Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(рівень вищої освіти)

на тему **"Розроблення мобільної гри "Галактика" засобами Unity3D"**

Виконав: студент **V** курсу, групи **КНз-51**

Спеціальності **122** -"Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Турок О. Г.
(прізвище та ініціали)

Керівник Капран І. Д., Карашецький В. П.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Львів – 2022 року

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут *деревообробних та комп'ютерних технологій і дизайну*
Кафедра *інформаційних технологій*
Рівень вищої освіти *перший (бакалаврський)*
Спеціальність *122 – "Комп'ютерні науки"*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
_____ Крошний І. М.
"____" _____ 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Турок Ользі Григорівній
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «*Розроблення мобільної гри "Галактика" засобами Unity3D»*
керівник проекту (роботи) *Капран Ігор Дмитрович, магістр, Карашецький Володимир Петрович, к. т. н.,доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "*13*" *грудня 2021* року №*С-618*
2. Термін подання студентом проекту (роботи) *11 квітня 2022року*
3. Вихідні дані до проекту (роботи) *Аналіз шляхів вирішення поставлених задач та переваг і недоліків аналогів мобільної гри, огляд алгоритмів та програмних засобів для розроблення мобільної гри «Галактика» під операційну систему Android*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *Вступ. Розділ 1. Стан проблемної області.*

Розділ 2. Інформаційне та математичне забезпечення.
Розділ 3. Програмне та технічне забезпечення.
Висновки. Список використаної літератури. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання *15 грудня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1.	Системний аналіз стану проблемної області. Огляд літературних джерел згідно досліджуваної теми. Збір потрібних матеріалів. Формування функціональних вимог та постановка задачі проекту.	*03. 02. 2022 р.* *14. 02. 2022 р.*	*Виконано*
2.	Огляд сучасного стану проблемної області. Оформлення першого розділу пояснювальної записки.	*14. 02. 2022 р.* *28. 02. 2022 р.*	*Виконано*
3.	Написання другого розділу. Аналіз інформаційного та математичного забезпечення.	*28. 02. 2022 р.* *16. 03. 2022 р.*	*Виконано*
4.	Оформлення третього розділу пояснювальної записки. Програмна реалізація	*16. 03. 2022 р.* *27. 03. 2022 р.*	*Виконано*
5.	Оформлення третього розділу пояснювальної записки. Формування апаратного забезпечення.	*27. 03. 2022 р.* *03. 04. 2022 р.*	*Виконано*
6.	Оформлення пояснювальної записки та здача на рецензування.	*03. 04. 2022 р.* *21. 04. 2022 р.*	*Виконано*

Студент _____ *Турок О. Г.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Капран І. Д.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Карашецький В. П.*
(підпис) (прізвище та ініціали)

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити та програмно реалізувати на основі комп'ютерних технологій мобільну гру «Галактика» під операційну систему Android. Для розроблення даної гри використати багатоплатформовий інструмент та рушій Unity3D. Логіку мобільної гри реалізувати за допомогою мови програмування C#. Розроблена гра повинна забезпечувати виконання наступних основних функцій:

1. Інтуїтивно зрозумілий інтерфейс;
2. Простота та легкість оформлення, дотримання загальної стилістики;
3. Велике ігрове поле на весь екран;
4. Користувацьке налаштування рівня;
5. Відсутність зайвих режимів та навантаження інтерфейсу;
6. Яскрава та мультиплікаційна графіка;
7. Музичний супровід.

За результатами проведеної роботи скласти пояснювальну записку, яка крім теоретичної частини має містити в собі детальне пояснення виконання кожного кроку завдання.

РЕФЕРАТ

Бакалаврська дипломна робота (проект): пояснювальна записка: 70 стор., 37 рис., 1 табл., 2 додатків, 20 джерел.

Дипломна робота присвячена проектуванню та розробці мобільної гри «Галактика» для пристроїв на операційній системі Android. Під час виконання дипломної роботи було проаналізовано основні тенденції у розробці мобільних застосунків, розглянуто аналоги даної гри, виявлено їх недоліки, на основі яких створено гру згідно із заявленим технічним завданням. Було розроблено графічне забезпечення гри, структуру гри, та саму гру. Проведене тестування розробленої мобільної гри, а також були розробленні вказівки щодо використання гри. Для розроблення гри використано багатоплатформовий інструмент Unity3D та об'єктно-орієнтовану мову програмування C#.

Ключові слова: мобільна гра, застосунок, алгоритм, інтерфейс, C#, Unity3D, Android.

ABSTRACT

Bachelor's thesis (project): explanatory note: 70 pages, 37 figures, 1 tables, 2 appendices, 20 sources.

Thesis is devoted to the design and development of the mobile game "Galaxy" for devices running the Android operating system. During the thesis the main trends in the development of mobile applications were analyzed, analogues of this game were considered, their shortcomings were identified, on the basis of which the game was created in accordance with the stated technical task. Graphics of the game, the structure of the game, and the game itself were developed. Testing of the developed mobile game was conducted, as well as instructions on the use of the game were developed. The multi-platform tool Unity3D and object-oriented programming language C # were used to develop the game.

Keywords: mobile game, application, algorithm, interface, C #, Unity3D, Android.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Android - операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux.

Unity - багатоплатформовий інструмент для розробки відеоігор і застосунків та рушій, на якому вони працюють.

C# - об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET.

Ассет - цифровий об'єкт, що переважно складається з однотипних даних, неподільна сутність, яка представляє частину ігрового контенту і має деякі властивості.

Спрайт - графічний об'єкт у комп'ютерній графіці.

Префаби - це особливий тип ассетів, що дозволяє зберігати весь GameObject з усіма компонентами та значеннями властивостей.

ВСТУП

Розроблення мобільних ігор на сьогоднішній день є не лише цікавою, а й прибутковою справою. Обсяг ігрового ринку вже перевершує кіноіндустрію та наздоганяє спортивний ринок, а доходи деяких ігрових компаній перевищують бюджети відомих фірм.

В умовах пандемії за 2021 рік ринок відеоігор у світі виріс на 35-40% і досяг позначки обсягу $2,3 мільярда. Ігровий ринок став одним із найбільш швидко зростаючих у цифровому середовищі, і зараз входить до десятки найбільших у світі.

Актуальність теми. Ринок мобільних ігор зростає з кожним днем. Ця величезна галузь розширюється в арифметичній прогресії і зупинятися поки що не збирається. Різко збільшується армія розробників, б'є рекорди кількість самих мобільних ігор. Дохід, створюваний індустрією мобільних ігор, досяг захмарних показників.

Об’єкт дослідження - процеси проектування та розроблення програмного забезпечення для трьохвимірного ігрового застосунку у жанрі «Аркади» для операційної системи Android.

Предмет дослідження - методи, прийоми та інформаційні технології розробки ігрового програмного забезпечення для пристроїв на базі Android платформ.

Мета кваліфікаційної бакалаврської роботи полягає у розробці мобільної гри «Галактика» для операційної системи Android.

Результатом роботи є розроблена та впроваджена мобільна гра «Галактика» для платформи Android.

1. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. **Аналіз ринку мобільних ігор**

Експерти компанії **App Annie** опублікували звіт про завантаження ігрових програм за перші шість місяців 2021 року. Відповідно до звіту, щотижневі завантаження ігор зросли на 25% порівняно з тим самим періодом 2019 року. Крім того, користувачі почали витрачати на мобільні ігри на 40% більше, ніж до пандемії (рис. 1. 1).

Рисунок 1. 1. Завантаження ігрових застосунків у 2021 році

Минулого року про зростання інтересу до ігор на тлі пандемії коронавірусу повідомило **Hollywood Reporter**. Згідно зі звітом, що базується на даних провідної американської телекомунікаційної компанії **Verizion**, з початку пандемії в США на 75% зросла популярність відеоігор, на 12% - потокових сервісів. Загальний веб-трафік виріс на 20%.

Тоді ж аналітики компанії **Cowen** передбачили зростання інтересу до ігор. Цю тенденцію підтвердили і спеціалісти **App Annie**. За їх словами, пандемія коронавірусу стала причиною стрімкого зростання популярності мобільних ігор і зараз немає жодних ознак його уповільнення. Компанія **App Annie** провела дослідження, щоб простежити, як саме розвивається ринок відеоігор. За підсумками аналізу зібраних

даних, аналітики опублікували докладний звіт. Насамперед у звіті вказані суми, витрачені користувачами на відеоігри. Фахівці враховували програми, що завантажуються з магазинів iOS, Google Play, Windows Phone, Amazon, Samsung Galaxy та сторонніх магазинів Android. В аналіз також включили ігри та ігрові підписки на Xbox Live, PlayStation Plus, Nintendo Switch Online, Nintendo 3DS та Switch Lite.

Найвищі показники зростання зафіксовані у мобільних ігор. Аналітики очікують, що до кінця цього року загальні витрати користувачів на цей напрямок перевищать $120 млрд. Це на 19% більше, ніж за минулий рік. Крім того, у першому півріччі щотижневі витрати на ігри в магазинах iOS App Store і Google Play склали $1,7 млрд. Це на 40 % вище за показники за аналогічний період 2019 року (рис. 1. 2).

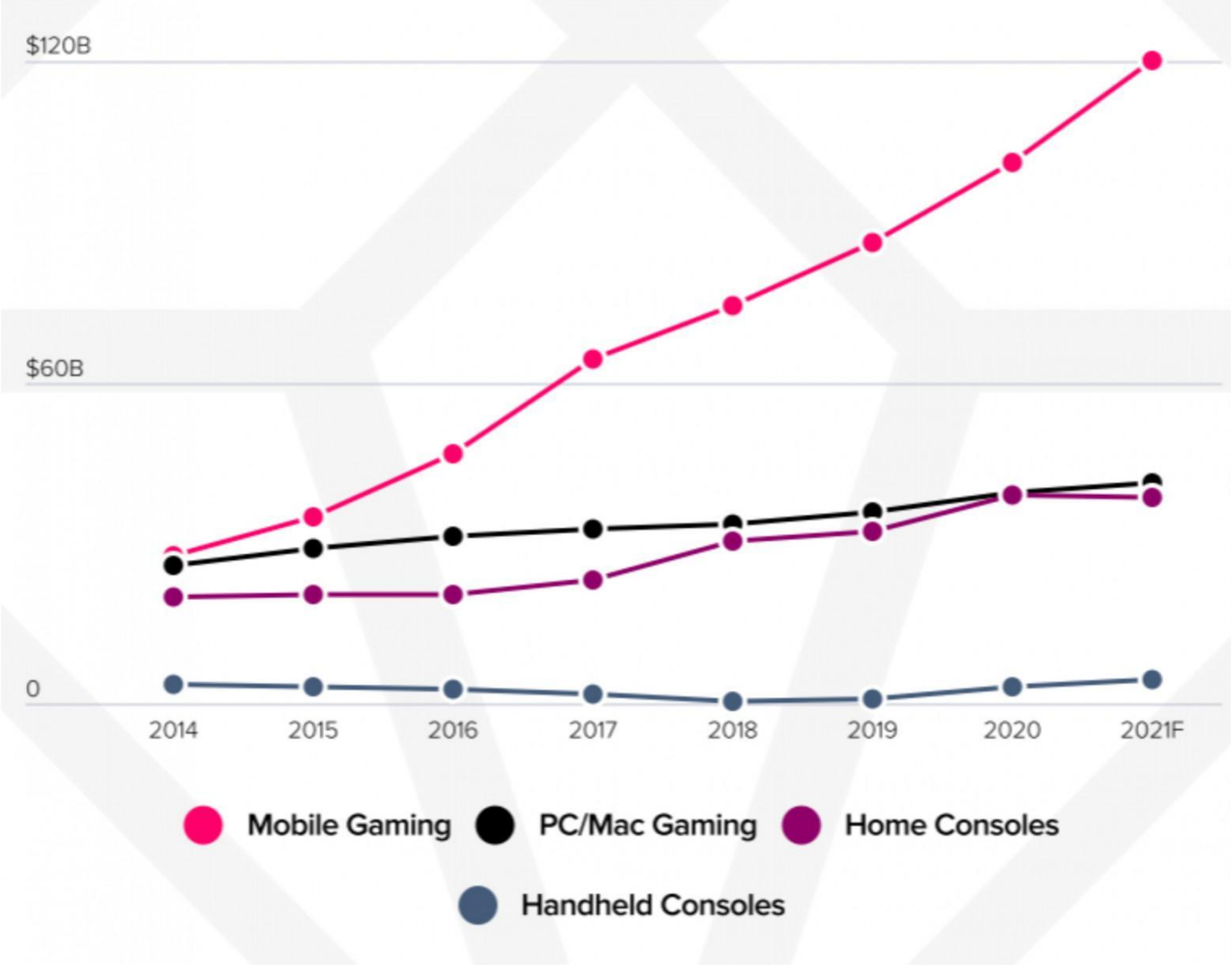


Рисунок 1. 2. Розвиток ринку відеоігор

Найчастіше ігри завантажують індійські користувачі – понад 4 млрд завантажень за перші шість місяців 2021 року. На другому місці знаходиться США і на третьому Бразилія — кожна країна має понад 2 млрд завантажень. Загальні щотижневі завантаження ігор перевищили 1 млрд, що на 25% більше, ніж у 2019 році (рис. 1. 3).

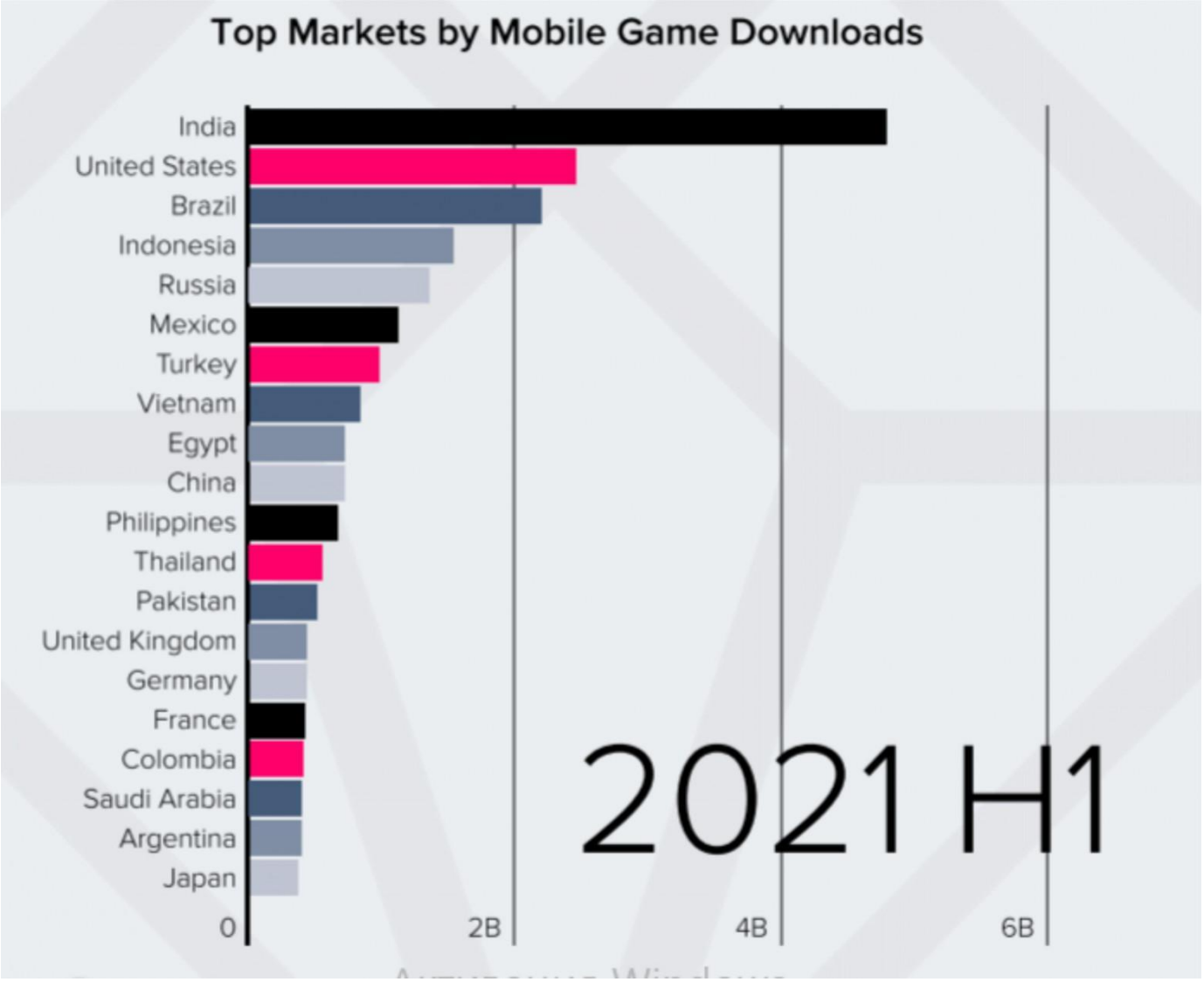


Рисунок 1. 3. Кількість завантажень мобільних ігор

Дослідники зазначають, що у минулому основний дохід розробники ігор отримували з Азіатсько-Тихоокеанського регіону. Він становив приблизно 45% ринку. Наразі його обганяють США, Німеччина та Великобританія. Також стрімке зростання витрат на ігри відбувається у Саудівській Аравії та Туреччині - на 60% і 35% порівняно з попереднім роком. Автори встановили, що загальний час,

проведений у мобільних іграх, збільшився на 35% порівняно з попереднім роком. Наразі він становить приблизно 5 млрд годин на тиждень. Також на основі цього показника аналітики склали рейтинг ігор, розподіливши його за різними віковими групами та статтю (рис. 1. 4). Користувачі від 16 до 24 років найбільше часу проводять в іграх Among Us!, Pokémon GO та ROBLOX. Користувачі від 25 до 44 років (Millenials) частіше грають у Homescapes, Toon Blast та Coin Master. Користувачі старше 45 років (Gen X/Baby Boomers) віддають перевагу Candy Crush Saga, Wordscapes і New Words With Friends. Серед користувачів чоловічої статі найбільшою популярністю користуються ігри Pokémon GO, Call of Duty: Mobile та Clash of Clans, у жіночої – ROBLOX, Among Us! та Candy Crush Saga.

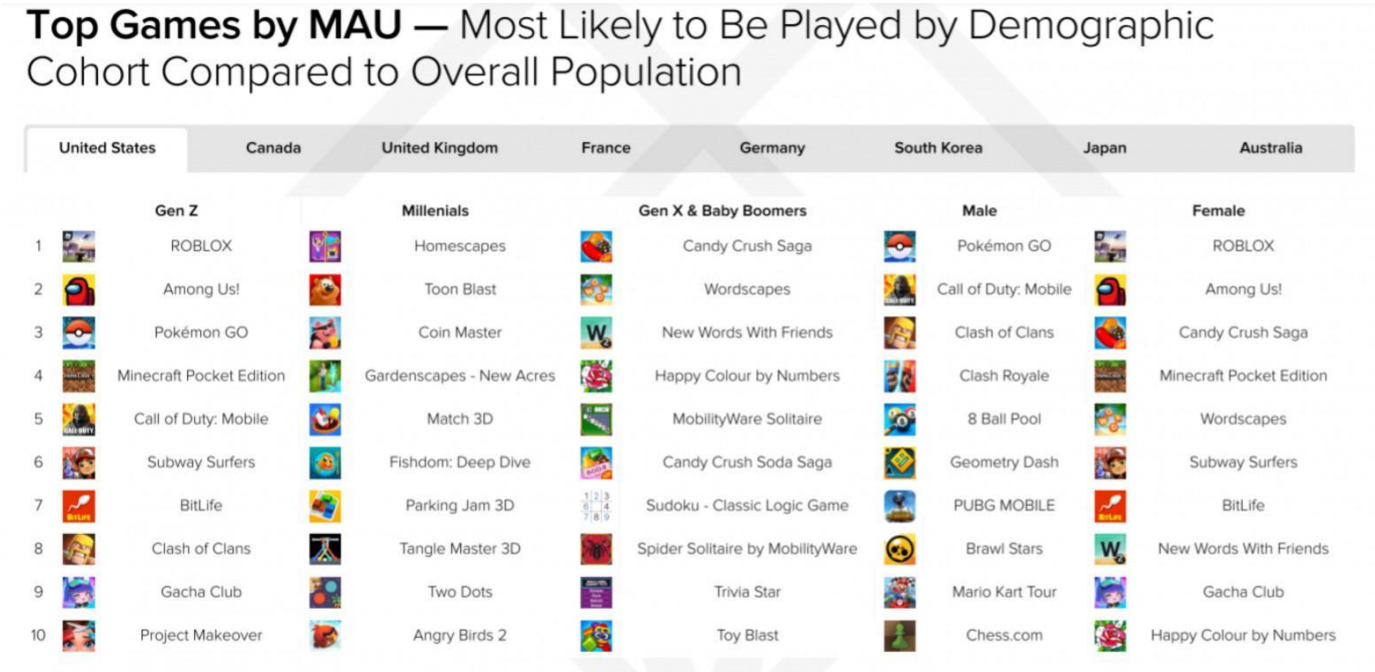


Рисунок 1. 4. Рейтинг мобільних ігор за віковими групами та статтю

З рисунку 1. 4 випливає, що у першій половині цього року користувачі по всьому світу найчастіше завантажували ігри Join Clash 3D, Free Fire та Bridge Race. Більшість коштів гравців йшла на ROBLOX, Genshin Impact і Honour of Kings. Найвищий відсоток активних користувачів встановили в іграх PUBG MOBILE, Honour of Kings та Among Us!.

Українські гравці найчастіше завантажували DOP 2: Delete One Part від білоруської компанії SayGames. За обсягом фінансових вкладень та кількістю активних користувачів перше місце посіла гра Brawl Stars від фінського видавця

Supercel. Загалом в Україні найбільшим попитом користуються ігри китайського, американського та білоруського виробництва (рис. 1. 5).

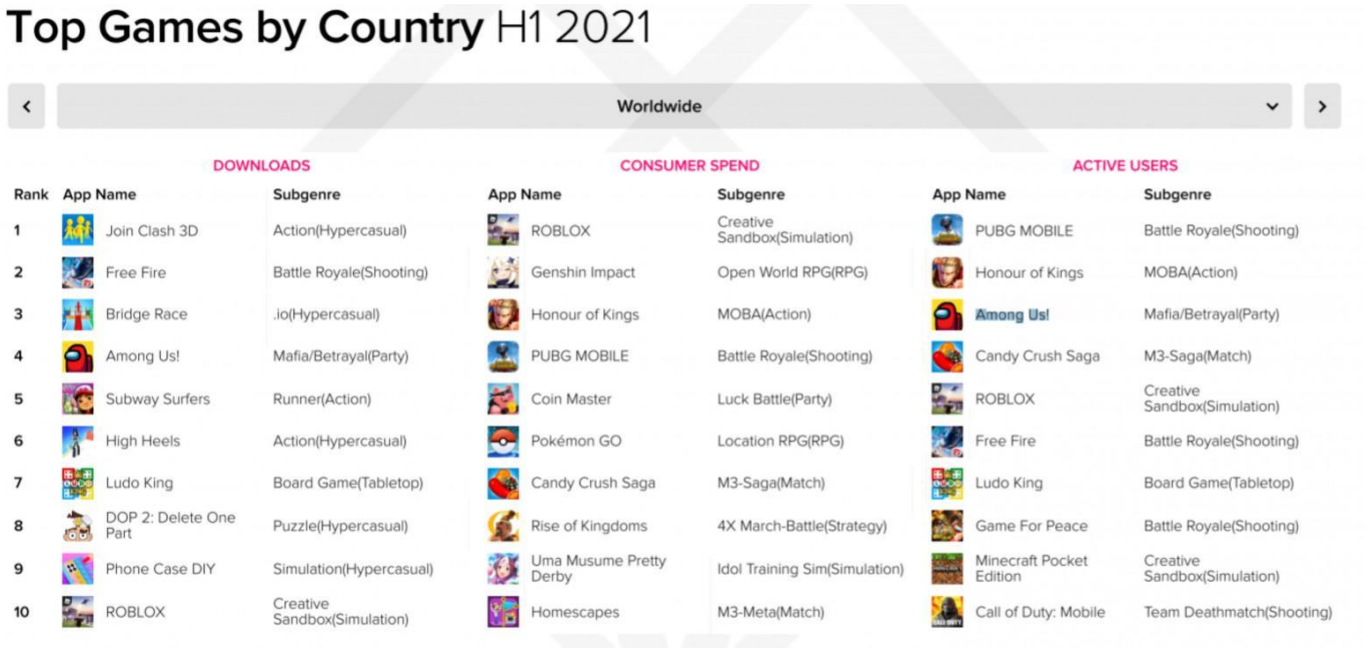


Рисунок 1. 5. Рейтинг мобільних ігор у світі

Аналітики **App Annie** пов'язують зростання інтересу до мобільних ігор не тільки з пандемією, але й з поширенням сучасніших смартфонів та досягненнями в галузі технічних можливостей мобільних пристроїв. Дослідники прогнозують швидке зростання показників із завантажень ігор у Бразилії, Індонезії та Україні. За їхніми словами, ці країни забезпечують широкі можливості для місцевих та іноземних видавців мобільних ігор та інвесторів.

1.2. **Тенденції та перспективи ринку мобільних ігор**

За даними App Annie, галузь мобільних ігор створила колосальні $41,1 млрд. валового річного доходу, очікується зростання цього показника до $50,9 млрд. Згідно з прогнозами Statista, у 2022 році валовий річний дохід перевищить $189 млрд. Незважаючи на те, що дані різних джерел дещо відрізняються, загальний висновок такий: ринку далеко до насичення. Прогнози App Annie підтвердилися у звітах Forrester про те, що до кінця 2016 лише 46% населення світу були власниками смартфонів. Це свідчить про те, що мобільна революція, що широко обговорюється, тільки починається.

Інша статистика Forrester демонструє величезний розрив між провідними компаніями, для яких мобільні пристрої стали каталізатором перетворення їхнього бізнесу, та компаніями, які відносяться до мобільних пристроїв просто як ще до одного напрямку для розвитку. На початку 2021 року лише 18% опитаних компаній належали до першої категорії. Цей показник, як очікується, подолає 25% вже наступного року. Споживачі еволюціонують набагато швидше, ніж бізнес. Сьогодні мобільний інтернет для багатьох користувачів став нагальною потребою.

Що стосується популярності, на передній план, ймовірно, вийдуть програми-агрегатори. Це інструменти, що дозволяють отримати контент із різних онлайн-ресурсів та об'єднати його у простому зрозумілому інтерфейсі. Контент може бути різноманітним: від гарячих новин до інших інтересів. Агрегатори призначені для тих, хто не має часу або бажання відвідувати багато сайтів або встановлювати багато застосунків. Серед популярних програм-агрегаторів - Flipboard, News360, Feedly та IFTTT. Програми-агрегатори зазвичай завойовують популярність серед користувачів у тих випадках, коли вони зручні або покращують процес покупок. Наприклад, Facebook зробив це з додатком Messenger, який дозволяє користувачам читати свої стрічки та викликати Uber.

Високопродуктивні мобільні процесори, підтримка потужної графіки, якісні екрани та швидке інтернет-з'єднання перетворили смартфони на ігрові пристрої. Згідно зі звітами App Annie, мобільні ігри, на які у 2019 році припадало менше 50% доходу від усіх мобільних застосунків, генерували 85% доходів ринку мобільних додатків у 2021-му.

Різко збільшився час, який користувачі проводять у різноманітних застосунках. При цьому неігрові програми обійшли ігри. За даними Flurry Analytics, до кінця 2021 року ринок мобільних застосунків встановив новий рекорд з використання.

Отже, вибухове зростання ринку мобільних застосунків найближчим часом збережеться. Незважаючи на посилення конкуренції в цій галузі, розробники застосовують нові методи монетизації і створюють все більш цікаві та корисні рішення для користувачів. Щонайменше дві нові моделі монетизації добре себе зарекомендували і останнім часом нарощують популярність. Розробники та видавці навчилися застосовувати ці моделі відповідно до ситуації. Наприклад, модель з підпискою працює лише у певних нішах, але є найприбутковішою з усіх. У той же час модель Freemium, яку так критикують за потенційну недобросовісність, при продуманому застосуванні дає приголомшливі результати, наприклад Clash of Clans. І лише платні програми впадають у немилість, хоча все ще виправдовують себе в окремих випадках.

Гібридні моделі монетизації, такі як вбудована реклама та покупки з програми, явно набирають популярності у світі бізнесу. Більшість досліджень припускають, що вбудована реклама буде основним фактором зростання мобільного ринку найближчими роками.

Сьогодні iOS та Android — провідні мобільні операційні системи, а технічні гіганти Apple та Google володіють найбільшими магазинами мобільних застосунків.

Як розвиватиметься ринок мобільних ігор, покаже час. Ринкові тенденції припускають, що в найближчому майбутньому ринок продовжить генерувати все більший і більший дохід. Цілком зрозуміло, що революція мобільних застосунків, що широко обговорюється, тільки починається.

2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. **Багатоплатформовий інструмент Unity**

За даними сайту gamedatacrunch.com, у 2021 році 49,48% усіх платних ігор, що вийшли в Steam, були розроблені на Unity. А у сфері мобільних ігор цей показник уже давно перевищив 50%. Зараз існує безліч відкритих платформ, але найпопулярніші і найбільші з них - це Unity і Unreal Engine 4. Важливі відмінності між двома рушіями лежать на рівні мови програмування - C # для Unity і C + + для UE4. Перший суворіший і має менший поріг входу, а другий надає більше можливостей, але вимагає більшої дисципліни від розробника.

Ігровий рушій Unity не просто так став одним із найпопулярніших у всьому світі. Його цінують за:

- **простоту** - у рушія низький поріг входу, тому його можуть освоїти навіть новачки;
- **універсальність** - за його допомогою можна зробити гру для будь-якої сучасної популярної платформи (ПК, iOS, Android, PlayStation, Xbox, Switch);
- **гнучкість** – Unity можна легко налаштувати під конкретний проект, щоб зробити розробку максимально ефективною.

Unity найчастіше використовується для інді-проектів або мобільних ігор - в рушії менше зайвих функцій і його можна охарактеризувати як "кімнату", з якої можна створити майстерню, зручну для роботи над конкретним жанром або серією ігор. Головна перевага Unity - це простота та гнучкість. Гнучкість Unity дозволяє розробникам у короткий термін і з мінімумом вкладень підлаштовувати рушій під власні потреби. Компанії можуть розширювати набір функцій під конкретний проект. Це стосується як масштабу, так і жанру Unity однаково добре підходить і для створення невеликої головоломки, і для величезної класичної RPG.

Ще одна перевага Unity у тому, що навколо нього сформувалося величезне співтовариство, яке готове ділитися досвідом — у мережі є безліч докладних уроків та різноманітних туторіалів.

У магазині рушія є багато платних та безкоштовних ассетів. Там можна знайти як прості 3D-моделі та нехитрі механіки, так і складні системи, які допоможуть реалізувати окремі ігрові аспекти, наприклад, штучний інтелект (рис. 2. 1).

Рисунок 2. 1. Багатоплатформовий інструмент та ігровий рушій Unity

Unity підходить навіть для тих, хто взагалі не володіє мовою програмування C# - рушій вже давно підтримує систему візуального програмування Bolt, яку можна використовувати для написання ігрової логіки без коду (рис. 2. 2). Це означає, що в Unity можуть працювати й розробники, які не спеціалізуються на програмуванні. Наприклад, Bolt дозволяє гейм-дизайнерам швидко прототипувати свої задуми, щоб одразу ж випробувати механіки у дії, а левел-дизайнерам самостійно налаштовувати тригери на локаціях. Тим не менш, Bolt не зсуває Unity до рівня простих конструкторів - це все те ж написання логіки, але в більш зрозумілому і простому вигляді. Хоча візуальний скриптинг і спрощує процес програмування, розробник має розуміти базові принципи логіки.

Unity дуже часто використовується в ігровій індустрії - цей рушій використовують як для великих консольних ігор, так і для невеликих мобільних

проектів. Завдяки цьому розробник на Unity має широкі можливості при виборі місця роботи — він однаково легко може піти і в інді-геймдев, і в сферу мобільних ігор.

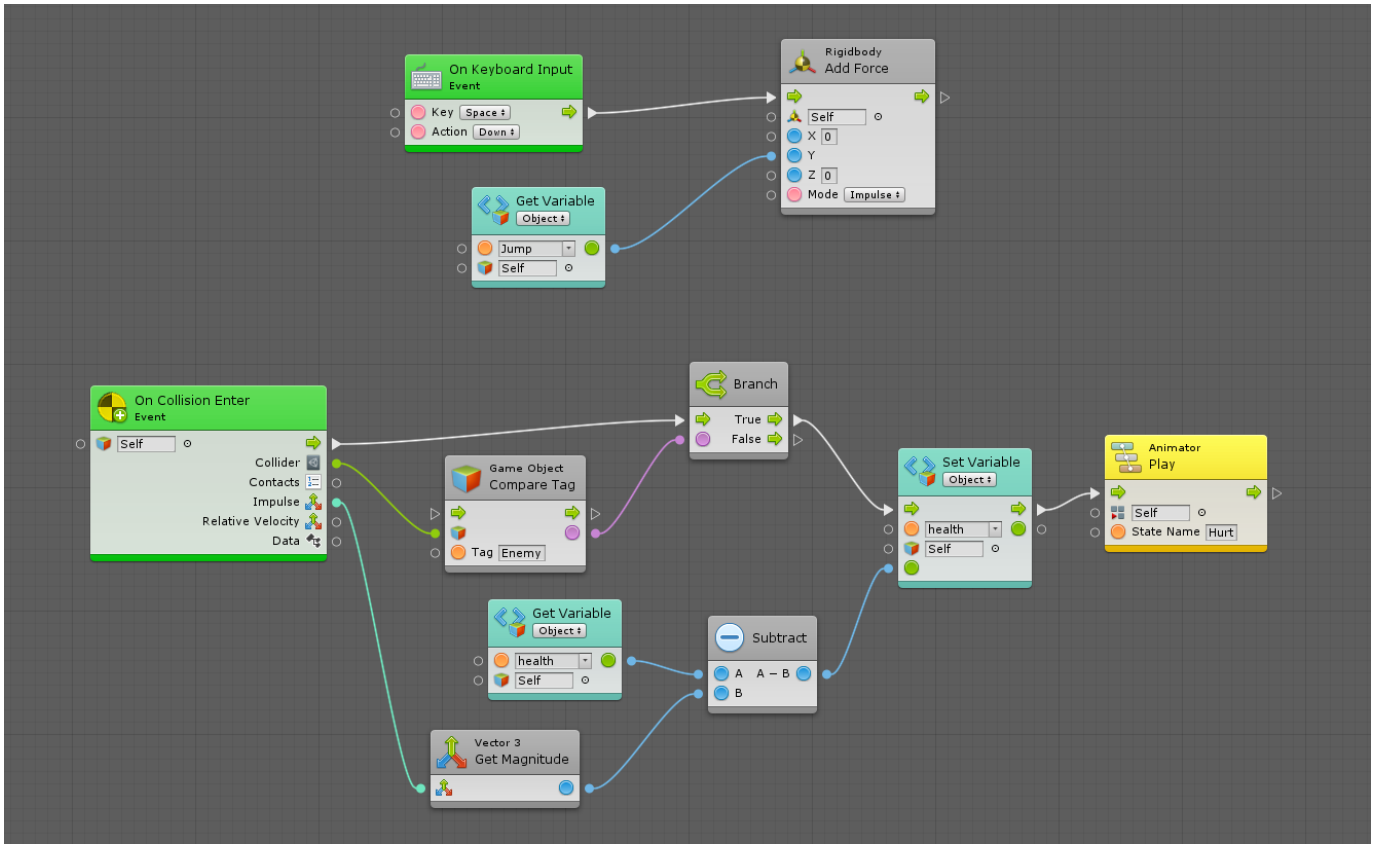


Рисунок 2. 2. Приклад використання візуального скриптингу у Bolt

2.2. **Ігровий рушій Unity3D**

Unity3D є сучасним кросплатформним рушієм для створення ігор та програм, розроблений Unity Technologies. За допомогою цього рушія можна розробляти не тільки програми для комп'ютерів, але і для мобільних пристроїв (наприклад, на базі Android), ігрових приставок та інших аксесуарів.

По-перше, варто відзначити те, що у середовище розробки Unity вбудований ігровий рушій, іншими словами, ми можемо протестувати свою гру не виходячи з редактора. По-друге, Unity підтримує імпорт величезної кількості різних форматів, що дозволяє розробнику гри конструювати самі моделі у зручнішому застосунку, а Unity використовувати за прямим призначенням – розробки продукту. По-третє,

написання сценаріїв (скриптів) здійснюється найбільш популярними мовами програмування — C# і JavaScript.

Таким чином, Unity3D є актуальною платформою, за допомогою якої ми можемо створювати свої власні програми та експортувати їх на різні пристрої, чи то мобільний телефон чи приставка Nintendo Wii.

Переваги Unity3D:

1. Unity3D візуально виглядає більш узгоджено на всіх пристроях, крім того, швидко розгортається одним клацанням на будь-якій платформі;
2. Unity3D займає на пристрої набагато менше місця, менше впливає на роботу кінцевого користувача. Компактний розмір особливо важливий у Google Play Store, де APK доводиться ділити на частини, якщо цей файл виявляється більшим за 50mb;
3. Unity3D набагато простіше вивчити та зрозуміти. Озброївшись Unity3D, недосвідчений користувач може розпочати роботу швидше та створювати продукти, підтримку яких гарантує велика спільнота фахівців;
4. Тривалість ітерації в Unity3D набагато менша (розгортання та компіляція вихідного коду відбувається швидше, шейдери компілюються майже миттєво).

Недоліки:

1. Вихідний код рушія закритий;
2. Нова система UI цілком задовільна. У ній відсутній спеціальний редактор, всі зміни вносяться прямо в сцені — а сцена дуже велика. Коли ви відкриваєте сцену і хочете відредагувати UI, вам спочатку доведеться добряче збільшити масштаб області, що вас цікавить.

Отже, Unity3D - один з найпопулярніших ігрових двигунів. В останні роки все більше відмінних ігор виходять завдяки тому, що Unity є простим у використанні і пропонує розробникам багато готових рішень.

2.3. **Об'єктно-орієнтована мова програмування C#**

.NET – це середовище або платформа. Вона розташована між кодом та Windows, дозволяючи надавати потрібні служби. Платформа .NET складається із двох основних компонентів. Це Common Language Runtime та .NET Framework Class Library.

Common Language Runtime (скорочено CLR) можна назвати рушієм платформи .NET. CLR займається управлінням пам'яттю, компіляцією та виконанням коду, роботою з потоками управління, забезпеченням безпеки тощо.

.NET Framework Class Library - це набір класів на всі випадки життя.

Однією з основних ідей .NET є сумісність різних служб, написаних різними мовами. Наприклад, служба, написана на C++ для .NET, може звернутися до методу класу бібліотеки, написаної на Delphi; на C# можна написати клас, успадкований від класу, написаного на Visual Basic .NET, а виняток, створений методом, написаним на C#, може бути перехоплений та оброблений у Delphi.

C# є жорсткою типізованою мовою. При її використанні ми повинні оголошувати тип кожного об'єкта, який створюється (наприклад, цілі числа, числа з плаваючою точкою, рядки, вікна, кнопки, і т. д.), і компілятор допоможе нам уникнути помилок, пов'язаних із присвоєнням змінних значень тільки того типу, що їм відповідає. Тип об'єкта вказує компілятор розмір об'єкта (наприклад, об'єкт типу int займає в пам'яті 4 байта) і його властивості (наприклад, форма може бути видима і невидима, і т.д.).

Мова C# надає програмісту широкий спектр вбудованих типів. Всі вони відповідають CLS (Common Language Specification) і це гарантує, що об'єкти, створені на C#, можуть успішно використовуватися поряд з об'єктами, створеними будь-якою іншою мовою програмування, що підтримує .NET. Кожен тип має строго заданий йому розмір, який може змінюватися.

Об'єкти одного типу можуть бути перетворені на об'єкти іншого типу неявно або явно. Неявні перетворення відбуваються автоматично, компілятор робить це замість

нас. Явні перетворення здійснюються, коли ми «наводимо» значення іншого типу. Неявні перетворення також гарантують, що дані не будуть втрачені.

У C# тип object є вихідним предком для всіх типів. Посилання object можна використовувати для прив'язки до будь-якого об'єкта. Також корисно у відображенні, коли код повинен працювати з об'єктами тип яких невідомий. Object надає нам кілька чудових методів. Equals(), GetHashCode(), GetType() та ToString(). Ці методи доступні будь-якому об'єкту. C# має свій тип string. Тому такі операції як копіювання та злиття відбувається миттєво.

C# дозволяє визначати свої складні типи на основі наявних простих типів. Як і примітиви, складні типи можуть поділятися на типи за посиланням та за значенням. У C# структура – це особливий тип класу. Оскільки структура розміщується в стеку, вона може створюватися та копіюватися більш ефективно, ніж клас. Струтура в C# абсолютно нове поняття, не тільки тому, що їх члени за замовчуванням є закритими, але екземпляри структур та екземпляри класів розміщуються у різних місцях пам'яті. Структури можуть підтримувати більшість функцій класу (хоч і не підтримує успадкування реалізацій), але їх потрібно використовувати обережно. Найкраще структури підходять для представлення невеликих об'єктів.

Класи є основним визначеним для користувача типом в C# і платформі .NET. Практично всі програми мають принаймні один клас (теоретично можна використовувати структуру замість класу), який містить метод Main() - точку входу до програми. Класи - це складові типи даних, які включають члени даних і функції. Класи так само містять у собі вкладені типи даних.

Інтерфейс C# містить тільки абстрактні елементи, що не мають реалізації. Безпосередньо реалізація цих елементів має бути у класі, похідному від цього інтерфейсу. Інтерфейси C# можуть містити методи, властивості та індексатори, але на відміну, наприклад, від Java, вони не можуть містити константних значень. Наприклад, якщо інтерфейс містить у собі метод, він працюватиме, але сам код реалізації методу всередині нього нічого очікувати.

Делегати це типи, які посилаються на методи. Вони схожі на показники функцій в C++, але дозволяють створювати екземпляр класу і викликати як статичні способи, і способи конкретного екземпляра класу.

Масив задає метод організації даних. Масивом називають упорядковану сукупність елементів одного типу. Кожен елемент масиву має індекси, що визначають порядок елементів. Число індексів характеризує розмірність масиву. У мові C#, як і в багатьох інших мовах, індекси задаються цілим типом. У мові C# знято значне обмеження мови C++ на статичність масивів. Масиви у мові C# є реальними динамічними масивами (рис. 2. 3).

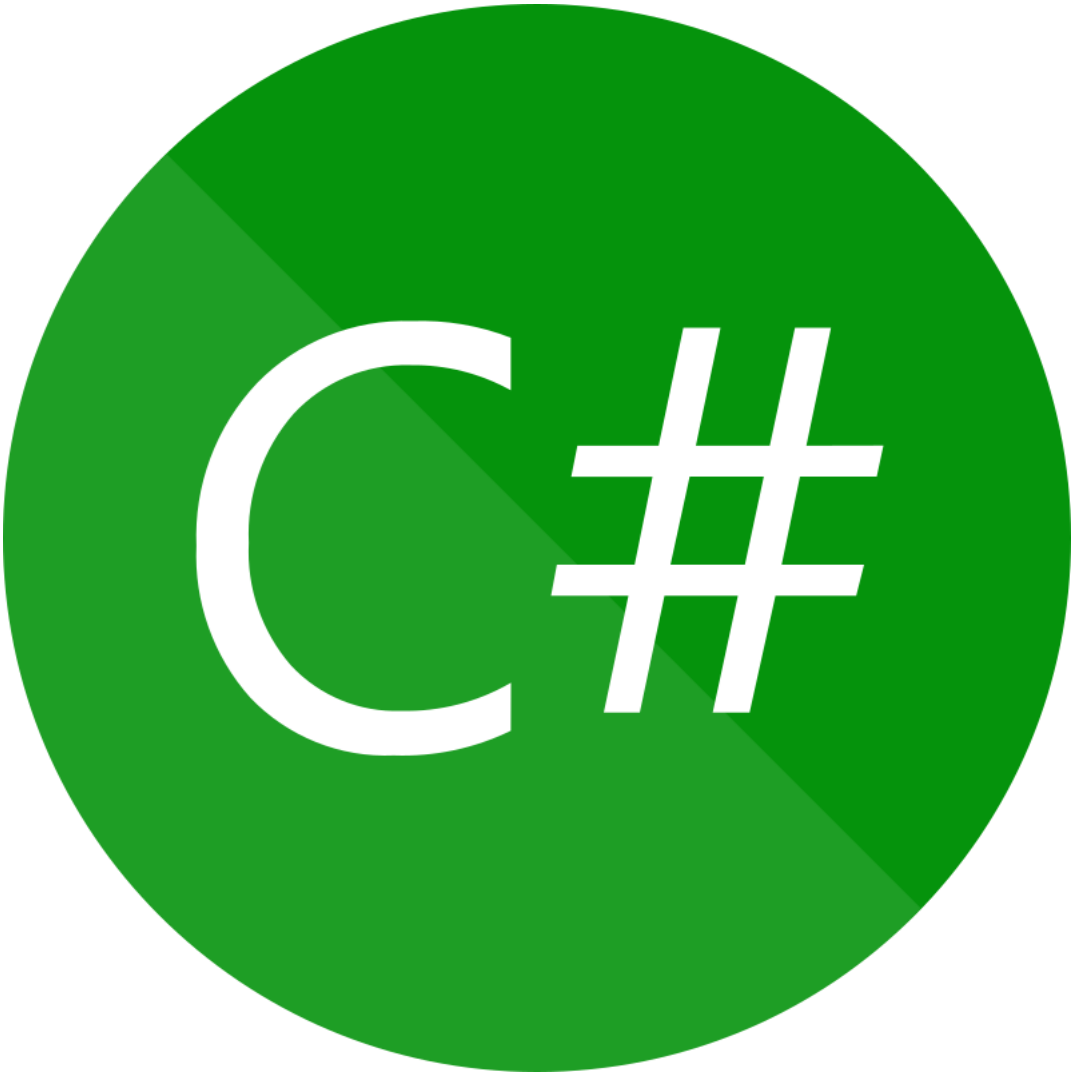


Рисунок 2. 3. Об'єктно-орієнтована мова програмування C Sharp

3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Перші кроки в Unity

Unity - це один з найпопулярніших рушіїв зараз. Скласти йому конкуренцію може хіба що **Unreal Engine 4**, але що перший, так і другий має свої плюси і свої мінуси. Unity підтримує майже 30 платформ, у тому числі мобільні, віртуальну реальність, настільні комп'ютери, консолі тощо. Unity - це не просто добрий варіант для старту, це ідеальний варіант для старту! Тут закладено використання гнучкої модульної системи при створенні сцен та персонажів у грі. Навіть новачок здатний створити хороший проект, використовуючи готові спрайти та конструктор рушія.

Першим кроком буде встановлення ПЗ. На офіційному сайті є чотири різні версії програми для встановлення:

1. **Unity Personal:** Ця версія безкоштовна. Проблема в тому, що в кожній грі буде відображатися заставка «Зроблено Unity», яку не можна видалити.
2. **Unity Plus:** ця версія коштує 35 доларів на місяць. Вона поставляється з інструментами звітності про продуктивність, оболонкою Unity Pro та деякими додатковими функціями. Ця версія дозволяє відключати або настроювати заставку "Зроблено Unity".
3. **Unity Pro:** це найвищий доступний рівень. Він коштує 125 доларів на місяць і поставляється з корисними сервісами Unity, професійними надбудовами для iOS та Android та не має заставки.

Першим встановиться Unity Hub. Unity Hub — це автономна програма, яка спрощує процес пошуку, завантаження та управління вашими проектами та установками Unity.

Щоб створити новий проект (і вказати, в якій версії редактора його відкрити), виконайте одну з таких дій:

- Натисніть кнопку **New**. У рядку заголовка діалогового вікна Новий проект відображається версія редактора, яку повинен використовувати проект.
- Клацніть стрілку списку, що розкривається, поруч із кнопкою «Створити», щоб вибрати версію редактора, яку ви хочете використовувати (рис. 3. 1).

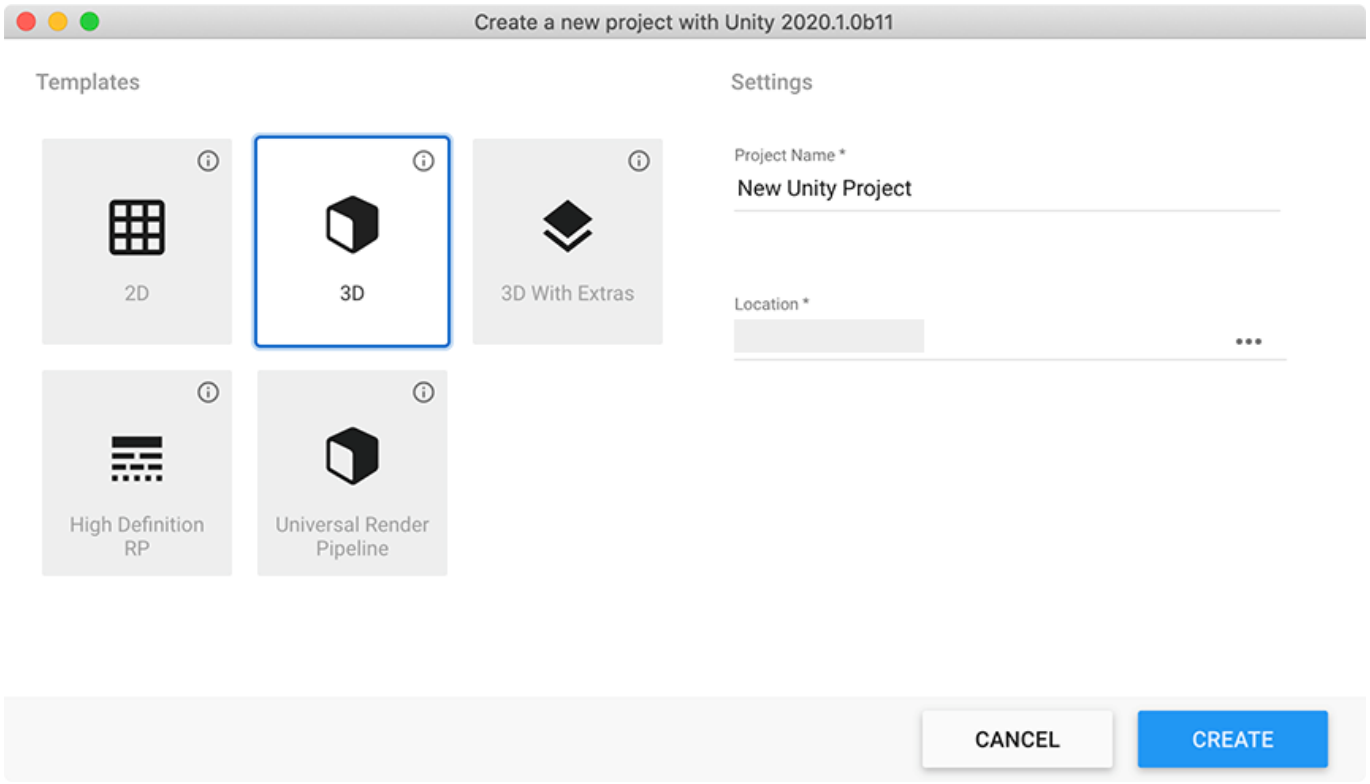


Рисунок 3. 1. Вкладка "Проекти"

Коли проект завантажиться, з'явиться екран, заповнений інформацією. Наш макет, ймовірно, виглядатиме так (рис. 3. 2):

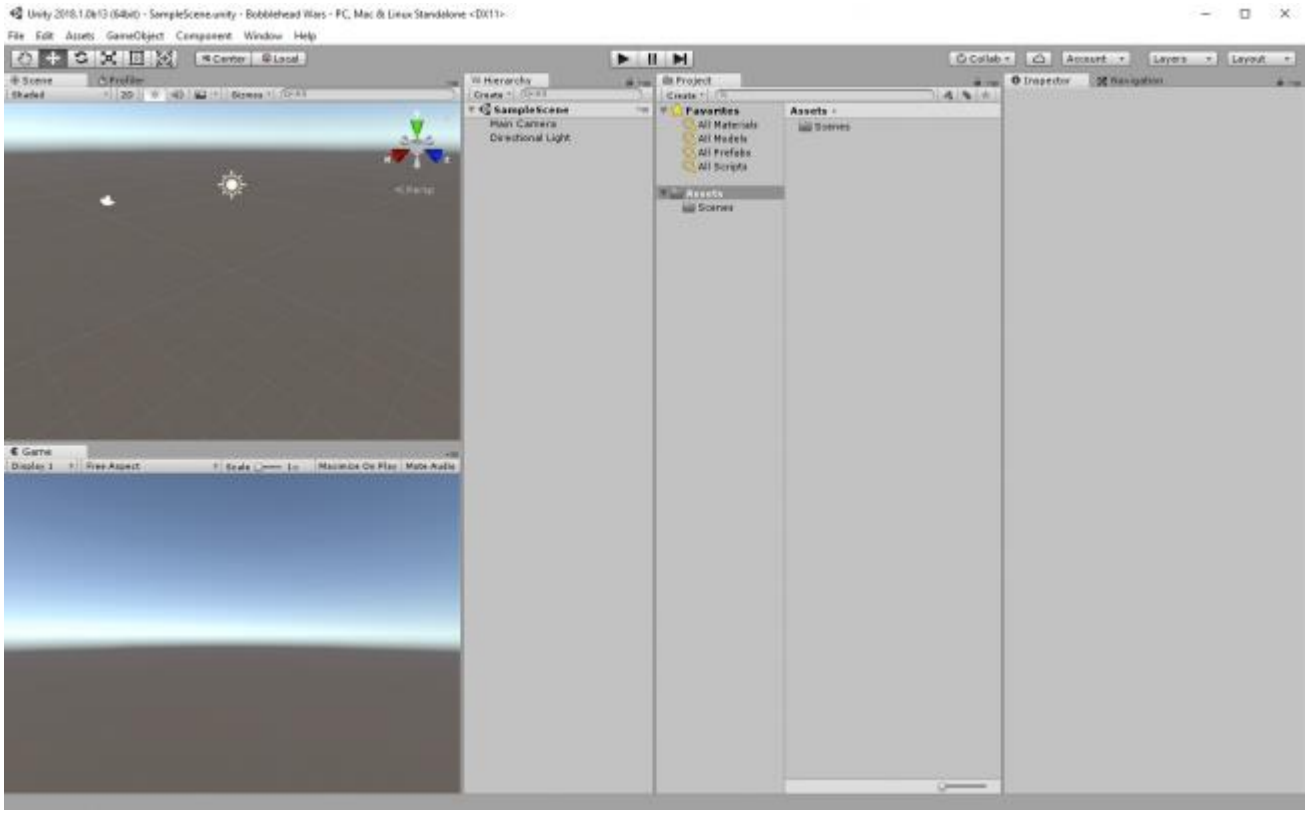


Рисунок 3. 2. Макет з інформацією про проект

Кожен макет складається з кількох різних представлень. Просто панель інформації, яку ви використовуєте для керування. Наприклад, є представлення для розміщення об'єктів. Ось як виглядає інтерфейс, розбитий на окремі представлення (рис. 3. 3).

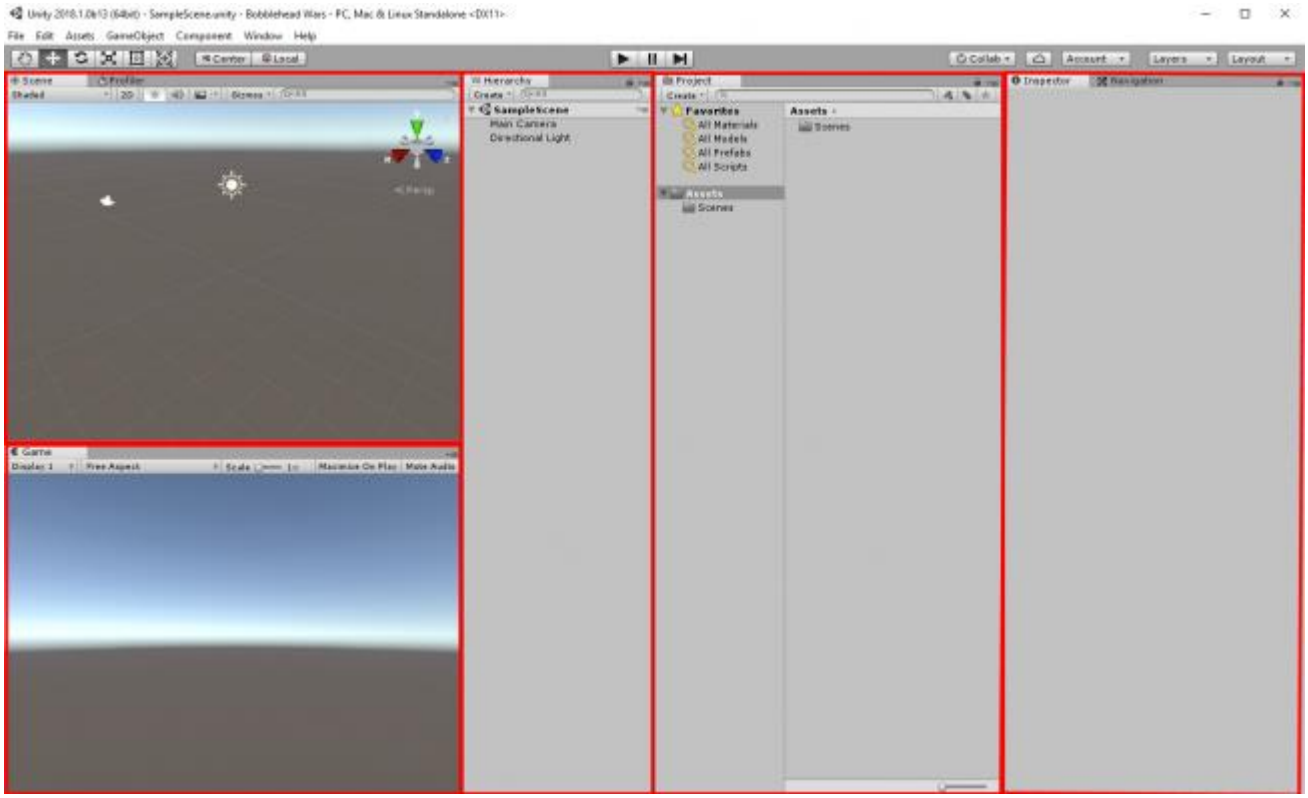


Рисунок 3. 3. Інтерфейс розбитий на окремі представлення

Щоб переглянути список усіх представлень, необхідно клацнути на параметр **«Вікно»** у рядку меню (рис. 3. 4).

Рисунок 3. 4. Список представлень

Користувальницький інтерфейс Unity повністю настроюється, тому ми можемо додавати, видаляти та впорядковувати представлення на свій розсуд. У редакторі необхідно знайти вкладку **«Гра»** і клацнути правою кнопкою миші . У розкривному

меню виберіть **"Додати вкладку"**, потім виберіть **"Профіль"**. Представлення **Profiler** дозволяє аналізувати нашу гру під час її роботи. На жаль, профіль також блокує перегляд гри, тому ми не зможемо грати в гру, поки ми її профілюємо (рис. 3. 5).

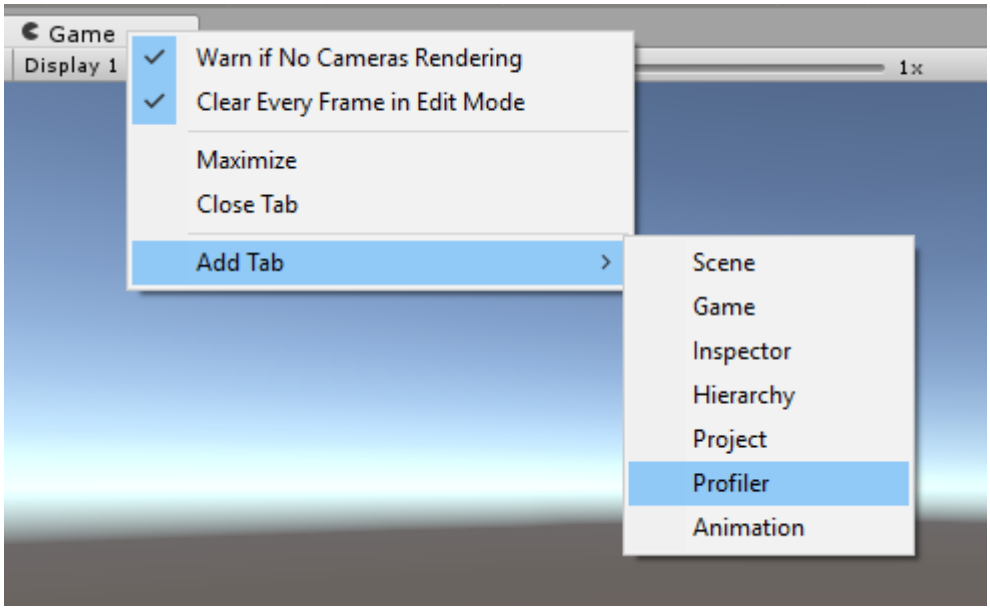


Рисунок 3. 5. Представлення Profiler

Натисніть та утримуйте вкладку **Профілі** і перетягніть її на вкладку Сцена вище (рис. 3. 6).

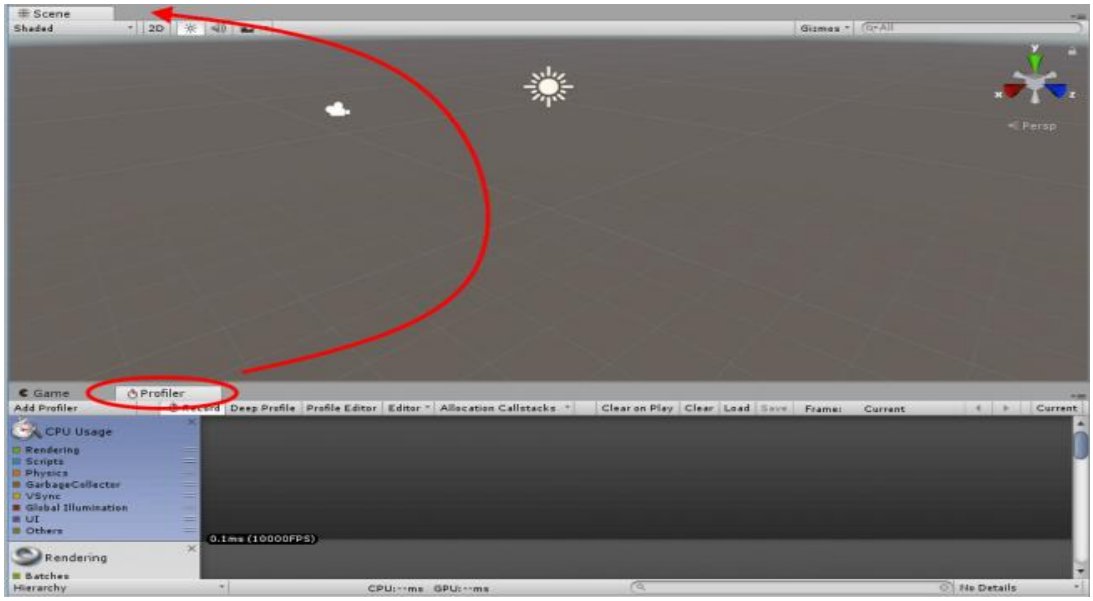


Рисунок 3. 6. Редагування видів

Представлення, в якому ми імпортуємо та впорядковуємо активи, називається **менеджер проекту**. Він імітує організацію нашої файлової системи (рис. 3. 7).

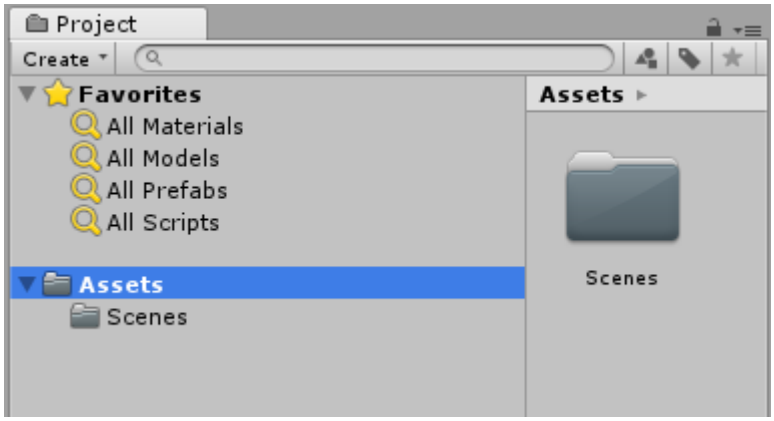


Рисунок 3. 7. Менеджер проектів

У Менеджері проектів виберіть папку **Assets** і натисніть кнопку **Create**. У розкриваючому меню виберіть **«Папка»** та назвіть її **«Моделі»**. Це буде місце для всіх моделей. Unity створює метадані для кожного активу. Створення, зміна або видалення ресурсів у файловій системі може порушити ці метадані та вашу гру. Створіть такі папки: **Animations**, **Materials**, **Prefabs**, **Presets**, **Scripts** та **Textures**. Ваш менеджер проекту має виглядати так (рис. 3. 8):

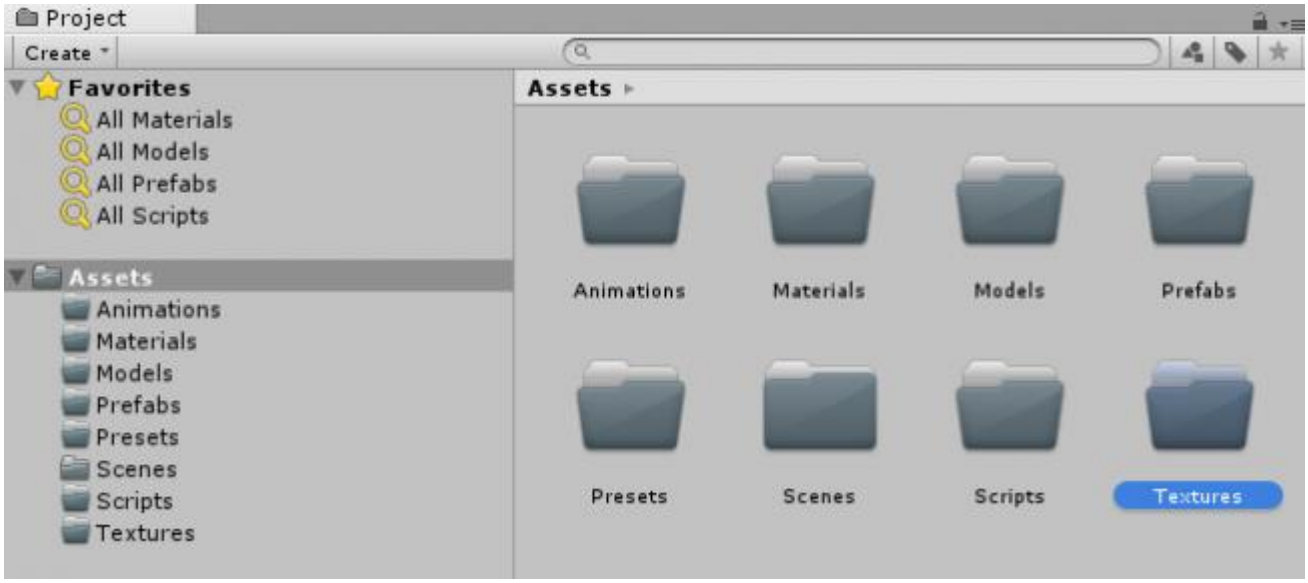


Рисунок 3. 8. Створення папок у диспетчері проекту

Нарешті ми можемо змінити назву активу. Наприклад, наша поточна сцена називається **SampleScene**. Виберіть папку **Scenes**, а потім виберіть **SampleScene**. Ім'я буде видалено. Один раз клацніть на ньому ще раз, і ви напишете нове ім'я. Змініть його на **Main** (рис. 3. 9).

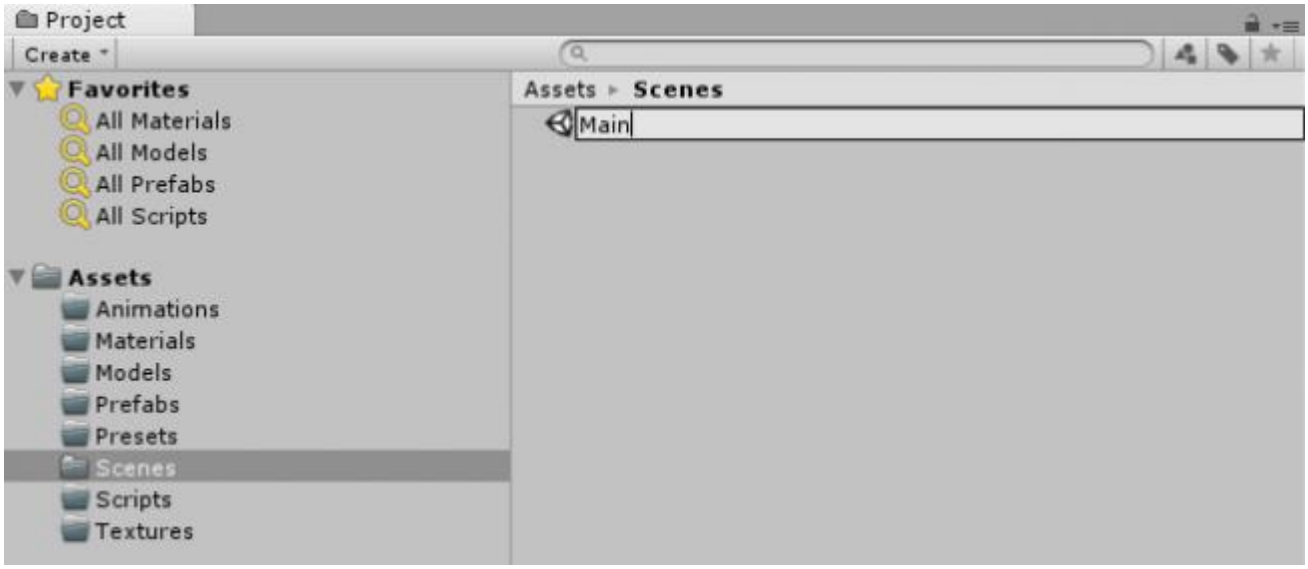


Рисунок 3. 9. Перейменування активу (сцени) в диспетчері проекту

3.2. Розроблення мобільної гри «Галактика»

Концепція гри полягає в тому, що вам потрібно взяти на себе управління зорельотом і знищити якомога більшу кількість метеоритів. На вашому шляху з'являтимуться ворожі космічні кораблі, які заважатимуть вам і після їх знищення з'являтимуться «капсули» після підбору яких буде доступний новий тип зброї. Гра називатиметься «Галактика» (рис. 3. 10).

Рисунок 3. 10. Мобільна гра «Галактика»

Графіка гри складається з наступних текстур (рис. 3. 11):

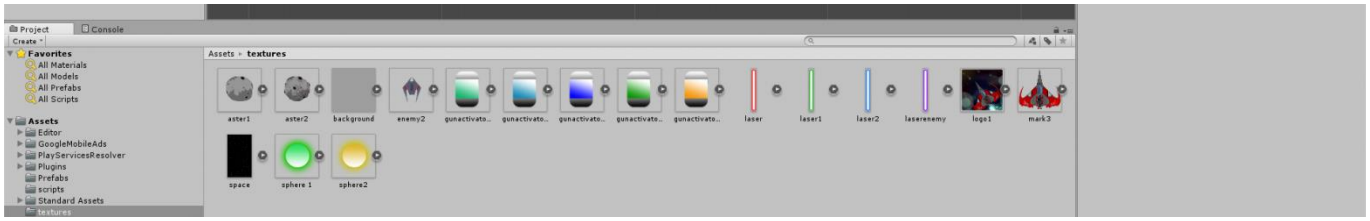


Рисунок 3. 11. Текстури гри

На основі створених текстур були розроблені такі префаби (рис. 3. 12):

Рисунок 3. 12. Префаби мобільної гри «Галактика»

Де **asteroidrotate** - метеорит, який потрібно знищувати;

enemy - ворожий зореліт;

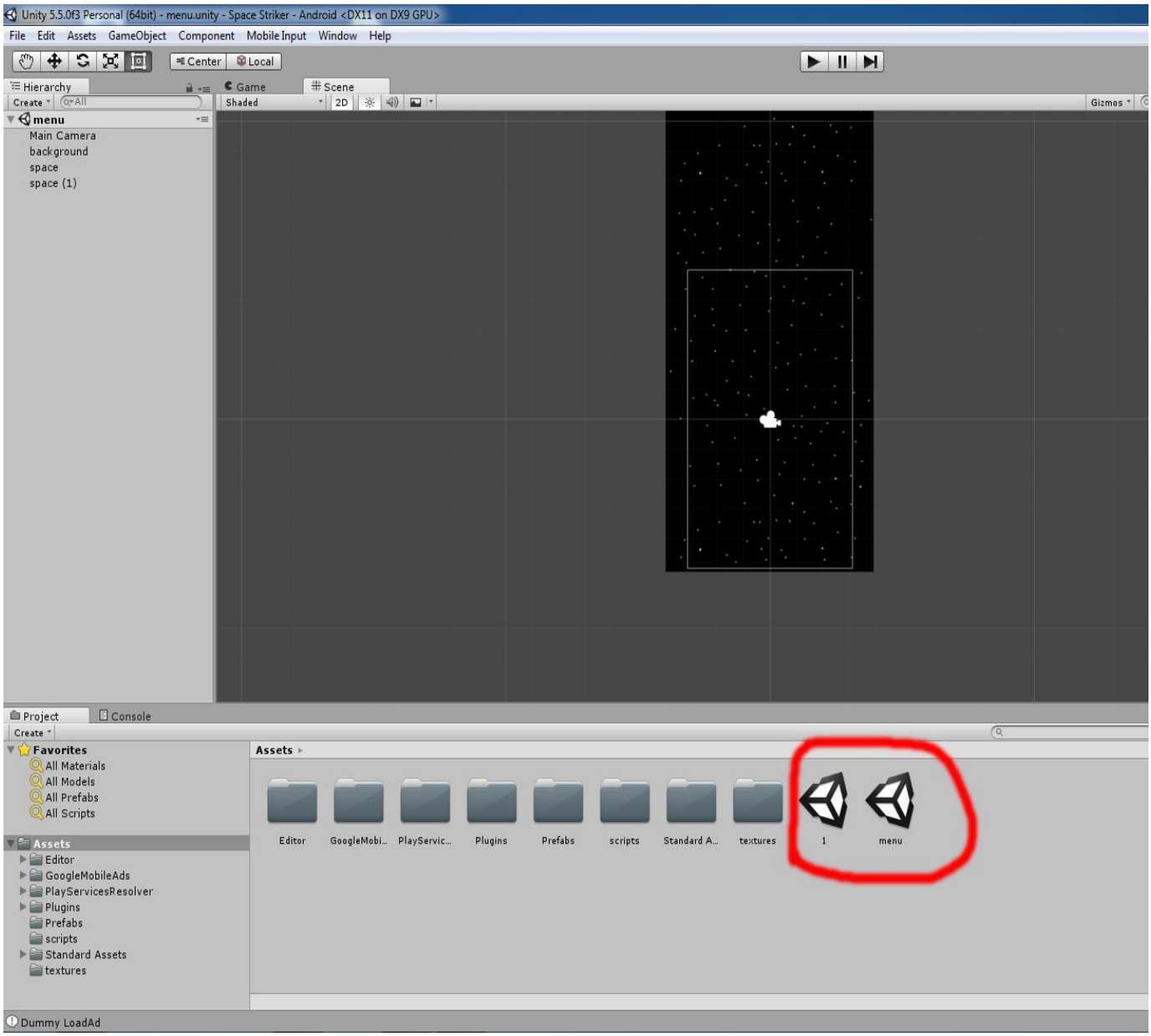
explosionasteroid, explosionenemy, explosionplayer - це анімації вибуху створені з використанням particle system;

gunactivator(s) — це капсули, які активуватимуть різний тип зброї у грі;

Решта префабів типу **laser** і тому подібні це і є зброя.

3.3. Розроблення головного меню гри

Мобільна гра «Галактика» буде включати в себе 2 сцени: *головне меню* та *ігрову сцену* (рис. 3. 13).



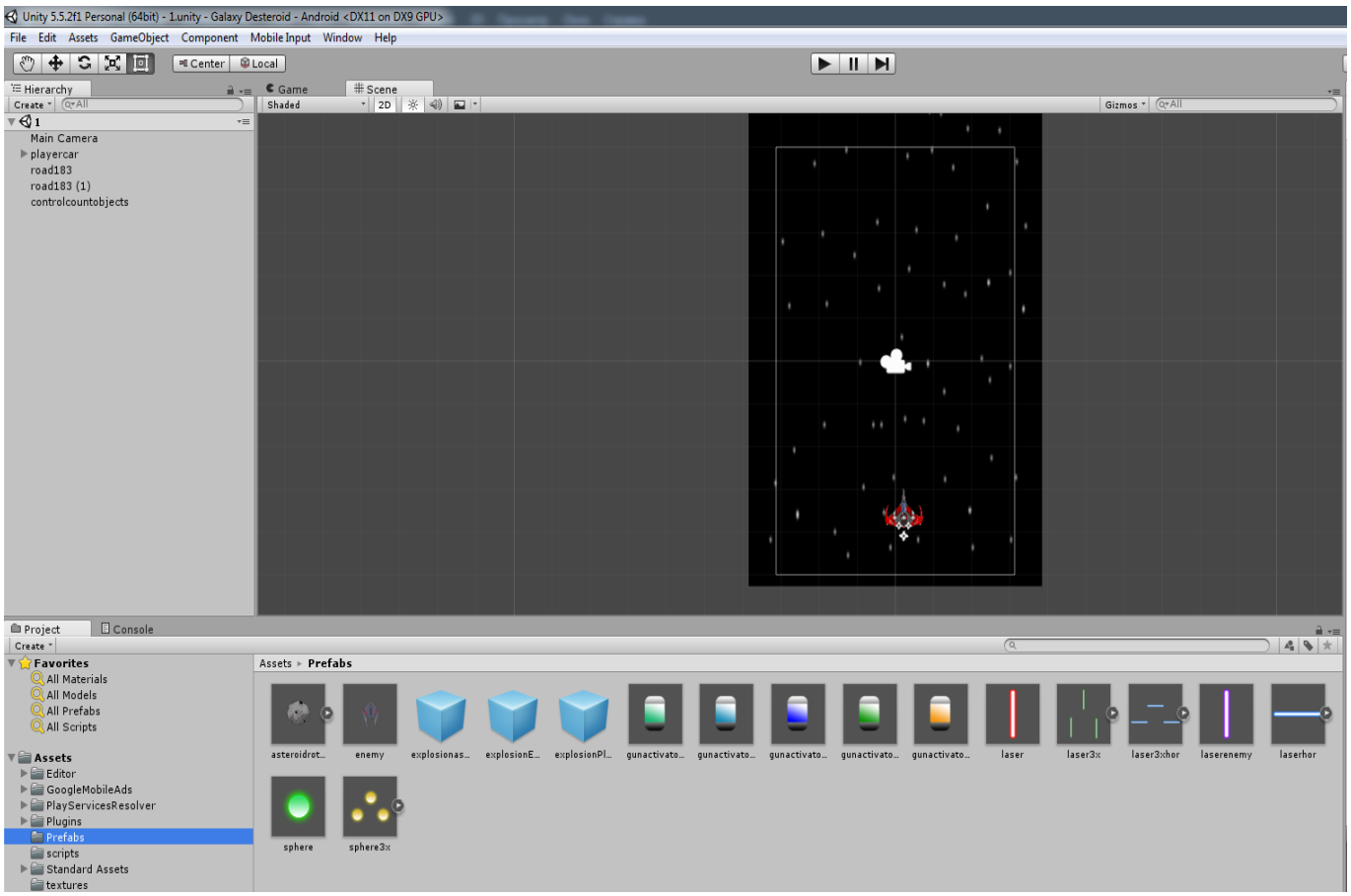


Рисунок 3. 15. Ігрова сцена

В якості фона використаємо спрайт з ім'ям **«space»**, на ньому зображений космос (рис. 3. 16).

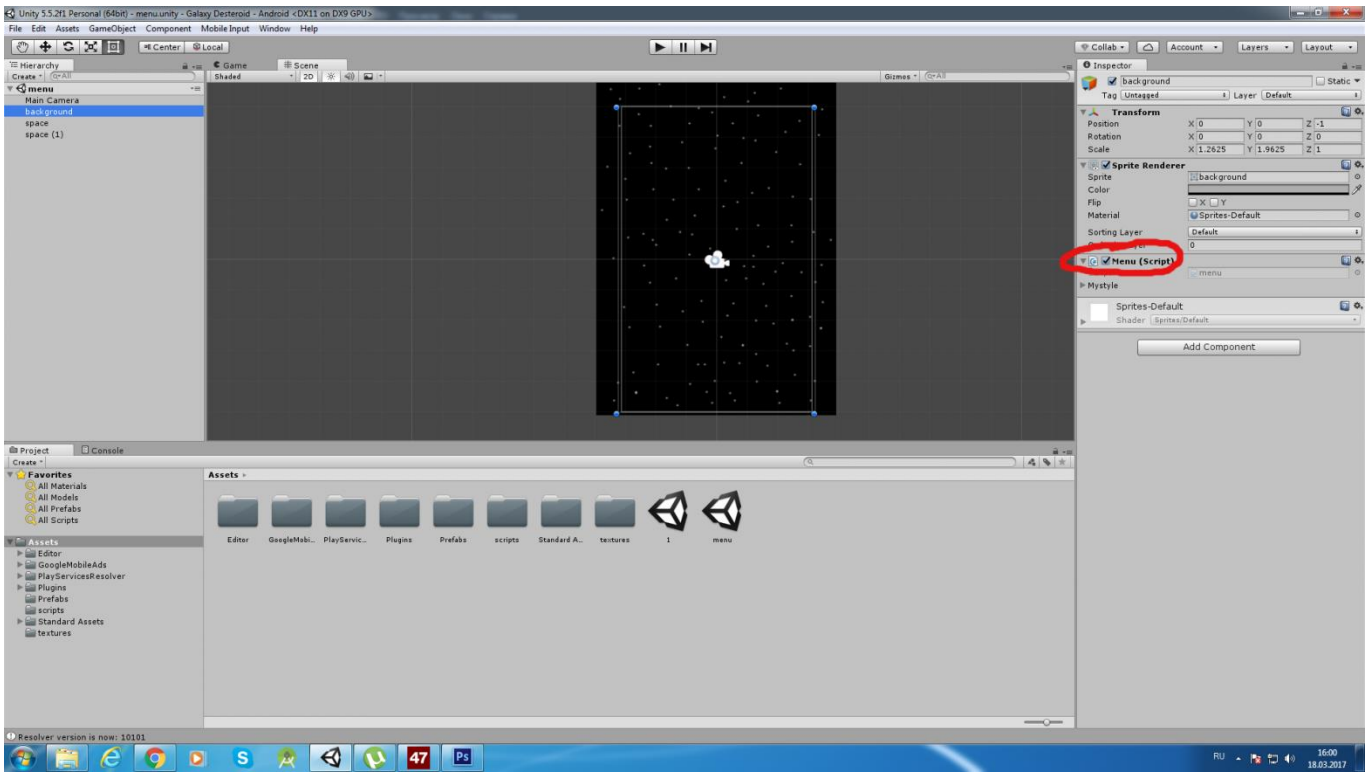


Рисунок 3. 16. Вибір фону гри

Далі створюємо скрипт **«menu.cs»** (клацаємо правою кнопкою → вибираємо **Create → C# Script**) і прикріплюємо його на **background**. **background** це спрайт зі 100% прозорістю, на якому розробляються основні елементи керування (**START/EXIT**), а **space** служить просто для декорації.

Вміст скрипту:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using UnityEngine.SceneManagement;


public class menu : MonoBehaviour {

	public GUIStyle mystyle;// оголошується для того щоб змінювати зображення GUI компонентів (шрифт, розмір і т. п.)
	string score;
	void Start ()
	{
		StreamReader scoredata = new StreamReader (Application.persistentDataPath + "/score1.gd");// створення файлової змінної
```

```
            score = scoredata.ReadLine ();//зчитування рядка
            scoredata.Close ();//закриття файлової змінної
      }

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

      }
      void OnGUI(){
            GUI.Box (new Rect (Screen.width*0.15f, Screen.height*0.8f,
Screen.width*0.7f, Screen.height*0.1f), "MAX DESTROYED:"+score,mystyle);
            if (GUI.Button (new Rect (Screen.width*0.15f, Screen.height*0.25f,
Screen.width*0.7f, Screen.height*0.1f), "START",mystyle))
            {
                  SceneManager.LoadScene (1);// Завантаження ігрової сцени
            }
            if (GUI.Button (new Rect (Screen.width*0.15f, Screen.height*0.4f,
Screen.width*0.7f, Screen.height*0.1f), "EXIT",mystyle))
            {
                  Application.Quit();//Вихід з гри
            }
      }

}
```

Ще не забуваємо повісити на **"space"** скрипт **activemenu**. Він служить для створення анімації руху фону меню. Потім створюємо копію **«space»** і вставляємо її трохи вище. Вміст скрипта **activemenu**:

```
using UnityEngine;
using System.Collections;

public class activemenu : MonoBehaviour {

      float speed=-0.1f;
      void Start () {

      }

      // Update is called once per frame
      void Update () {
            transform.Translate (new Vector3 (0f,speed,0f));
            if (transform.position.y < -12f)
            {
                  transform.position=new Vector3(0f,13f,0f);
            }
      }

}
```

Повинно вийти приблизно так, як показано на рис. 3. 17.

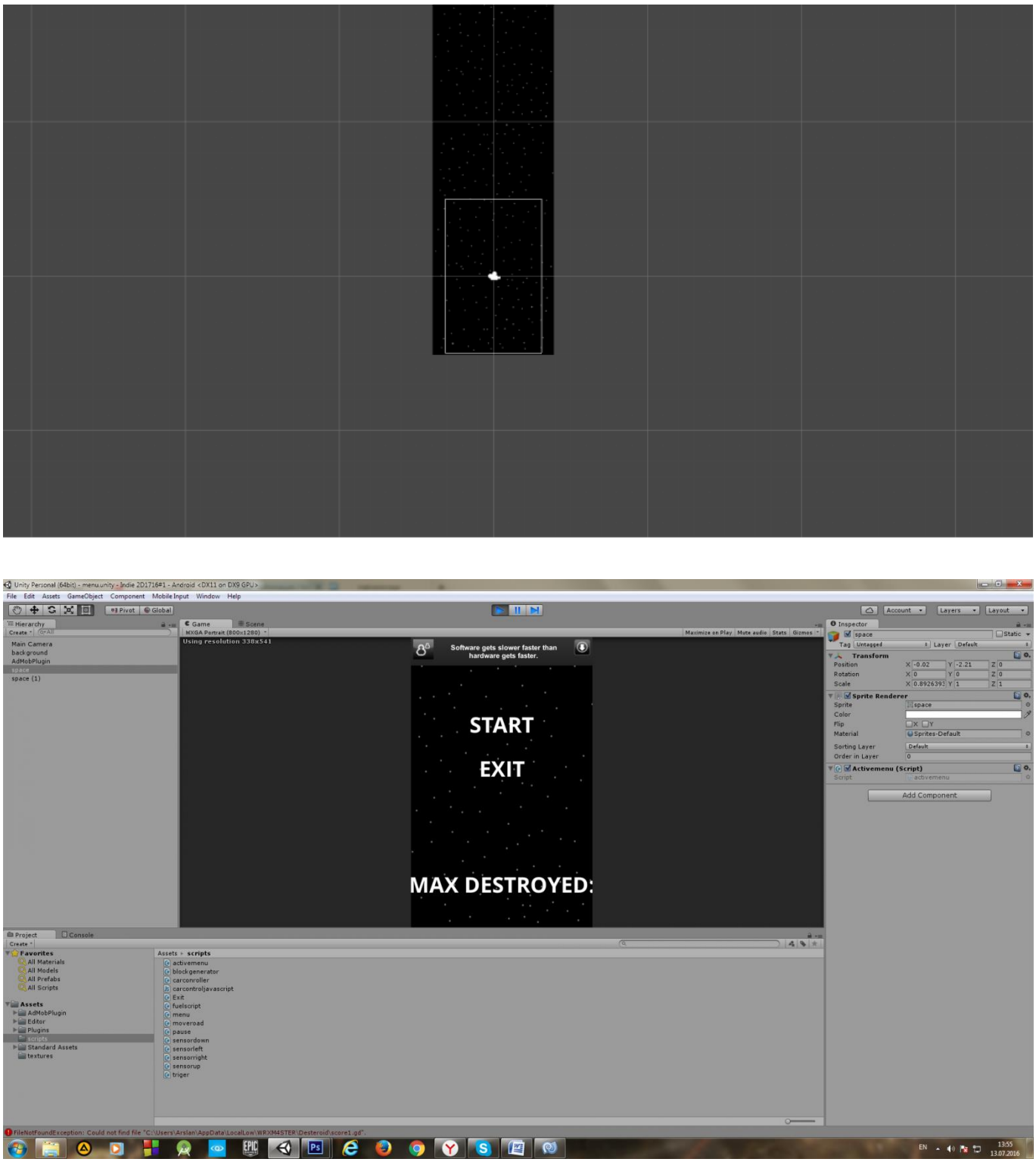


Рисунок 3. 17. Основні елементи керування меню гри «Галактика»

3.4. Створення ігрової сцени

Ігрова сцена буде складатися з наступних ключових об'єктів:

- космос (спрайт "**space**");
- гравець;
- метеорити;
- ворожі кораблі;
- інше (лазери та вибухи).

Космос організований так, як у головному меню. Тут можна нічого не чіпати (рис. 3. 18).

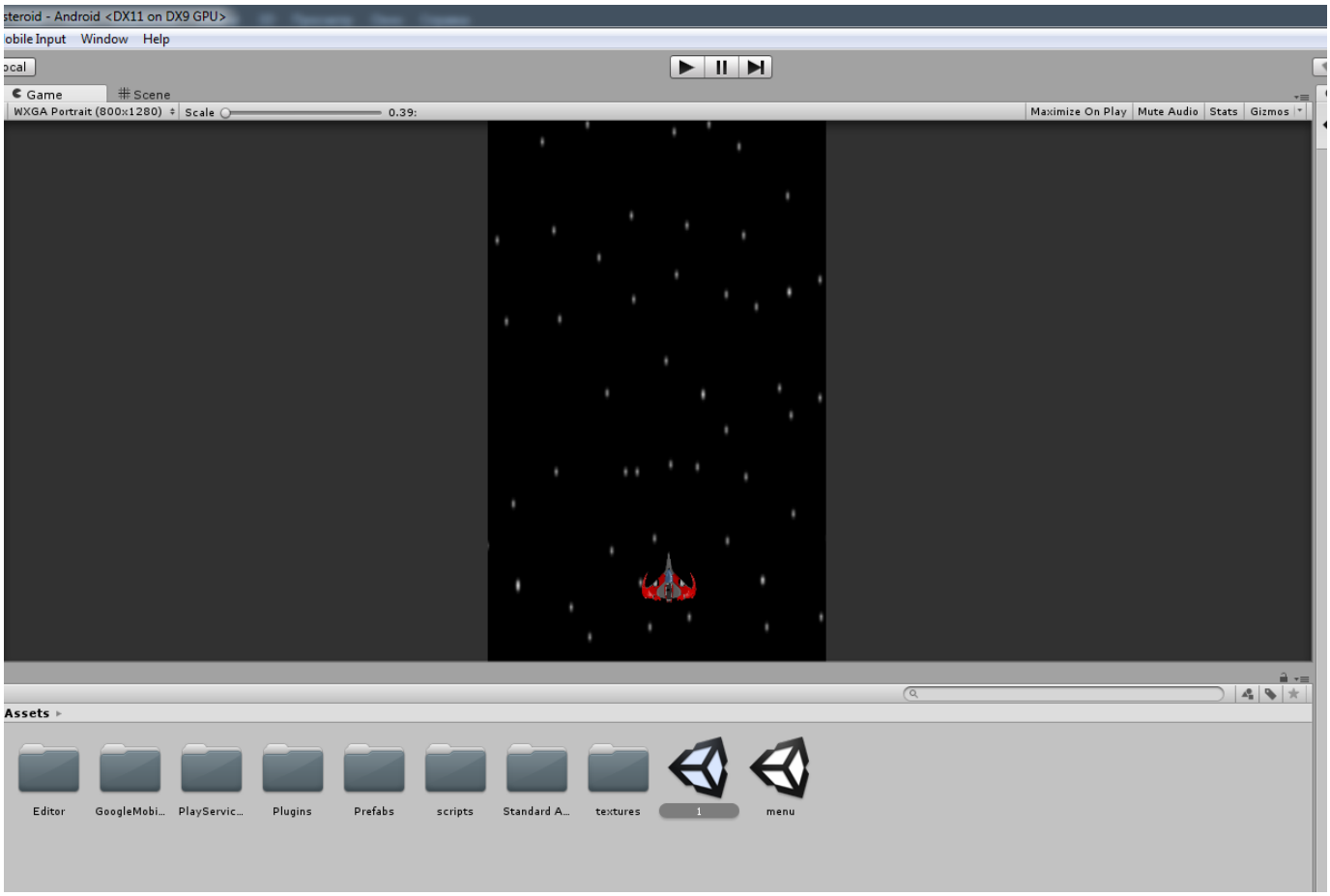


Рисунок 3. 18. Ігрова сцена

Далі потрібно звернути увагу на те, що в грі постійно генеруватимуться з префабів, такі об'єкти як метеорити, постріли та вороги. І ті об'єкти, які упустив гравець, потрібно видаляти, щоб вкотре не навантажувати пам'ять. Це можна зробити в такий спосіб. Створюємо новий ігровий об'єкт на сцені (у нас це "**controlcountobjects**"), додаємо до нього компонент **boxcollider** і розтягуємо його навколо ігрової зони (рис. 3. 19).

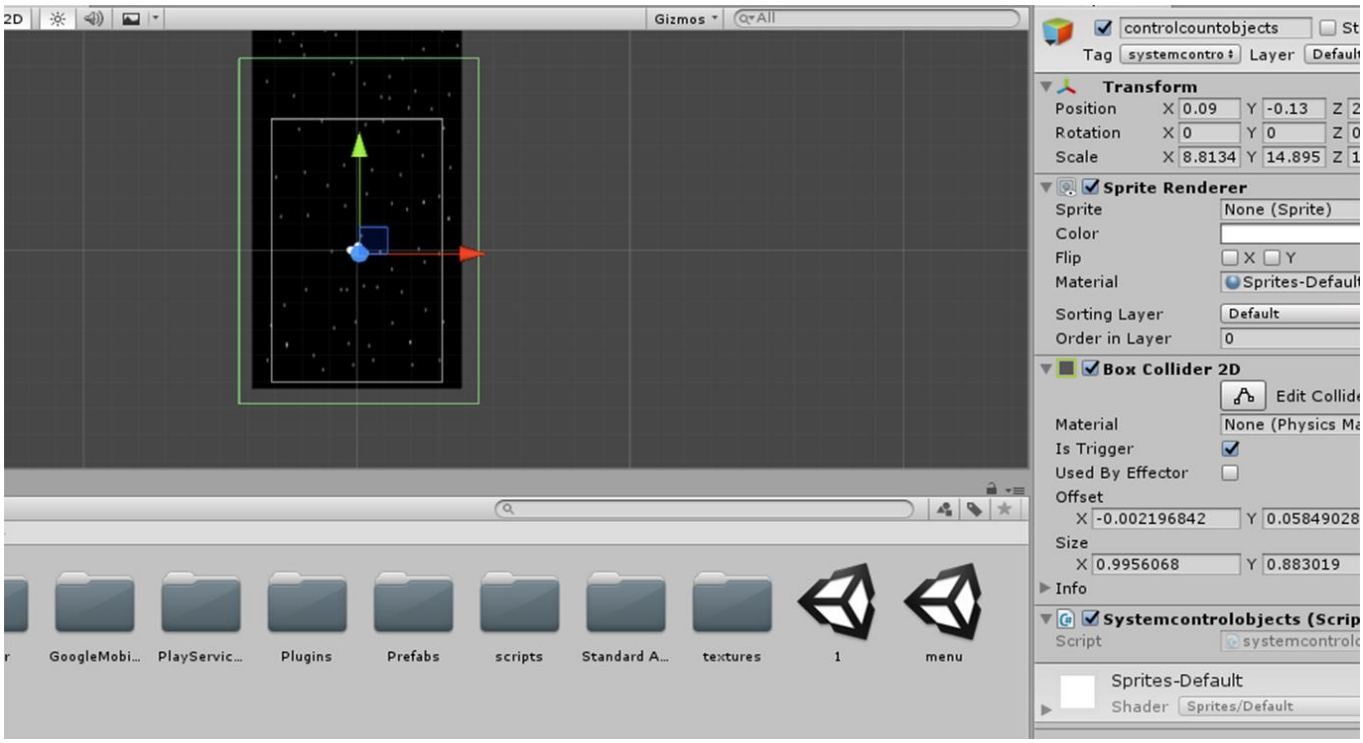


Рисунок 3. 19. Створення нового ігрового об'єкту на сцені

Далі додаємо на нього скрипт з наступним вмістом:

```
using UnityEngine;
using System.Collections;

public class systemcontrolobjects : MonoBehaviour {

	void Start ()
	{

	}

	void Update () {

	}
	void  OnTriggerExit2D(Collider2D col)
```

```
	{
		Destroy(col.gameObject);
	}

}
```

У цьому скрипті відбувається видалення елементів при виході з **box collider** об'єкта **controlcountobject**. Таким чином, утворюється певна зона при виході з якої відбувається видалення об'єктів.

Генерація метеоритів

Додаємо даний скрипт на камеру:

```
using UnityEngine;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class blockgenerator : MonoBehaviour {

	public GameObject asteroid;// Додаємо сюди метеорит із префабів
	float x,y,timer;
	float timerespawn=0.25f;// Період відродження метеоритів. За допомогою цієї змінної можна контролювати щільність трафіку метеоритів.
	bool trigtime=false;// Перемикач для відліку часу. Якщо значення true то тоді відраховується час для генерації наступного метеорита.
	public int score;// Підрахунок загальної кількості підбитих метеоритів за поточну гру і якщо це число перевищить рекорд тоді воно буде записано у файл рекорду. З цією змінною взаємодіятимуть інші скрипти. Кожен створений метеорит передаватиме команду, яка збільшуватиме цю змінну на одиницю.
	public float data;


	void Start ()
	{
		score = 0;//
		timer = timerespawn;
		StreamReader scoredata = new StreamReader (Application.persistentDataPath + "/score1.gd");
		data = float.Parse(scoredata.ReadLine ());
		scoredata.Close ();
	}

	void Update ()
	{
		if (timer==timerespawn)// Якщо встановлено час для відліку генерації метеориту, то:		{
			x = Random.Range (-2.5f, 2.5f);//1) Вибираємо рендомну координату, в якій з'явиться метеорит з динамічного діапазону.
```

```
                  Instantiate(asteroid, new
            Vector3(x,5.5f,2.17f),transform.rotation);//2) Генерація метеориту із
            префабу. x вибирається рандомно із зазначеного вище діапазону.
                  trigtime = true;//3) Активуємо перемикач для відліку часу. Після його
активації протягом часу вказаного в timerrespawn метеорити не будуть з'являтися.
            }
            if (trigtime==true)// Перевірка перемикача
            {
                  timer = timer-Time.deltaTime;// Відлік часу для генерації наступного
метеориту.
            }
            if (timer < 0)// Якщо час періоду спливає тоді:
            {
                  timer = timerespawn;//1) Скидання часу на число записане в timerespawn
                  trigtime = false;//2) Блокування відліку часу
                        // Тут відбувається повернення значень на "за умовчанням". При
їх скиданні знову відбуватиметься генерація метеоритів та відлік часу. І так по колу
            }
      }

}
```

Метеорити які з'являються на сцені різного розміру і ще вони обертаються. Це все тому, що метеорит реалізований за допомогою двох **GameObject**, де один знаходиться всередині іншого.

Зовнішній об'єкт **"asteroidrotate"** описує рух метеорита, містить **circle collider** і запускає ефект вибуху, у разі зіткнення, а **"aster"** рандомно при своєму створенні задає швидкість напряму обертання та розмір (рис. 3. 20).

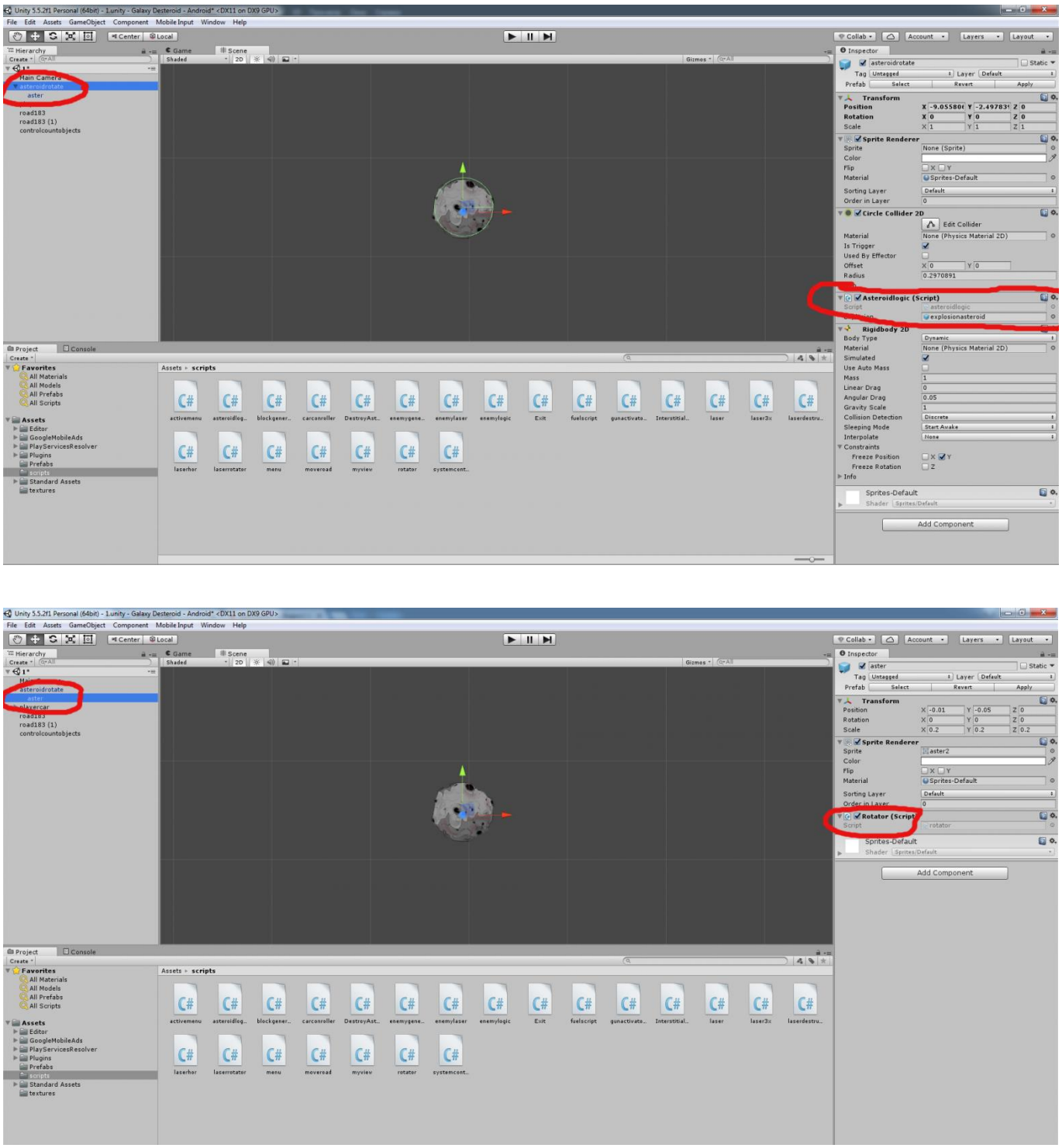


Рисунок 3. 20. Генерація метеоритів у грі «Галактика»

Скрипт для **asteroidrotate:**

```
using UnityEngine;
using System.Collections;

public class asteroidlogic : MonoBehaviour {
```

```
	public GameObject explosion;// Тут потрібно додати префаб вибуху
	float speedasteroid=-0.1f;// Швидкість руху метеориту

	void Start ()
	{

	}

	void Update ()
	{
		transform.Translate (new Vector3 (0, speedasteroid, 0f));// Рух метеорита
вниз по екрану із заздалегідь зазначеною швидкістю
	}

	void  OnTriggerEnter2D(Collider2D col)
	{
		if (col.tag == "systemcontrol")
		{
			return;// Повернення необхідне для уникнення конфлікту з об'єктом
controlcountobjects
		}
		if (col.tag == "laser") // Перевірка на попадання пострілів гравця у
метеорит
		{
			Destroy (col.gameObject);// Видаляємо сам постріл
			GameObject.Find ("Main Camera").GetComponent<blockgenerator>
().score++;// Додаємо одиницю до загальної кількості підбитих метеоритів
			Instantiate(explosion,
transform.position,transform.rotation);//Генеруємо вибух із префабів
			Destroy (this.gameObject);// Видалення метеориту
		}
	}

}
```

Скрипт для **aster:**

```
using UnityEngine;
using System.Collections;

public class rotator : MonoBehaviour {

	int f;
	float sc;
	void Start ()
	{
		f = Random.Range (-5, 5);
		sc = Random.Range (0.1f,0.2f);
		transform.localScale = new Vector3 (sc,sc,sc);//розмір
	}

	void Update ()
	{
		transform.rotation *= Quaternion.AngleAxis (f,new
Vector3(0,0,1));//обертання
	}
```

```
}
```

Гравець

Завантажуємо спрайт із зорельотом на ігрову сцену, додаємо на нього **box collider** (не забуваємо відзначати **Is Triger**), додаємо **rigibody** (потрібно заморозити Y координату). Все це необхідно, щоб гравець міг взаємодіяти з іншими ігровими об'єктами (рис. 3. 21).

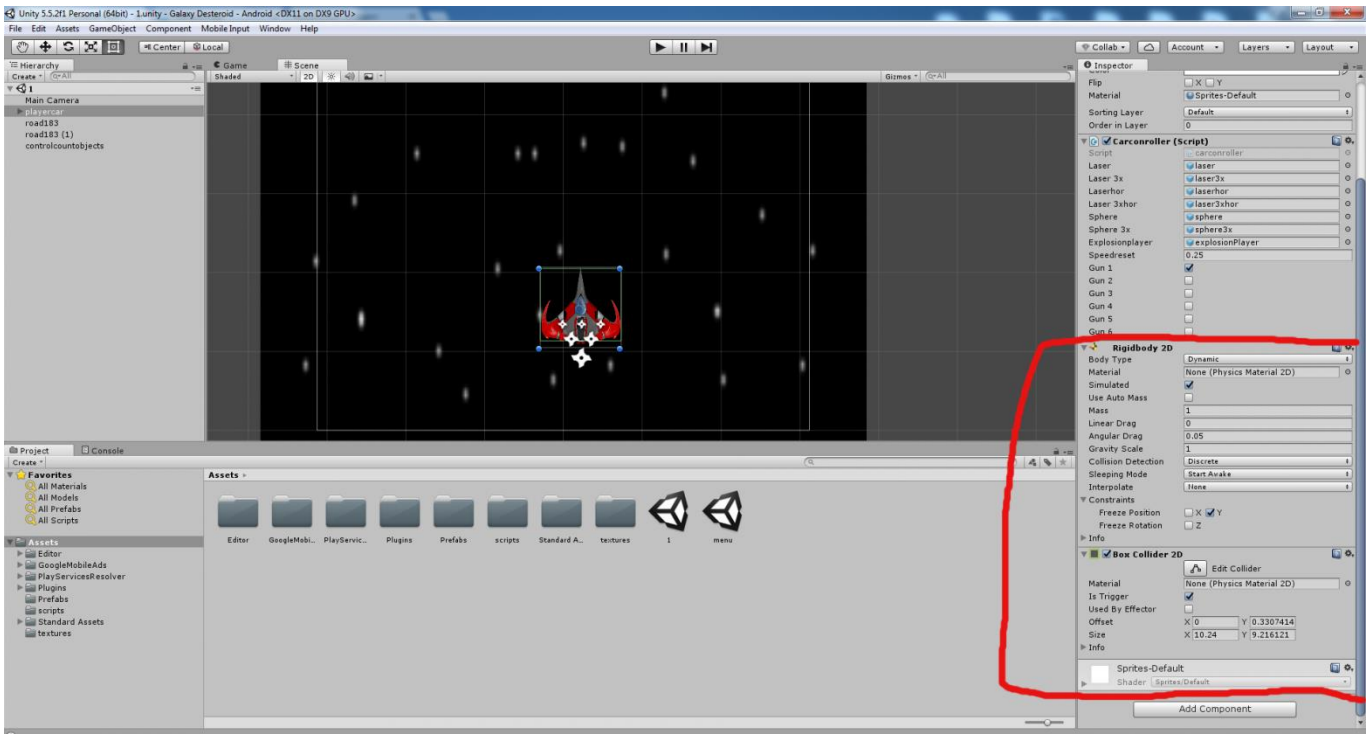


Рисунок 3. 21. Спрайт із зорельотом на ігровій сцені

Можна ще додати компонент **particle system**, який буде зображати струмінь диму з двигуна (рис. 3. 22).

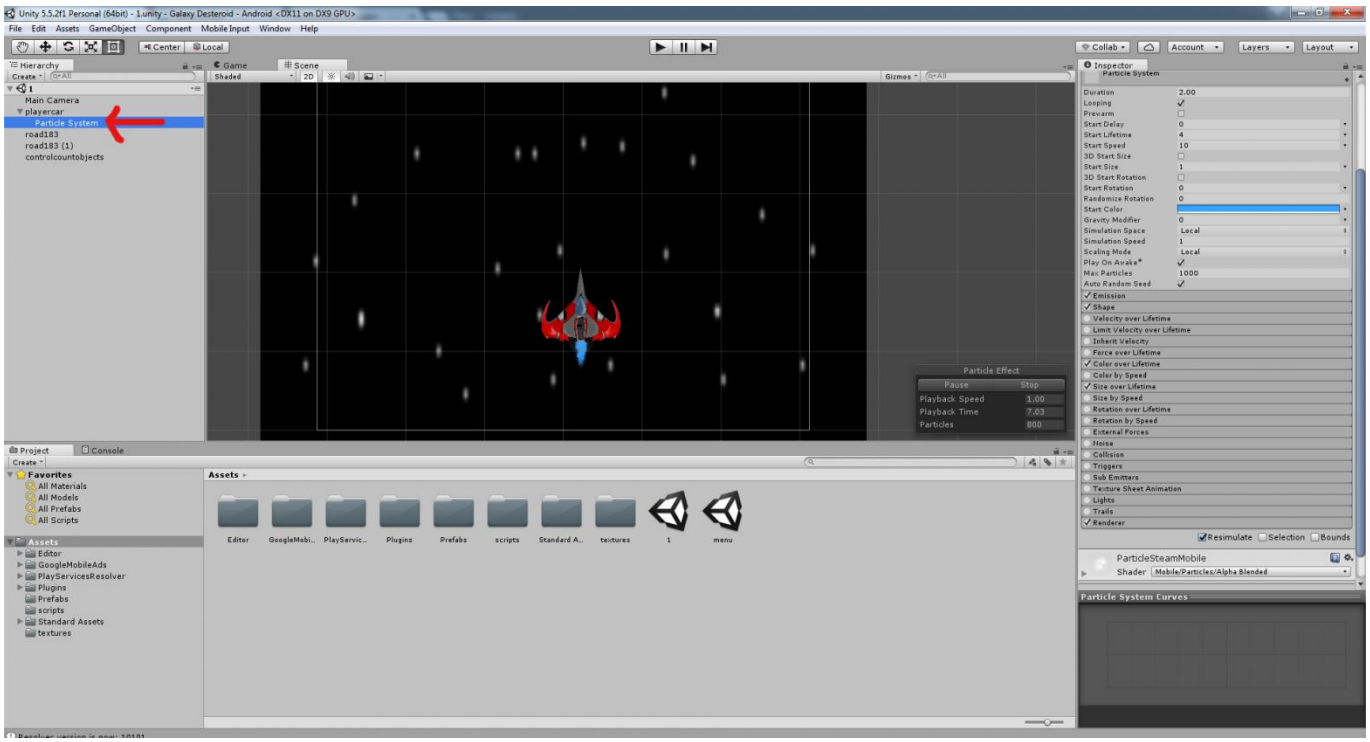


Рисунок 3. 22. Додавання компоненту particle system

У скрипті для гравця відбувається встановлення "зв'язку" з такими префабами як **explosionplayer**, **laser**, **laser3x**, **laser3xhor** і т. п., а також організація управління та підбір «капсул» для активації інших видів зброї. Керування влаштоване так що ігровий об'єкт просто рухається за пальцем користувача та попутно веде вогонь.

Вміст скрипту:

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class carconroller : MonoBehaviour
{
	public GameObject laser;// Ця змінна буде пов'язана з префабом laser
	public GameObject laser3x;// Ця змінна буде пов'язана з префабом laser3x
	public GameObject laserhor;// Ця змінна буде пов'язана з префабом laserhor
	public GameObject laser3xhor;// Ця змінна буде пов'язана з префабом laser3xhor
	public GameObject sphere;// Ця змінна буде пов'язана з префабом sphere
	public GameObject sphere3x;// Ця змінна буде пов'язана з префабом sphere3x
	public GameObject explosionplayer;// Ця змінна буде пов'язана з префабом
explosionplayer

	float x,y,z,x1,y1;
```

```
	bool trigtime=false;
	public float speedreset=0.25f;// період перезарядки зброї в мілісекундах
	float timer;
	Vector2 startpos;
	Vector2 startcar;
	  // об'єкти laser...sphere3x є зброєю, а нижче булеві змінні їх відповідні активатори
	public bool gun1 = true;// За замовчуванням активована 1 зброя (laser), вона є базовою та має необмежений боєзапас, на відміну від інших видів (laser3x...)
	public bool gun2 = false;
	public bool gun3 = false;
	public bool gun4 = false;
	public bool gun5 = false;
	public bool gun6 = false;
	  // При активації зброї відмінної від базового боєзапас закінчуватиметься. Коли боєзапас закінчиться, буде автоматично активована базова зброя.
	int guncount=0;// Кількість боєзапасу для зброї, яка відрізняється від базової. Для кожного з додаткового виду зброї встановлюється своє число боєзапасу.

	void Start ()
	{
		timer = speedreset;
		y = laser.transform.position.y;
		z = laser.transform.position.z;
	}

	public void Update ()
	{
		if (timer < 0)
		{
			timer = speedreset;
			trigtime = false;
		}

		if (Input.GetMouseButton(0))// Відстеження натискання на екран
		{
			Vector2 pos = Camera.main.ScreenToWorldPoint (Input.mousePosition);// Запис у змінну pos координат місця, де відбулося торкання екрана.
			transform.position = pos;// присвоєння позиції ігровому об'єкту координат із змінної pos
			transform.position = new Vector2 (transform.position.x,transform.position.y+1f);// коригування координат гравця (необов'язково)
			if (timer==speedreset)// перевірка закінчення часу (час перезарядки)
			{
				if (gun1 == true)// перевірка базової зброї
				{
					Instantiate (laser, new Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);// Генерація пострілів із префабу laser у місці поточної позиції гравця (з деякими поправками)
					trigtime = true;
				}
				if (gun2 == true && guncount > 0)// У разі активації 2-ї зброї постріли генеруватимуться за цим кодом
				{
					guncount--;// Зниження кількості боєприпасів
```

```
                                if (guncount == 0) // Якщо боєприпаси закінчаться то
активуємо 1 зброю, а інші блокуємо
                                {
                                        gun1 = true;
                                        gun2 = false;
                                        gun3 = false;
                                        gun4 = false;
                                        gun5 = false;
                                        gun6 = false;
                                }
                                Instantiate (laser3x, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);//
Генерація пострілів із префабу laser3x
                                trigtime = true;
                        }
                                 // Нижче для інших видів "озброєння" все відбувається
аналогічно
                        if (gun3 == true && guncount > 0)
                        {
                                guncount--;
                                if (guncount == 0)
                                {
                                        gun1 = true;
                                        gun2 = false;
                                        gun3 = false;
                                        gun4 = false;
                                        gun5 = false;
                                        gun6 = false;
                                }
                                Instantiate (laserhor, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                                trigtime = true;
                        }
                        if (gun4 == true && guncount > 0)
                        {
                                guncount--;
                                if (guncount == 0)
                                {
                                        gun1 = true;
                                        gun2 = false;
                                        gun3 = false;
                                        gun4 = false;
                                        gun5 = false;
                                        gun6 = false;
                                }
                                Instantiate (laser3xhor, new
Vector2(transform.position.x,transform.position.y+2f), transform.rotation);
                                trigtime = true;
                        }
                        if (gun5 == true && guncount > 0)
                        {
                                guncount--;
                                if (guncount == 0)
                                {
                                        gun1 = true;
                                        gun2 = false;
                                        gun3 = false;
                                        gun4 = false;
```

```
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (sphere, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                              trigtime = true;
                        }
                        if (gun6 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (sphere3x, new
Vector2(transform.position.x,transform.position.y+2f), transform.rotation);
                              trigtime = true;
                        }
                  }
                  if (trigtime == true)
                  {
                        timer = timer - Time.deltaTime;// Відлік часу для перезаряджання
                  }
            }
      }

      void  OnTriggerEnter2D(Collider2D col)
      {
            if (col.tag == "systemcontrol")
            {
                  return;// Повернення необхідне для уникнення конфлікту з об'єктом
controlcountobject
            }
            if (col.tag == "gunactivator2")// перевірка на перетин з "капсулою" для
gunactivator2
            {
                  gun2 = true;// активація другої зброї
                       guncount = 85;// встановлення кількості боєприпасів
                  gun1 = false;// відключення інших видів зброї
                  gun3 = false;
                  gun4 = false;
                  gun5 = false;
                  gun6 = false;
                  Destroy (col.gameObject);// знищення "капсули" з якою стався перетин
            return;
            }
                // Нижче все аналогічно
            if (col.tag == "gunactivator3")
            {
                  gun3 = true; guncount = 50;
                  gun1 = false;
                  gun2 = false;
                  gun4 = false;
```

```
                gun5 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
            }
            if (col.tag == "gunactivator4")
            {
                gun4 = true; guncount = 15;
                gun1 = false;
                gun3 = false;
                gun2 = false;
                gun5 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
            }
            if (col.tag == "gunactivator5")
            {
                gun5 = true; guncount = 100;
                gun1 = false;
                gun3 = false;
                gun4 = false;
                gun2 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
            }
            if (col.tag == "gunactivator6")
            {
                gun6 = true; guncount = 50;
                gun1 = false;
                gun3 = false;
                gun4 = false;
                gun5 = false;
                gun2 = false;
                Destroy (col.gameObject);
                return;
            }
                // Якщо зіткнення з "капсулами" не сталося це означає, що гравець
зіткнувся з лазером супротивника, метеоритом або з ворожим зорельотом. Значить, потрібно
видалити гравця зі сцени
            Handheld.Vibrate ();//активація вібрації (для смартфонів)
            if (GameObject.Find ("Main Camera").GetComponent<blockgenerator>
().score>GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().data)// Якщо
кількість збитих метеоритів у поточній ігровій сесії вище, ніж у файлі з ігровим
рекордом, то робимо запис нового рекорду у файл
            {
                StreamWriter scoredata=new
StreamWriter(Application.persistentDataPath + "/score1.gd");
                scoredata.WriteLine(GameObject.Find ("Main
Camera").GetComponent<blockgenerator> ().score);
                scoredata.Close();
            }
            Instantiate (explosionplayer, transform.position, transform.rotation);//
Генерація вибуху зорельоту гравця
            Destroy (col.gameObject);// Видалення об'єкта з яким відбувся перетин
            Destroy(this.gameObject);// Видалення гравця з ігрової сцени
        }
```

}

На рисунку 3. 23. зображені активатори зброї та результат роботи скрапта:

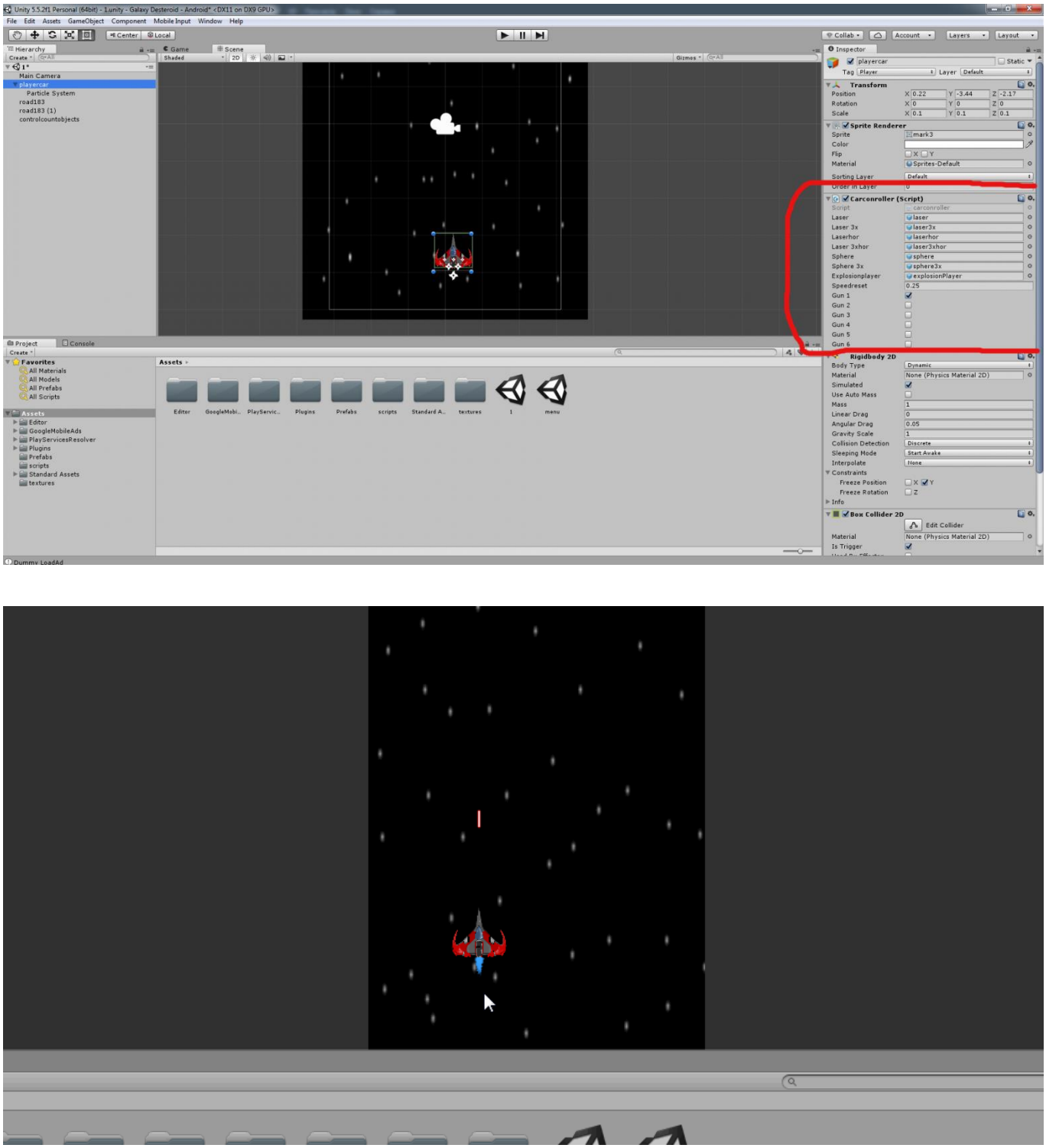


Рисунок 3. 23. Активатори зброї та результат роботи скрапта

Коли гравець буде видалений, а у файл буде записаний новий рекорд знищених метеоритів, тоді потрібно переходити на головне меню гри.

Створюємо скрипт з наступним вмістом та підключаємо його на камеру:

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class Exit : MonoBehaviour {

	public GameObject target;// Додаємо сюди зореліт гравця
	float timer=3f;

	void Start ()
	{

	}

	void Update ()
	{
		if (Input.GetKey (KeyCode.Escape))// Якщо буде натиснуто кнопку назад під час гри, то:
		{
			if (GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().score>GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().data)
			{
				StreamWriter scoredata=new StreamWriter(Application.persistentDataPath + "/score1.gd");
				scoredata.WriteLine(GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().score);
                                // Запис у файл рекорду змінну score з "blockgenerator", якщо вона більша за неї
				scoredata.Close();
			}
			SceneManager.LoadScene (0);// Завантаження головного меню
		}
		if (!GameObject.Find ("playercar"))// Якщо гравець був видалений, то після закінчення часу (в секундах) вказаного в timer буде відкрито головне меню
		{
			timer = timer - Time.deltaTime;
			if (timer < 0)
			{
				SceneManager.LoadScene (0);
			}
		}
	}

}
```

Даний скрипт спостерігає за станом гравця.

Вибухи

Ефект вибуху можна створити за допомогою системи **Particle System** (рис. 3. 24). Але тоді він буде зацикленим і вічно повторюватиметься. Щоб цього не було, на префаб вибуху слід додати наступний скрипт.

```
using UnityEngine;
using System.Collections;

public class DestroyAsteroid : MonoBehaviour {

	void Start ()
	{

	}

	void Update ()
	{
		Destroy (this.gameObject,0.4f);// Видалення ігрового об'єкта після його створення. Час життя 400 мілісекунд. Значення можна змінити.
	}

}
```

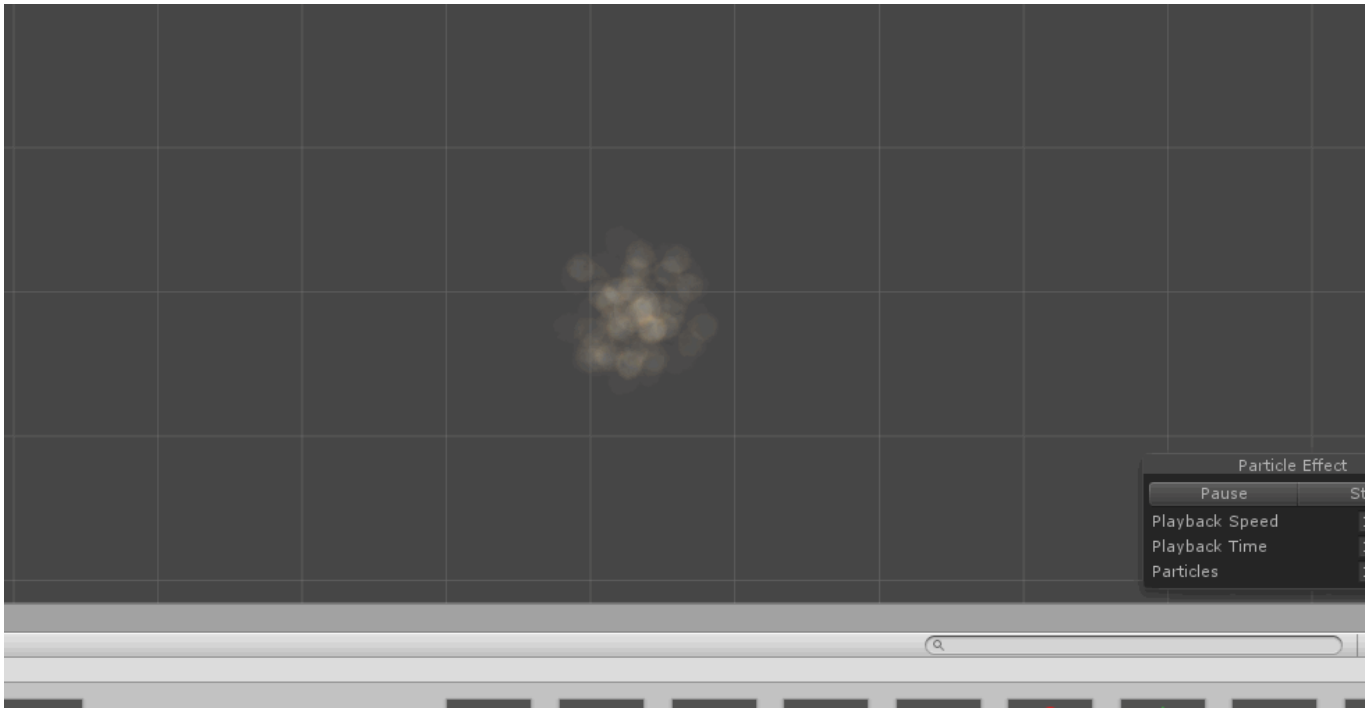


Рисунок 3. 24. Ефект вибуху створений з допомогою системи Particle System

Постріли

Постріли генеруються із місця знаходження гравця під час його руху (рис. 3. 25). У скрипті для пострілу потрібно задати лише швидкість та напрямок.

```
using UnityEngine;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class laser : MonoBehaviour {


	float speedlaser=0.5f;

	void Start ()
	{

	}

	void Update ()
	{
		transform.Translate (new Vector3 (0, speedlaser, 0f));
	}

}
```

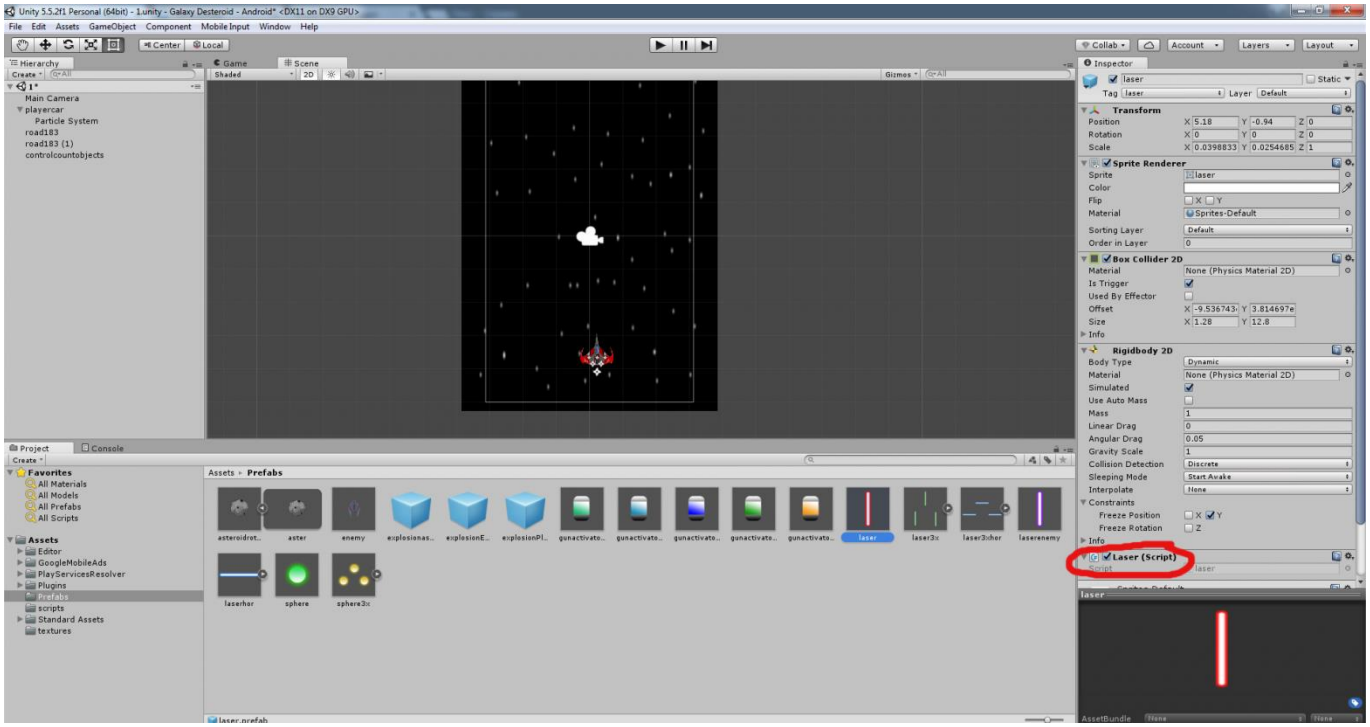


Рисунок 3. 25. Генерація пострілів

<u>Генерація супротивників</u>

Генерація противників влаштована аналогічно до генерації метеоритів (рис. 3. 26). Вміст скрипта **Enemygenerator**:

```
using UnityEngine;
using System.Collections;

public class enemygenerator : MonoBehaviour {

	public GameObject enemy;
	bool trigtime=false;
	float speedreset=2f;
	float timer,x;

	void Start ()
	{
		timer = speedreset;
	}

	void Update ()
	{
		if (timer < 0)
		{
			timer = speedreset;
			trigtime = false;
		}
		if (timer == speedreset)
		{
			x = Random.Range (-2.5f, 2.5f);// Задаємо місце розташування
			Instantiate(enemy, new
	Vector2(x,5.5f),transform.rotation);// Генерація супротивника з префабу
			trigtime = true;
		}
		if (trigtime == true)
		{
			timer = timer - Time.deltaTime;
		}
	}

}
```

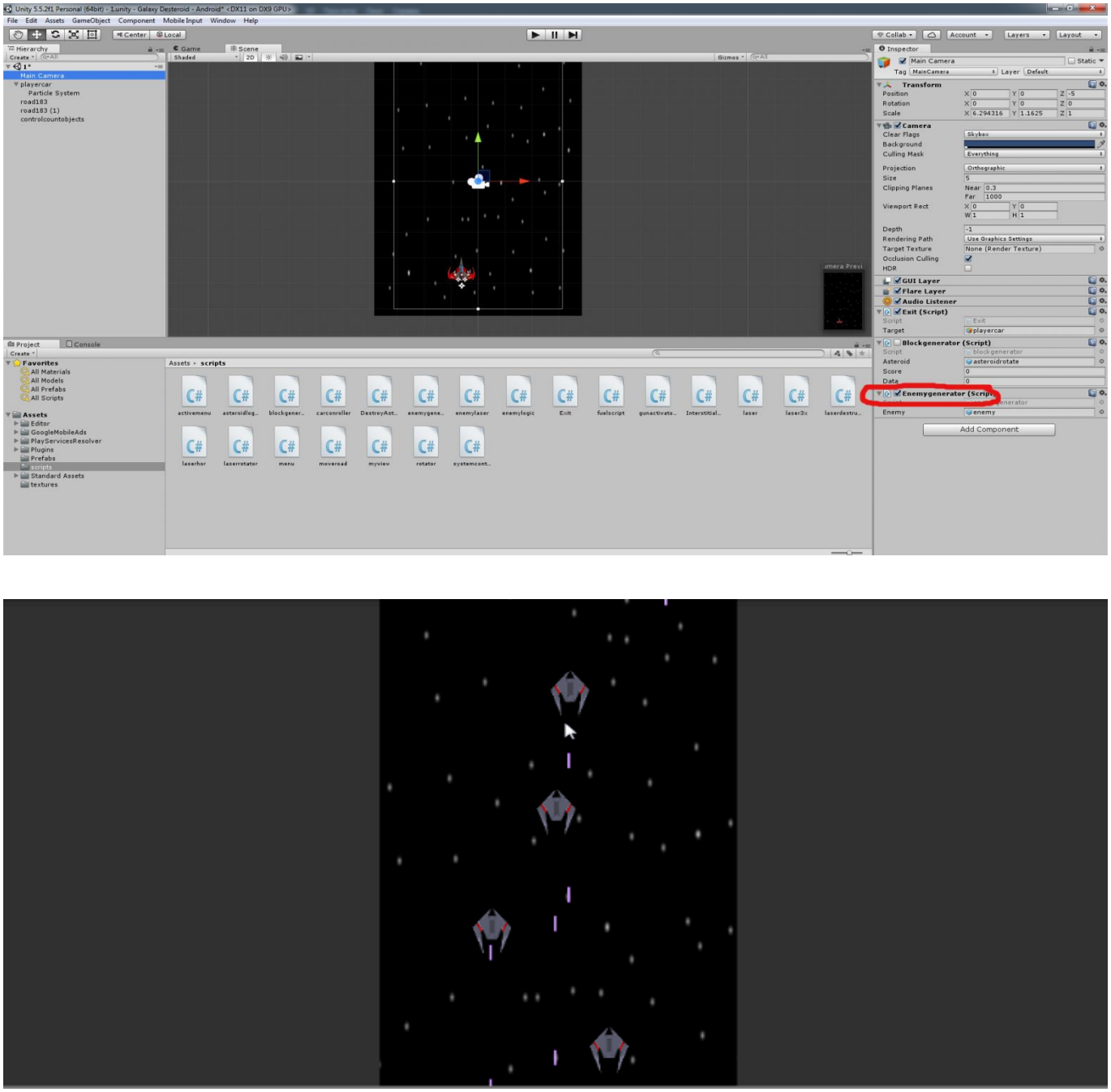
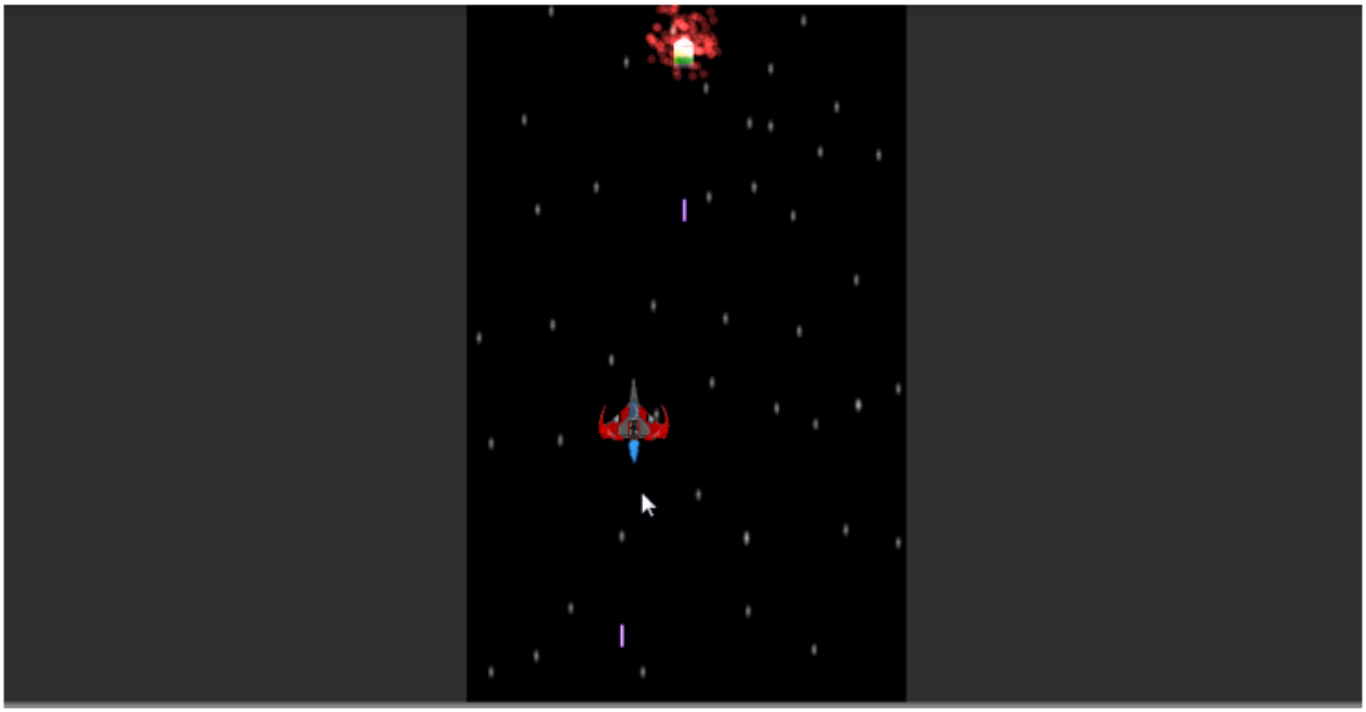


Рисунок 3. 26. Генерація супротивників

Логіка супротивника

Поведінка противника організована таким чином, що він просто летить по прямій траекторії, веде вогонь і у разі враження може залишити капсулу для активації гравцем нової зброї (рис. 3. 27).



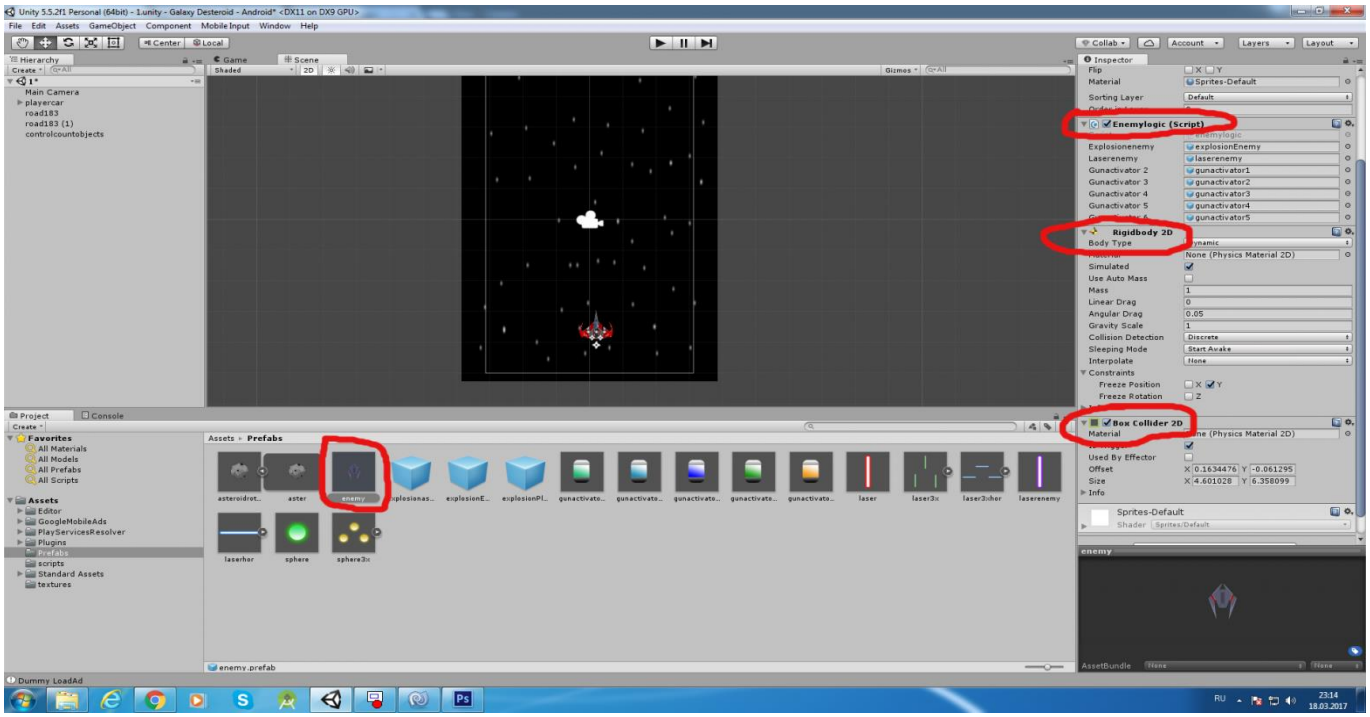


Рисунок 3. 27. Поведінка противника

```
using UnityEngine;
using System.Collections;

public class enemylogic : MonoBehaviour {

      public GameObject explosionenemy;// Префаб вибуху
      public GameObject laserenemy;// Префаб пострілу
        // Префаби капсул для активації інших видів зброї
```

```
	public GameObject gunactivator2;
	public GameObject gunactivator3;
	public GameObject gunactivator4;
	public GameObject gunactivator5;
	public GameObject gunactivator6;
	  //
	bool trigtime=false;
	float speedreset=1.5f;// Час перезаряджання
	float timer;
	float speedenemy = -0.02f;// Швидкість та напрямок

	float x;

	void Start ()
	{
		timer = speedreset;
	}


	void Update ()
	{
		if (timer < 0)
		{
			timer = speedreset;
			trigtime = false;
		}
		if (timer == speedreset)
		{
			Instantiate (laserenemy, new
Vector2(transform.position.x,transform.position.y-0.4f), transform.rotation);
			trigtime = true;
		}
		if (trigtime == true)
		{
			timer = timer - Time.deltaTime;
		}
		transform.Translate (new Vector3 (0, speedenemy, 0f));
	}

	void  OnTriggerEnter2D(Collider2D col)
	{
		if (col.tag == "systemcontrol")
		{
			return;
		}
		if (col.tag == "Player")
		{
			Instantiate (explosionenemy, transform.position, transform.rotation);
			Destroy(this.gameObject);
		}
		if (col.tag == "laser")
		{
			x = Random.Range (0f, 100f);// Генерується будь-яке число від 0 до 100
			if (x > 1f && x < 5f) // якщо число в діапазоні від 1 до 5, то
створюється капсула gunactivator2
		{
				Instantiate (gunactivator2, transform.position,
transform.rotation);
```

```
                    }
                        // Внизу все аналогічно
                    if (x > 20f && x < 25f)
                    {
                        Instantiate (gunactivator3, transform.position,
transform.rotation);
                    }
                    if (x > 40f && x < 45f)
                    {
                        Instantiate (gunactivator4, transform.position,
transform.rotation);
                    }
                    if (x > 60f && x < 65f)
                    {
                        Instantiate (gunactivator5, transform.position,
transform.rotation);
                    }
                    if (x > 80f && x < 85f)
                    {
                        Instantiate (gunactivator6, transform.position,
transform.rotation);
                    }
                    Instantiate (explosionenemy, transform.position, transform.rotation);
                    Destroy(this.gameObject);
                }
        }

}
```

Кожен префаб «капсули» зі зброєю просто рухається до гравця і у разі перетину знищується (рис. 3. 28).

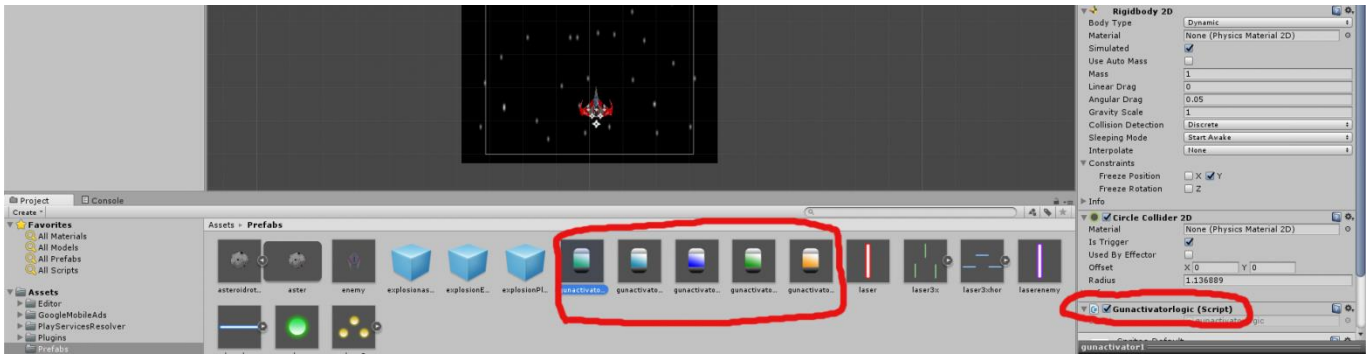


Рисунок 3. 28. Капсули зі зброєю

```
using UnityEngine;
using System.Collections;

public class gunactivatorlogic : MonoBehaviour {

        float speed = -0.025f;
```

```
        void Start ()
        {

        }

        void Update ()
        {
                transform.Translate (new Vector3 (0, speed, 0f));
        }

}
```

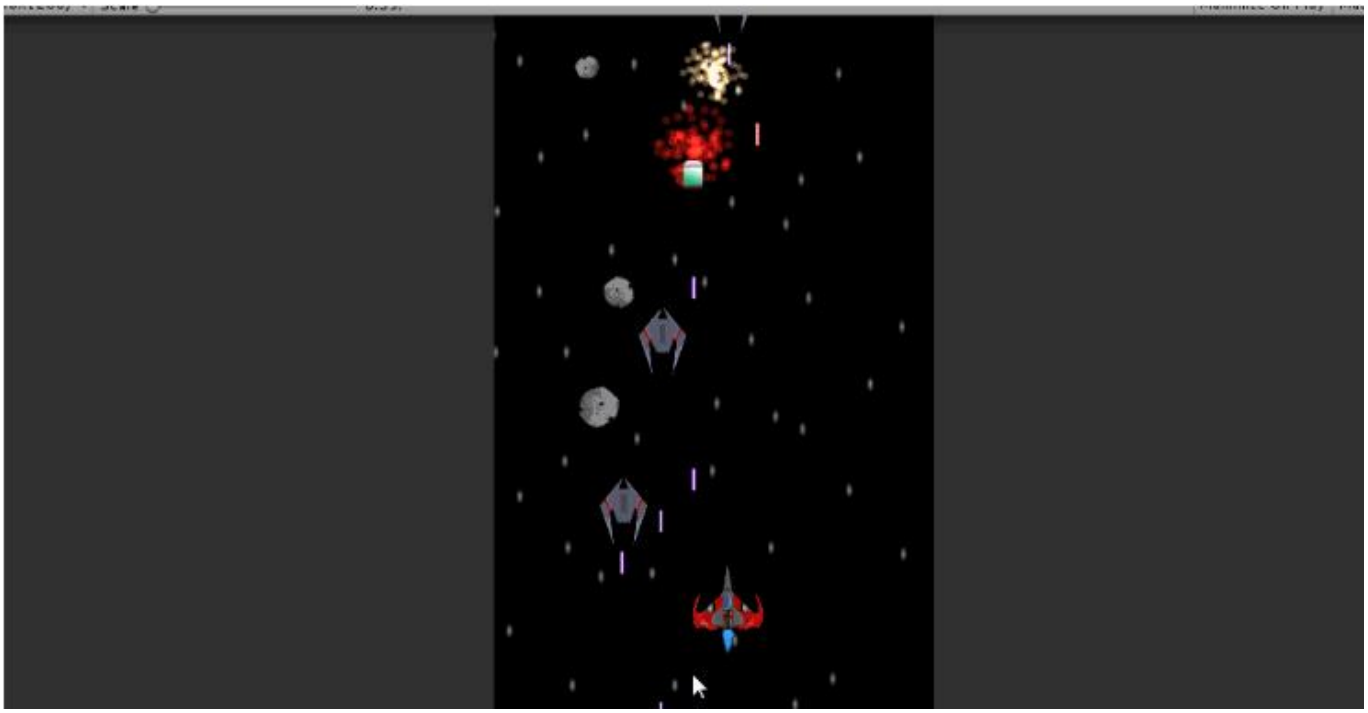


Рисунок 3. 29. Тестування мобільної гри «Галактика»

3.5. Вимогоги до програмного та апаратного забезпечення

Для розроблення мобільної гри «Галактика» необхідне відповідне програмне забезпечення та комп'ютерна техніка: ноутбук. Його перевага в порівнянні з персональним комп'ютером - мобільність - можна показати частину виконаної роботи, невисока вартість, немає необхідності купувати додатково дорогий монітор.

Отже для успішної роботи необхідні наступні характеристики технічного та системного забезпечення, які приведені в таблиці 3. 1.

Таблиця 3. 1 Основні характеристики технічного та системного забезпечення

Тип	Ноутбук
Операційна система	Windows 10 HomeBasic
Процесор	Core i5 460M 2533 МГц
Ядро процесора	Arrandale
Кількість ядер процесора	2
Об’єм кеша	512 Кб
Пам’ять	4 Гб DDR3
Екран	15.6 дюймів, 1366x768, широкоформатний
Відеопроцесор	Intel GMA HD
Безпровідний зв’язок	Wi-Fi IEEE 802.11n, Bluetooth 3.0 HS
Інтерфейси	USB 2.0x3, VGA (D-Sub), HDMI, вхід мікрофонний, вихід аудіо/навушники, LAN (RJ-45)

Для роботи використовувалися безкоштовні програми: Unity3D, C#.

ВИСНОВКИ

Під час виконання дипломної роботи було вивчено низку джерел з практичного та професійного програмування. У результаті були набуті знання необхідні для розроблення власного продукту – мобільної гри «Галактика». Так само в ході розв'язання поставлених завдань було здійснено знайомство з принципами розвитку індустрії мобільних ігор, проаналізовано досвід роботи в галузі розроблення демонстраційних мобільних ігор.

Також був вивчений багатоплатформовий інструмент та рушій Unity3D. Внаслідок чого стало можливим розроблення невеликої мобільної гри. Цей процес докладно описаний у роботі.

Під час виконання дипломної роботи було проаналізовано основні тенденції у розробці мобільних застосунків, розглянуто аналоги даної гри, виявлено їх недоліки, на основі яких створено гру згідно із заявленим технічним завданням. Було розроблено графічне забезпечення гри, структуру гри, та саму гру. Проведене тестування розробленої мобільної гри, а також були розробленні вказівки щодо використання гри. Для розроблення гри використано багатоплатформовий інструмент Unity3D та об'єктно-орієнтовану мову програмування C#.

ПЕРЕЛІК ПОСИЛАНЬ

1. Game Engine // Unity. URL: http://unity3d.com
2. Джозеф Хокинг Unity в действии, мультиплатформенная разработка на C#.
3. A* Pathfinding Project. URL: http://arongranberg.com/astar/
4. Become a Developer // Brackeys. URL: http://brackeys.com/
5. Игровой дизайн, гейм дизайн (game design) // GameDev.ru – Разработка игр. URL: http://www.gamedev.ru/gamedesign/terms/gameplay
6. Параллакс // Астронет. URL: http://astronet.ru/db/msg/1178033
7. Создание игр. Один в поле воин! Игрострой GCUP. 2021 // http://gcup.ru...
8. Создание игр. О начинающем разработчике инди-игр. Игрострой GCUP. 2021 // http://gcup.ru...
9. Конструктор игр. Википедия - свободная энциклопедия. - 2021 // http://ru.wikipedia.org/wiki/...
10. Создание 3D игр на языке Blitz3D. Blitz-school - центр обучения созданию компьютерных игр. - 2021 // http://www.blitz-school.info/
11. Несколько популярных игровых движков и конструкторов игр. 3D Graphics and me - блог о 3d графике и разработке игр, статьи, уроки, аналитика. 2021 // http://3dg.me/ru/gamedev/...
12. Перечень коммерческих игровых движков. Википедия - свободная энциклопедия. - 2021 // http://ru.wikipedia.org/wiki/...

13. Games software revenues to reach $110 billion by 2018. [Electronic resource] / James Brightman.– Access mode: http://www.gamesindustry.biz/articles/2015- 05-04-games-software-revenues-to-reachusd110-billion-by-2018-digi-capital.

14. Bethke, E. Game development and production [Text] / E. Bethke. – Wordware Publishing, Inc, 2003.– 415 c. – ISBN 1-55622-951-8.

15. Rogers, S. Level Up [Text] / S. Rogers.– John Wiley & Sons, Ltd, 2010. – 492 c. – ISBN 978-0-470-68867-0.

16. Unity, Source 2, Unreal Engine 4, or CryENGINE- Which Game Engine Should I Choose? [Electronic resource] / Mark Masters. – Access mode: http://blog.digitaltutors.com/unity-udk-cryengine-gameengine-choose/.

17. Game engine. [Electronic resource] :Материал из Википедии – свободной энциклопедии :Версия 68823161, / Авторы Википедии // Википедия,свободная энциклопедия. – Электрон. дан. – Сан-Франциско: Фонд Викимедиа, 2015. – https://en.wikipedia.org/wiki/Game_engine.

18. Video games starting to get serious. [Electronic resource] / Steve Barberich. – http://www.gфzette.net/stries/083107/businew11739_32356.shtml.

19. Physics engines survey results. [Electronic resource]/ Real–Time Physics Simulation. –http://bulletphysics.org/wordpress/?p=88

20. Another million unity developers in the house.[Electronic resource] / David Helgason. –http://blogs.unity3d.com/2013/07/09/anothermillion-unity-developers-in-the-house.

ДОДАТКИ

ДОДАТОК А

Лістинг програмного коду скрипта «menu.cs»

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using UnityEngine.SceneManagement;
public class menu : MonoBehaviour {
	public GUIStyle mystyle  string score;
	void Start ()
	{
		StreamReader scoredata = new StreamReader (Application.persistentDataPath +
"/score1.gd");
		score = scoredata.ReadLine ();
		scoredata.Close ();
	}
	void Update () {
	}
	void OnGUI(){
		GUI.Box (new Rect (Screen.width*0.15f, Screen.height*0.8f,
Screen.width*0.7f, Screen.height*0.1f), "MAX DESTROYED:"+score,mystyle);
		if (GUI.Button (new Rect (Screen.width*0.15f, Screen.height*0.25f,
Screen.width*0.7f, Screen.height*0.1f), "START",mystyle))
		{
			SceneManager.LoadScene (1);
		}
		if (GUI.Button (new Rect (Screen.width*0.15f, Screen.height*0.4f,
Screen.width*0.7f, Screen.height*0.1f), "EXIT",mystyle))
		{
			Application.Quit();
		}
	}

}


using UnityEngine;
using System.Collections;
public class activemenu : MonoBehaviour {

	float speed=-0.1f;
	void Start () {

	}

	void Update () {
		transform.Translate (new Vector3 (0f,speed,0f));
		if (transform.position.y < -12f)
		{
			transform.position=new Vector3(0f,13f,0f);
		}
```

```
	}

}
```

ДОДАТОК Б

Лістинг програмного коду ігрової сцени

```
using UnityEngine;
using System.Collections;

public class systemcontrolobjects : MonoBehaviour {

	void Start ()
	{

	}
	void Update () {
	}
	void  OnTriggerExit2D(Collider2D col)
	{
		Destroy(col.gameObject);
	}

}
```

```
using UnityEngine;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class blockgenerator : MonoBehaviour {

	public GameObject asteroid;
	float x,y,timer;
	float timerespawn=0.25f
	bool trigtime=false
	public int score
	public float data;

	void Start ()
	{
		score = 0;//
		timer = timerespawn;
		StreamReader scoredata = new StreamReader (Application.persistentDataPath +
"/score1.gd");
		data = float.Parse(scoredata.ReadLine ());
		scoredata.Close ();
	}

	void Update ()
	{
		if (timer==timerespawn		{
			x = Random.Range (-2.5f, 2.5f);
```

```
                  Instantiate(asteroid, new Vector3(x,5.5f,2.17f),transform.rotation);
                  trigtime = true;          }
            if (trigtime==true)
            {
                  timer = timer-Time.deltaTime          }
            if (timer < 0)
            {
                  timer = timerespawn;
                  trigtime = false;
            }
      }

}


using UnityEngine;
using System.Collections;

public class asteroidlogic : MonoBehaviour {

      public GameObject explosion;
      float speedasteroid=-0.1f;

      void Start ()
      {

      }

      void Update ()
      {
            transform.Translate (new Vector3 (0, speedasteroid, 0f));
      }

      void  OnTriggerEnter2D(Collider2D col)
      {
            if (col.tag == "systemcontrol")
            {
                  return;
            }
            if (col.tag == "laser")
            {
                  Destroy (col.gameObject);
                  GameObject.Find ("Main Camera").GetComponent<blockgenerator>
().score++;
                  Instantiate(explosion, transform.position,transform.rotation);
                  Destroy (this.gameObject);
            }
      }

}


using UnityEngine;
using System.Collections;

public class rotator : MonoBehaviour {
      int f;
      float sc;
```

```
	void Start ()
	{
		f = Random.Range (-5, 5);
		sc = Random.Range (0.1f,0.2f);
		transform.localScale = new Vector3 (sc,sc,sc);
	}

	void Update ()
	{
		transform.rotation *= Quaternion.AngleAxis (f,new Vector3(0,0,1));
	}

}


using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class carconroller : MonoBehaviour
{
	public GameObject laser;
	public GameObject laser3x;
	public GameObject laserhor;
	public GameObject laser3xhor;
	public GameObject sphere;
	public GameObject sphere3x;
	public GameObject explosionplayer;

	float x,y,z,x1,y1;
	bool trigtime=false;
	public float speedreset=0.25f;
	float timer;
	Vector2 startpos;
	Vector2 startcar;

	public bool gun1 = true;
	public bool gun2 = false;
	public bool gun3 = false;
	public bool gun4 = false;
	public bool gun5 = false;
	public bool gun6 = false;

	int guncount=0;

	void Start ()
	{
		timer = speedreset;
		y = laser.transform.position.y;
		z = laser.transform.position.z;
	}

	public void Update ()
	{
		if (timer < 0)
		{
```

```
                  timer = speedreset;
                  trigtime = false;
            }

            if (Input.GetMouseButton(0))
            {
                  Vector2 pos = Camera.main.ScreenToWorldPoint (Input.mousePosition);
                  transform.position = pos;
                  transform.position = new Vector2
(transform.position.x,transform.position.y+1f);
                  if (timer==speedreset)
                  {
                        if (gun1 == true)
                        {
                              Instantiate (laser, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                              trigtime = true;
                        }
                        if (gun2 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (laser3x, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                              trigtime = true;
                        }

                        if (gun3 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (laserhor, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                              trigtime = true;
                        }
                        if (gun4 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
```

```
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (laser3xhor, new
Vector2(transform.position.x,transform.position.y+2f), transform.rotation);
                              trigtime = true;
                        }
                        if (gun5 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (sphere, new
Vector2(transform.position.x,transform.position.y+1.1f), transform.rotation);
                              trigtime = true;
                        }
                        if (gun6 == true && guncount > 0)
                        {
                              guncount--;
                              if (guncount == 0)
                              {
                                    gun1 = true;
                                    gun2 = false;
                                    gun3 = false;
                                    gun4 = false;
                                    gun5 = false;
                                    gun6 = false;
                              }
                              Instantiate (sphere3x, new
Vector2(transform.position.x,transform.position.y+2f), transform.rotation);
                              trigtime = true;
                        }
                  }
                  if (trigtime == true)
                  {
                        timer = timer - Time.deltaTime;
                  }
            }
      }

      void  OnTriggerEnter2D(Collider2D col)
      {
            if (col.tag == "systemcontrol")
            {
                  return;
            }
            if (col.tag == "gunactivator2")
            {
                  gun2 = true;
```

```
                guncount = 85;
                gun1 = false;
                gun3 = false;
                gun4 = false;
                gun5 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
        }

        if (col.tag == "gunactivator3")
        {
                gun3 = true; guncount = 50;
                gun1 = false;
                gun2 = false;
                gun4 = false;
                gun5 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
        }
        if (col.tag == "gunactivator4")
        {
                gun4 = true; guncount = 15;
                gun1 = false;
                gun3 = false;
                gun2 = false;
                gun5 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
        }
        if (col.tag == "gunactivator5")
        {
                gun5 = true; guncount = 100;
                gun1 = false;
                gun3 = false;
                gun4 = false;
                gun2 = false;
                gun6 = false;
                Destroy (col.gameObject);
                return;
        }
        if (col.tag == "gunactivator6")
        {
                gun6 = true; guncount = 50;
                gun1 = false;
                gun3 = false;
                gun4 = false;
                gun5 = false;
                gun2 = false;
                Destroy (col.gameObject);
                return;
        }

        Handheld.Vibrate ();
        if (GameObject.Find ("Main Camera").GetComponent<blockgenerator>
().score>GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().data)
```

```
            {
                  StreamWriter scoredata=new
StreamWriter(Application.persistentDataPath + "/score1.gd");
                  scoredata.WriteLine(GameObject.Find ("Main
Camera").GetComponent<blockgenerator> ().score);
                  scoredata.Close();
            }
            Instantiate (explosionplayer, transform.position, transform.rotation);
            Destroy (col.gameObject);
            Destroy(this.gameObject);
      }

}


using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class Exit : MonoBehaviour {

      public GameObject target;
      float timer=3f;

      void Start ()
      {

      }

      void Update ()
      {
            if (Input.GetKey (KeyCode.Escape))
            {
                  if (GameObject.Find ("Main Camera").GetComponent<blockgenerator>
().score>GameObject.Find ("Main Camera").GetComponent<blockgenerator> ().data)
                  {
                        StreamWriter scoredata=new
StreamWriter(Application.persistentDataPath + "/score1.gd");
                        scoredata.WriteLine(GameObject.Find ("Main
Camera").GetComponent<blockgenerator> ().score);

                        scoredata.Close();
                  }
                  SceneManager.LoadScene (0);
            }
            if (!GameObject.Find ("playercar"))
            {
                  timer = timer - Time.deltaTime;
                  if (timer < 0)
                  {
                        SceneManager.LoadScene (0);
                  }
            }
      }

}
```

```
using UnityEngine;
using System.Collections;

public class DestroyAsteroid : MonoBehaviour {

	void Start ()
	{

	}

	void Update ()
	{
		Destroy (this.gameObject,0.4f);
	}

}


using UnityEngine;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class laser : MonoBehaviour {

	float speedlaser=0.5f;

	void Start ()
	{

	}

	void Update ()
	{
		transform.Translate (new Vector3 (0, speedlaser, 0f));
	}

}


using UnityEngine;
using System.Collections;

public class enemygenerator : MonoBehaviour {

	public GameObject enemy;
	bool trigtime=false;
	float speedreset=2f;
	float timer,x;

	void Start ()
	{
		timer = speedreset;
	}

	void Update ()
	{
		if (timer < 0)
```

```
            {
                  timer = speedreset;
                  trigtime = false;
            }
            if (timer == speedreset)
            {
                  x = Random.Range (-2.5f, 2.5f);
                  Instantiate(enemy, new
      Vector2(x,5.5f),transform.rotation);
                  trigtime = true;
            }
            if (trigtime == true)
            {
                  timer = timer - Time.deltaTime;
            }
      }

}


using UnityEngine;
using System.Collections;

public class enemylogic : MonoBehaviour {

      public GameObject explosionenemy;
      public GameObject laserenemy;
      public GameObject gunactivator2;
      public GameObject gunactivator3;
      public GameObject gunactivator4;
      public GameObject gunactivator5;
      public GameObject gunactivator6;
        //
      bool trigtime=false;
      float speedreset=1.5f;
      float timer;
      float speedenemy = -0.02f;

      float x;

      void Start ()
      {
            timer = speedreset;
      }

      void Update ()
      {
            if (timer < 0)
            {
                  timer = speedreset;
                  trigtime = false;
            }
            if (timer == speedreset)
            {
                  Instantiate (laserenemy, new
Vector2(transform.position.x,transform.position.y-0.4f), transform.rotation);
                  trigtime = true;
            }
```

```
            if (trigtime == true)
            {
                  timer = timer - Time.deltaTime;
            }
            transform.Translate (new Vector3 (0, speedenemy, 0f));
      }

      void  OnTriggerEnter2D(Collider2D col)
      {
            if (col.tag == "systemcontrol")
            {
                  return;
            }
            if (col.tag == "Player")
            {
                  Instantiate (explosionenemy, transform.position, transform.rotation);
                  Destroy(this.gameObject);
            }
            if (col.tag == "laser")
            {
                  x = Random.Range (0f, 100f);
                  if (x > 1f && x < 5f)
            {
                        Instantiate (gunactivator2, transform.position,
transform.rotation);
                  }
                  if (x > 20f && x < 25f)
                  {
                        Instantiate (gunactivator3, transform.position,
transform.rotation);
                  }
                  if (x > 40f && x < 45f)
                  {
                        Instantiate (gunactivator4, transform.position,
transform.rotation);
                  }
                  if (x > 60f && x < 65f)
                  {
                        Instantiate (gunactivator5, transform.position,
transform.rotation);
                  }
                  if (x > 80f && x < 85f)
                  {
                        Instantiate (gunactivator6, transform.position,
transform.rotation);
                  }
                  Instantiate (explosionenemy, transform.position, transform.rotation);
                  Destroy(this.gameObject);
            }
      }

}


using UnityEngine;
using System.Collections;

public class gunactivatorlogic : MonoBehaviour {
```

```
	float speed = -0.025f;

	void Start ()
	{

	}

	void Update ()
	{
		transform.Translate (new Vector3 (0, speed, 0f));
	}

}
```