практиці.

Програма практики

1-й тиждень: Ознайомлення з програмою практики. Оформлення щоденника. Огляд сучасних інструментальних засобів розробки ігрових додатків. Основні принципи роботи в середовищі розробки ігрового додатку.

2-й тиждень: Створення ігрового середовища з використанням обраних інструментальних засобів. Розробка геймплея ігрового додатку. Динамічне створення об’єктів.

3-й тиждень: Розробка інтерфейсу з використанням обраних інструментальних засобів. Оформлення звіту з практики та його захист.

Зміст практики

У відповідності до мети та завдань технологічна (навчальна) практика включає:

1. Виконання індивідуального завдання самостійно під контролем керівника практики.
2. Самостійне вивчення нових технологій, програмних засобів розробки тощо.
3. Збір та вивчення матеріалів для виконання майбутніх науково-практичних задач.
4. Оформлення та захист звіту з технологічної (навчальної) практики.

Індивідуальні завдання

Індивідуальні завдання для кожного студента розробляються керівником практики, завдання студент отримує на першому занятті з практики. Виконання контролюється безпосередньо керівником практики.

Методичні рекомендації

Для виконання завдання на навчальну практику за студентом на постійно закріпляється комп’ютер та всі завдання конкретизуються відповідно до нього. Керівник практики повинен ознайомити студентів з положеннями цієї програми та роздати відповідну навчальну та додаткову літературу в друкованому або в електронному вигляді.

Завдання практики повинні виконуватися студентом тільки у межах навчального класу на тому комп’ютері, що закріплений за ним. Додаткова

робота може проводитись також і самостійно студентом поза навчальними класами, користуючись допомогою відповідної довідкової літератури та інтерактивних довідкових систем. При цьому варто користуватися тими знаннями, що отримані при вивчені навчальних дисциплін: «Алгоритмізація та програмування», «Операційні системи», «Алгоритми та структури даних».

Методи контролю

Після закінчення практики оформлений звіт здається викладачу на перевірку. Якщо вся програма практики виконана в повному обсязі, про що свідчать відповідні записи у щоденнику практики, завірені підписом викладача та звіт оформлений відповідно до вимог, студент отримує залік (60%, «E»). Якщо студент бажає отримати вищий бал, то він захищає звіт (відповідає на питання викладача).

Вимоги до звіту

Основний зміст звіту повинен відповідати тематиці технологічної (навчальної) практики. Перелік та кількість розділів звіту визначає керівник практики. При цьому один з розділів повинен містити результати виконання індивідуальних завдань.

Звіт про виконання навчальної практики повинен містити:

1. Титульний лист.
2. Мета роботи, постановка завдання.
3. Лістинг коду скриптів проекту.
4. Скріншот проекту в Unity Editor.
5. Скріншот запущеної гри на мобільному пристрої.
6. Відповіді на контрольні питання.
7. Висновок.

Два однакових звіту з практики не можуть бути зараховані навіть при умові виконання роботи з того ж самого варіанту.

Правила техніки безпеки та охорони праці

Технологічна практика (навчальна) проводяться у лабораторіях кафедри інформаційних технологій, які оснащені комп'ютерною технікою з відповідним програмним забезпеченням. Студенти зобов'язані дотримуватися правил техніки безпеки та правил користування обчислювальною технікою в лабораторіях кафедри.

Згідно з «Правилами техніки безпеки в лабораторіях» студентам забороняється:

- з`являтися та знаходитись приміщенні в нетверезому стані;
- ставити поруч з клавіатурою ємності з рідиною;
- перебувати в приміщенні у верхньому одязі та завалювати ним робочі столи та стільці;
- працювати в лабораторії більше 6-ти годин на день (для вагітних жінок більше 4-х годин);
- за власною ініціативою змінювати закріплені за ними робочі місця та знаходитись в приміщенні під час роботи іншої учбової групи;
- самостійно виконувати вмикання електроживлення лабораторії та заміну складових частин ПК, що вийшли із ладу.

У випадку виявлення несправності обчислювальної техніки студент повинен сповістити про це викладача чи будь-кого з навчально-допоміжного персоналу лабораторії.

Кожен студент перед початком учбової практики мусить вивчити правила роботи з комп'ютерною технікою в лабораторіях кафедри інформаційних технологій, пройти співбесіду з інструктором по техніці безпеки та розписатися в журналі по техніці безпеки.

ПРАКТИЧНА ЧАСТИНА

Встановлення необхідного ПЗ та створення проекту

Unity – це крос-платформне інтегроване середовище розробки (IDE) для створення ігор та інших візуальних додатків, що працюють на різних платформах, таких як комп'ютери, мобільні телефони, консолі та інші. Вона була створена компанією Unity Technologies та перші версії з'явилися у 2005 році.

Unity має вбудовану підтримку різних платформ, таких як iOS, Android, Windows, macOS, Linux, Xbox, PlayStation та інші. Розробники можуть створювати одну версію своєї гри та експортувати її на різні платформи.

В Unity є вбудовані інструменти для тестування та налагодження застосунків, що дозволяє розробникам швидко виявляти та виправляти помилки.

Unity має потужний рушій рендерингу, який дозволяє створювати вражаючі візуальні ефекти та графіку в 2D та 3D. Розробники можуть користуватися багатьма інструментами та функціями, щоб створювати ігри, такі як фізичне моделювання, штучний інтелект, системи анімації, звукові ефекти та інші. Unity підтримує велику кількість різноманітних форматів файлів, таких як FBX, OBJ, WAV, MP3, PNG, JPG та інші, що дозволяє імпортувати зовнішні ресурси для використання в проекті.

Unity підтримує різні мови програмування, такі як C#, JavaScript, Boo та інші. C# є найбільш популярною мовою програмування для Unity.

Unity Hub – це додаток-менеджер для роботи з Unity, який дозволяє легко встановлювати та управляти версіями Unity, налаштовувати проекти та завантажувати різноманітні компоненти, такі як модулі платформ, шаблони проектів та різні версії рушія Unity.

Завантажити та встановити Unity Hub можна з офіційного сайту Unity за посиланням: https://unity.com/ru/download.

Після запуску Unity Hub вам буде доступна можливість встановити різні версії Unity та додатки, які залежать від версії рушія Unity, такі як Unity Editor, Standard Assets, та інші. Для роботи з Unity Hub необхідно увійти до свого облікового запису Unity або створити новий.

У вкладці Installs (рис. 1) натисніть на кнопку Install Editor. Бажано встановлювати LTS (Long-Term Support) версії Unity, а не останню, адже вони випускаються з меншим темпом і тому мають додатковий час для тестування та виправлення помилок. Це забезпечує надійність та стабільність роботи рушія, що особливо важливо для розробки великих та складних проектів. До того, LTS версії Unity мають довший термін підтримки, ніж остання версія. Це означає, що вони отримують оновлення безпеки та ремонтні патчі протягом тривалого періоду, що дозволяє уникнути можливих проблем, які можуть виникнути внаслідок застосування застарілих версій.

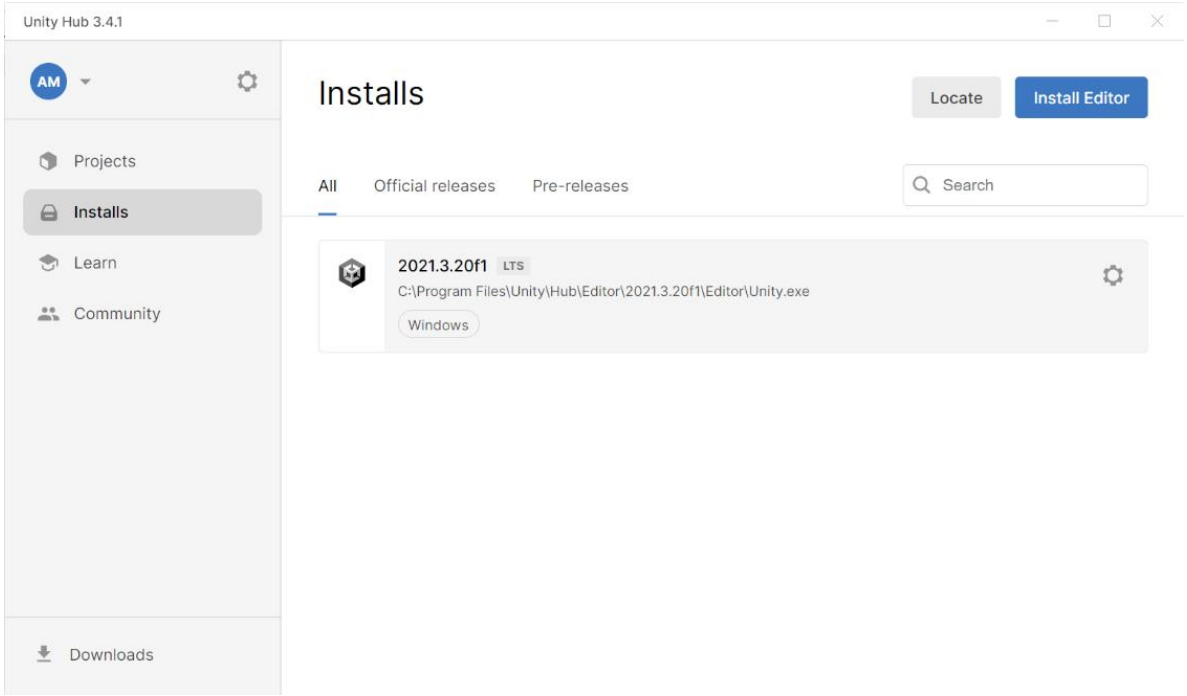

Рис. 1 – Вікно встановлення (installs) версії Unity 2021.3.21f1

Також важливо, що LTS версії Unity зазвичай мають більшу сумісність з різними версіями платформ, бібліотеками та іншими залежностями, що

дозволяє зменшити кількість можливих проблем та збільшити ефективність розробки.

Рекомендованою LTS версією Unity на березень 2023 року є Unity 2021.3.21f1. В даних методичних вказівках буде використовуватись саме ця версія.

Виберіть версію Unity та компоненти, які ви бажаєте встановити. Дочекайтеся завантаження.

Після встановлення Unity, ви можете створювати нові проекти або відкривати існуючі проекти в Unity Editor.

Запустіть Unity Hub. Натисніть New Project, щоб створити новий проект. Оберіть порожній шаблон 2D. Дайте назву проекту, Flappy Bird, та вкажіть папку для зберігання файлів проекту. Натисніть Create project (рис. 2).

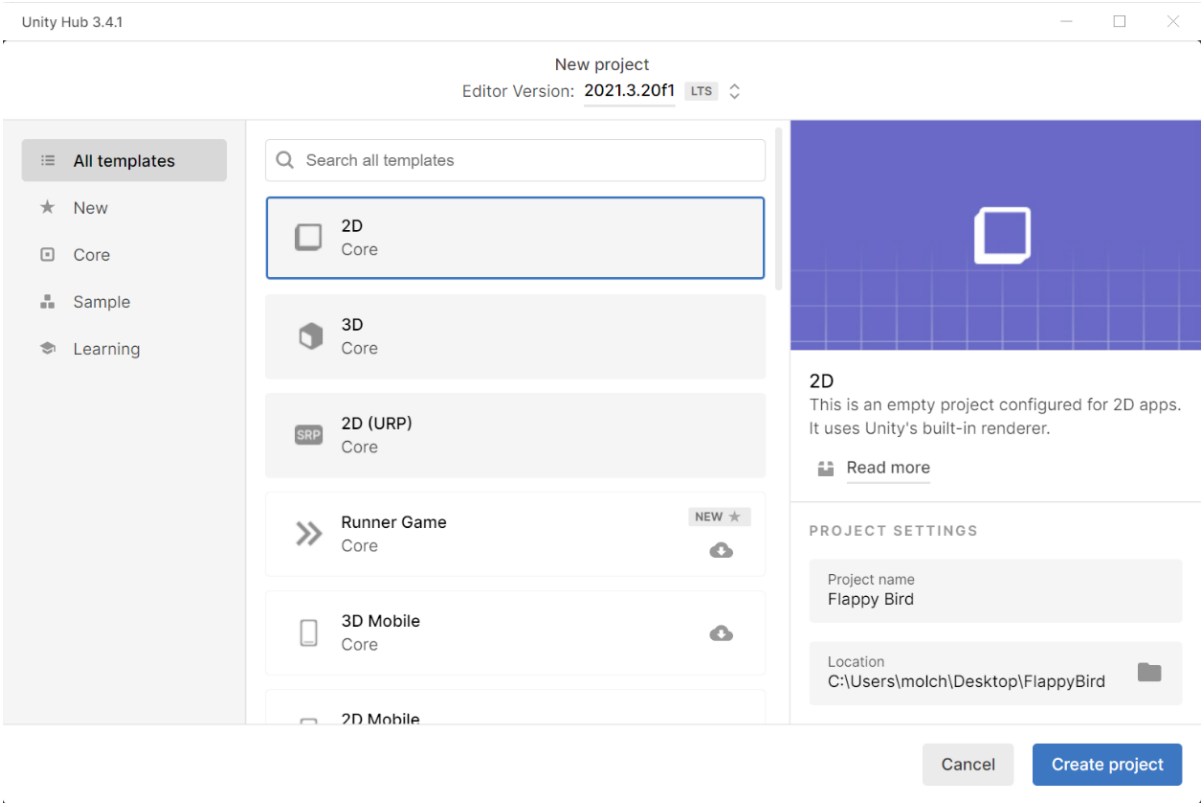

Рис. 2 – Вікно створення нового проекту

Буде створений порожній проект (рис. 3). Перший запуск нового проекту в Unity може займати більше часу, тому що Unity потрібно створити новий проект, що включає створення нового каталогу зі всіма необхідними налаштуваннями та файлами проекту, встановити та налаштувати середовище розробки та завантажити необхідні пакети та бібліотеки.

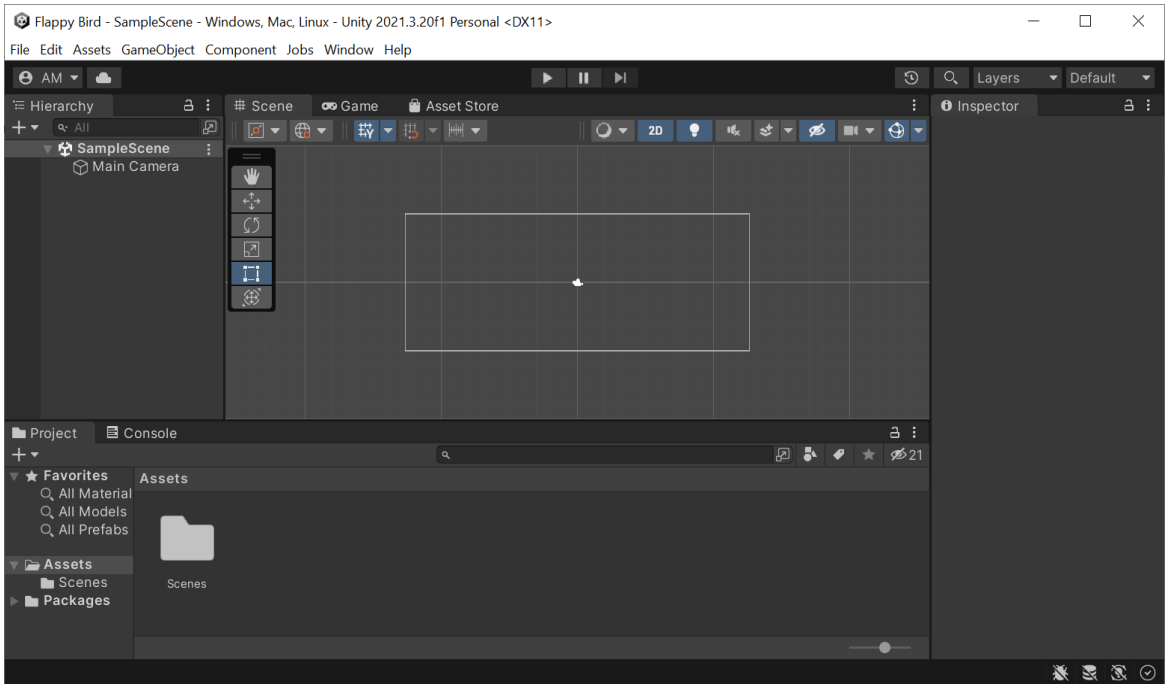

Рис. 3 – Головне вікно редактора

Знайомство з інтерфейсом Unity Editor

Головне вікно редактора складається з кількох вкладок, які називаються видами (Views) або вікнами (Windows). Є 6 основних видів:

– Project

– Scene

– Hierarchy

– Inspector

– Toolbar

– Game

Розташування вікон за замовчуванням дає доступ до цих основних вікон.

Вікно Project – це оглядач проекту (рис. 4). Ліва панель оглядача відображає структуру папок проекту як ієрархічний список. Ресурси проекту називаються assets. В Unity, asset – це будь-який ресурс, що використовується в проекті для створення ігрового вмісту. Це можуть бути текстури, звукові ефекти, моделі 3D об'єктів, скрипти, анімації, матеріали, плагіни, налаштування проекту, сцени та інші елементи. Asset є ключовим елементом процесу створення ігрового контенту в Unity. Вони можуть бути створені у редакторі Unity або імпортовані з зовнішніх ресурсів.

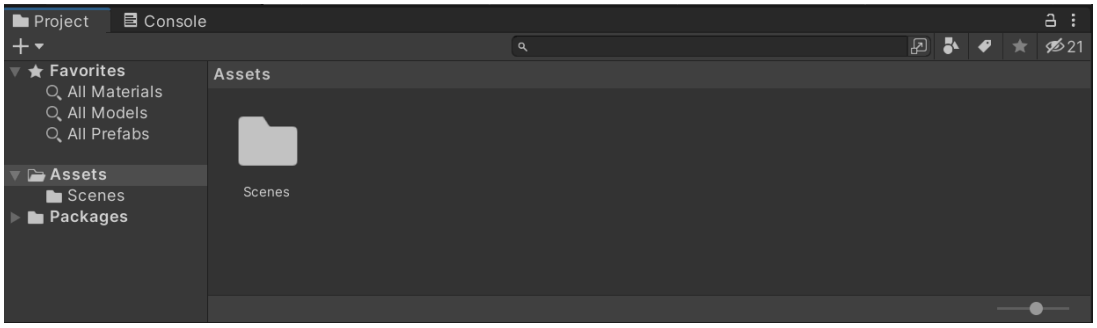

Рис. 4 – Вікно Project

Над списком структури проекту знаходиться розділ Favorites (вибране), в якому можна зберігати посилання на assets, що часто використовуються, для швидкого доступу до них.

В порожньому проекті в Assets є тільки папка Scenes – стандартна папка проекту, яка містить сцени, які використовуються для створення візуального представлення гри або застосунку. Сцени містять об'єкти гри. Вони можуть використовуватися для створення головного меню, окремих рівнів та інших цілей. Кожен файл сцени можна вважати окремим ігровим рівнем. У кожній сцені можна розмістити різні об'єкти оточення, загородження, декорації.

Вікно Scene – це інтерактивна пісочниця (рис. 5). Ви будете використовувати Scene View для вибору та розташування оточень, гравця, камери та всіх інших ігрових об'єктів.

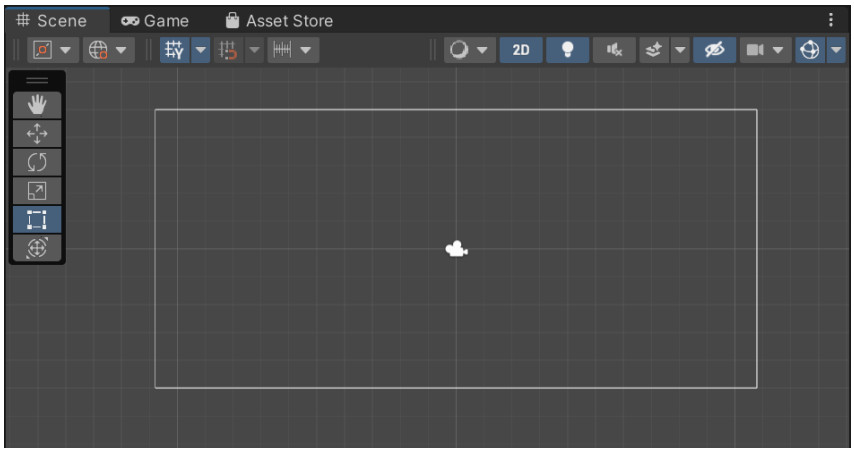

Рис. 5 – Вікно Scene

Вікно Hierarchy – це ієрархічне текстове представлення кожного об’єкта (GameObject) поточної сцени (рис. 6). Кожен елемент у сцені має запис в ієрархії, тому два вікна за своєю суттю пов’язані. При додаванні та видаленні об'єктів у сцені вони також будуть з'являтися та зникати з Ієрархії. Ієрархія розкриває структуру того, як об’єкти прикріплені один до одного.

Об’єкти можуть бути типів asset або prefab. У Unity asset та prefab – це два різних типи ресурсів, які використовуються для створення ігрового вмісту.

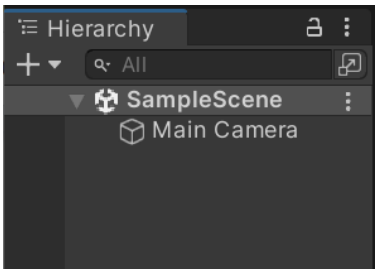

Рис. 6 – Вікно Hierarchy

Asset – це будь-який ресурс, що використовується в проекті для створення ігрового контенту, такий як моделі 3D об'єктів, текстури, скрипти, звукові ефекти, анімації, матеріали, плагіни та інші елементи. Asset може бути використаний в багатьох місцях проекту та зберігається в окремих файлах з розширенням *.asset або в папках проекту.

Prefab – це префаб, або «префабрикат», готовий об'єкт або набір об'єктів, який містить в собі всі необхідні компоненти, налаштування та дочірні об'єкти. Prefab можна розглядати як шаблон або макет, який може бути використаний для створення копій об'єкта в будь-якому місці проекту. Кожен екземпляр префаба може бути змінений незалежно від оригіналу, що забезпечує зручне управління проектом.

Панелі інструментів Toolbar (рис. 7) та Tools (рис. 8) забезпечують доступ до найважливіших робочих функцій. Кнопки ліворуч надають доступ до хмарних служб Unity та облікового запису Unity. У центрі знаходяться елементи керування відтворенням, паузою та кроками. Праворуч міститься меню макета редактора, яке надає кілька альтернативних макетів для вікон редактора та дозволяє зберігати власні макети. На панелі Tools містяться основні інструменти для керування видом сцени та об’єктами в ньому.

Рис. 7 – Вікно Toolbar

Рис. 8 – Вікно Tools

Вікно Inspector дозволяє переглядати та редагувати всі властивості поточного вибраного об’єкта (рис. 9). Оскільки різні типи об’єктів мають різні набори властивостей, макет і вміст вікна інспектора відрізнятимуться.

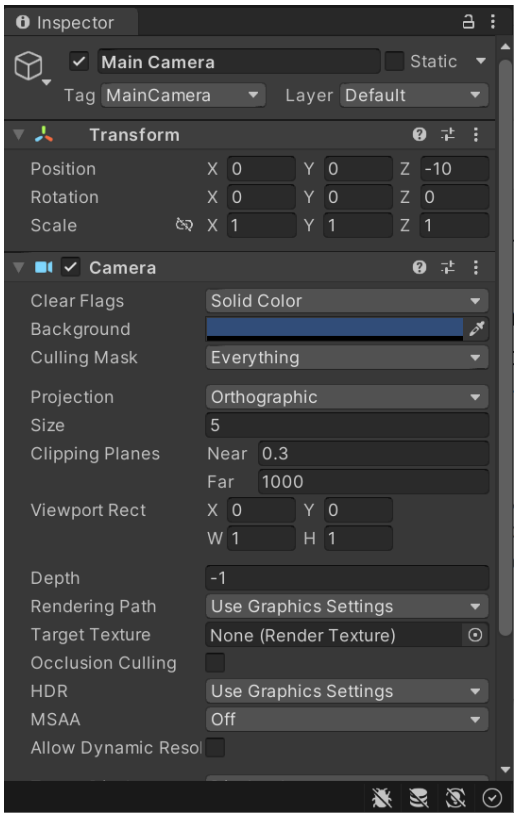

Рис. 9 – Вікно Inspector

Game View відрисовується з камери (камер) у грі (рис. 10). Це відображення фінальної, опублікованої гри. Вам потрібно буде використовувати одну або більше Cameras для контролю того, що гравець

фактично побачить коли гратиме у вашу гру. Game View для порожнього проекту відображає тільки блакитний фон за замовчуванням.

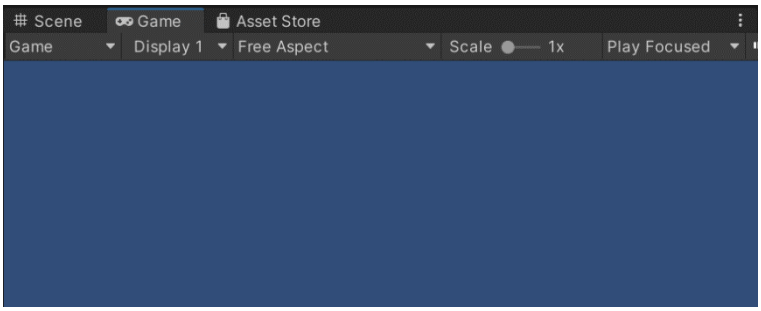

Рис. 10 – Вікно Game View

Для зручності, налаштуйте паралельне відображення вікон Scene та Game (рис. 11). Для цього натисніть на заголовок вікна Game і перетягніть його в праву частину вікна Scene. Так як гра передбачена в першу чергу для мобільних пристроїв, створіть та встановіть відображення в вертикальному Full HD (оберіть існуючий варіант або створіть свій).

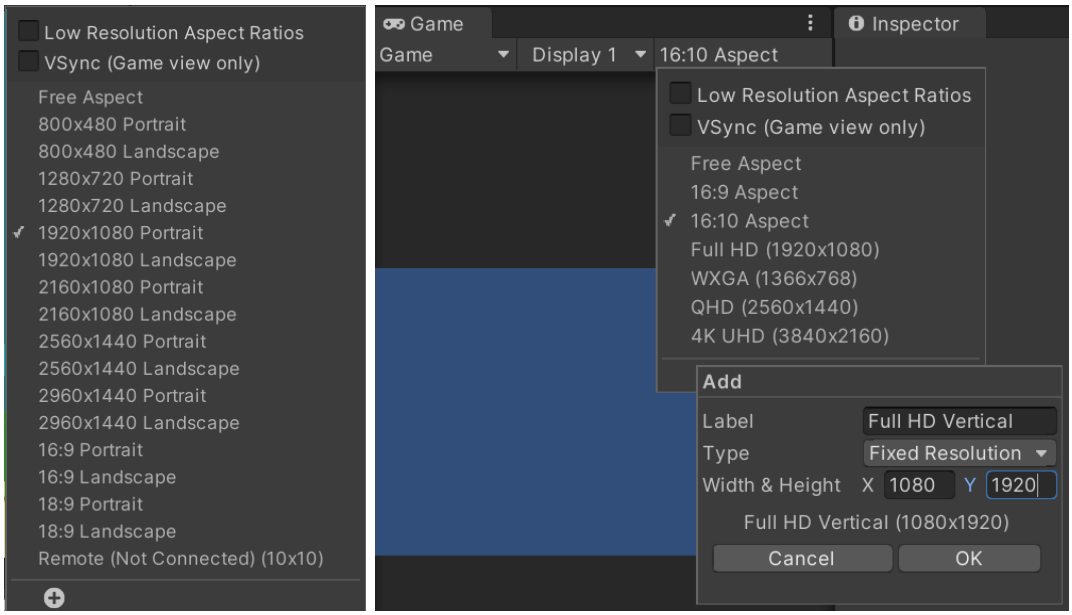

Рис. 11 – Вікно налаштування Game View

Налаштування сцени

Клацніть правою кнопкою миші у вікні Assets та створіть нову папку за допомогою команди Create > Folder. Назвіть її Sprites. Sprites в Unity – це двовимірні графічні елементи, які можуть бути використані для створення двовимірної графіки в іграх.

Додайте до папки необхідні для гри спрайти (рис. 12). Можна використовувати зображення, надані викладачем, або завантажити власні.

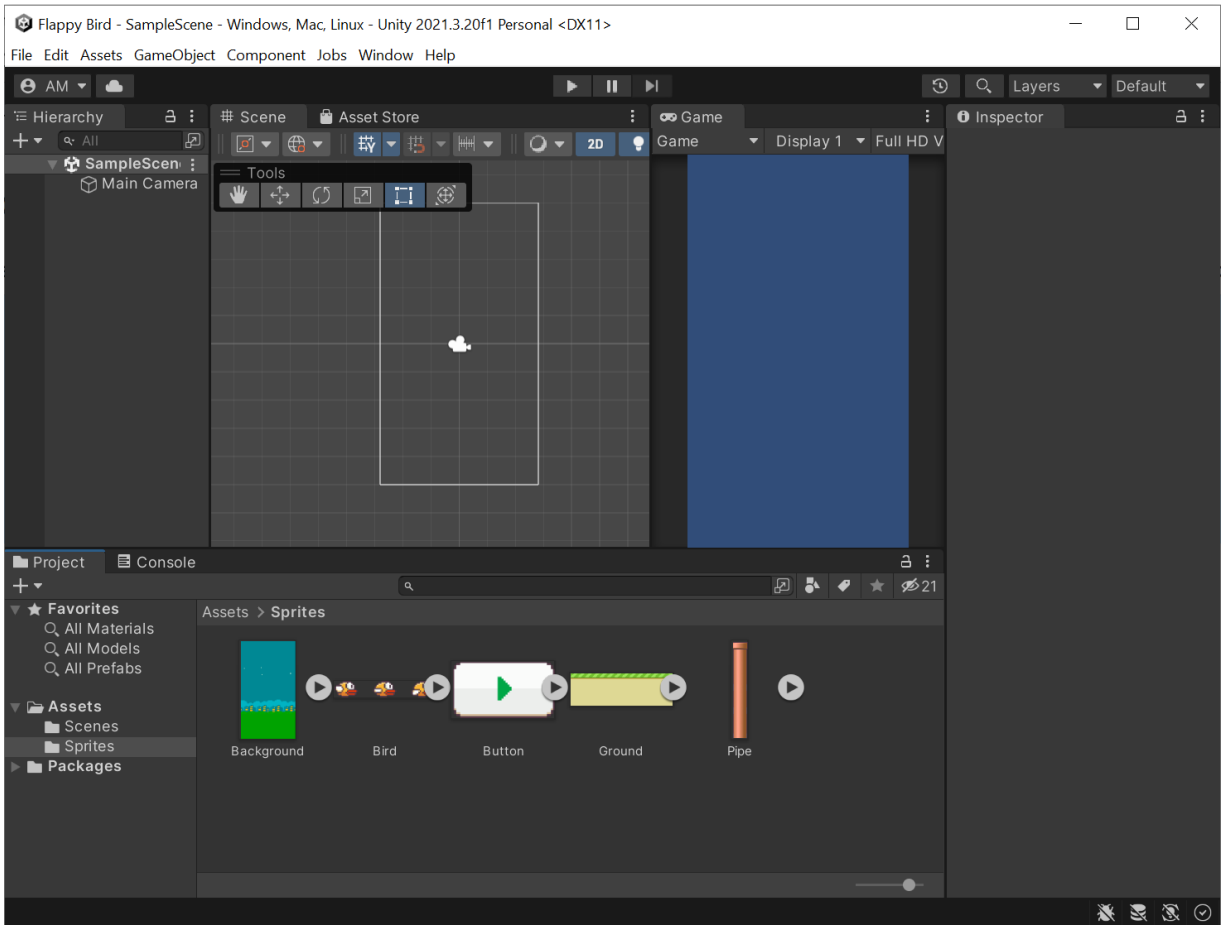

Рис. 12 – Додавання до папки необхідних спрайтів

Перетягніть файл Background у вікно Scene. Справа відкриється вікно налаштувань. В розділі Sprite Renderer в параметрі Order in Layer вкажіть -3. Це означає, що даний спрайт буде відображатись на фоні і не буде

перекривати інші ігрові елементи. Щоб вирівняти позицію зображення, в розділі Transform > Position задайте 0 для всіх вісей.

Таким самим чином додайте до сцени зображення землі (файл Ground). Order in Layer для цього спрайту = -1. Вкажіть позицію x=0.5, y=-4.5. Це розмістить шар з невеликим зміщенням вниз і ліворуч. Натисніть Add Component та додайте компонент Box Collider 2D (введіть назву в рядку пошуку або знайдіть компонент через Physics 2D > Box Collider 2D).

Box Collider 2D – це компонент, що дозволяє визначити границі зіткнення для 2D об'єктів в грі. Він допомагає забезпечити правильну взаємодію об'єктів в грі і дозволяє створювати реалістичну ігрову фізику. В даному випадку цей компонент потрібний для майбутньої взаємодії птаха з землею.

Натисніть на спрайт Bird. У вікні Inspector змініть Sprite Mode на Multiple. Натисніть Apply. Sprite Mode Multiple означає, що спрайт складається з декількох під-спрайтів, які можна використовувати окремо або в поєднанні. Кожен під-спрайт може бути розміщений в різних місцях на основному спрайті, що дозволяє зручно розміщувати і анімувати різні елементи графіки.

При використанні спрайтів з режимом Multiple необхідно також вказати налаштування для розміщення і розміру кожного під-спрайту. Натисніть Sprite Editor (рис. 13). У вікні, що відкриється, можна редагувати кожен під-спрайт окремо, вказавши його розташування та розміри в пікселях. Після цього кожен під-спрайт можна буде використовувати окремо, вказавши його ім'я в коді або в іншому компоненті Unity, який використовує спрайти.

У верхньому рядку натисніть Slice і ще раз підтвердіть Slice. Натисніть Apply.

Перетягніть спрайт Bird на сцену, щоб автоматично створити анімацію птаха. Створіть у папці Assets папку Animations та збережіть туди анімацію. Для чистоти проекту у вікні Hierarchy переназвіть об'єкт Bird_0 на Bird. Вкажіть позицію птаха x=-1, y=1. Натисніть Add Component та додайте

компонент Rigidbody 2D (введіть назву в рядку пошуку або знайдіть компонент через Physics 2D > Rigidbody 2D).

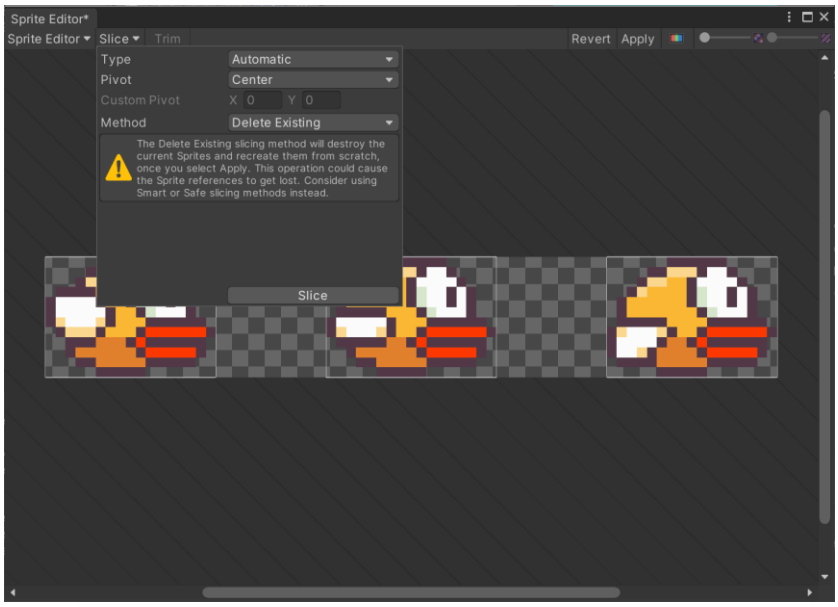

Рис. 13 – Використання Sprite Editor

Rigidbody 2D є компонентом фізичного рушія Unity, який дозволяє об'єктам взаємодіяти з іншими об'єктами в сцені за допомогою реалістичної фізики. Він дозволяє керувати рухом та поведінкою об'єктів в залежності від фізичних законів, таких як гравітація, тертя, зіткнення і т.д.

В налаштуваннях в розділі Constraints відмітьте галочками Freeze Position (x) та Freeze Rotation (z). Це заблокує рух об'єкта в певних напрямках: Freeze Position (x) означає, що об'єкт не може рухатися вздовж осі X; аналогічно Freeze Rotation (z) означає, що об'єкт не може обертатися навколо осі Z.

Додайте ще один компонент – Circle Collider 2D. Circle Collider 2D є компонентом фізичного рушія Unity, який дозволяє створювати колізії на об'єктах з формою кола в 2D-світі. Він дозволяє об'єктам взаємодіяти з іншими об'єктами в сцені з урахуванням їх форми та розмірів. Коли два об'єкти з компонентами Collider 2D зіткнуться, фізичний рушій Unity

визначає точку зіткнення та напрямок взаємодії між ними. Це дозволяє об'єктам взаємодіяти реалістично, з урахуванням їх форми та розміру, і може бути корисним для створення різноманітних ігрових ефектів, таких як зіткнення, рух та взаємодія між об'єктами в грі. В даному випадку Circle Collider 2D необхідний для основної логіки гри – відстежування зіткнення птаха з іншими ігровими об'єктами (трубами або землею).

Двічі клацніть на Bird у вікні Hierarchy, щоб наблизитись до цього об'єкта на сцені та побачити межі Circle Collider 2D. Встановіть радіус Circle Collider 2D, щоб він якомога точніше охоплював птаха (близько 0.25-0.3).

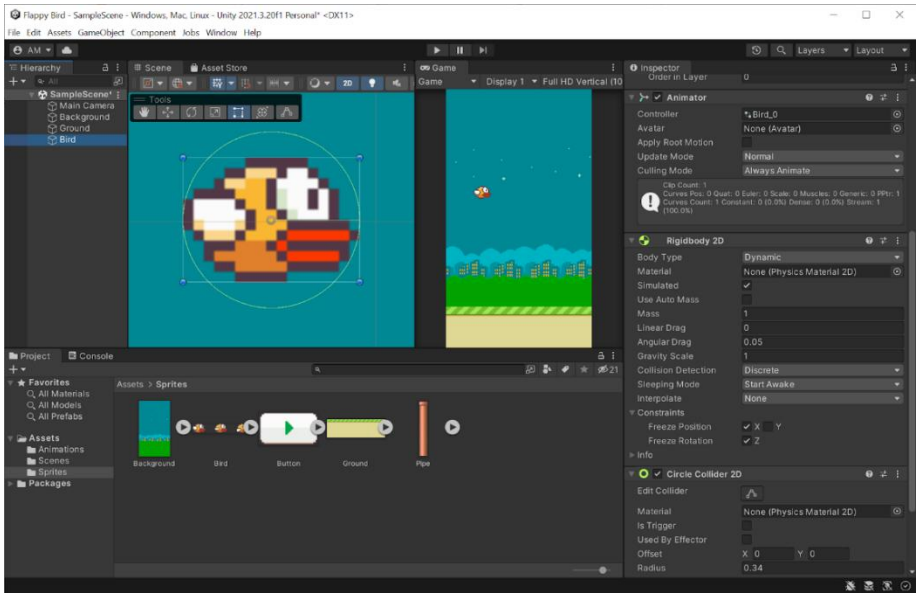

Рис. 14 – Встановлення радіусу Circle Collider 2D

Для перевірки правильності всіх налаштувань запустіть гру. Для цього натисніть на значок трикутника на панелі Toolbar. Птах повинен мати анімацію (махати крилами) та впасти на землю під дією ігрової гравітації. Ще раз натисніть на значок трикутника, щоб зупинити гру.

Управління головним об’єктом

Щоб керувати різними аспектами поведінки ігрових об'єктів, такі як рух, анімація, зіткнення та інші, в Unity можна створювати скрипти мовою C#. Скрипти на C# можуть містити різноманітний код, що дозволяє керувати ігровими об'єктами. Наприклад, можна використовувати методи, які виконують рух об'єкта, змінюють його позицію та орієнтацію, змінюють його параметри, такі як швидкість, масштаб, поворот та інші. Також можна додавати логіку до скрипту, щоб керувати об'єктом в залежності від дій гравця або різних умов у грі.

Наприклад, скрипт на C# може використовувати Input клас, щоб отримати інформацію про натискання клавіш гравцем, і використовувати її для керування рухом об'єкта. Або скрипт може використовувати Physics 2D клас, щоб керувати зіткненнями між об'єктами та іншими фізичними властивостями у грі. Загалом, використання C# скриптів дозволяє створювати складні та динамічні ігри з великою кількістю інтерактивності та деталізацією.

В папці Assets створіть папку Scripts. В ній створіть файл C# Script та назвіть його Bird. Додайте його до об'єкта Bird на сцені. Для цього натисніть на Bird у вікні Hierarchy і перетягніть створений файл скрипта з вікна Project до вікна параметрів Inspector. Двічі клацніть на назву скрипта, щоб відкрити його. Буде запущена встановлена версія Visual Studio.

Код скрипта Bird.cs:

```
using System;
using UnityEngine;
public class Bird : MonoBehaviour
{
    [SerializeField] private Rigidbody2D _rigidbody;
    [SerializeField] private float _force;
    [SerializeField] private float _yBound;
    private void Update()
```

```
    {
        if (Input.GetMouseButtonDown(0) && _rigidbody.position.y < _yBound)
        {
            Flap();
        }
    }
    private void Flap()
    {
        _rigidbody.velocity = Vector2.zero;
        _rigidbody.AddForce(Vector2.up * _force);
    }
}
```

Це клас, який представляє об'єкт птаха, яким можна керувати за допомогою миші.

Клас містить три серіалізованих поля (SerializeField):

– Rigidbody2D _rigidbody, яке вказує на компонент Rigidbody2D, який прикріплений до GameObject на сцені;

– float _force, що відповідає силі польоту;

– float _yBound, що вказує на максимальну висоту польоту.

Серіалізовані поля можуть бути використані для конфігурації гри, дозволяючи розробникам змінювати її параметри без необхідності редагування коду (через параметри об'єктів у вікні Inspector).

Rigidbody2D.position – це вектор, який визначає позицію центра мас об'єкта з фізикою на сцені. Таким чином запис _rigidbody.position.y вказує на координату Y об'єкта _rigidbody (на якій він висоті).

Rigidbody2D.velocity – це вектор швидкості об'єкта. Цей вектор визначає напрямок та швидкість руху об'єкта на сцені. При встановленні значення velocity = Vector2.zero вектор швидкості об'єкта встановлюється у (0, 0), тобто об'єкт зупиняється. Метод AddForce додає фізичну силу до об'єкту Rigidbody2D у вказаному напрямку.

У методі Update() перевіряється, чи натиснуто ліву кнопку миші (Input.GetMouseButtonDown(0)) та чи координата Y птаха менше ніж _yBound. Якщо ці умови виконуються, викликається метод Flap().

Метод Flap() скидає швидкість птаха та додає силу в напрямку вгору, щоб птах підлітав вище.

Збережіть скрипт. Можна закрити Visual Studio або просто повернутися до Unity Editor. Змінений скрипт автоматично оновиться. У вікні Inspector об’єкта Bird в компоненті скрипта тепер відображаються серіалізовані поля (рис. 15). Вкажіть для них значення. Для цього перетягніть компонент птаха Rigidbody 2D на поле Rigidbody, а в полях Force та Y Bound задайте відповідні значення. Наприклад, Force можна вказати 250, а Y Bound - 4.2.

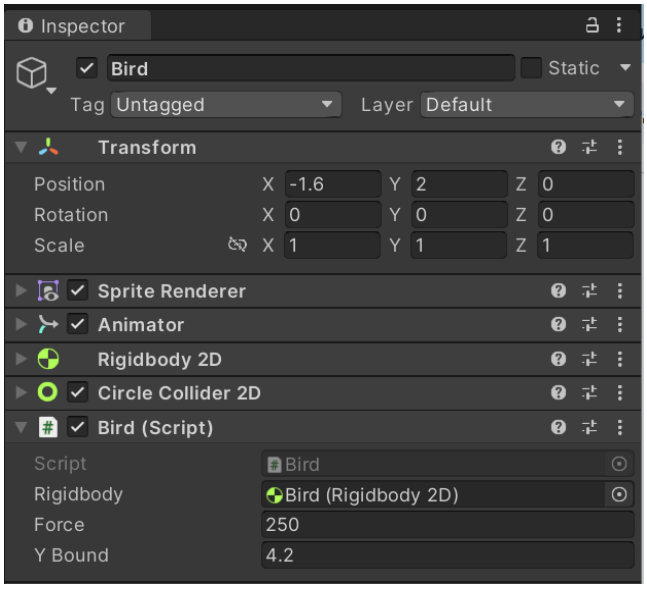

Рис. 15 – Відображення серіалізації поля об’єкта Bird у вікні Inspector

Запустіть гру та перевірте управління. При натисканні лівої кнопки миші птах повинен підлітати вгору, але не вище заданої границі (верхнього краю екрана). По експериментуйте з різними значеннями та оберіть ті, що підходять найкраще.

Додавання елементів перешкоджання головному об’єкту

Для реалізації гри необхідно додати перешкоди для птаха у вигляді труб, між якими він повинен буде пролетіти.

Натисніть правою кнопкою миші у вікні Hierarchy та створіть новий пустий об'єкт за допомогою пункта Create Empty. Назвіть його PipeGate. Перетягніть з вікна Project з папки Sprites об'єкт Pipe на новостворений об'єкт у вікні Hierarchy (рис. 16).

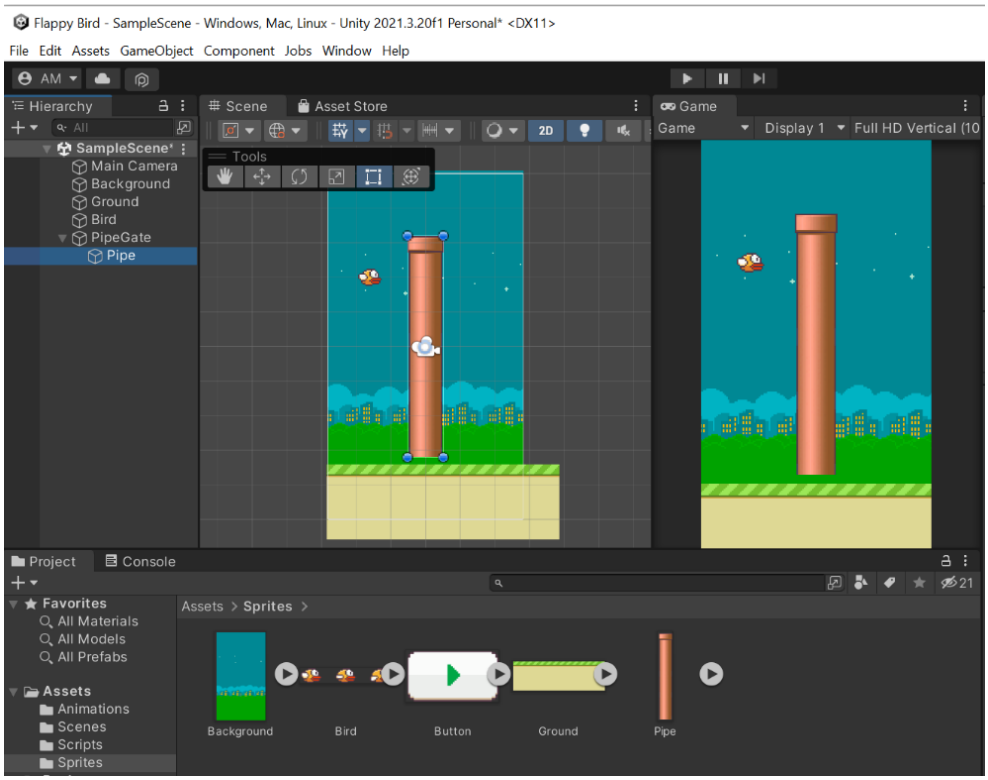

Рис. 16 – Додавання нового об’єкту PipeGate

У вікні Inspector вкажіть для нього позицію Y = -4, а значення Order in Layer = -2, щоб труба знаходилася внизу і за шаром землі. Також додайте до труби новий компонент - Box Collider 2D.

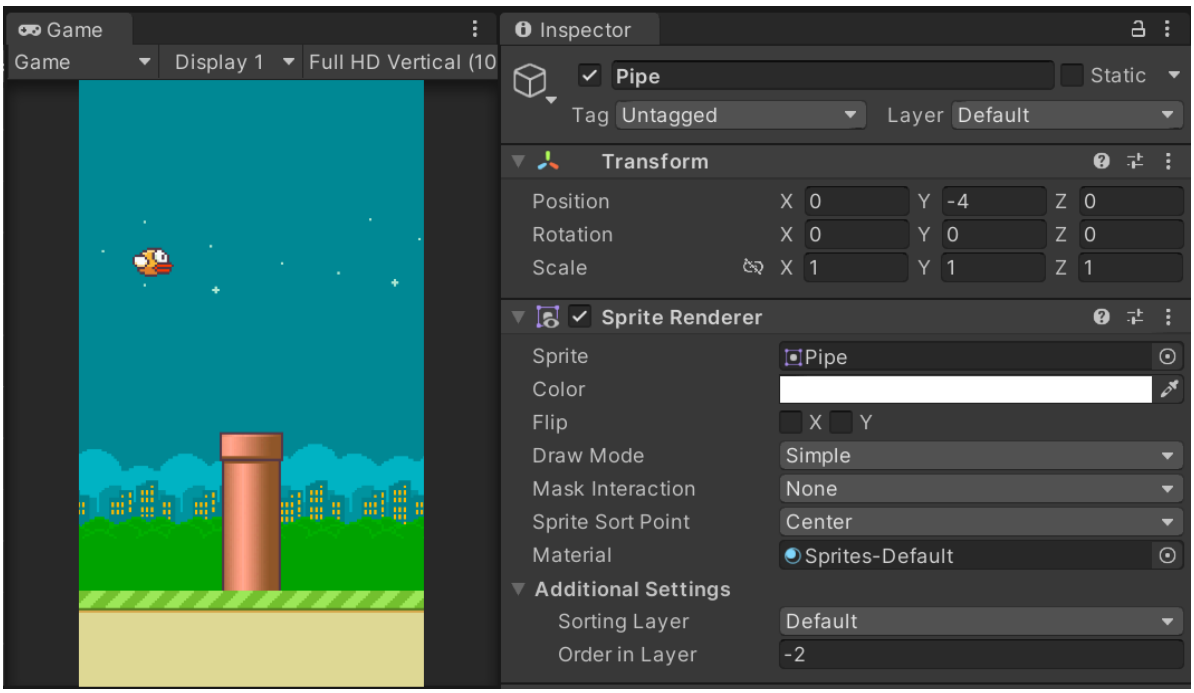

Рис. 17 – Налаштування нового об’єкту PipeGate у вікні Inspector

Дублюйте об'єкт труби. У вікні Hierarchy натисніть на Pipe та натисніть Ctrl+D або натисніть правою кнопкою миші та оберіть Duplicate. Буде створений об'єкт Pipe (1). Вкажіть для нової труби значення Y = 4.5 та Scale Y = -1. В результаті труба буде розміщуватись вгорі та буде перевернута необхідним кінцем донизу.

Переназвіть Pipe у LowerPipe, а Pipe (1) в UpperPipe.

Створіть ще один порожній об'єкт всередині PipeGate. Назвіть його Trigger. Тригери – це компоненти, які дозволяють об'єкту проходити через інший об'єкт без спрацьовування фізичного стику. Тригери використовуються для спрацьовування подій, коли об'єкти зіштовхуються або наближаються один до одного (рис. 18).

В даному випадку тригер необхідний для визначення того, чи пролетів птах між трубами. Додайте до нього компонент Box Collider 2D. Встановіть галочку в параметрі Is Trigger. Вкажіть значення Offset Y = 0.25, щоб тригер розмістився по центру між трубами та задайте розміри Size X = 0.5, Y = 2.

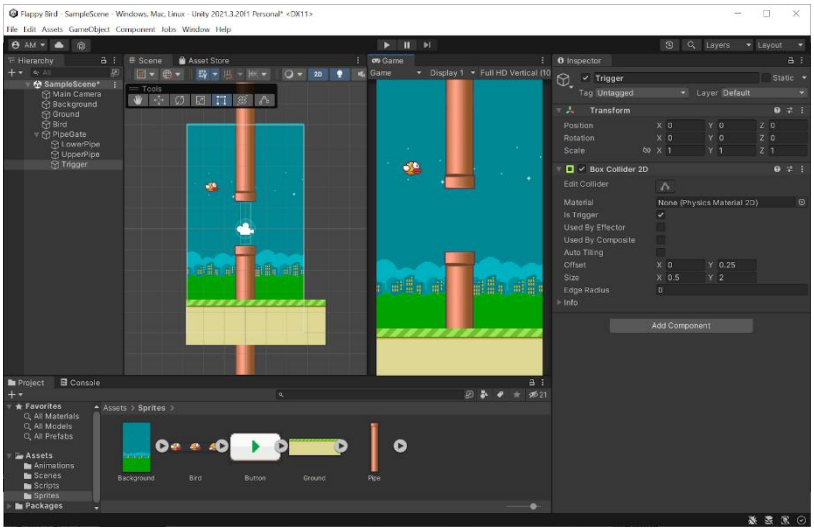

Рис. 18 – Вікно створення Trigger

Створіть нову папку в Assets та назвіть її Prefabs (рис. 19). Перетягніть до неї весь об'єкт PipeGate. Тепер цей об'єкт став префабом. Prefab – це готовий шаблон об'єкта, який містить всі компоненти, властивості та налаштування, необхідні для створення екземплярів цього об'єкта в грі.

Після цього можна видалити PipeGate у вікні Hierarchy, адже у грі ми будемо створювати перешкоди у вигляді труб не вручну, а програмно.

Також зробіть префабом об'єкт Ground – перетягніть його з вікна Hierarchy до папки Prefabs, але не видаляйте зі сцени.

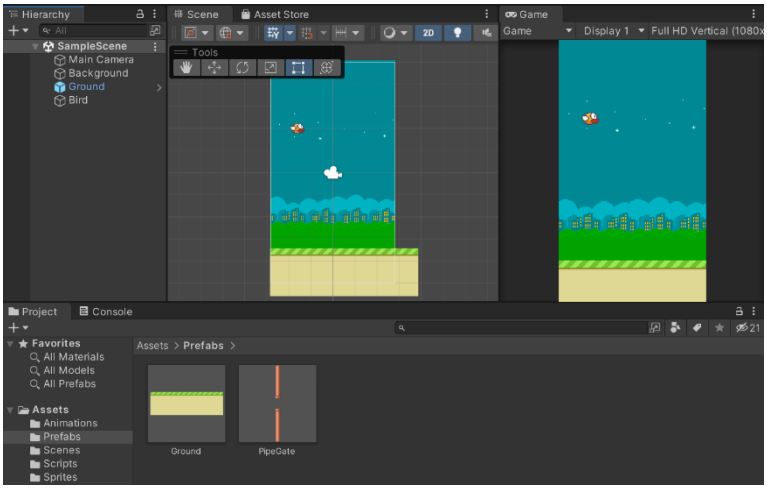

Рис. 19 – Вікно створення Prefabs

У папці Scripts створіть новий C# скрипт та назвіть його MovingObject. Двічі клацніть на ньому, щоб відкрити файл в Visual Studio.

В серіалізованих полях визначені дві змінні типу float – швидкість (_speed) і кордон по осі X (_xBound).

У функції Update() виконується переміщення об'єкту вліво на відстань _speed * Time.deltaTime, де Time.deltaTime – це час, який пройшов з моменту останнього оновлення фрейму.

Якщо координата по осі X цього об'єкту стає меншою за кордон _xBound, то об'єкт знищується за допомогою функції Destroy (this.gameObject).

Код скрипта MovingObject.cs:

```
using UnityEngine;
public class MovingObject : MonoBehaviour
{
    [SerializeField] private float _speed;
    [SerializeField] private float _xBound;
    private void Update()
    {
        this.transform.position -= Vector3.right * _speed * 
Time.deltaTime;
        if (this.transform.position.x < _xBound)
        {
            Destroy(this.gameObject);
        }
    }
}
```

Цей скрипт буде використовуватися для реалізації руху землі та труб в грі, які будуть переміщуватися вліво та зникати, коли будуть досягати певного кордону.

Збережіть скрипт та поверніться в Unity Editor. Перейдіть в папку Prefabs та відкрийте властивості префабу PipeGate, натиснувши на нього лівою кнопкою миші. Додайте компонент MovingObject (знайдіть через рядок пошуку або Scripts > MovingObject). Встановіть Speed = 1.5, а X Bound = -8.

Також додайте цей скрипт до префабу Ground. Можна повторити ті самі дії, як для префабу PipeGate, або скопіювати за допомогою команди Copy Component і вставити через Paste Component As New.

Для перевірки роботи скрипта перетягніть префаб PipeGate до вікна Scene та запустіть гру (рис. 20).

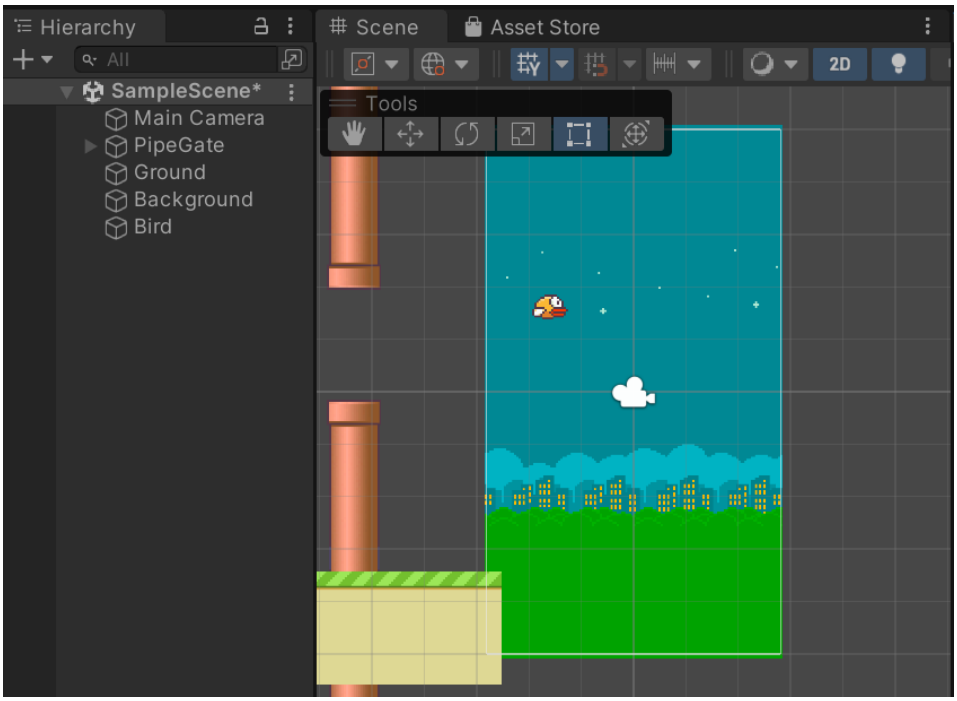

Рис. 20 – Перевірка роботи гри

Труби і земля рухаються і потім зникають, коли виходять з ігрової сцени. Знову видаліть PipeGate зі сцени (з вікна Hierarchy).

Генерація елементів перешкоджання та земельних ділянок

В папці Scripts створіть новий скрипт C# та назвіть його Spawner. Він буде відповідати за програмну нескінченну генерацію нових ділянок землі та перешкод у вигляді труб з різним зміщенням по вертикалі.

У цьому класі використовуються наступні поля:

– _prefab – префаб, з якого будуть створюватися інстанси (унікальні екземпляри префабу).

– _time – час між кожним створенням інстансу.

– _yClamp – максимальне значення зміщення, яке використовується для зміни висоти створення перешкод.

Код скрипта Spawner.cs:

```
using System;
using UnityEngine;
public class Spawner : MonoBehaviour
{
    [SerializeField] private GameObject _prefab;
    [SerializeField] private float _time;
    [SerializeField] private float _yClamp;
    private float _elapsedTime;
    private void Update()
    {
        _elapsedTime += Time.deltaTime;
        if (_elapsedTime > _time)
        {
            SpawnObject();
            _elapsedTime = 0f;
        }
    }
    private void SpawnObject()
    {
        float offsetY = UnityEngine.Random.Range(-_yClamp, _yClamp);
        Vector2 pos = new Vector2 (this.transform.position.x,
            this.transform.position.y + offsetY);
```

```
        Instantiate(_prefab, pos, Quaternion.identity, this.transform);
    }
}
```

Метод Update() викликається кожний кадр і перевіряє, чи пройшов достатній час для створення наступного інстансу. Якщо так, викликається метод SpawnObject ().

Метод SpawnObject() використовується для створення нового інстансу префабу _prefab. Він генерує випадкове зміщення по y-координаті в діапазоні від –_yClamp до _yClamp, щоб змінювати висоту, на якій будуть створюватися перешкоди.

У вікні Hierarchy створіть новий порожній об’єкт (Create Empty) та назвіть його PipeSpawner (рис. 21). Встановіть для нього позицію X = 5, Y = 0, Z = 0. Додайте компонент Spawner. Щоб вказати префаб, натисніть кнопку Обрати (круг з точкою посередині) в кінці рядка Prefab та у вікні, що відкрилося, перейдіть до вкладки Assets. Оберіть префаб PipeGate.

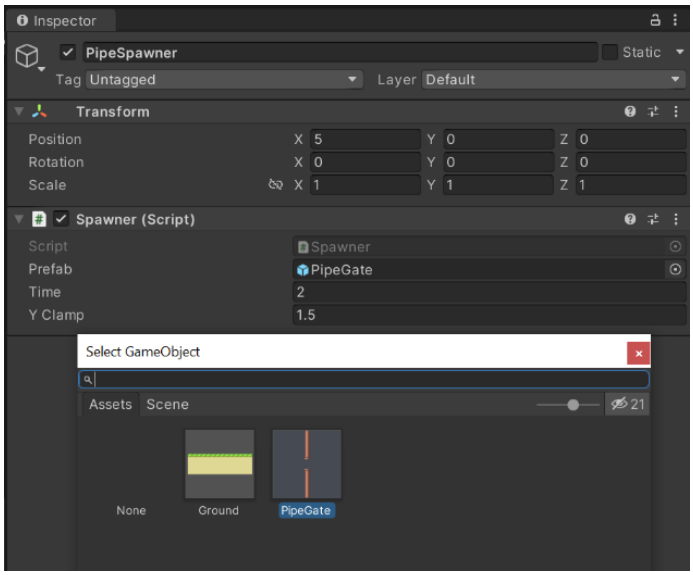

Рис. 21 – Налаштування об’єкта PipeSpawner

Встановіть Time = 2, Y Clamp = 1.5. Це означає, що кожні 2 секунди буде генеруватись нова перешкода у вигляді пари труб з випадковим зміщенням від 0 до 1.5 по осі Y.

Створіть ще один порожній об’єкт (Create Empty) та назвіть його GroundSpawner. Для нього вкажіть позицію X = 6.25, Y = -4.5, Z = 0 (координати центра ділянки землі, яка буде створена після вже існуючої на сцені). Додайте компонент Spawner. Оберіть префаб Ground та вкажіть Time = 4.45, Y Clamp = 0.

Перетягніть у вікні Hierarchy об'єкт Ground на GroundSpawner та дублюйте його. Для Ground (1) вкажіть Position X = 0. Переназвіть Ground у Ground_1, а Ground (1) в Ground_2 (рис. 22).

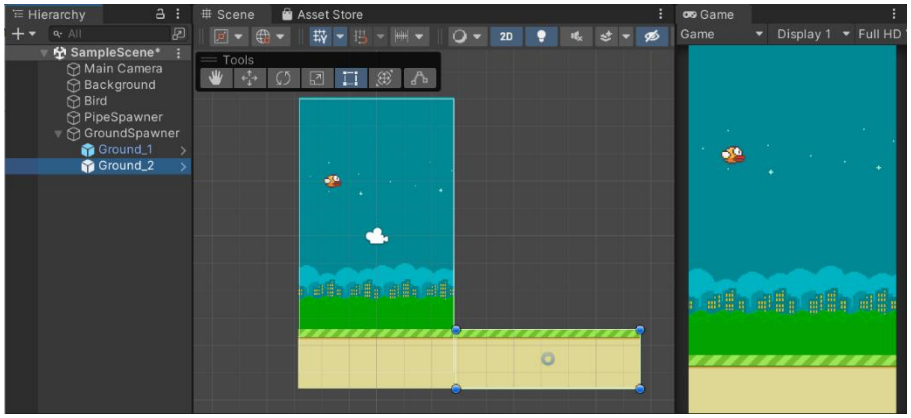

Рис. 22 – Відображення об’єктів Ground_1 та Ground_2

Запустіть гру (рис. 23). Ділянки землі і труби генеруються та видаляються безперервно.

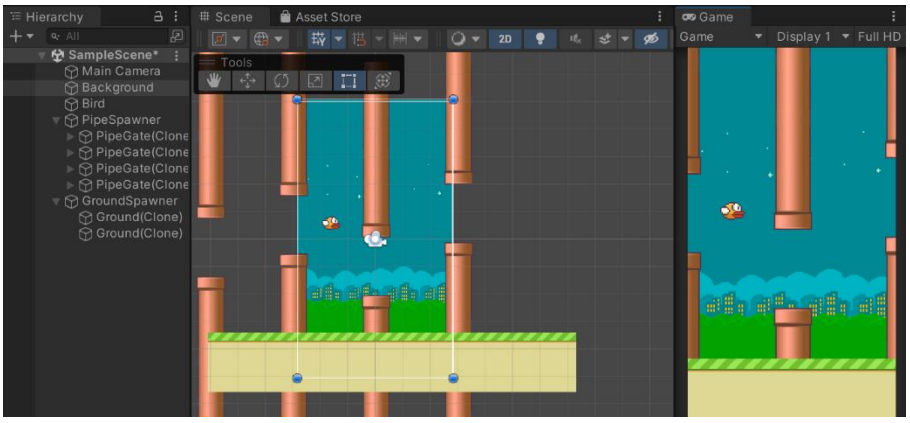

Рис. 23 – Перевірка роботи гри

Завершення етапу «Раунд гри»

Кінець гри наступає, коли птах торкається землі або труб. Щоб відстежити це у грі, необхідно доповнити код скрипта Bird.cs.

Додайте статичну подію:

```
public static event Action OnDeath;
```

Метод Start():

```
private void Start()
{
    Time.timeScale = 1f;
}
```

Метод OnCollisionEnter2D():

```
private void OnCollisionEnter2D()
{
    OnDeath?.Invoke();
    Time.timeScale = 0f;
}
```

Коли гравець запускає сцену в Unity, викликається метод Start() на всіх скриптах, які прикріплені до об’єктів у сцені. У цьому випадку, при запуску сцени, метод Start() встановлює значення швидкості часу на 1.0 за допомогою властивості Time.timeScale. Це означає, що ігровий час буде проходити зі стандартною швидкістю.

Метод OnCollisionEnter2D() викликається, коли об’єкти з компонентами Collider зіштовхуються. В даному випадку – коли птах зіштовхується з землею або трубою. Тоді встановлюється Time.timeScale на 0, що зупиняє рух гри. Також спрацьовує подія onDeath шляхом виклику методу Invoke(), який виконує всі підписані методи.

Шаблон розробки Observer є одним з найпоширеніших шаблонів для реалізації зв'язку між об'єктами програми. В Unity цей шаблон можна використовувати для реалізації підписки на події, що виникають в грі. Основна ідея шаблону полягає в тому, що єдиний об'єкт, який називається

видавцем (або суб'єктом), надсилає повідомлення про свої зміни всім підписникам (або спостерігачам), які на нього підписалися.

Знак питання в коді використовується для перевірки, чи onDeath не дорівнює null, перед тим, як викликати метод Invoke(). Це робить код більш безпечним, оскільки уникне виклику події, якщо на неї ніхто не підписаний.

Збережіть скрипт та поверніться до Unity Editor. У вікні Hierarchy створіть новий об’єкт UI > Legacy > Button. Це буде кнопка для запуску гри після її закінчення. У списку об’єктів з’явиться ієрархія Canvas > Button > Text та об’єкт EventSystem.

Canvas – це контейнер, який містить всі UI елементи і відповідає за їх розміщення та відображення на екрані. Можна налаштувати різні параметри Canvas, такі як розмір, шари, налаштування рендерингу, масштабування і т. д. Наприклад, можна налаштувати положення кнопки відносно верхнього краю Canvas, або змінювати колір тексту на текстовому полі. Крім того, Canvas дозволяє налаштовувати порядок рендерингу UI елементів. Наприклад, якщо є дві кнопки, одна з них повинна бути вище за іншу, можна налаштувати порядок їх рендерингу в межах Canvas.

EventSystem в Unity – це система, що дозволяє обробляти події (event) від UI елементів на сцені. Вона автоматично створюється разом з першим UI елементом, доданим до сцени. EventSystem відповідає за обробку введення користувача (такого як кліки миші або натискання клавіш на клавіатурі) від UI елементів і передачу цих подій відповідним компонентам UI. Наприклад, якщо користувач клікає на кнопку, EventSystem зберігає цю подію та передає її до компонента Button на цій кнопці. Потім компонент Button виконує відповідну дію, наприклад, викликає функцію, яка пов'язана з натисканням кнопки.

Видаліть елемент Text з кнопки, адже на ній буде тільки зображення без тексту. Оберіть Button та встановіть потрібні налаштування у вікні Inspector. Позиція X = 0, Y = 0, Z = 0. Scale X = 5, Y = 5. В розділі Image в

полі Source Image оберіть спрайт Button та встановіть галочку Preserve Aspect, щоб зберегти співвідношення сторін зображення (рис. 24).

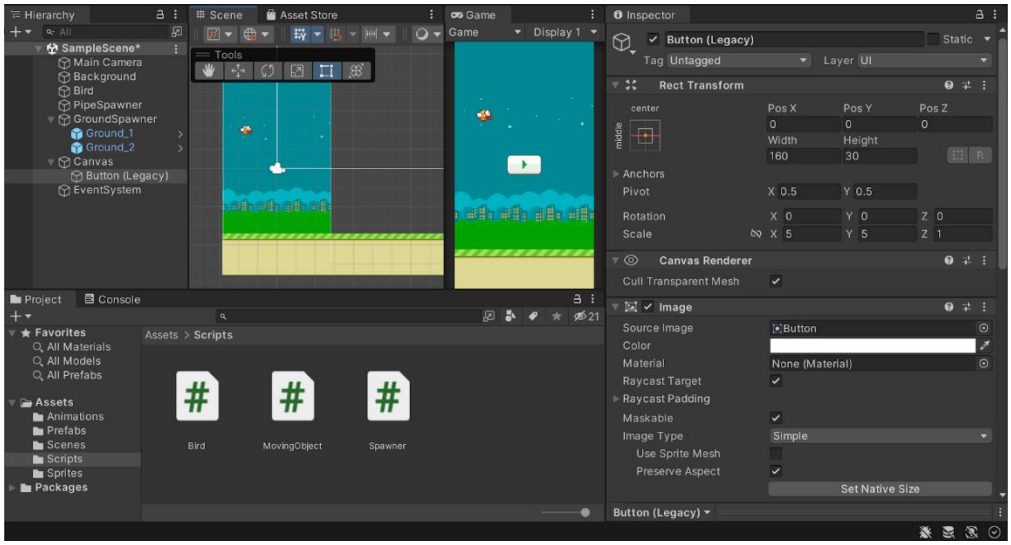

Рис. 24 – Налаштування компоненту Button

Відключіть відображення кнопки, знавши галочку біля назви об’єкта в Inspector-i.

Створіть новий скрипт UIManager та додайте його до об’єкту Canvas.

Код UIManager.cs:

```
using TMPro;
using UnityEngine.SceneManagement;
using UnityEngine;
public class UIManager: MonoBehaviour
{
    [SerializeField] private GameObject _playButton;
    private void Awake()
    {
        Bird.OnDeath += OnGameOver;
    }
    private void OnDestroy()
    {
        Bird.OnDeath -= OnGameOver;
    }
    public void RestartGame() => SceneManager.LoadScene
```

```
        (SceneManager.GetActiveScene().buildIndex);
    private void OnGameOver() => _playButton.SetActive(true);
}
```

Цей код представляє клас UIManager, який управляє інтерфейсом користувача гри. Він містить приватне поле _playButton, яке відповідає за посилання на ігрову кнопку Play.

Клас містить метод Awake(), який додає метод OnGameOver до події OnDeath в класі Bird. Це означає, що коли птах зіштовхнеться з перешкодою, UIManager викличе метод OnGameOver, який зробить ігрову кнопку Play видимою. Метод OnDestroy() видаляє метод OnGameOver з події OnDeath, щоб запобігти витоку пам'яті та невірній роботі програми.

Клас також містить публічний метод RestartGame(), який завантажує сцену знову, коли гравець натисне на ігрову кнопку Play. Оператор => визначає вираз-лямбда функцію, яка може бути використана для визначення коротких методів. В даному випадку, RestartGame() – це лямбда функція, яка замінює традиційний синтаксис методу.

Збережіть скрипт та поверніться до Unity Editor. Перетягніть Button (Legacy) на поле Play Button скрипта UIManager, який прикріплений до об’єкта Canvas. Перейдіть до параметрів Button. В розділі Button знайдіть налаштування OnClick() (рис. 25). Натисніть +, щоб вказати метод, який буде викликатися при натисканні на цю кнопку. В поле None (Object) перетягніть об’єкт Canvas та оберіть функцію UIManager > RestartGame().

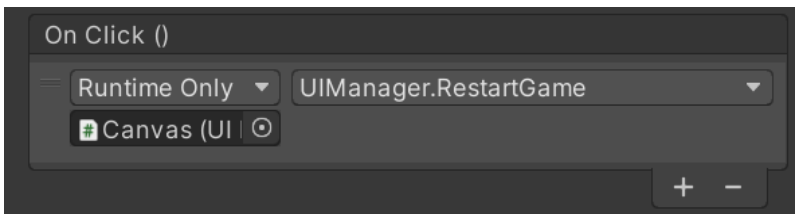

Рис. 25 – Налаштування методу OnClick() компоненту Button

Запустіть гру, щоб перевірити всі налаштування. Тепер при зіткненні птаха з перешкодою гра зупиняється і з’являється кнопка Play, при натисканні на яку гра починається спочатку.

Створення елементу «Рахунок гри»

Необхідно додати текстове поле до сцени, в якому буде відображатися поточний рахунок гри – кількість успішно пройдених перешкод.

Правою кнопкою миші клацніть на Canvas та створіть новий об’єкт UI > Text – TextMeshPro.

TextMeshPro – це пакет для Unity, призначений для створення високоякісного тексту в застосунках. Він є більш продвинутою альтернативою звичайному компоненту текста в Unity. TextMeshPro надає значно більше можливостей для налаштування вигляду тексту, включаючи різноманітні шрифти, розміри, кольори, тіні, градієнти, ефекти та багато іншого. Крім того, він підтримує відображення тексту з високою деталізацією та гладкість анімацій.

При першому використанні пакету TextMeshPro, необхідно імпортувати необхідні матеріали – натиснути Import TMP Essentials. Після завантаження вінко TMP Importer можна закрити.

У вікні Hierarchy з’явився об’єкт Text (TMP) (рис. 26). Переназвіть його на Score. У вікні Inspector вкажіть позицію X = 0, Y = 700, Z = 0 (у верхній частині екрану) та розміри: Width = 500, Height = 200. Замість тексту New Text впишіть 0. Зробіть текст жирним, розмір: 200 та вирівнювання по центру по вертикалі та горизонталі. Необхідно доповнити скрипт Bird.cs, щоб відображати в полі Score поточний рахунок гри.

Додайте подію onScore:

```
public static event Action OnScore;
```

Додайте метод OnTriggerEnter2D():

```
private void OnTriggerEnter2D()
{
    OnScore?.Invoke();
}
```

Він буде спрацьовувати, коли птах буде успішно пролітати між трубами – тобто при проході птаха через об’єкт Trigger між перешкодами.

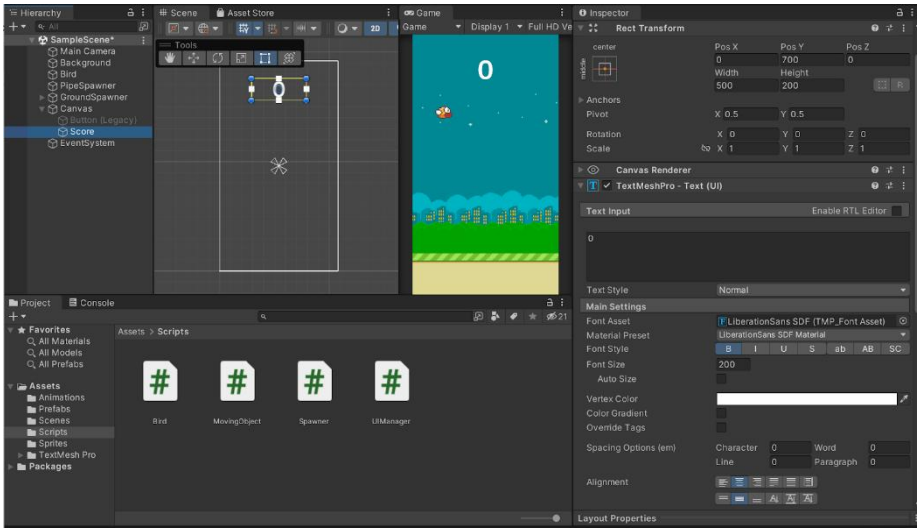

Рис. 26 – Налаштування об’єкта Score

Збережіть цей скрипт та відкрийте UIManager.cs.

Додайте поле _score, яке буде вказувати на елемент TextMeshPro:

```
[SerializeField] private TMP_Text _score;
```

В метод Awake() додайте рядок:

```
Bird.OnScore += OnScore;
```

і відповідний рядок в метод Destroy():

```
Bird.OnScore -= OnScore;
```

Створіть метод onScore():

```
private void OnScore() => _score.text = (int.Parse(_score.text) + 1).ToString();
```

Збережіть скрипт та поверніться до Unity Editor. Відкрийте параметри Canvas та перетягніть об’єкт Score у відповідне поле скрипта UIManager (рис. 27). Запустіть гру та перевірте роботу скриптів.

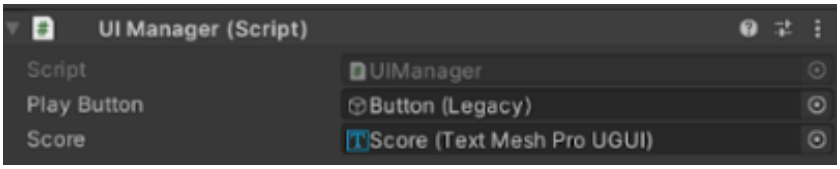

Рис. 27 – Вікно UI Manager (Script)

Створення звукового супроводу

Звукові ефекти в грі відіграють дуже важливу роль у створенні реалістичного та захоплюючого ігрового досвіду. Основна функція звукових ефектів полягає в тому, щоб забезпечити гравцям звукову інформацію про події, які відбуваються в грі, збільшити емоційну насиченість гри та створити певну атмосферу.

Створіть в папці Assets папку Audio. Додайте в неї звукові файли для озвучування підйому птаха, успішного проходження перешкод та кінця гри. Можна використовувати файли, надані викладачем: flap.ogg, score.ogg та hit.ogg.

Оберіть у вікні Hierarchy об'єкт птаха Bird та додайте до нього компонент Audio > Audio Source.

Відкрийте скрипт Bird.cs. Додайте чотири серіалізованих поля: компонент Audio Source та 3 звукових ефекта.

```
[SerializeField] private AudioSource _audioSource;
[SerializeField] private AudioClip _flapSound;
[SerializeField] private AudioClip _hitSound;
[SerializeField] private AudioClip _scoreSound;
```

В метод OnCollisionEnter2D() додайте рядок:

```
_audioSource.PlayOneShot(_hitSound);
```

В даному випадку метод PlayOneShot() використовується для відтворення звукового ефекту, який ми присвоїли змінній _hitSound, використовуючи AudioSource _audioSource.

В метод OnTriggerEnter2D() додайте рядок:

```
_audioSource.PlayOneShot(_scoreSound);
```

В метод Flap() додайте рядок:

```
_audioSource.PlayOneShot(_flapSound);
```

Збережіть скрипт та поверніться до Unity Editor. Перетягніть компонент Audio Source та файли звукових ефектів до відповідних полів скрипта (рис. 28).

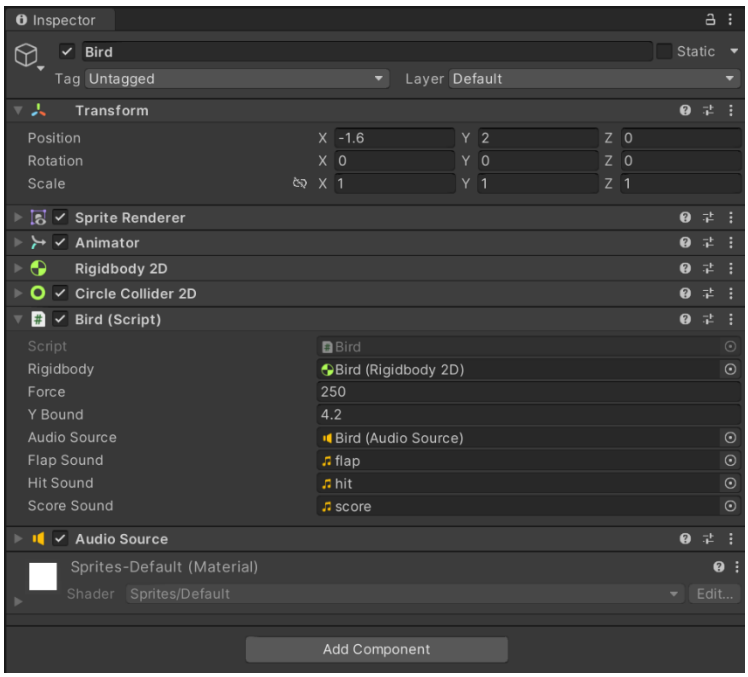

Рис. 28 – Налаштування об’єкта Bird у вікні Inspector

Запустіть гру та перевірте налаштування (рис. 29). Гра повністю готова.

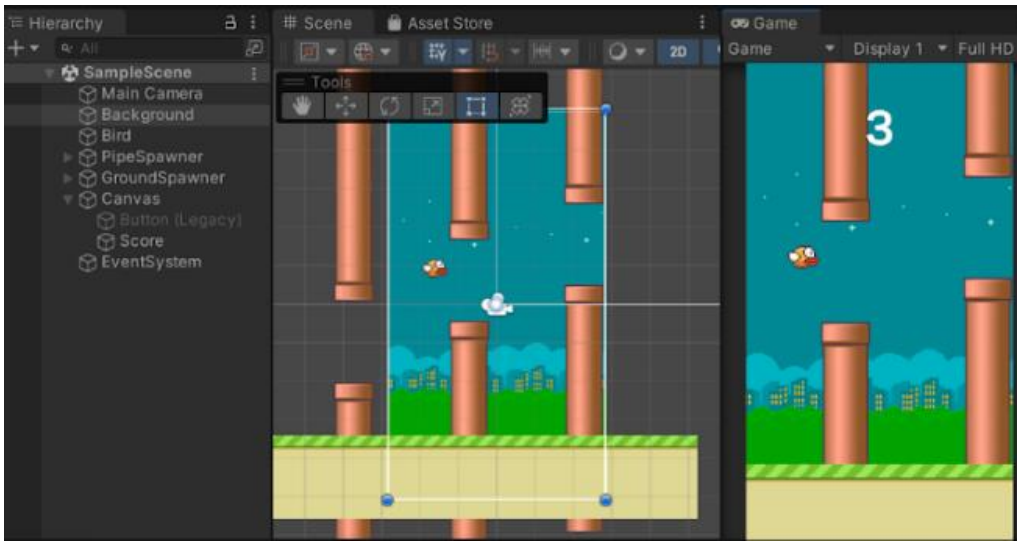

Рис. 29 – Перевірка роботи гри

Експортування проекту на ОС Android

Стандартно проект створюється для комп'ютерів з ОС Windows. Гра Flappy Bird більше підходить для мобільних пристроїв. Необхідно змінити платформу та зберегти гру для пристроїв з ОС Android.

Відкрийте Налаштування збірки через File > Build Settings (рис. 30).

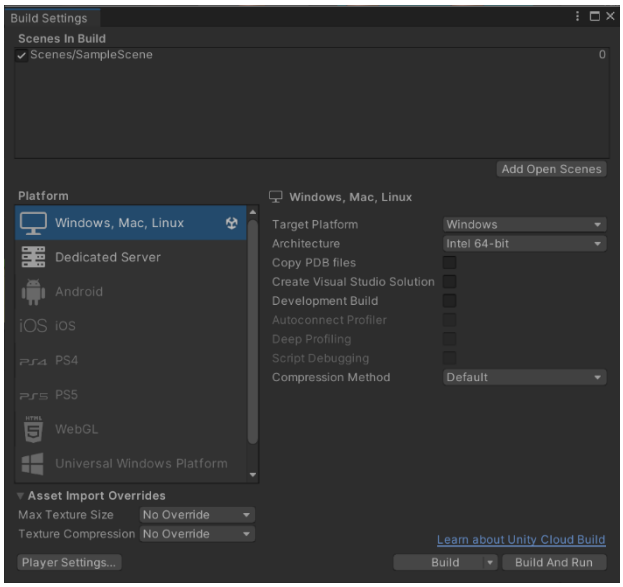

Рис. 30 – Вікно Build Settings

В цьому вікні можна обрати платформу для збереження проекту та встановити необхідні параметри. Якщо при встановленні Unity ви не обрали модуль для Android, в цьому вікні він буде відображатися неактивним (рис. 31).

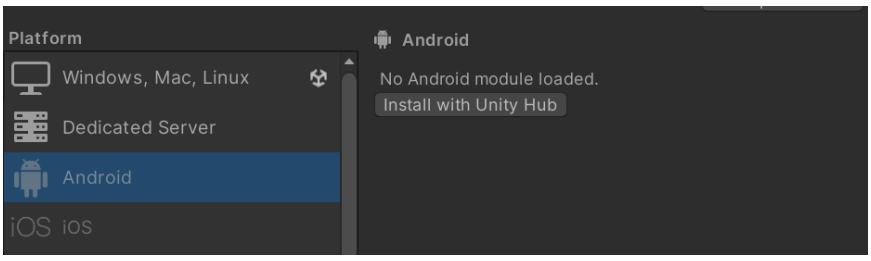

Рис. 31 – Вікно Build Settings, вкладка Android

В такому випадку натисніть на рядок Android та встановіть модуль зараз. Він може займати близько 5 Гб. Дочекайтеся завантаження та встановлення модуля (рис. 32).

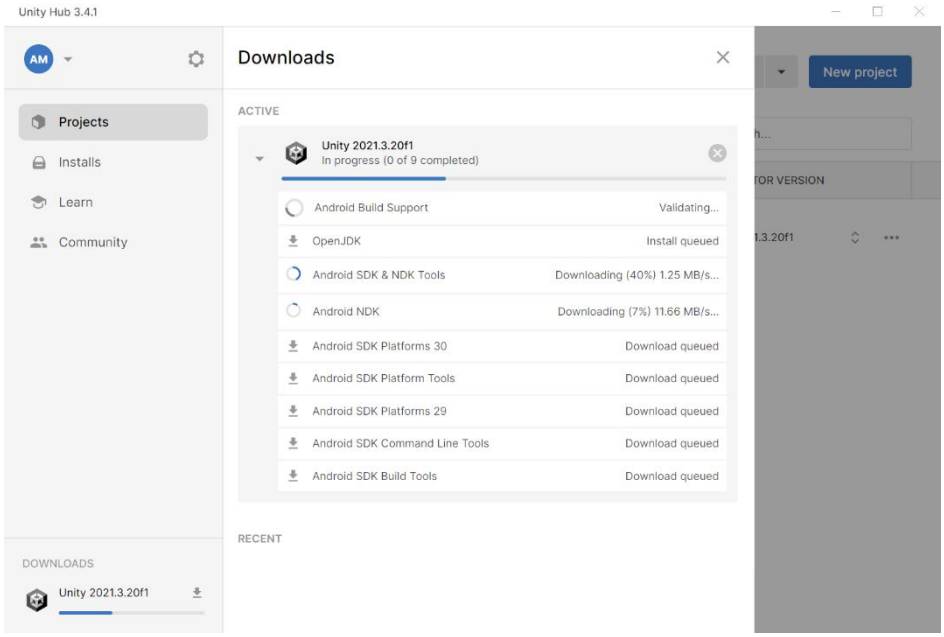

Рис. 32 – Встановлення додаткового модулю для Android

Після успішного завантаження перезапустіть Unity Editor та знову відкрийте File > Build Settings. Тепер платформа Android активна і при її виборі у правій частині відображаються доступні параметри для побудови проекту під Android. Натисніть Switch Platform, щоб переключитися на цю платформу. Це може зайняти деякий час (рис. 33).

Натисніть Player Settings в нижньому лівому куті вікна Build Settings. Player Settings – це вікно налаштувань, яке дозволяє настроїти параметри, пов'язані зі створенням готової гри, яку користувач може запустити на різних платформах.

У Player Settings можна встановити різні налаштування, такі як:

– Application Identifier (Ідентифікатор додатку) – унікальний ідентифікатор додатку, що використовується в магазинах застосунків.

– Company Name (Назва компанії) – назва компанії або студії.
– Product Name (Назва продукту) – назва гри або застосунку.
– Default Icon (Іконка за замовчуванням) – іконка, яка відображається на робочому столі та в меню застосунків на пристроях.
– Splash Image (Екран запуску) – зображення, яке відображається при запуску застосунку або гри.
– Graphics Settings (Налаштування графіки) – параметри, пов'язані зі створенням графічного вмісту.
– Scripting Backend (Система обробки скриптів) – тип системи обробки скриптів, яку буде використовувати застосунок або гру.
– Minimum API Level (Мінімальний рівень API) – мінімальна версія Android або iOS, на яких можна запустити застосунок або гру.

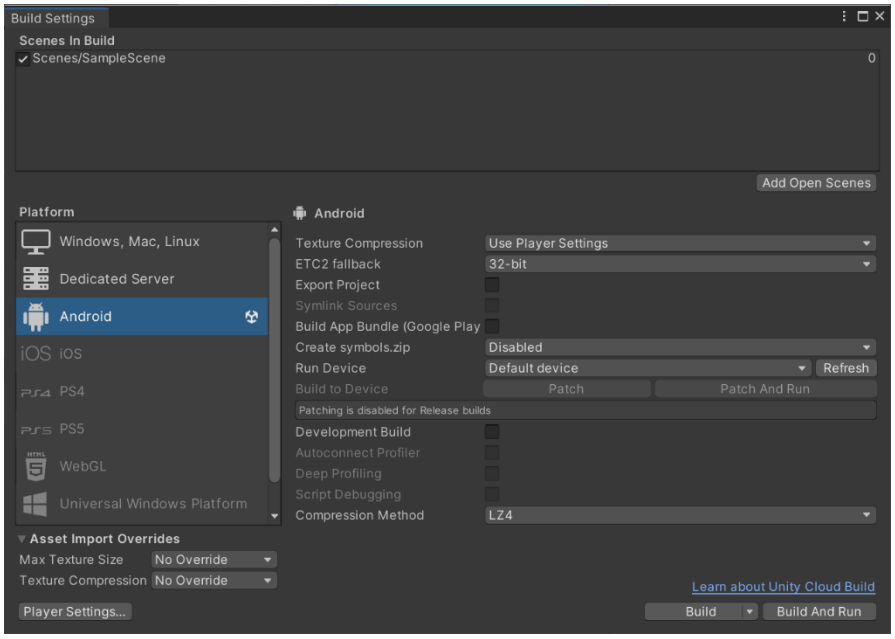

Рис. 33 – Вікно Build Settings, вкладка Android налаштувань

Для збірки проекту на телефон достатньо мінімальних налаштувань. Вкажіть Company Name (ваше прізвище англійською) та Product Name, а також оберіть Default Icon (наприклад, Icon.png) (рис. 34). В розділі

Resolution and Presentation вкажіть Default Orientation – Portrait, адже гра вертикальна.

В Other Settings під заголовком Configuration виберіть Scripting Backend IL2CPP.

IL2CPP – це система обробки скриптів, яку можна використовувати для перетворення коду C# в низькорівневий код C++ для певних платформ. В разі Android, IL2CPP перетворює C# код на C++ код, який може бути виконаний на пристроях з Android. Основна перевага використання IL2CPP на Android полягає в тому, що він забезпечує більшу швидкодію та зменшує обсяг пам'яті, який потрібен для роботи застосунку.

Також встановіть галочку для параметра Target Architectures – ARM64. Це означає включення підтримки 64-бітних процесорів ARM. 64-бітні процесори ARM є стандартом для нових моделей мобільних пристроїв та дозволяють отримати кращу швидкодію та підвищену продуктивність. Завдяки підтримці 64-бітних процесорів ARM, застосунок може використовувати більше пам'яті та обчислювальних ресурсів, що дозволяє створювати більш потужні та ефективні додатки.

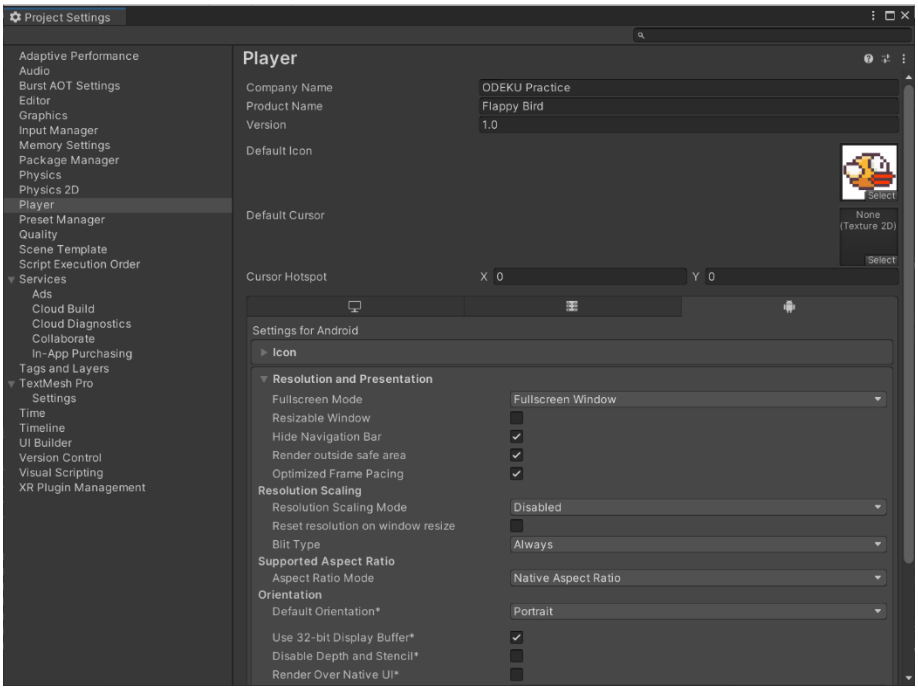

Рис. 34 – Вікно Project Settings

Окрім того, починаючи з серпня 2019 року, Google вимагає, щоб всі нові застосунки та оновлення застосунків, які планують публікувати в Google Play, мали підтримку 64-бітних процесорів ARM. Це означає, що без підтримки 64-бітних процесорів ARM застосунок може бути відхилений від публікації в Google Play.

Перевірте, щоб був відключений параметр Override Default Package Name. Тоді назва пакету застосунку буде сформована автоматично з назви компанії та назви продукту.

Закрийте вікно Project Settings та поверніться до Build Settings. Функція Build App Bundle (Google Play) дозволяє створити пакет застосунку в форматі, який оптимізований для розповсюдження через Google Play. Однак, щоб побудувати проект і мати змогу одразу запустити його на мобільному телефоні, перевірте, щоб опція Build App Bundle (Google Play) була відключена.

Натисніть Build та вкажіть назву та місце для збереження apk-файлу. Дочекайтесь побудови проекту. Це може зайняти 5-10 хвилин.

Встановіть створений apk на телефоні (рис. 35).

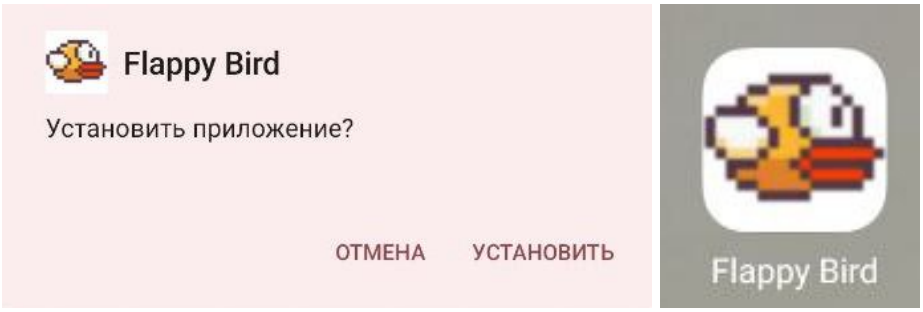

Рис. 35 – Встановлення додатку на Android

Гра відкривається в повноекранному режимі, працює коректно, містить всі звукові ефекти.

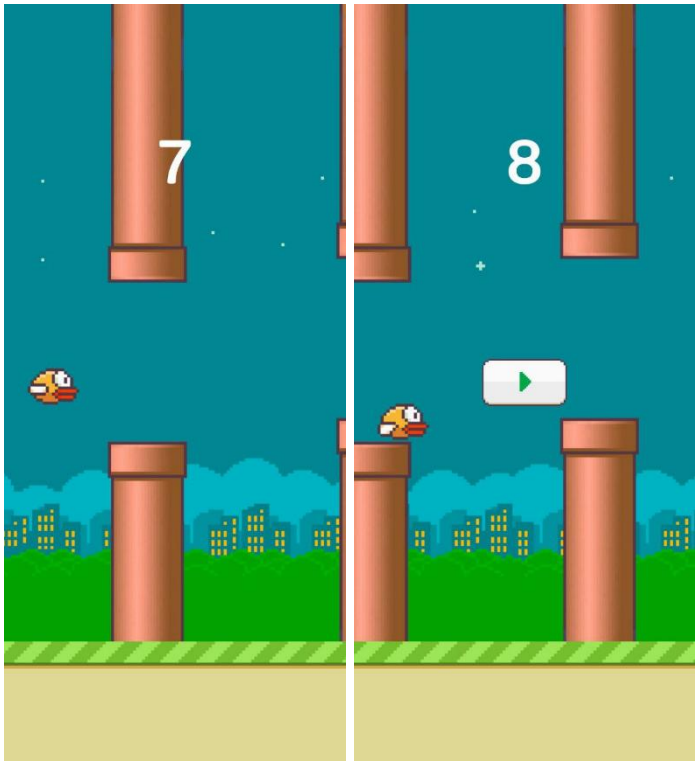

Рис. 36 – Результат встановлення гри

Звичайно, на даному етапі в грі не реалізовані вихід із застосунку, збереження максимального результату та рейтинг гравців, програвання фонової музики, налаштування звуків тощо. Детальні інструкції для реалізації цих та інших функцій, а також приклади готових проектів доступні у документації Unity на офіційному сайті за посиланням: https://docs.unity3d.com.

ПОСТАНОВКА ЗАВДАННЯ І ВИХІДНІ ДАНІ ДЛЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Для виконання індивідуального завдання потрібно реалізувати збереження даних в грі. Найпростіший варіант – використати PlayerPrefs.

PlayerPrefs – це клас, який зберігає налаштування гравця між сеансами гри. Він може зберігати рядкові дані, плаваючі та цілі числа в реєстрі платформи користувача. Unity зберігає PlayerPrefs у локальному реєстрі без шифрування, тому ніколи не використовуйте дані PlayerPrefs для зберігання конфіденційних даних.

Unity зберігає дані PlayerPrefs по-різному залежно від того, на якій операційній системі працює програма. Стандартні місця зберігання:

- Android: /data/data/pkg-name/shared_prefs/pkg-name.v2.playerprefs.xml
- iOS: використовує [NSUserDefaults standardUserDefaults] API для зберігання PlayerPrefs.
- Linux: ~/.config/unity3d/ExampleCompanyName/ExampleProductName
- Windows: HKCU\Software\ExampleCompanyName\ExampleProductName
- Windows Universal Platform: %userprofile%\AppData\Local\Packages\[ProductPackageId]\LocalState\playerprefs.dat

Таблиця 1. Статичні методи PlayerPrefs:

Назва методу	Призначення
DeleteAll	Видаляє всі ключі та значення з параметрів. Використовуйте з обережністю.
DeleteKey	Видаляє вказаний ключ із PlayerPrefs. Якщо ключ не існує, DeleteKey не має ефекту.

Продовження табл. 1. Статичні методи PlayerPrefs:

Назва методу	Призначення
GetFloat	Повертає значення, що відповідає ключу у файлі параметрів, якщо він існує.
GetInt	Повертає значення, що відповідає ключу у файлі параметрів, якщо він існує.
GetString	Повертає значення, що відповідає ключу у файлі параметрів, якщо він існує.
HasKey	Повертає true, якщо вказаний ключ існує в PlayerPrefs, інакше повертає false.
Save	Зберігає всі змінені налаштування.
SetFloat	Встановлює значення з плаваючою точкою параметра, визначеного вказаним ключем. Ви можете використовувати PlayerPrefs.GetFloat, щоб отримати це значення.
SetInt	Встановлює ціле числове значення параметра, визначеного вказаним ключем. Ви можете використовувати PlayerPrefs.GetInt, щоб отримати це значення.
SetString	Встановлює строкове значення параметра, визначеного вказаним ключем. Ви можете використовувати PlayerPrefs.GetString, щоб отримати це значення.

Для прикладу буде реалізоване зберігання за допомогою PlayerPrefs кількості пройдених перешкод в останньому зіграному раунді.

В цьому випадку необхідно оновлювати збережене значення щоразу при успішному проходженні перешкоди та анулювати значення на початку кожної нової гри. Метод, що викликається при проходженні між трубами – OnTriggerEnter2D() у скрипті Bird.cs.

Створіть нову змінну:

```
private static int lastScore;
```

В метод OnTriggerEnter2D() додайте код для зберігання поточного значення рахунку гри. Один з варіантів реалізації:

```
PlayerPrefs.SetInt("lastScore", ++lastScore);
PlayerPrefs.Save();
```

Для скидання значення на 0 на початку нової гри, додайте рядок в методи OnCollisionEnter2D() та Start():

```
lastScore = 0;
```

Можна перевірити правильність налаштувань, виводячи збережене значення в консоль під час гри. Для цього призначений метод Debug.Log(). Додайте рядок в метод OnTriggerEnter2D():

```
Debug.Log("SCORE IS "+PlayerPrefs.GetInt("lastScore"));
```

Запустіть гру. Значення зберігаються коректно та відображаються в консолі. Тепер ці дані можна відображати будь-де у грі (рис. 37).

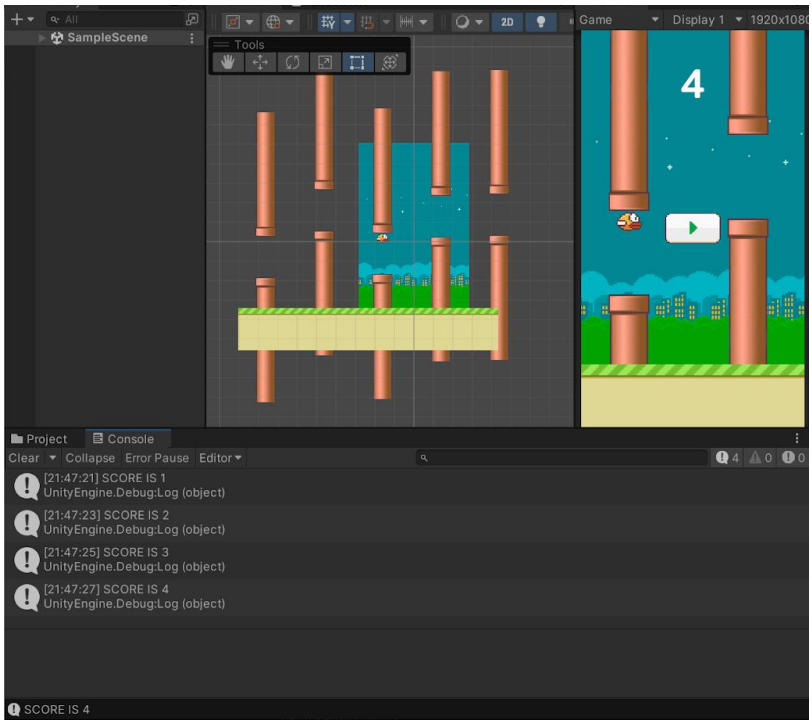

Рис. 37 – Результат відображення значень у консолі

Таблиця 2. Варіанти індивідуального завдання

№ вар.	Завдання
1	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість успішно пройдених перешкод та максимальна кількість успішно пройдених перешкод за гру.
2	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість успішно пройдених перешкод та загальна кількість зіграних раундів.
3	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість успішно пройдених перешкод та середня кількість перешкод, пройдених за одну гру.
4	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість успішно пройдених перешкод та загальна кількість взмахів крилами птаха (натискань на екран, при яких птах підлітає вгору).
5	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: максимальна кількість успішно пройдених перешкод за гру та загальна кількість зіграних раундів.
6	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: максимальна кількість успішно пройдених перешкод за гру та середня кількість перешкод, пройдених за одну гру.
7	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: максимальна кількість успішно пройдених перешкод за гру та загальна кількість взмахів крилами птаха (натискань на екран, при яких птах підлітає вгору).
8	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість зіграних раундів та середня кількість перешкод, пройдених за одну гру.

Продовження табл. 2. Варіанти індивідуального завдання

№ вар.	Завдання
9	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: загальна кількість зіграних раундів та загальна кількість взмахів крилами птаха (натискань на екран, при яких птах підлітає вгору).
10	Додати до сцени кнопку «Статистика», при натисканні на яку буде відображатись: середня кількість перешкод, пройдених за одну гру та загальна кількість взмахів крилами птаха (натискань на екран, при яких птах підлітає вгору).

Контрольні питання:

1. Які кроки потрібно виконати, щоб створити новий проект 2D гри в Unity?
2. Які основні вікна в Unity Editor? Опишіть їхнє призначення.
3. За допомогою яких компонентів створюється реалістична фізика ігрових об'єктів?
4. Для чого використовується пакет TMP?
5. Як використовувати скрипти в Unity для створення логіки гри та управління ігровими об'єктами?
6. Опишіть алгоритм експорту застосунку на пристрої з ОС Android.

ЛІТЕРАТУРА

Основна

1. Кузніченко С.Д., Коваленко Л.Б. Основи алгоритмізації та програмування. Навчальний посібник – Одеса: ТЕС, 2019. 338 с.
2. Кузніченко С.Д., Коваленко Л.Б. Алгоритмізація та програмування. Конспект лекцій – Одеса, ОДЕКУ, 2015. 326 с.

Додаткова

3. Will Goldstone. Unity Game Development Essentials Paperback, 1st edition – Packt Publishing, 2009. – 316 p. ISBN-10: 184719818X, ISBN-13: 978-1847198181
4. Jeremy Gibson Bond. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#, 1st edition – Addison-Wesley Professional, 2014. – 944 p. ISBN-10: 0321933168, ISBN-13: 978-0321933164
5. Harrison Ferrone. Learning C# by Developing Games with Unity 2019: Code in C# and build 3D games with Unity, 4th Edition – Packt Publishing, 2019. – 342 p. ISBN-10: 1789532051, ISBN-13: 978-1789532050
6. Jesse Glover. Complete Virtual Reality and Augmented Reality Development with Unity: Leverage the power of Unity and become a pro at creating mixed reality applications, 1st Edition, [Kindle Edition] / Jesse Glover, Jonathan Linowes – Packt Publishing, 2019. – 670 p. ISBN 978-1-83864-818-

Інформаційні ресурси

7. Unity Documentation. URL: https://docs.unity3d.com/Manual/index.html
8. GitHub. Flappy-Bird-Demo. URL: https://github.com/origami99/Flappy-Bird-Demo

Репозитарій ОДЕКУ URL: http://eprints.library.odeku.edu.ua/