МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Дипломна бакалаврська робота

на тему:

«Розробка 2D гри на Unity»

Виконав студент 4 курсу, групи БІТск-21

спеціальності 122 Комп'ютерні науки

Ігор МОСКАЛЕНКО

Керівник Тетяна ДЕМКІВСЬКА

Рецензент Віктор ЧУПРИНКА

Київ 2023

АНОТАЦІЯ

Москаленко Ігор Андрійович. Розробка 2D гри на Unity.

Дипломна бакалаврська робота за спеціальністю 122 — «Комп’ютерні науки» — Київський національний університет технологій та дизайну, Київ, 2023 рік.

В дипломній бакалаврській роботі проведено дослідження процесу розробки відеоігор, засобів їх поширення та ринків збуту. Здійснено аналіз розвитку ігрової індустрії, її впливу на розвиток сучасних технологій і еволюцію комп’ютерної графіки. Вивчено вплив комп’ютерних ігор на суспільство та можливі методи взаємодії з ним через даний вид програмного продукту.

Розроблена демонстраційна версія комп’ютерної гри у жанрі платформер на базі ігрового рушія Unity, та стилізованою двовимірною графікою згідно стилю Pixel art.

Ключові слова: відеоігри, розробка ігор, 2D гра, Unity, ігрова індустрія, комп’ютерна графіка, Pixel art.

ABSTRACT

Ihor Moskalenko. Development of a 2D game with Unity.

Bachelor’s thesis on specialty 122 — “Computer science” — Kyiv National University of Technology and Design, Kyiv, 2023.

In the bachelor's thesis, a study of the process of developing video games, means of their distribution and sales markets was carried out. An analysis of the development of the game industry, its influence on the development of modern technologies and the evolution of computer graphics was carried out. The impact of computer games on society and possible methods of interaction with it through this type of software product have been studied.

A demo version of a platformer computer game based on the Unity game engine and stylized two-dimensional graphics in the Pixel art style has been developed.

Keywords: video games, game development, 2D game, Unity, game industry, computer graphics, Pixel art.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет Мехатроніки та комп'ютерних технологій
Кафедра Комп'ютерних наук
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри
Комп'ютерних наук

_____ Володимир ЩЕРБАНЬ
(підпис)
«_____» _____ 2023р.

ЗАВДАННЯ

НА ДИПЛОМНУ БАКАЛАВРСЬКУ РОБОТУ

студенту
Москаленку Ігорю Андрійовичу

1. Тема роботи: Розробка 2D гри на Unity

Науковий керівник роботи Демківська Тетяна Іванівна

затверджені наказом КНУТД від " 08 " 11 2022 року № 224-уч.

2. Строк подання студентом дипломної роботи 10.06.2023 р.

3. Вихідні дані до дипломної роботи (проекту) Розробки кафедри комп'ютерних наук, рекомендована література.

4. Зміст дипломної бакалаврської роботи: 1) Аналіз предметної області, 2) Проєктування та дизайн, 3) Розробка програмного продукту.

5. Дата видачі завдання 15.03.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної бакалаврської роботи	Терміни виконання етапів		Примітка про виконання
1	Вступ	20.04.2023		
2	Розділ 1. Аналіз предметної області	25.04.2023		
3	Розділ 2. Проектування та дизайн	06.05.2023		
4	Розділ 3. Розробка програмного продукту	09.05.2023		
5	Висновки	10.05.2023		
6	Оформлення дипломної бакалаврської роботи (чистовий варіант)	15.05.2023		
7	Здача дипломної бакалаврської роботи на кафедру для рецензування (за 14 днів до захисту)	25.05.2023		
8	Перевірка кваліфікаційної роботи на наявність текстових співпадінь та помилок (за 10 днів до захисту)			
9	Подання дипломної бакалаврської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)			

Студент _____ Ігор МОСКАЛЕНКО
(підпис)

Науковий керівник _____ Тетяна ДЕМКІВСЬКА
(підпис)

Рецензент _____ Віктор ЧУПРИНКА
(підпис)

ЗМІСТ

ВСТУП

Актуальність дослідження: в сучасному світі все більше аспектів людської діяльності, слід за розвитком технологій, перекочує у цифрову форму та віртуальну реальність. Розваги не є виключенням, серед різних сервісів можна виділити відеоігри, які розвинулись в потужну індустрію та стали звичайним споживчим продуктом у житті людини. Комп'ютерні ігри є складним програмним продуктом, що об'єднує в собі досягнення різноманітних сучасних технологій та впливає на розвиток різних сфер досліджень та галузей промисловості. Ігри мають великий культурний, соціальний та економічний вплив на суспільство та технології.

Мета за завдання роботи: метою дипломної роботи є вивчити процес розробки відеоігор, історію їх виникнення та популяризацію. Проаналізувати вплив комп'ютерних ігор на розвиток технологій та суспільство, формування і склад сучасного ринку комп'ютерних ігор, їх види та жанри. Дослідити технології та види програмного забезпечення, що використовується для розробки відеоігор; а також методи розробки комп'ютерної графіки, її види та застосування. Розробити програмний продукт у вигляді відеогри з двовимірною графікою у жанрі платформер.

Об'єкт дослідження: об'єктом дослідження є відеогра, як складний програмний продукт, що об'єднує в собі різноманітні сучасні технології та мистецькі течії.

Предмет дослідження: предметом даного дослідження є вивчення процесу розробки відеоігор.

Методи розробки: мова програмування C# та ігровий рушій Unity, а також графічний редактор Aseprite.

Структура, зміст та обсяг роботи: дипломна робота складається з 3 розділів, містить 44 рисунків та 30 джерел. У Розділі 1 проведений аналіз ігрової індустрії та опис процесу розробки. У Розділі 2 наведено основні види та методи розробки комп'ютерної графіки. У Розділі 3 зображено процес розробки програмного продукту. У додатках наведено лістинги скриптів до програми.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Відеоігри, що це?

Відеоігри або комп’ютерні ігри – це програмні продукти, розроблені з розважальною метою, вони можуть бути як цифровими варіантами настільних чи будь-яких ігор, так і повністю новими видами ігор.

Загалом відеоігри можна охарактеризувати як електронну гру, в процесі якої гравець використовує периферійні пристрої для вводу інформації і своїх дій, комп’ютерний інтерфейс користувача і пристрої виводу для отримання зворотної інформації з пристрою що компілює гру.

Відеоігри пройшли довгий шлях, від простих ігор на аркадних автоматах та появи перших портативних грального приставок, до сучасних мультиплатформних проєктів з багатомільйонними бюджетами. Вони змінювали світ і змінювались самі. Колись купка друзів грала в настільну Dungeon & Dragons, збираючись у когось вдома чи в гаражі, а тепер кілька десятків або навіть сотень людей збираються разом на серверах в Discord і грають у World of Warcraft. До речі Discord і багато подібних програм були розроблені завдяки відеоіграм, у відповідь на потребу гравців вільно спілкуватись у мережі в реальному часі, під час гри чи просто так, більше того, вплив відеоігор на розвиток всесвітньої мережі та технологій передачі даних теж суттєвий, як і на інформаційні технології і суспільство у цілому.

1.2. Наука чи мистецтво?

Комп’ютерні ігри вже давно стали частиною буденності, для дітей це улюблена забавка, для дорослих гарний спосіб відпочити, для спортсменів опанувати нову дисципліну та нові вершини. Кіберспорт, книжки та фільми по іграм й навпаки, вони вже давно стали частиною поп-культури, частиною нашого життя.

З наукової точки зору це розважальний програмний продукт, з суспільної - витвір мистецтва. Розробка ігор була і є одним з ключових рушіїв розвитку комп'ютерних технологій. Саме відеоігри розвивали інтерес до комп'ютерних наук у молодого покоління, ще за часів коли Інтернет лише починав прясти свої

тенета. Саме завдяки комп'ютерним іграм, майбутні програмісти починали програмувати власні ігри, а згодом програми. Зараз ця тенденція зберігається, але також слід зауважити що ігрова індустрія допомагає розвивати різноманітні технології віртуальної візуалізації та обчислення даних, розвиток та популяризацію обчислювальних чипів.

В гонитві за більш реалістичною графікою та ігровою логікою, в меті створити ідеальну фізику, спецефекти та звук, виникала потреба розробляти більш потужні технологічні пристрої для відтворення задуманої візуалізації. Розвивались програмні чипи, згодом стало замало одного ЦП для відтворення більш сучасних ігор зі складнішими технологіями, що викликало потребу у використанні додаткового відео прискорювача, який згодом виріс у потужну відеокарту з власною пам’яттю, процесорами та охолодженням. Крім цього виникала потреба в більш якісному способі виведення зображення, розвивалась і сфера моніторів, нові технології екранів. Більша роздільна здатність, вища частота оновлення, мільйони відтінків кольорів, найменша затримка виводу зображення. Згодом з’явився попит на більше занурення у віртуальну реальність, що викликав розвиток шоломів віртуальної реальності і навіть розробку приладу для передачі запахів. З розвитком відеоігор збільшувався попит на продукцію, з ними пов’язану. Починаючи від перших ігрових консолей і аж до сучасних спеціальних периферійних пристроїв з яскравим дизайном для геймерів, шоломами віртуальної реальності.

Ігри ставали популярними, все більше людей проявляли цікавість, все більше прибутків приносила індустрія, все більше ставало розробників і компаній, які були готові інвестувати в індустрію відеоігор. Серед таких інвесторів є і компанії які займаються розробкою сучасних технологій візуалізації та обчислення. Технологій, які покращували і візуалізацію самих ігор, і могли використовуватись для розробки іншого ПЗ чи геть в інших напрямах.

Наприклад, потужні відеокарти стали основою для створення і популяризації видобутку криптовалюти; технології віртуальної візуалізації,

зокрема ігрові рушії, використовують для створення спецефектів у кінематографі або для створення віртуальних моделей на виробництві; віртуальні симулятори допомагають тренувати різних фахівців і навіть військових, особливо у цьому допомагають шоломи віртуальної реальності.

Мабуть, одна з найголовніших функцій відеоігор це суспільна. Ігри об'єднують людей. Дають їм змогу спілкуватись, проводити час разом, знаходити однодумців, вчитись і просто насолоджуватись мистецтвом. Адже це таке саме мистецтво, як книги, картини чи фільми, але з більшим ефектом занурення, можливістю відчути і вплинути на сюжет.

Комп'ютерні ігри так сильно вплелись в сучасну культуру, що вже досить складно виокремити їх вплив на неї. Всередині культури відеоігор навіть з'явились свої субкультури, розділені за певними жанрами ігор, яких зараз існує вже дуже багато.

Відеоігри це грандіозний культурний та технологічний здобуток людства.

1.3. Актуальність відеоігор

Три мільярди

Це приблизна кількість геймерів у світі, тобто більше третини населення планети грає у відеоігри, і ця цифра продовжує збільшуватись. А загальна капіталізація ігрового ринку становить близько 200 млрд доларів. Щороку кількість компаній в напрямі геймдеву (game development – розробка ігор) збільшується, одні поглинають інші, створюються нові незалежні студії. Тобто відбуваються звичайні процеси для світу великого бізнесу. Геймдев перетворився із угруповань ентузіастів і маленьких творчих проєктів, у потужну комерцію, з усіма відповідними проблемами. Але незважаючи на це, в індустрію продовжують входити ті самі ентузіасти, які прагнуть створити щось чудове і неймовірне.

Ігри це можливості

Для розробників ігри це спосіб розповісти історію, поділитись думками та ідеями, баченням, мріями. Це можливість розкрити проблему, вплинути на суспільство, донести певне послання, популяризувати ідею. Це можливість

створювати світи із власними правилами, це новий етап образотворчого мистецтва, що виділяється взаємодією з ним, чого немає у фільмах чи тим паче у книжках. Коли автори книжок можуть тільки частково передати уявлення про персонажів чи краєвиди, розробники ігор можуть буквально оживити персонажів, дозволити відчути атмосферу свого світу і зробити це настільки точно, наскільки дозволяють технології, які покращуються з кожним роком.

З точки зору споживача, тобто гравця, ігри це можливість відволіктись від реальності, від власних проблем, психологи рекомендують інколи використовувати ігри для емоційної розрядки чи так званого «перезавантаження», коли людині треба відійти від всіх стресів і оточуючих проблем. Сучасні технології у іграх дозволяють досить глибоко зануритись у вигаданий світ, або у реалістичні реконструкції минулого чи найбільш імовірні проекції майбутнього, хороші чи погані, уявити їх наслідки.

Частина нас

Часом ігри залишають значимий відбиток в суспільстві, найчастіше у вигляді мемів – одиниць культурної інформації, що поширюється між людьми, і як описує видатний біолог Річард Докінз, являє собою своєрідний «культурно-інформаційний ген» що передається між поколіннями і народами крізь час. Один з найвідоміших ігрових мемів «Press F to pay respects», що виник з моменту у грі Call of Duty: Advanced Warfare, коли потрібно було натиснути вказану клавішу аби вшанувати пам’ять полеглого товариша. Типова для ігор інструкція «press F», що застосовувалась для більшості видів взаємодій у різних іграх, набула нового сенсу і ця фраза вже автоматично вважалась проявом вшанування когось чи чогось після невдачі. Згодом для відображення цього мему почали використовувати просто літеру F. Таких прикладів і мемів є безліч, і всі вони вже давно стали частиною нас.

Інформація це важливо

Ігри також можуть бути інструментом донесення інформації. Часто в них, як і в будь-якому мистецтві піднімають різні важливі теми, як політичні чи соціальні, так і, наприклад, питання глобального потепління чи вимирання. Ігри

можуть бути потужним засобом інформації, деякі з них навіть забороняють у певних країнах, в першу чергу через схований підтекст. Іноді це може бути просто відсилка на якусь іншу гру чи мем, а іноді весь сюжет будується на одному політичному підтексті, який є дуже важливою проблемою у реальному світі. Виражатись все це може або в підтексті сюжету, непрямих рисах персонажів, їх мотивах, або у вигляді об'єктів оточення: листівок, плакатів, будівель, місцевих назв, а в деяких випадках додають навіть QR-коди, які можна просканувати і вийти на ресурс з певною інформацією.

Яскравим прикладом гри з потужним інформаційним посилом в сюжеті є гра Valiant Hearts The Great War від студії Ubisoft Montpellier, яку розробники створювали під 100 річчя Першої світової війни, аби показати і нагадати людям жахи Великої війни. До створення гри були залучені найкращі історики Європи, а також особисті історії і фронтові листи родичів розробників. Гра робить акцент на історичності і драматизмі, та намагається показати всі жахі війни, спрощуючи їх мультиплікаційною графікою, оскільки гра і так досить драматична. Колірна гама, яка використовується, нагадує старі фото, які кольоризували, сюжет розповідає про реалістичні і трагічні історії персонажів у різних країнах. Аби продемонструвати погляди на світ з різних боків, очами простих людей різних країн того часу. Також є можливість знаходити колекційні предмети, які відкривають певні історичні довідки чи навіть реальні фронтові листи. Ігри, на відміну від інших видів мистецтва, дозволяють відчути себе частиною історії, керуючи персонажем ти не дивишся на героя збоку, як у книгах чи кіно, ти сам ведеш його цим шляхом, від початку і до кінця.

Тому тут інформація грає дуже важливу роль, вона визначає що буде переживати гравець, що відчувати і що залишиться у його свідомості після проходження гри: радість, смуток, розуміння чи прозріння, душевний спокій, відчуття наповнення, катарсис.

Тут раді всім

У геймдеві задіяні безліч різноманітних креативних професій. Ми створюємо світи, тож нам потрібні ті, хто зможе їх будувати, наповнювати,

наділяти сенсом. Програмісти мають змогу випробувати специфічні логічні моделі, інтегрувати бази даних і пов’язувати графічні елементи. Художники можуть створювати унікальних деталізованих персонажів, безмежні продумані світи, використати всю палітру фантазії. Сценаристи - створити і описати неповторну історію і світ, який згодом «оживе» у віртуальному просторі і може покласти початок популярній франшизі і згодом перекочувати у інші індустрії. Звукорежисери - створити найрізноманітніші звуки та мелодії для реалізації ігрового світу. Актори – подарувати свою зовнішність чи голос ігровим персонажам, допомогти у створенні анімацій. Маркетологи - створити унікальні бізнес-моделі, які будуть приносити більше доходів, але такі рішення, зазвичай, досить не дуже тепло зустрічає спільнота гравців. Проєкт-менеджери – організувати роботу команди, різних відділів та сфер, використання технологій та сервісів, для ефективної експлуатації та ведення проекту. Ігрова індустріє налічує сотні тисяч додаткових робочих місць по всьому світу, і відкриває безліч можливостей для творчості у всіх сферах.

Це простір для експериментів

Різноманітність професій і культурне багатство співробітників у ігровій індустрії створює можливість поєднувати найрізноманітніші ідеї та традиції. В ігри додаються персонажі з яскравими проявами їх етнічного походження, демонструються різноманітні традиції і вірування, які навряд можна було б побачити чи прийняти участь у реальному житті, або в силу віддаленості певних регіонів, або через те, що деякі з них вже забуті. Створюючи ігри можна створити світ в якому відбудеться те, чого не сталось у реальності, те що вже попередили або чому ще можна завадити. І зробити з цього висновки, нагадавши чому це важливо. Прикладом може виступити серія ігор Wolfenstein, яка показує світ, де під час Другої світової війни перемогла Німеччина, і наслідки для світу. Або гра Homefront від студій Kaos Studios і Digital Extremes, яка демонструє майбутнє, в якому Південна Корея захоплює Азію, а потім і США. Не слід забувати і про культові, відомі на весь світ, вітчизняні проєкти, такі як серії ігор S.T.A.L.K.E.R. та Metro, які демонструють погляд розробників та письменників на наслідки

ядерної війни чи катастрофи. Експериментувати можна і в інших площинах, наприклад в музиці та звуках. Відкривати і застосовувати нові технології та способи записування і відтворення звуків у віртуальній реальності. Створювати реалістичні об'ємні звуки або вплітати музику у ігровий процес. Останнє, до речі, викликало появу нового жанру відеоігор - Rhythm або (Rhythm-game/Rhythm-action), особливість якого заключається в мотивації гравця робити певні дії синхронно з ритмом. Яскравим представником цього жанру є Metal: Hellsinger, в розробці якої, до речі, безпосередню участь приймали й українці. Нові жанри з'являються постійно, а сучасні ігри часто взагалі не обмежуються лише одним жанром. А навіщо, коли створюєш світи, з власними правилами і законами? Так і виникли масштабні Одіссеї на зразок легендарної The Elder Scrolls V: Skyrim, Final Fantasy XV, The Witcher 3, і як не дивно Assassins Creed: Odyssey.

1.4. Процес створення

Концепт

Створення відеоігор, як і будь-чого, починається з ідеї (Рис. 1.1). Вона приходить сама або для її створення наймають спеціаліста.

Ідея може полягати у створенні певної історії чи сюжету, або ж це може бути ідея нового ігроладу – способу гри, системи взаємодій гравця із грою. В будь-якому випадку, в процесі обговорення такої ідеї залучаються концепт-художники, які займаються візуалізацією концептів. Вони допомагають розробити попередній вигляд елементів проєкту, так би мовити креслення. Це можуть бути пейзажі ігрових світів чи локацій; варіанти зображення персонажів, їх спорядження; можливі способи зображення ігрового процесу, комбінування різних концептів; візуалізація всього що можна візуалізувати.

Загалом залучаються багато концепт-художників для більшого різноманіття запропонованих варіантів і подальшого вибору найбільш підходящого. Але іноді один концепт-художник може керувати всім візуальним стилем проєкту, як у випадку гри God of War 2018 року від Santa Monica Studio, яка сфокусувала весь візуальний стиль даного проєкту на баченні концепт-

художника Хосе Кабрера. Або навіть керувати всією компанією, як Хідео Коджіма, який займається не лише концептами але й безпосередньо ігровим дизайном та сюжетом.

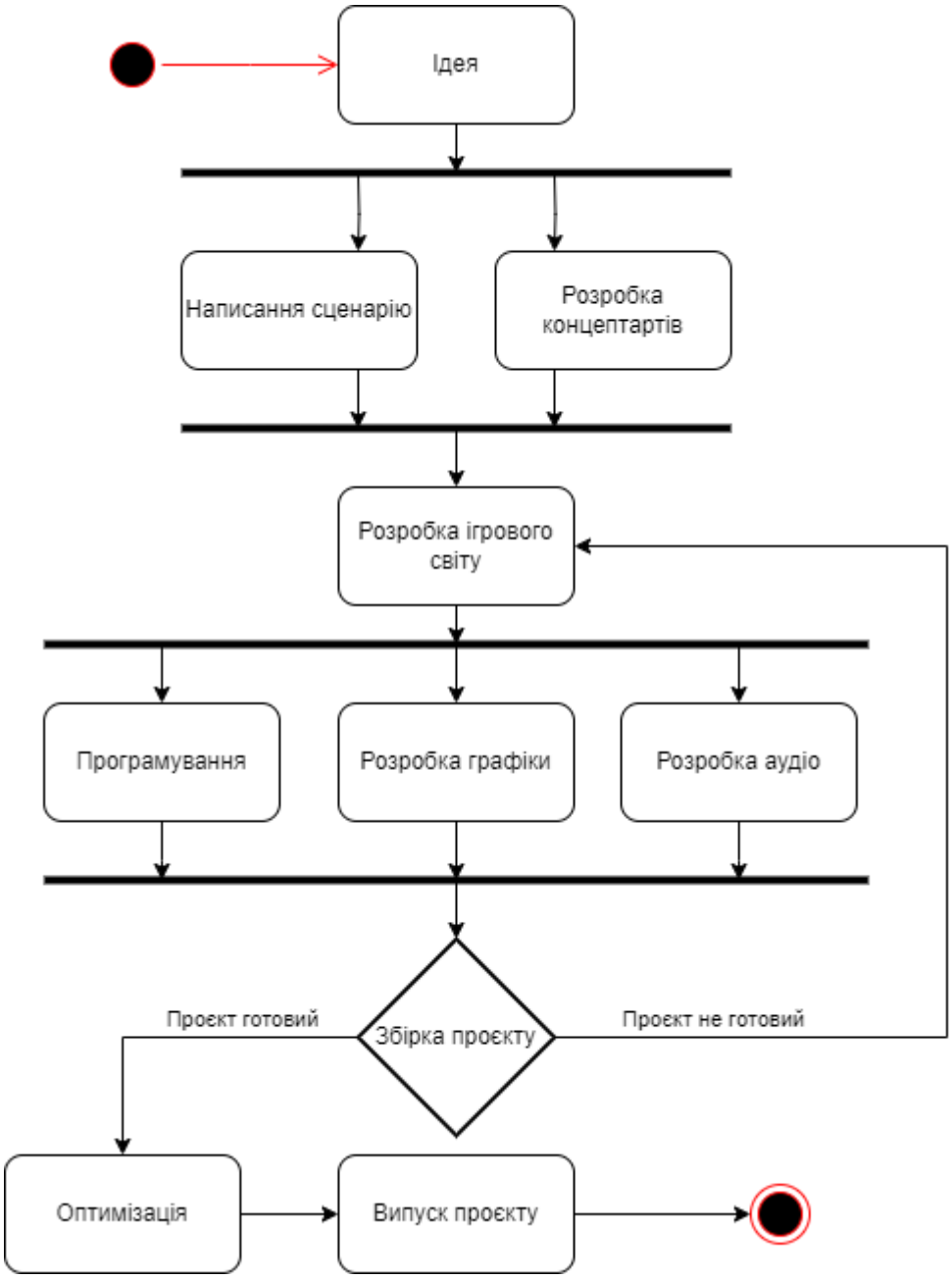


Рис. 1.1 UML-діаграма процесу розробки гри

Сюжет є не менш важливою складовою концепту, оскільки саме по ньому і орієнтуються концепт-художники, і з рештою навколо нього і будується майбутня гра. Гра може не мати унікального захопливого ігроладу чи графіки, натомість мати потужний сюжет який і захоплює гравців, до прикладу культова Undertale з ретро графікою але складним сюжетом, на який своїми діями впливає гравець. Або навпаки гра може комбінувати чудову графіку, цікавий ігролад та

захопливий сюжет, як в God of War 2018, The Witcher 3 чи The Last of Us. Але навіть якщо на перший погляд здається, що сюжету в грі немає, наприклад, в Dota 2, в самому ігровому процесі він ніяк не зображається, проте в концепті, в історії персонажів він фігурує, він будує фундамент ігрового світу, через що той здається простим та звичним або ж навпаки загадковим.

Ігровий дизайн

Це наступний етап після розробки концептів. Ігровий дизайн має за мету втілення запропонованих і замальованих концепт-художниками ідей, безпосередньо у грі. Мова йде не лише про створення графіки та анімацій, а й про постановку рівня, систему внутрішньої взаємодії різних елементів ігроладу, розробка ігрової логіки. Розуміння даного етапу можна розширити до поняття менеджменту проєктом в цілому, але все ж більш наближено до візуальної та логічної складової.

Стосовно логічної складової, одну з головних робіт виконує дизайнер рівнів. Йому необхідно побудувати структуру локації та усього світу гри так, щоб це мало сенс та змогло зацікавити гравця. Це стосується як просто розташування умовних платформ по яким переміщається персонаж, так і розробка усіляких пасток, розміщення ворогів, розробка та реалізація головоломок та загадок. Дизайнер рівнів створює завдання для гравців, він співпрацює з дизайнером квестів, який розробляє цікаві сюжети та завдання, а також з дизайнером оточення, який займається наповненням логічної моделі рівня візуальною складовою, створенням графічних об’єктів, деталізацією ігрового світу, побудовою загальної картини відповідно до концепт артів, враховуючи дизайн рівнів.

Так само працюють і дизайнери персонажів, які використовують все ті ж принципи, тільки вже в площині персонажів, їх історій, ігрового процесу, пов’язаного з ними.

Звук

Звук дуже важлива складова комунікації гри з гравцем. І справа навіть не в діалогах персонажів. Звук – це потужна складова будь-якого продукту, якщо

його правильно використовувати.

Застосування звуків в іграх дуже широке, це буквально додатковий вимір. Він допомагає гравцям орієнтуватись в просторі і натякає як слід реагувати на певну подію. Наприклад звук може вказати на точне перебування ворога, постріли чи кроки якого видають його місцезнаходження. Також орієнтування в просторі покращується за рахунок відповідних звуків оточення. Набагато зручніше і звичніше, коли разом з положенням камери у грі змінюється і розташування звуків, наприклад, водоспаду чи заведеного автомобілю. Або ж коли пересуваючись по траві чути її шелест, чи по багнюці чути відповідний звук кроків.

Все це допомагає краще поринути у віртуальний світ і орієнтуватись в ньому, відчути атмосферу та настрій гри. Останнє досягається шляхом підбору відповідної мелодії. Музика в іграх, як і в кіно, часто використовується для емоційного забарвлення певної ситуації, місця чи персонажу. Для нас вже звично визначати настрій і характер певної події за мелодією, це дозволяє краще зрозуміти творчий задум та підсилити емоційну реакцію на щось. В іграх застосовують як вже випущені мелодії певних виконавців, так і цілі альбоми написані спеціально для проєкту.

Але вершина аудіо в іграх – це звісно спеціальне створення звуків самою студією. Багато чого можна просто записати з допомогою мікрофону, але коли справа доходить до унікальних звуків вигаданих істот, предметів чи локацій, тут залучаються звукові дизайнери. Вони записують різноманітні звуки існуючих предметів чи істот, і компонують їх у щось неймовірно нове, страшне і загрозливе або приємне і мелодійне. Таким чином можна розробити унікальну палітру звуків для будь-чого в грі, в тому числі так розробляють і спеціально скомпоновані самою ігровою студією композиції, які можуть максимально точно передавати атмосферу.

Програмування

Майже всі процеси, під час створення гри, пов’язані з програмуванням. Оскільки гра це програмний продукт, саме її існування зав’язано на програмному

коді. Програмісти в ігровій індустрії повинні уміти пов’язати всі процеси створення гри за допомогою коду. Без належної роботи коду, всі інші розробки будуть марними, оскільки вони або не зможуть працювати, або будуть працювати некоректно, викликаючи глюки та баги і псуючи враження від продукту.

Програмісти проводять чи не найбільшу роботу, оскільки втілення різноманітних рішень в процесі створення вимагають грамотну адаптацію коду, можливості оптимізації та відтворення задуманих процесів програмно. Часто цікаві та прогресивні ідеї в розробці відеоігор обмежувалися програмними можливостями свого часу, а іноді приводили до революції та просуванню інноваційних технологій.

Програмісти супроводжують проєкт на всіх стадіях його життя. На початку розробки – будуючи кістяк програми та вказуючи що можливо відтворити програмно, а що доведеться спростити. Безпосередньо в процесі розробки – розробляючи різноманітні системи та функції, пов’язуючи окремі елементи проєкту в монолітний процес, наприклад графіку, анімацію, звук та логічний зв’язок зі світом гри. Після готовності проєкту – вони продовжують працювати над оптимізацією та вдосконаленням коду, а також над випусками оновлень, додаткового контенту та розробці сиквелів.

Умовно кажучи програмісти створюють закони і взаємозв’язки ігрового світу та стежать щоб всі процеси не полишали їх рамки (Рис. 1.2).

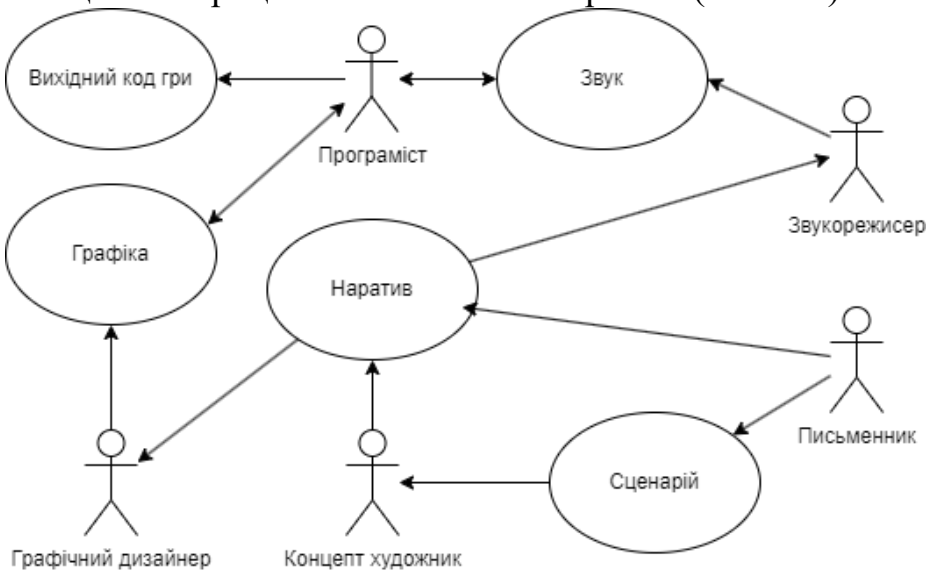


Рис. 1.2 UML-діаграма прецедентів в процесі розробки гри

Підтримка

Після випуску продукту, його потрібно підтримувати не лише програмно. Оскільки це багатогранний продукт, підтримка має бути відповідною. В першу чергу треба забезпечити популяризацію продукту. Рекламування гри на рівні з адаптацією до потреб споживачів, починається ще на ранніх етапах розробки. Великі студії готують аудиторію до виходу нового продукту заздалегідь, розпалюючи та підтримуючи цікавість до нього. Оскільки від цього напряму залежить чи стане проєкт успішним, чи буде він популярний та прибутковим.

Крім реклами, необхідно потурбуватись і про підтримку інтересу аудиторії, особливо якщо гра є грою-сервісом, тобто замість випуску нових частин гри, зміни та нововведення пропонуються в поточному проєкті. Якщо гра залишається однаковою, гравці досить швидко втрачають цікавість і переводять її на інші проєкти, тому ігрові студії розробляють плани з підтримки проєктів на довгостроковій основі.

В них, окрім регулярних рекламних кампаній, входять контентні оновлення, які мають продовжити пригоди гравців. Це можуть бути як DLC (Download Content) – додатковий завантажувальний вміст, тобто додаткова частина продукту, в площині відеоігор це додаткові локації, рівні, нові пригоди та завдання. Також це можуть бути тимчасові сезонні події, до прикладу, новорічні, пасхальні, гелловінські чи інші оновлення, які додають відповідні тимчасові пригоди, нові способи гри чи ігрові предмети, змінюють зовнішній вигляд гри відповідно до тематики певного сезону.

Наразі також використовуються різноманітні колаборації з іншими проєктами та навіть індустріями. Зокрема персонажі або бренди з однієї гри можуть з’явитись у проєкті іншої студії, які заздалегідь домовились про колаборацію для взаємної реклами. Крім цього колаборація може відбуватись і поза іграми, наприклад у вигляді спеціальної лінійки одягу чи колекційних предметів. З такою ж метою випускаються комікси, книжки, фільми та серіали.

Таким чином увага споживача до продукту постійно утримується і цінується з боку гравців, які хочуть продовжувати грати у певну гру, але також

хочуть регулярно споживати новий пов’язаний з нею контент.

1.5. Програмне забезпечення

Основним ПЗ для розробки ігор є ігровий рушій. Ігровий рушій (game engine) – це програмне забезпечення яке використовується для розробки ігор, це каркас, який буквально створює закони фізики будь-якої гри і відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури

Ігровий рушій це той самий простір, в якому програмісти пов’язують між собою всі складові розробки гри з допомогою коду. Одна з головних його функцій це передача інструкцій відеокарті для візуалізації графіки та візуальних ефектів, але крім того в залежності від самого рушія він може виконувати ряд більш складних функцій. Наприклад створювати логічні зв’язки між різними елементами віртуального світу, від банальних «цей предмет слідує за тим» до імітації відбивання світлових променів в реальному часі та симуляції руйнування ігрових об’єктів згідно законів фізики.

Також більшість ігрових рушіїв пропонують інструментарій візуального програмування, з допомогою якого можна програмувати деякі процеси без безпосереднього написання програмного коду. Принцип побудовано на використанні так званої дошки, на яку можна додавати блоки, що означають певний алгоритм. Кожен блок має спеціальні параметри, що налаштовуються, а самі блоки можна з’єднувати з іншими блоками і тип з’єднання визначає залежність одно блоку від іншого і кінцевого результату. Іншими словами, ми оперуємо готовими функціями та методами, обираючи способи їх взаємодії та залежності змінних від попередніх функцій. Таким чином можна програмувати як внутрішню логіку та залежності, так і графіку та візуальні ефекти. Це корисно, коли гру розробляє маленька команда без програмістів, або взагалі одна людина. Крім того це запобігає виникненню багів чи небажаного результату та економить час на розробку.

Багато ігрових студій використовують власні ігрові рушії, які забезпечують необхідну гнучкість та бажаний функціонал в залежності від їх

потреб, крім того створюючи та зберігаючи характерний стиль для проєктів даної студії. Вони є закритими і використовуються лише всередині компанії. Такі рушії є цінним надбанням студії, яке розроблялось та вдосконалювалось, інколи навіть десятиліттями, провокуючи виникнення нових рішень та технологій. Найяскравішими прикладами таких ігрових рушіїв є Frostbite Engine студії DICE, CryEngine студії Crytek, Creation Engine студії Bethesda, Source Engine студії Valve, ігровий рушій української студії 4A Games – 4A Engine.

Окрім внутрішніх рушіїв студій існує багато ігрових рушіїв у відкритому доступі. Їх досить багато але найпопулярнішими наразі є Unity та Unreal Engine, за рахунок своєї зручності у використанні, інтеграції власних програм навчання для користувачів, широкої бази побудованих проєктів на їх базі, підтримки багатьох платформ, постійної та активної підтримки з боку розробників та спільноти. Ці два рушії постійно оновлюються, розробляють та інтегрують новітні технології, і пропонують широкий спектр функцій.

Головна різниця між Unity та Unreal Engine полягає в можливостях. Вони можуть виконувати однакові функції але.

Unity використовує C# для програмування, завдяки чому він простіше і безпечніше оперує ресурсами задіяними у компілюванні програми, відповідно гри. Його часто позиціонують як рішення для новачків, самотніх розробників чи невеликих студій, що займаються невеликими проєктами або проєктами з не дуже вимогливою графікою, наприклад 2D ігри, або мобільні проєкти. Хоча функціонал цього рушія дозволяє створювати як ігри з якісною тривимірною графікою, так і високоякісні рендери відео. Тобто Unity це інструмент для простої роботи без зайвого клопоту.

Unreal Engine в свою чергу більш складний і потужний інструмент. Перш за все він використовує C++, за рахунок чого дозволяє більш точно оперувати ресурсами, але з іншого боку це вимагає більшої уваги розробника до оптимізації. Цей рушій часто застосовують в кіно та проєктах з реалістичною графікою, оскільки він має потужний функціонал для роботи саме з деталізацією та реалізмом. Unreal Engine також можна використовувати для створення

мобільних ігор та 2D проєктів, але він досить громісткий і нераціонально його використовувати для досить простих проєктів, що не використовують його потенціал на повну.

І Unity, і Unreal Engine мають внутрішній магазин готових асетів (assets) - графічних об’єктів, технологій та рішень, де вони поширюються безкоштовно або на платній основі. Вони розробляються користувачами або пропонуються самими розробниками і їх можна інтегрувати у свій проєкт. Кількість асетів постійно збільшується, а магазини і технології постійно оновлюються. Наприклад Unreal Engine пропонує магазини асетів MetaHuman, який дозволяє використовувати і редагувати реалістичні моделі облич з анімаціями міміки, та Quixel Megascans що пропонує функціонал застосування тривимірних асетів оточення, створених з використанням фотограметрії – тривимірного сканування місцевості.

Крім ігрових рушіїв при розробці ігор використовується безліч інших спеціалізованих програм, відповідно до необхідних функцій. Найчастіше у великих компаній є власні рішення, або корпоративні комплекти відомих програм та сервісів.

Із найбільш популярного ПЗ ще можна виокремити графічні редактори та програми для моделювання чи анімацій. Зокрема використовують Zbrush – для моделювання, частіше органічних об’єктів, Maya – найчастіше для анімацій, 3ds Max – для моделювання різноманітних складних об’єктів, Blender – ультимативний продукт, що поєднує в собі функціонал для створення та редагування графіки, моделювання, анімацій, відео та навіть ігрового рушія, крім має відкритий доступ. Також застосовують пакет програмних засобів Adobe та подібні.

1.6. Платформи та ринки збуту

Колись ігри продавали в спеціальних магазинах, на фізичних носіях, сьогодні ж їх можна придбати майже буть де. Але головним чином це відбувається в мережі на спеціальних сервісах.

В першу чергу все залежить від цільової платформи гри, оскільки ігри для

мобільних пристроїв, комп’ютерів чи ігрових консолей, створюються окремо і працюють лише на платформі під яку були створені, це обумовлено технічними складностями програмного коду. Але більшість проєктів розробляють версії одночасно для декількох платформ і продаються у відповідних сервісах. Наразі основні платформи поширення ігор це ПК на Windows, Linux та MacOS, ігрові консолі Xbox та PlayStation, портативна консоль Nintendo Switch, мобільні пристрої на Android та IOS, крім того для консолей існують різні версії ігор в залежності від моделі консолі, наприклад PS3, PS4 та PS5 або Xbox 360, Xbox One, Xbox Series X.

В залежності від платформи розповсюдження ігор відбувається в цифрових магазинах і сервісах. Ігрові консолі Xbox, PlayStation та портативна консоль Nintendo Switch мають власні сервіси де поширюють ігри. Мобільні пристрої мають сервіси залежно від ПЗ, для Android це Play Market, для IOS це App Store. Персональні комп’ютери не залежно від ПЗ можуть застосовувати однакові цифрові сервіси, які в свою чергу крім сайту мають застосунки через які відбувається завантаження ігор та програм, в деяких випадках можна отримати файл завантаження на email. Найпопулярніші сервіси для покупки та завантаження ігор на ПК наступні: Steam, GOG Galaxy, Humble Bundle, Epic Games, Xbox (store). Також деякі видавці випускають ігри у власних сервісах, таких як Ubisoft Connect, EA Play, Battle.net, тощо.

Steam є найпопулярнішою платформою поширення ігор на ПК. Крім цифрового магазину він має інтегровану структуру спільноти, систему форумів, майстерню для розробників модифікацій і контенту по іграм, регулярно організовує спеціальні ігрові премії та сезонні події зі знижками. Steam має одну з найкращих систем багатокористувацької інтеграції, має такі можливості як Remote Play, Big Picture, а віднедавна і власну портативну консоль Steam Deck.

Найбільшими споживачами ігрових продуктів серед країн є Китай, США та Японія, що за різними підрахунками мають 700, 200 і 75 мільйонів гравців відповідно. Загалом ринки збуту ігор, на рівні з ігровими серверами поділяють на регіони, для зручності врегулювання ігор та сервісів на законодавчому рівні,

через різницю часових поясів, через регіональне кодування та в залежності від складності підключення гравців до багатокористувацьких матчів через затримку передачі інформації. Зазвичай це регіони Північна Америка – 219 млн, Латинська Америка – 315 млн, Європа – 430 млн, Африка та Близький Схід – 488 млн, Азія та Океанія 1,746 млрд. Іноді регіони можуть об’єднуватись або додатково розділятись в залежності від характеру ігрового продукту та сервісів.

В залежності від регіонів змінюються і вподобання гравців. Наприклад в США більш популярні ігрові консолі, тоді як в Європі переважають ПК, а в Китаї популярнішими є мобільні пристрої. З цього випливає і той факт що більшу частину ринку наразі займає саме сегмент мобільних ігор.

Регіони використовуються також для регулювання цін на програмні продукти, згідно міжнародного законодавства та враховуючи економіку окремих країн, для прикладу ігри, що коштують в США 60 доларів, в Україні можуть коштувати 20 доларів, все завдяки регіональним цінам. Саме тому контролюється розповсюдження продуктів в межах того регіону де вони були придбані.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ДИЗАЙН

2.1. Графіка

Для початку розробки гри, після затвердження тематики, слід визначитись з типом графіки та візуальним стилем. Загалом визначають два основні типи комп'ютерної графіки в іграх – 2D та 3D.

3D графіка

3D графіка частіше застосовується в AAA-проєктах ("triple-A", у ігровій індустрії, високобюджетні якісні проєкти великих і середніх студій) орієнтованих на реалістичну графіку та фізику, з увагою до деталей та спецефектів. Прикладом може виступити серія ігор Call of Duty, українські франшизи Metro та S.T.A.L.K.E.R, польський The Witcher, та інші подібні високоякісні блокбастери. Але 3D графіка не завжди є фото реалістичною, також існує безліч проєктів з низько полігональною чи стилізованою графікою, крім цього вона може застосовуватись в 2D орієнтації умовного платформеру. Зазвичай вибір між 2D та 3D, в першу чергу, залежить від фінансової спроможності компанії, та від художнього бачення проєкту. Оскільки 3D графіка потребує багато ресурсів не лише для відтворення на пристроях споживачів, вона також потребує багато коштів та ресурсів під час розробки, глибокої оптимізації та продуманої будови проєкту.

2D графіка

2D графіка в свою чергу є дешевшою в розробці, простіше візуалізується і тому більш популярна в інді-іграх («indie-ігри» та «indie-студії», скорочення від independent – незалежний, тобто невеликі незалежні студії та їх проєкти). Переважно 2D ігри розробляються у жанрах платформер та сайдскролер, різниця між ними полягає у побудові рівнів. У випадку платформера вони складаються з умовних платформ, розташованих у просторі, на яких розміщуються ігрові об'єкти та здійснюється переміщення гравця. Сайдскролер (side-scroller) схожий на платформер за виключенням відсутності різних платформ і наявності лиш однієї, якою здійснюється переміщення гравця вліво або в право. Загалом дані стилі побудови рівнів часто поєднують, утворюючи різноманіття рівнів, як

горизонтальних в стилі сайдскролерів, так і вертикальних з елементами платформера. Також бувають 2D ігри з видом згори в одній площині, найяскравішим прикладом є культова гра Hotline Miami, яка завоювала серця геймерів своєю динамічністю, яскравою графікою та неповторним саундтреком, і вже стала класикою сучасної індустрії відеоігор та поп культури.

2.5D графіка

Окремо слід виділити так звану 2.5D графіку, це вид 2D графіки, який різними способами імітує 3-й вимір, наприклад:

- за допомогою карт тіней та нормалей текстур, які надають плоским об’єктам об’ємного вигляду (також застосовується і в 3D графіці для більшої деталізації та зменшення навантаження на графічний процесор);
- зображення об’єктів під іншим кутом, з врахуванням перспективи;
- застосування ефекту паралаксу, який імітує глибину за рахунок різної швидкості відносно камери об’єктів, що знаходяться на різній відстані – чим далі об’єкт від камери тим повільніше він рухається, переважно застосовується для заднього фону в 2D іграх.

Але, мабуть найпопулярнішим варіантом 2.5D графіки є використання ізометричного відображення ігрових об’єктів і гри в цілому, так звана 2D ізометрія. В такому зображенні об’єктів ігнорується перспектива, вони подаються в ортографічному вигляді – лінії проєкції перпендикулярні до площини зображення, приклад наведено на Рис. 2.1.

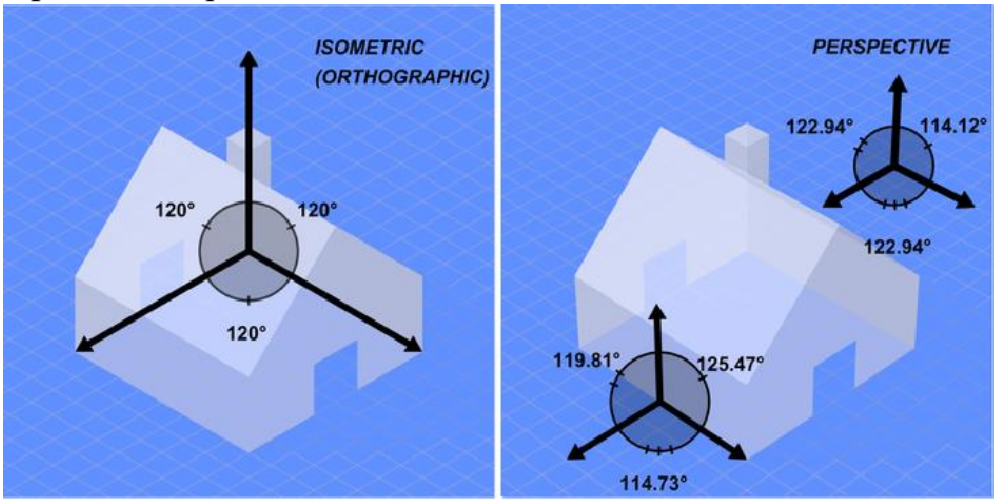


Рис. 2.1 Порівняння відображення об’єктів з використанням ізометрії (праворуч) та перспективи (ліворуч)

Візуальний стиль

Для дипломної роботи була обрана гра з 2D графікою у жанрі платформер, далі необхідно визначитись з візуальним стилем. Візуальний стиль сучасних 2D ігор в першу чергу можна розділити за типом графіки – векторна та растрова. Векторну графіку зручно використовувати для адаптації зображення до дисплеїв з різним розширенням, оскільки це не призводить до погіршення чіткості зображення при його масштабуванні, вона здатна краще передавати чіткі лінії та форми (Рис. 2.2). Але існує певна складність у створенні такого типу графіки, оскільки створення векторних зображень відбувається шляхом застосування математичних формул та їх розрахунків, і спеціалізованого ПЗ для її створення в рази менше ніж для растрової графіки. Тому переважна більшість проєктів створена із застосуванням саме растрової графіки.

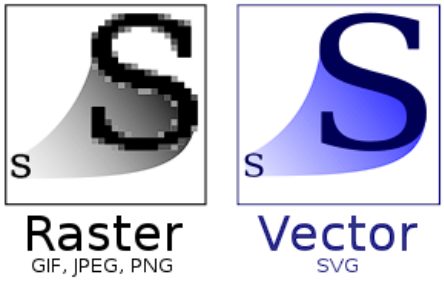


Рис. 2.2 Приклад результату збільшення растрового і векторного зображень, а також формати для збереження файлів

Растрова графіка в 2D іграх має дві основні течії, звичайна, так би мовити, мальована графіка, та піксельна графіка з низьким розширенням (Рис. 2.3).

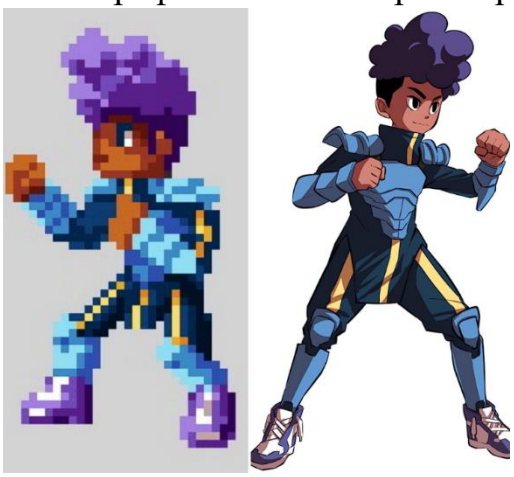


Рис. 2.3 Порівняння: ліворуч – піксельна стилізація зображення, праворуч – традиційний цифровий малюнок

Мальована графіка в іграх з’явилась відносно нещодавно, з розвитком

графічних редакторів та збільшенням роздільної здатності дисплеїв. Сучасні ігри з такою графікою нагадують мультфільми, а іноді створюються по вже відомим мультиплікаційним продуктам. Але справжньою класикою жанру 2D ігор, особливо платформерів, є піксельна графіка, яка з'явилась у часи перших відеоігор та комп'ютерів, вона настільки полюбилась геймерам, що виник окремий напрямок цифрового мистецтва, який називається піксель арт (Pixel art).

2.2. Pixel art

Піксель арт це те з чого все починалось. Коли з'явились перші ігри, для відображення ігрових об'єктів програмувались конкретні пікселі дисплею, а оскільки їх роздільна здатність була досить мала, так і виникли перші піксельні зображення (Рис. 2.4). До речі попередником піксель арту або нецифровим аналогом можна вважати традиційну вишивку хрестиком або бісером.

Рис. 2.4 Один з перших піксель артів, піксельне зображення прибульця з гри Space Invaders (1978)

По мірі розвитку технологій, збільшувалась роздільна здатність дисплеїв, через що збільшувалась роздільна здатність ігор і піксельної графіки загалом. На заміну монохромним дисплеям прийшли кольорові з чіткішим зображенням і більшою частотою оновлення (Рис. 2.5). Почалась ера ігрових приставок.

Рис. 2.5 Зображення гри The Chaos Engine (1993)

З появою та розвитком 3D графіки, її спершу почали використовувати для попереднього рендеру 3D асетів та переносу їх у гру у вигляді піксельних зображень, а згодом використовувати для розробки повноцінних 3D ігор. Ігрова індустрія поступово почала переходити на новий вид графіки, а 2D графіка лишалась на старих платформерах та нових не дуже дорогих проєктах. Ностальгія від старих ігор і нові технології, які дозволяли буть кому самостійно створювати 2D графіку, дали підґрунтя для відродження піксель арту як окремої течії сучасного цифрового мистецтва (Рис. 2.6). Піксель арт зміг вибороти своє місце у сучасному мистецтві, еволюціонував, але зберіг при цьому свою сутність і відродив моду на 2D ігри уже в окремому жанрі.

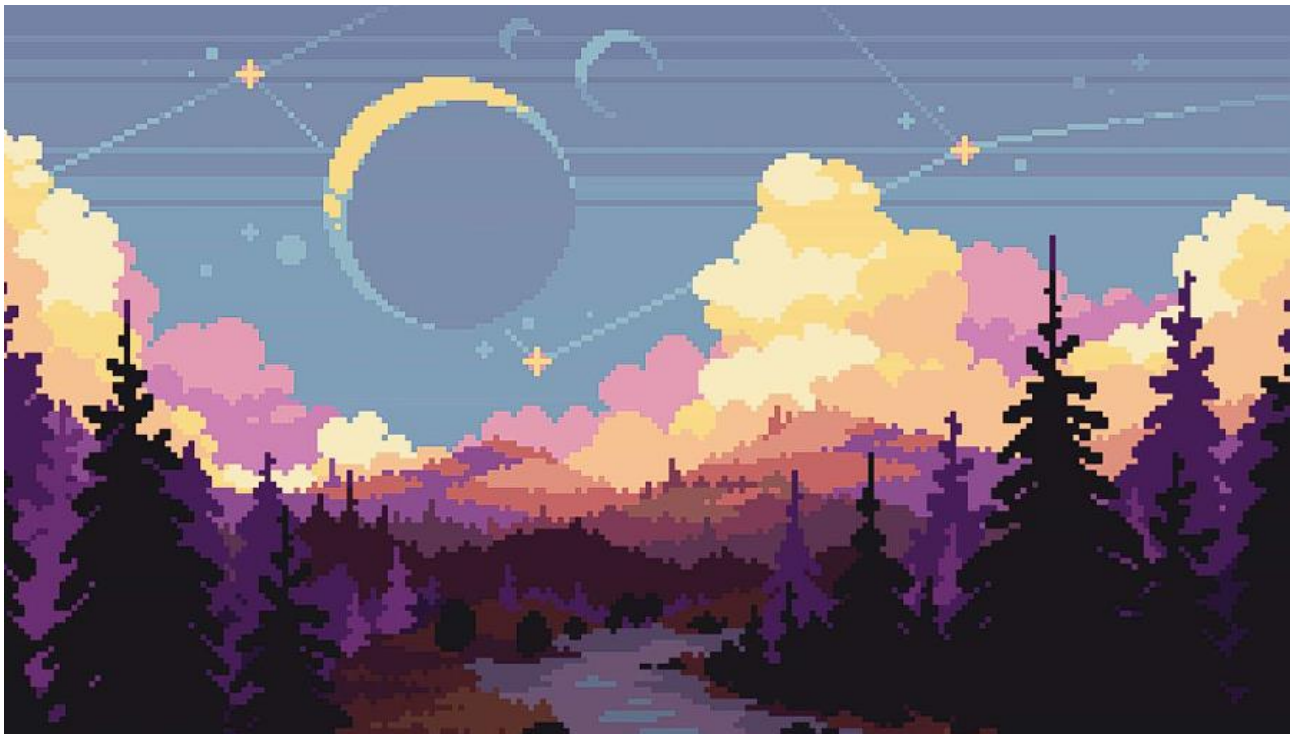


Рис. 2.6 Приклад сучасної композиції в стилі піксель арт

В сучасному піксель арті розвинулись певні правила і техніки створення зображень, адже тепер вони створюються не через обмеження дисплеїв і обчислювальної здатності пристроїв, а виключно з естетичною метою.

В першу чергу постає питання масштабу. Яку інформацію буде нести 1 піксель, яка роздільна здатність буде у того чи іншого об’єкту, який розмір полотна. Все це вибір художника, він вирішує буде персонаж чи об’єкт заввишки 16 пікселів або 64, наскільки він буде деталізованим, пропорційним та скільки кольорів буде застосовано. Є певна складність у зображенні чого-небудь у

маленькій роздільній здатності. Зображення повинно зберігати впізнаваність і дотримуватись загального стилю та масштабу, іноді один піксель має велике значення. Піксель арт це не про маленьку роздільну здатність, це про вибір які пікселі варто лишити.

Форма

Під час аналізу спрощених піксельних зображень, наш мозок, не свідомо, отримує різного роду інформацію. Правильно сформована композиція та образ зможуть донести потрібну візуальну інформацію та задум автора. Так само як художники починають розробку композиції з силуету, вбачаючи у вигинах і формах деталі майбутнього персонажу чи об’єкту, так і глядач може сприймати різні об’єкти в першу чергу за формою (Рис. 2.7). Форма це один з ключових принципів передачі візуальної інформації, який активно використовують і в процесі розробки відеоігор, наприклад:

- трикутна форма – частіше всього означає небезпечні об’єкти, вона привертає увагу і асоціюється з чимось гострим та небезпечним, під час дизайну злих персонажів її також часто застосовують;
- квадратна форма – зазвичай створює уявлення про велику вагу і громіздкість об’єкту, твердість та міцність, також нейтральність і стійкість, у дизайні персонажів це зазвичай означає велику силу та непохитність;
- кругла форма – використовується для зображення гладкості, м’якості та безпеки, круглі об’єкти частіше всього виконують функцію допомоги, вони асоціюються з мобільністю та пружністю, круглі форми зазвичай застосовуються у дизайні добрих персонажів.

Рис. 2.7 Приклад силуетів різних геометричних форм

Змінювання пропорцій теж вносить певні корективи у сприйняття, наприклад загострення трикутної форми матиме більший ефект асоціації з

небезпекою, заокруглення квадрату матиме ефект пом’якшення та гладкості, а розтягування круглої форми може впливати на сприйняття його ваги та швидкості.

Колір

Окрім форми велике значення має колір. Колір може нести інформацію про тип об’єкту і наповнювати загальну атмосферу, зокрема завдяки використанню так званої температури кольорів: сині відтінки – холодні, жовті та червоні – гарячі. За допомогою кольорів позначають і доповнюють візуальну інформацію про об’єкти, із ключових прийомів та асоціацій можна виділити наступні:

- Червоний колір – використовують для позначення небезпечних об’єктів та ворогів, також ним зазвичай позначають шкалу здоров’я персонажів, об’єкти що відновлюють здоров’я;
- Жовтий колір – науково доведено, що цей колір помічається людським оком та мозком найшвидше, тому всюди і в тому числі в іграх, його переважно застосовують щоб привернути увагу до чогось, наприклад позначити шлях чи важливий об’єкт;
- Зелений колір – в основному зображає відновлення, позитивні об’єкти, дружні персонажі, природне походження об’єктів, часто його використовують для позначення об’єктів що відновлюють здоров’я персонажів, амуніцію і тд;
- Синій колір – в першу чергу він асоціюється з водою, спокоєм, рівновагою, добротою, його також застосовують для позначення дружніх персонажів, крім цього часто саме цей колір обирають для зображення предметів, пов’язаних з інтелектом або магією, в тому числі серед елементів інтерфейсу.

Приклади наведені на Рис. 2.8.

Рис. 2.8 Приклад поєднання форм та кольорів для відповідних асоціацій

Зміна тональності і різне комбінування форм та кольору може суттєво впливати на візуальну інформацію образу предмета чи персонажа. Особливу роль грає контраст, він може сильно вплинути як на самі кольори, так і на атмосферу яка утворюється. Наприклад, більш яскраві та насичені кольори створюють більш веселу атмосферу та сприйняття, в той час як зменшення контрасту сприяє утворенню більш драматичної картинки у сірих тонах (Рис. 2.9)

Рис. 2.9 Приклад впливу контрасту на сприйняття зображення

Shading

Кольори також є інструментом, завдяки якому передається відчуття об’ємності предметів та форм. Ця технологія називається шейдингом (shading) або затіненням. Вона полягає у використанні різної тональності або кольорів для виділення граней і форм об’єктів, таким чином імітуючи відбиття світлових променів від поверхней в залежності від розміщення їх відносно джерела світла (Рис. 2.10), хоча останнім у піксель арті іноді нехтують для спрощення технічних аспектів, зображаючи вплив лише одного джерела світла. Затінення один з ключових інструментів передавання візуальної інформації в піксель арті, воно вселяє життя у двовимірну графіку і дає змогу відчути вагу і форму купки пікселів на екрані.

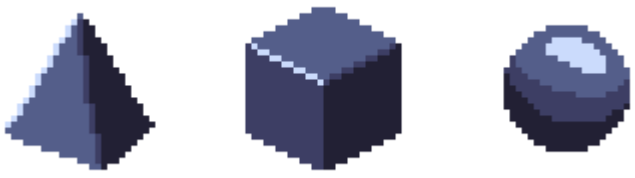


Рис. 2.10 Приклад застосування затінення для візуалізації об’єму

Anti-Aliasing

Для покращення сприйняття зображення і пом’якшення контрасту була

розроблена технологія, яка сьогодні активно застосовується в розробці відеоігор та реалізації комп’ютерної графіки – anti-aliasing або згладжування. Принцип даної технології полягає у додаванні проміжного шару кольору, який є середнім між двома кольорами що межують. Таким чином відбувається розмиття граней, пом’якшення зображення та його сприйняття.

В 3D графіці ця функція допомагає візуально згладжувати краї графічних об’єктів, роблячи зображення не таким різким і без пошарпаних контурів, але іноді це може викликати занадто сильний ефект і зображення стане надто розмитим, для виправлення цього винайшли кілька інших технологій та навіть інтегрували нейронні мережі для оптимізації процесу.

У 2D графіці, а саме у піксель арті все простіше, згладжування застосовується для зменшення різкості контрасту, вздовж контурів зустрічі різних кольорів, додаючи кілька пікселів змішаного кольору (Рис. 2.11). Таким чином наш мозок згладжує точки зустрічі кольорів і створює ефект гладкого переходу між ними, і чим вища роздільна здатність зображення, тим сильніший ефект.

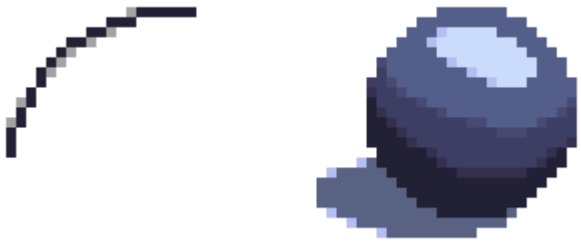


Рис. 2.11 Приклад застосування ефекту згладжування (anti-aliasing)

Dithering

За схожим принципом працює і наступна техніка, яка застосовується у піксель арті, а саме dithering або тремтіння. Ця техніка також застосовується для згладження переходу між кольорами в умовах низької роздільної здатності зображення, але вона більше націлена на створення додаткових відтінків двох кольорів з використанням лише цих самих кольорів. Іншими словами це градієнт але в умовах низької роздільної здатності. Виникла ця технологія в часи коли палітра кольорів у піксель арті і на дисплеях того часу загалом, була дуже обмежена, і тому це був єдиний варіант створення рівномірного переходу між

двома кольорами. Сьогодні ж кількість підтримуваних кольорів та відтінків вражаюча і досить просто підібрати потрібні відтінки для всіх потреб. Але ця техніка досі використовується в естетичних цілях або для повною мірою відтворення аутентичних зображень з минулої епохи.

Основний принцип полягає у накладанні двох кольорів решітчатим способом, умовно кажучи перехрещуючи матриці, поступово зменшуючи вкраплення кольорів один в одного (Рис. 2.12).

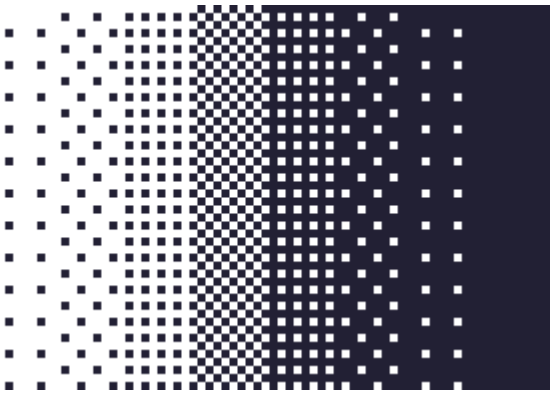


Рис. 2.12 Приклад застосування техніки Dithering (тремтіння)

Outline

Варто згадати виключно естетичну техніку виконання піксель арту. Вона застосовується за бажанням художника і несе стилістичну функцію. Мова про outline або контур. В багатьох піксель арт роботах або іграх з піксельною графікою можна зустріти даний стиль зображення (Рис. 2.13). Суть досить проста, окремим об’єктам додається контрастний контур. Таким чином можна краще відокремити будь-що від заднього фону, для кращого розпізнавання ігрових об’єктів. Частіше контур застосовується в піксель артах з досить низькою роздільною здатністю, щоб об’єкти не зливалися і мали кращий естетичний вигляд.

Рис. 2.13 Приклад застосування контуру для персонажів у грі Celeste

2.3 Анімація

Перейдемо до застосування піксель арту безпосередньо в процесі розробки відеоігор. У будь-якій комп’ютерній грі ігрові об’єкти мають бути інтерактивними, персонажі мають рухатись, вороги нападати, у цілому має щось відбуватись на екрані. Все це забезпечується в першу чергу завдяки створенню анімацій.

Анімація виникла ще у XIX столітті у вигляді стробоскопічного диску, на якому були зображення, що під час швидкого обертання диску викликали ефект руху, так виник сучасний принцип анімації. А вже на початку XX століття почали з’являтись перші мультфільми фільми та мультфільми з використанням плівки та проекторів, у випадку мультфільмів плівка складалась з фотографій малюнків.

Основний принцип роботи анімації чи відеоматеріалів це швидке заміщення одного зображення іншим, що трохи відрізняється від попереднього. Ці зображення називаються кадрами, і чим їх більше та менш помітні зміни, тим чіткішою і плавною буде анімація. Але все також залежить від частоти прокрутки кадрів, вона має бути достатньо швидкою, аби людське око не розпізнавало перехід між кадрами, а самі кадри мають бути розраховані на демонстрацію з конкретною частотою, аби не виникало ефекту пришвидшення чи сповільнення анімації. Загалом стандартом частоти кадрів при якому людське око не розпізнає перехід між кадрами прийнято вважати 24 кадри в секунду.

24 кадри в секунду

Чому саме 24? Перш за все це пов’язано з додаванням звуку у німе кіно, яке знімали із частотою кадрів від 16 до 24 в секунду. При цьому і камери і проектори були ручними, через що частота кадрів була не рівномірною і часто змінювалась у відповідності до сцени. Більше того частоту могли змінювати і під час показу кіно в залежності від ситуації на екрані або для нівелювання різної частоти зйомки та стабілізації зображення. Також німе кіно переважно показували пришвидшеним. Через це, з появою звукової плівки, маніпуляції з частотою прокрутки плівки припинили, оскільки зміна частоти звуків людським

слухом розпізнається досить добре і загалом це могло нести негативний характер. Тож частоту вирішили вирівняти, а оскільки більшість тогочасних кінотеатрів показували фільми з частотою в діапазоні від 22 до 26 кадрів в секунду було взято за стандарт 24 кадри.

Але якщо ми здатні розпізнавати частоту 24 кадри в секунду, навіщо тоді існують сучасні дисплеї з частотою 120 чи навіть 300 кадрів в секунду? Справа втім, що зоровий орган людини, який називається зоровим аналізатором здатен сприймати 1000 окремих зображень на секунду, але для очей частота стає непомітною на 150-240 кадрах, загалом ми здатні окремо сприймати 10-12 зображень в секунду, а вища частота вже буде сприйматись мозком як рух об’єктів. Розглянемо технологію демонстрації кадрів на проекторі та як працюють сучасні дисплеї, для цього використовується світло, а миготіння світла приблизно на рівні 50 Гц сприймається людським оком як стале світіння і називається порогом злиття миготіння. Саме від цього показника і відштовхуються під час планування частоти оновлення дисплеїв та кадрів в анімації. На цьому наголошував американський науковець і винахідник Томас Едісон ще за часів німого кіно, мовляв повільніша частота буде напружувати очі. Тому хоч фільми тоді показували з частотою 24 кадри в секунду, проектори були налаштовані на дво- або три-разове підсвічення кожного кадру, і тому фактично кіно показували з частотою 48 зображень в секунду.

Таким чином можна розділити поняття частоти кадрів в секунду, як послідовності унікальних зображень, від частоти оновлення дисплею чи демонстрації зображень проекторами. Для зручності розділення понять використовуються терміни Frame rate (частота кадрів, або FPS - frame per second, кадри в секунду) та Гц (герци, частота оновлення). FPS застосовується до анімацій та відео контенту, позначаючи плавність анімацій, а Гц використовуються для позначення частоти оновлення дисплеїв, їх здатності з показу окремих зображень за секунду.

Наразі стандартом частоти оновлення дисплею вважають 60 Гц і вище, це мінімальний поріг який оптимально сприймається очима не напружуючи їх, крім

того застосування частоти кадрів на такому ж рівні дає відчутну і оптимальну плавність анімації.

Види 2D анімацій

З часу виникнення анімації минуло багато часу, технології і техніки розвивались, слід виокремити 2 основні типи анімацій, що застосовуються у двовимірній графіці. Анімація з ключовими кадрами (Keyframe) та анімація за методом Straight Ahead (прямо вперед) започаткована аніматорами студії Disney і вперше згадується у книжці 1981 року «12 базових принципів анімації».

Основна різниця між цими двома методами полягає у наявності ключових кадрів. Якщо у варіанті анімації з ключовими кадрами процес відбувається зі створенням ключових положень об’єктів між якими згодом малюються проміжні кадри для згладжування переходу між ними. У випадку прямої анімації є тільки перши ключовий кадр, а далі «прямо вперед» аніматор малює кожен кадр окремо не знаючи куди це його приведе. Таким чином виникала певна творча свобода, занурення в процес анімації, але це мало певні ризики і могло нести негативні наслідки. Оскільки відсутні ключові кадри, на які можна орієнтуватись і підтримувати зображення однаковим, наприклад розмір об’єкта чи його форма. Перевірити це можна було лише в кінці, коли всі кадри були створені.

Ключові кадри допомагають простіше спланувати анімацію і відслідковувати її створення. Більше того, це дозволяє не створювати ідеально однакові проміжні кадри, наприклад під час якихось активних сцен де персонаж чи предмет рухається, в проміжних кадрах можна розтягнути чи розмити об’єкт, саму дію, щоб зробити перехід між ключовими кадрами більш гладким або навпаки прибрати проміжні кадри і зробити перехід біль різким та динамічним. Такий прийом в основному використовують в японській анімації, так званому Аніме, в ньому дуже багато динамічних сцен і переходів. Сприяло цьому перш за все те, що аніме створюється по манзі, японських коміксах, але на відміну від американських мультфільмів, аніме максимально намагається зберегти оригінальний візуальний стиль коміксів і пришвидшити процес анімації, тому може буквально використовувати ключові кадри з коміксів і зв’язувати їх

проміжними. І саме для полегшення роботи та збільшення плавності чи динаміки, проміжні кадри можуть не зберігати достовірні форми ключових кадрів, а їх кількість може зменшуватись або взагалі не використовуватись. Аніме в цілому є дуже важливим елементом розвитку анімації та кінематографу, через розвиток і поширення власних стилів, ракурсів, зображення емоцій, динамічних сцен і фантастичних сценаріїв в інші культури. Багато прийомів що виникли в процесі його розвитку застосовують в фільмах та мультфільмах по всьому світу.

З рештою можна сказати що в сучасній 2D анімації використовується 2 види анімацій, і обидва вони використовують ключові кадри. Перший, похідний від старого методу анімації студії Disney без ключових кадрів, тепер використовує ключові кадри, але вимагає детально промальовувати кожен кадр, цей метод більше притаманний західним студіям анімації. Другий спосіб це якраз метод що застосовується в японській анімації, коли в основному використовуються тільки ключові кадри, а проміжні кадри між ними виглядають плавніше або ж навпаки відсутні.

В розробці 2D ігор також застосовується прийом з аніме, крім того він здебільшого переважає, оскільки це додає картинці динаміки і спрощує процес анімації. Наприклад для анімації різкого удару мечем, можна використати ключові кадри початкового положення меча та кінцевого, а між ними один кадр що зображає лінію удару. Це можна застосувати і до інших моментів у грі, наприклад до різкого випаду вперед чи переміщення. Такий прийом з анімацією випаду називається Dash (ривок) і він вже давно виріс в окрему механіку в відеоіграх, а в деяких з них вона є ключовою, на якій побудований ігровий процес.

Sub-pixel

У відеоіграх з піксельною графікою також розвинувся всій підвид анімації, який покликаний збільшити плавність анімації зображень з низькою роздільною здатністю. Це так звана субпіксельна (sub-pixel) анімація. Ідея полягає в тому щоб зобразити плавний перехід і створити відчуття зміщення у наступному кадрі

зі швидкістю менше 1 пікселя на кадр. Це досягається за рахунок візуального сприйняття таких аспектів як:

- Розмазування елементів
- Анімація контурів
- Анімація кольорів

У випадку розмазування елементів відбувається додавання кількох додаткових пікселів для пом’якшення і візуального сповільнення анімації переміщення об’єктів. Наприклад, для анімації руху одного пікселя, замість зображення його на кожному наступному кадрі на 1 піксель далі від положення на початковому кадрі, малюється не один піксель а декілька, які плавно перетікають з одного положення в інше. Тобто замість руху одного пікселя, цей розтягується на декілька пікселів, рухається, потім так само плавно стискається до одного пікселя і зупиняється. Виходить такий собі ефект розмиття.

Стосовно анімації контурів, під нею розуміється деформація контурів об’єкту в процесі переміщення. Зменшення або збільшення округлості контурів на проміжних кадрах, в умовах піксельної графіки це виглядає як подовження або скорочення піксельних сходинок які імітують похилі лінії. З рештою це викликає ефект наче об’єкт плавно рухається і через це трохи деформуються його контури в низькій роздільній здатності.

Анімація кольорів схожа до розмазування елементів, але замість розтягування пікселя на декілька інших, відбувається затемнення кольору на початковій позиції пікселю і одночасне помірне зафарбування позиції куди цей піксель має переміститись. Можна назвати цей процес перетіканням кольору з одного пікселю в інший. Цей процес знову ж таки викликає ефект повільного переміщення пікселів.

Комплексно ці методи дозволяють суттєво покращити відчуття плавності анімацій повільних дій у піксельній графіці. Наприклад у дипломній роботі ці техніки застосовуються для покращення анімації каптура головного персонажу.

2.4 Проєктування гри

Концепція

Розглянемо нарешті безпосередньо сам проект гри. Як і має бути – розробка почалась з роботи над концептом, адже найважливіша частина роботи – концептуальна. Спершу треба придумати концепцію: «що?», «як?», «для чого?», «чому?», «навіщо?», «який все матиме вигляд і як співпрацюватиме?». Це мабуть найдовша й найголовніша частина розробки ігор. І, часом, видимих результатів у ній найменше на цьому етапі, бо, переважно, основний кістяк концепту залишається в головах розробників, і вже потім, коли з'явилася певна логіка та єдине бачення майбутнього проєкту, з'являються перші системи і моделі роботи. Перші вагомі результати роботи над концепцією з'являються вже на етапі розробки, але при цьому фазу концепції не можна вважати повністю завершеною, бо нові ідеї можуть виникати в процесі розробки, як через творчий порив, так і через складнощі в процесі реалізації конкретного аспекту.

Отже, концепція гри наступна. Події відбуваються у майбутньому, коли людство почало досліджувати космос та опановувати планети сонячної системи, оскільки материнська планета перестала бути придатною для життя. Безпосередньо події гри будуть відбуватись на Венері. З художньою метою можна недотримуватись реалізму стосовно сприятливості умов даної планети для життя людей. Тому у грі Венера буде зображена як заселена людьми, хмарна планета з мегаполісами вкритими густим туманом (Рис. 2.14).

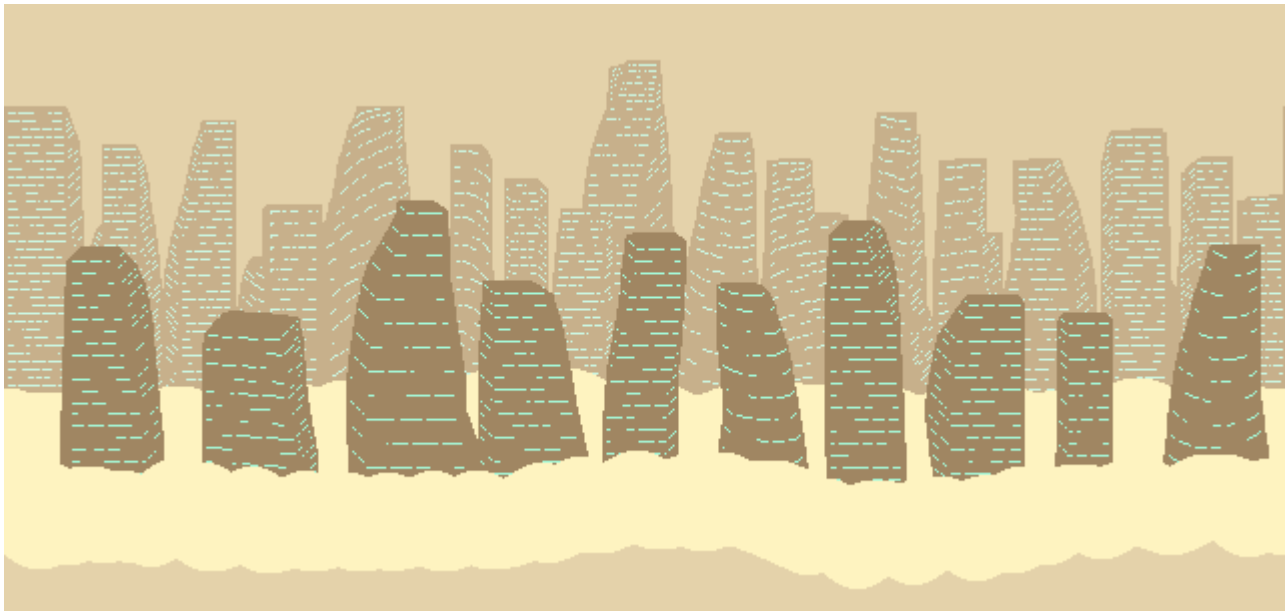


Рис. 2.14 Розроблений задній фон гри, що демонструє Венеру

Головний герой це найманець з досвідом та певними навичками. Він

загадковий і ніколи не знімає свого каптура та маски, що під ним ховається. В його арсеналі є зброя ближнього і дальнього бою. У нього є певні принципи та стиль поведінки, він також за своєю легендою є ветераном не одного конфлікту. Загалом характер персонажа похмурий і виважений, він робить те, що має і те що вважає за потрібне чи правильне. Дизайн персонажа зображений на Рис. 2.15.

Рис. 2.15 Дизайн головного персонажу гри

Під час роботи над концепцією персонажа я надихався серіалом Mandalorian, а всесвіт гри будувався під впливом аніме Cowboy Bebop та одно іменного серіалу адаптації від компанії Netflix. Початкові концепції світу гри змінювались і корегувались в залежності від різних нових джерел натхнення, випадкових думок, наукових статей та подій в житті. Врешті є багато різних ідей і переконань, які я збирав довгий час, що утворили у мене загальне уявлення про вигаданий світ, його правила й історію, від яких я відштовхуюсь в процесі розробки.

Від багатьох речей довелось відмовитись або переосмислити в процесі роботи над дипломним проєктом, в першу чергу через брак часу та або знань, необхідних для реалізації певних ідей. Для прикладу, напочатку гра розглядалась як проєкт жанру метроїдванія (metroidvania жанр відеоігор, що виник від поєднання ігроладу ігор Metroid та Castlevania), тобто пригодницька 2D гра з нелінійним сюжетом, різноманітними рівнями та планетами, купою механік та застосованих у розробці технологій. Але згодом було вирішено представити проєкт у вигляді демоверсії в жанрі платформер, для полегшення та пришвидшення процесу розробки, з орієнтацією на подальший розвиток у щось більше.

Aseprite

Вид та стиль графіки для гри, від самого початку було обрано як 2D та піксель арт. Для розробки графічної складової було використано програму Aseprite. Це графічний редактор, повністю орієнтований на створення піксельної графіки: піксель артів, асетів для ігор, анімацій та навіть тайлсетів (tileset) – набору так званих плиток текстур, з яких будується ігровий рівень. Слід зауважити, що піксельна графіка в іграх будується двома шляхами:

- з використанням тайлсетів – малюються окремі блоки/плитки, які мають зображати різні варіанти і типи поверхонь та переходів між ними (підлога, стеля, стіна, блок всередині стіни, трава, земля, тощо), які потім формують ігровий рівень з відповідною логікою розміщення цих блоків згідно дизайну рівнів (level design);
- з використанням асетів – всі ігрові об’єкти малюються цілими асетами, не розбиваються на плитки, рівень складається з великих асетів, що розміщуються відповідно творчому задуму та ігровому дизайну, такий вид графіки є привабливішим з точки зору зовнішнього вигляду але може нести певні втрати продуктивності через велику кількість великих і унікальних текстур, що також впливає на розмір файлів гри.

Aseprite виник, як проєкт з відкритим вихідним кодом і у вільному доступі у 2001 році, але починаючи з 2016 року перейшов на модель платної ліцензії і поширюється, зокрема через Steam. Цей продукт активно розвивався відповідно до відгуків спільноти піксель арт художників та тих, то просто спробував програму. Наразі вона налічує безліч корисних інструментів та функцій для піксель арту, серед яких можна виділити наступні.

Для кольору та малювання:

- налаштування палітр кольорів;
- налаштування контрасту;
- колірне колесо для зручного вибору відтінків кольорів;
- Shading Mode – режим затінення;
- Pixel Perfect Stroke – режим пензлика для малювання без подвійних

пікселів, лише діагональне сполучення;

- RotSprite Rotation – зміна ротації спрайтів (sprite) – окремих елементів малюнку;
- Tiled Mode – режим розробки тайлсетів;
- можливість додати власні пензлики;
- різноманітні режими змішування шарів та кольорів.

Для анімації та шарів:

- організація та налаштування шарів;
- створення кадрів анімацій та їх відтворення;
- впорядкування та налаштування кадрів анімацій;
- Onion Skin – режим відображення найближчих сусідніх кадрів анімації для використання в якості референсів;
- вивід в окремому вікну попереднього перегляду анімації і налаштування відображення;

Для імпорту та експорту файлів:

- збереження та відкриття послідовності кадрів у форматі .png
- створення GIF-зображень
- створення аркушів спрайтів для подальшої інтеграції в ігрові рушії
- збереження даних та файлів у разі аварійного вимкнення або неправильного закриття програми
- автоматизоване перетворення зображень інших форматів розширення файлів

Та ще безліч спеціальних специфічних функцій, які полегшують життя розробникам та художникам, наприклад збільшення масштабу зображення під час експорту, для коректного відображення на дисплеях з високою роздільною здатністю, зокрема для публікації у соцмережах, і навіть спеціальні автоматичні налаштування для публікації роботи у Twitter.

Unity

Основою для розробки гри виступає ігровий рушій Unity. Це

мультиплатформний програмний засіб розробки відеоігор, який використовується для їх створення та запуску. Програмування всередині Unity виконується на мові програмування C#, завдяки чому і забезпечується мультиплатформність. Редактор Unity працює на Windows, Linux та macOS, а додатки на базі рушія можуть запускатися на 25 різних платформах.

Також дуже зручною є можливість створювати проєкти з використанням різних версій рушія, наприклад, для продовження розробки і підтримки певних проєктів на їх поточній версії без необхідності переводити проєкт на нову версію ПЗ та його адаптацію до неї.

Unity має ряд функцій для створення проєктів орієнтованих як на 3D так і 2D графіку, у вікні створення проєкту можна побачити безліч спеціальних налаштувань для наперед визначених потреб проєктів. Також доступні вбудовані навчальні проєкти, що супроводжують користувача підказками в процесі розробки і зберігають прогрес навчання на сайті через синхронізацію акаунту користувача.

Для навчання користуванням всіма можливостями і аспектами рушія існує безліч офіційних навчальних програм та технічної документації. Команда розробників також створила багато інтеграцій в сам редактор для кращої взаємодії з навчальними програмами та додатковими системами.

Для Unity існує дуже багато додаткових програм, які автоматизують або покращують деякі процеси розробки, переважну більшість з них розробляють самі користувачі і поширюють на форумах, але також існують платні рішення від самої компанії.

Unity постійно розвивається, в нього інвестують передові компанії з розробки ПЗ та передових технологій. Його використовують для розробки ігор, додатків, різноманітного відео контенту та різного роду візуалізації. Від студентів початківців і самостійних розробників, до великих студій, підприємств та бізнесів. Це потужний і гнучкий інструмент для рішень багатьох задач з візуалізації та симуляції.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Створення проєкту

Для початку роботи необхідно створити проєкт в Unity, і в налаштуваннях обрати орієнтацію на 2D графіку. Програма сама створить кореневу папку проєкту і основу ієрархії для майбутньої системи файлів, крім того, доступ до неї буде безпосередньо з самого редактору, що дуже зручно і практично, приклад наведено на Рис. 3.1.

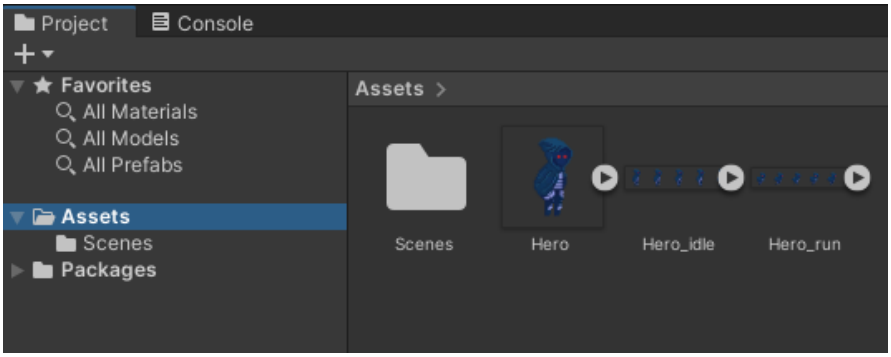


Рис. 3.1 Вікно Project з доступом до кореневої папки проєкту є всередині редактору Unity

Після створення проєкту, в ієрархії об’єктів вже одразу є створений об’єкт «Головна камера», яка демонструє межі простору редактора, які потраплять на екран під час запуску програми. Інше кажучи, це поле зору гравця. Камера, як і будь-який об’єкт в Unity має вікно параметрів Inspector, в якому відображається базовий набір налаштувань об’єкту (Рис. 3.2), а також додані з допомогою скриптів функції та параметри.

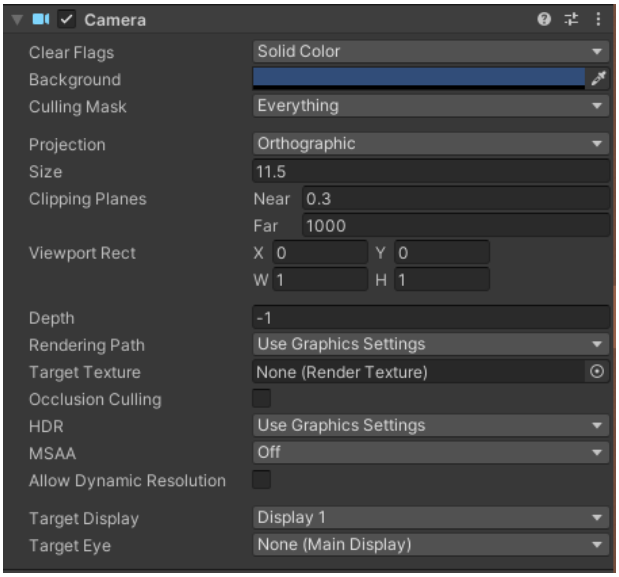


Рис. 3.2 Приклад набору параметрів об’єкту Камера

Наразі проєкт пустий, в ньому немає ніяких об’єктів, окрім Камери та набору пустих папок для майбутніх асетів. Нові об’єкти додаються через вікно ієрархії сцени (Рис. 3.3), там відображаються усі ігрові об’єкти, які є свого роду контейнерами для скриптів, налаштувань і асетів. Додаймо перший власний об’єкт Hero, який буде контейнером для програмування грального персонажу.

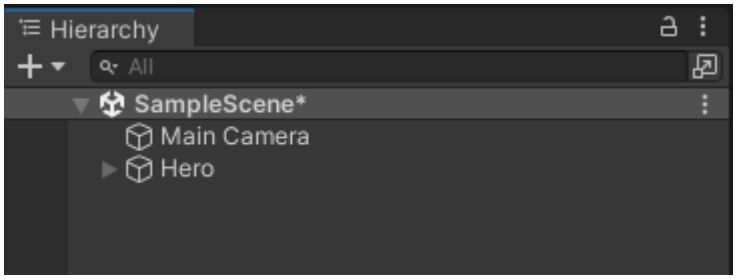


Рис. 3.3 Вікно ієрархії ігрових об’єктів поточної сцени

Нові додані об’єкти потрібно налаштовувати через вікно Inspector і з допомогою параметрів та скриптів перетворювати на необхідні нам елементи гри. Для візуального відображення об’єктів у сцені, необхідно зв’язати їх з асетами, використовуючи вікно Inspector потрібно додати новий компонент, в заданому випадку Sprite Renderer, і вже в параметрах цього компоненту, в поле з назвою Sprite треба перетягнути необхідний асет, після чого об’єкт візуалізується у вигляді обраного асету (Рис. 3.4).

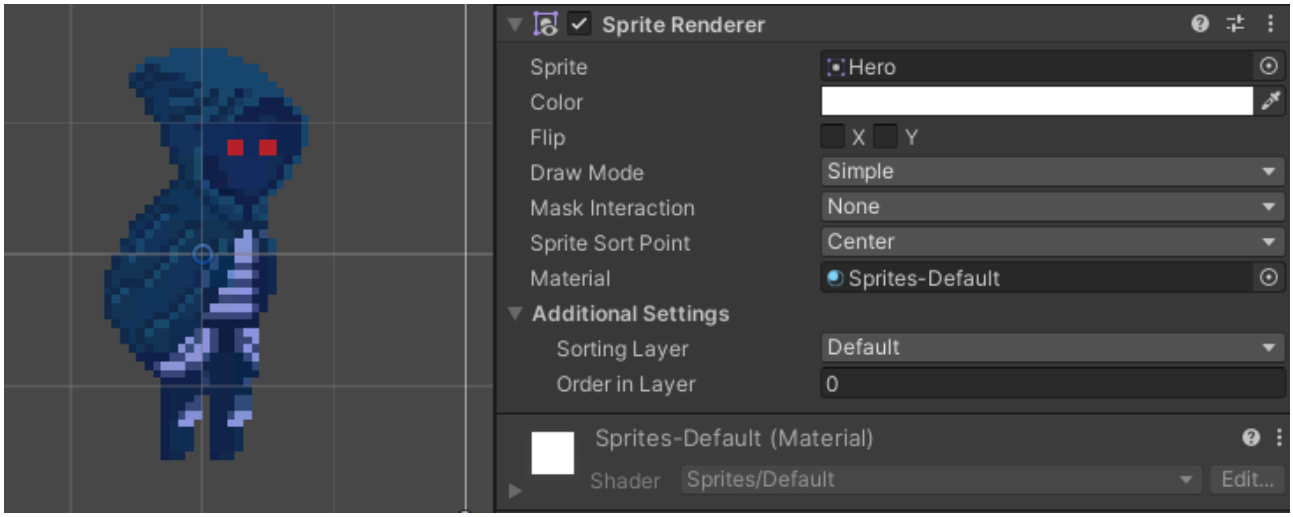


Рис. 3.4 Вікно компоненту Sprite Renderer і результат додавання асету Hero в поле Sprite.

Таким чином відбувається додавання нових об’єктів, підключення до них нових компонентів і параметрів, з допомогою звичайних функцій перетягування об’єктів та побудови логічних зв’язків.

3.2 Додавання заднього фону

Цікаво зауважити, що попри вибір орієнтації проєкту на 2D графіку, Unity зберігає 3D функції всередині проєкту. Наприклад, 2D сцена має функцію виду в тривимірному просторі, а об’єкт Камера має глибину по осі Z (Рис. 3.5).

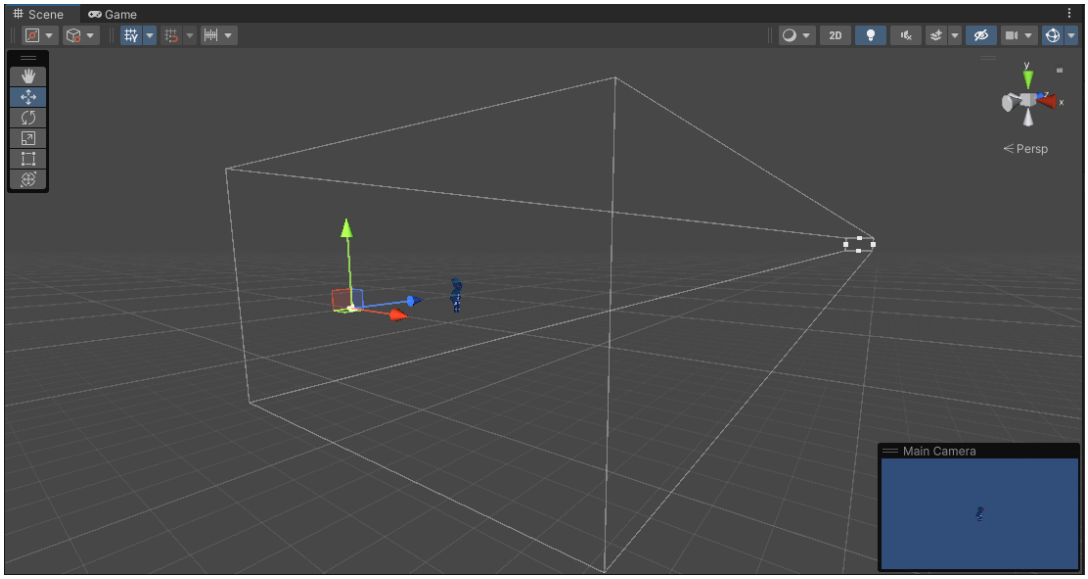


Рис. 3.5 Тривимірний вигляд сцени проєкту та об’єкту Камера

Цю функцію ми використаємо для створення заднього фону з ефектом паралаксу, для цього спершу слід створити відповідні ігрові об’єкти та додати асети у сцену. Для цього використовуємо асети заднього фону спеціально розроблені в редакторі Aseprite і розміщуємо їх в тривимірній сцені (Рис. 3.6).

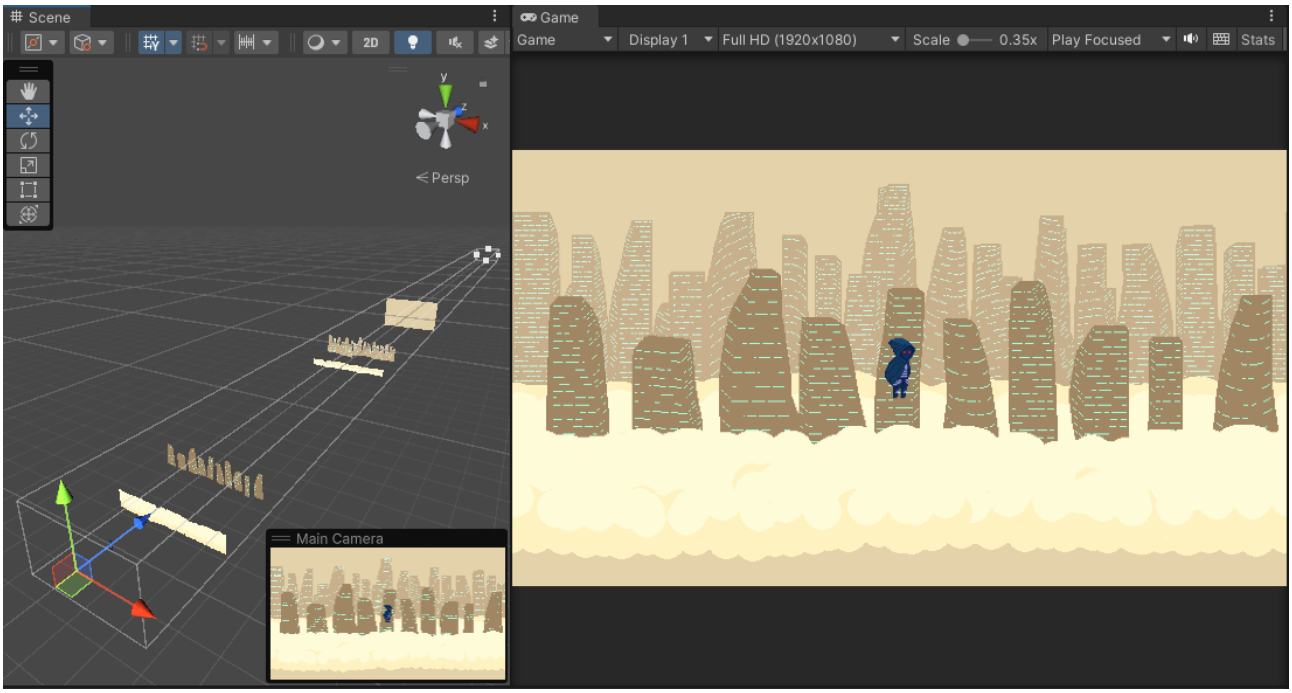


Рис. 3.6 Розміщення об’єктів у 3D сцені та їх зображення в камері

В незалежності від відстані розміщення об’єктів від камери, їх розмір у

вікні гри не зміниться, оскільки камера використовує ортографічну систему зображення, ігноруючи перспективу, тому всі об’єкти будуть зберігати пропорції в будь-якому випадку. Тепер можна перейди до програмування ефекту паралаксу, для цього необхідно додати об’єктам фону скрипт, який буде виконувати програмний код. Під час роботи зі скриптами, Unity автоматично створює файли формату .cs і підключає до них необхідні бібліотеки, включаючи бібліотеку UnityEngine, а також автоматично відкриває скрипти за допомогою Visual Studio і синхронізує зміни у файлах з проєктом в редакторі Unity.

Новостворений скрипт, як він відображається у редакторі Unity зображено на Рис. 3.7, цей самий скрипт після написання програмного коду виглядає так як зображено на малюнку Рис. 3.8.

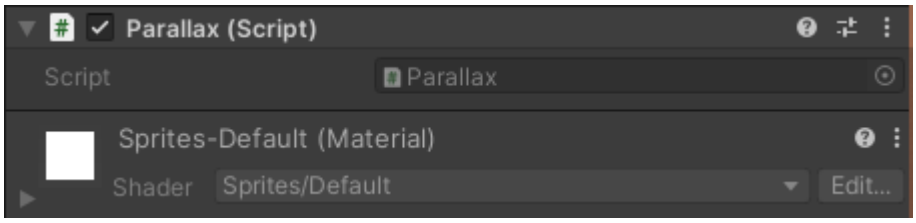


Рис. 3.7 Вікно новоствореного (пустого) скрипту

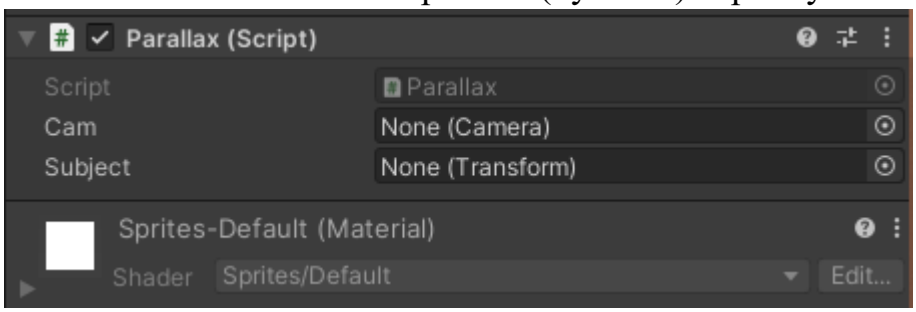


Рис. 3.8 Вікно скрипту після написання програмного коду

Скрипт Parallax – утворює взаємозв’язок між параметрами руху камери та об’єктів заднього фону в залежності від відстані між об’єктами і камерою. Чим далі об’єкт від камери, тим повільніше відбувається його зміщення в протилежний від камери бік, чим ближче об’єкт, тим швидше він рухається. Для завершення створення ефекту паралаксу лишається тільки увімкнути параметр клонування заднього фону, аби його розтягнути і утворити ефект безкінечного фону.

3.3. Створення рівнів

Створення рівнів буде виконуватись з використанням тайлсетів, тобто

викладатись вручну з маленьких плиток текстур, для початку цього процесу необхідно інтегрувати в проєкт готовий тайлсет. Після чого створити новий ігровий об’єкт Tilemap в ієрархії сцени (Рис. 3.9)

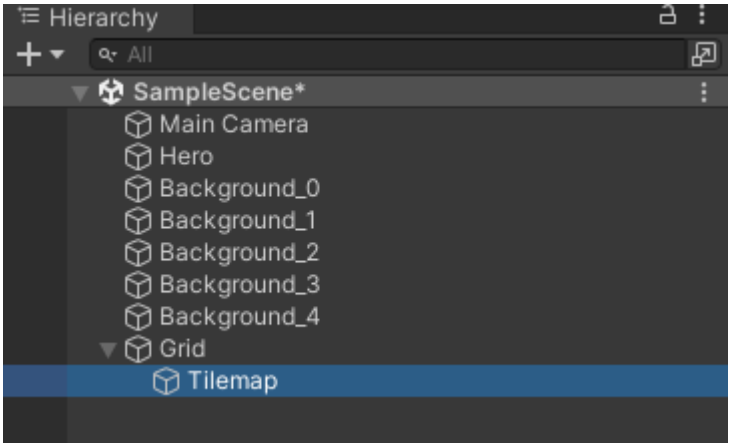


Рис. 3.9 Вікно ієрархії з новоствореним об’єктом Tilemap

Даний об’єкт додасть у вікно сцени сітку для побудови рівню з окремих тайлів. Для початку її заповнення необхідно створити нову палітру тайлів, зробити це можна шляхом додавання в редакторі Unity спеціального додаткового вікна Tile Palette, і створення нової палітри, в результаті чого отримаємо палітру з заданим тайлсетом, в цьому випадку з одним тайлом (Рис. 3.10)

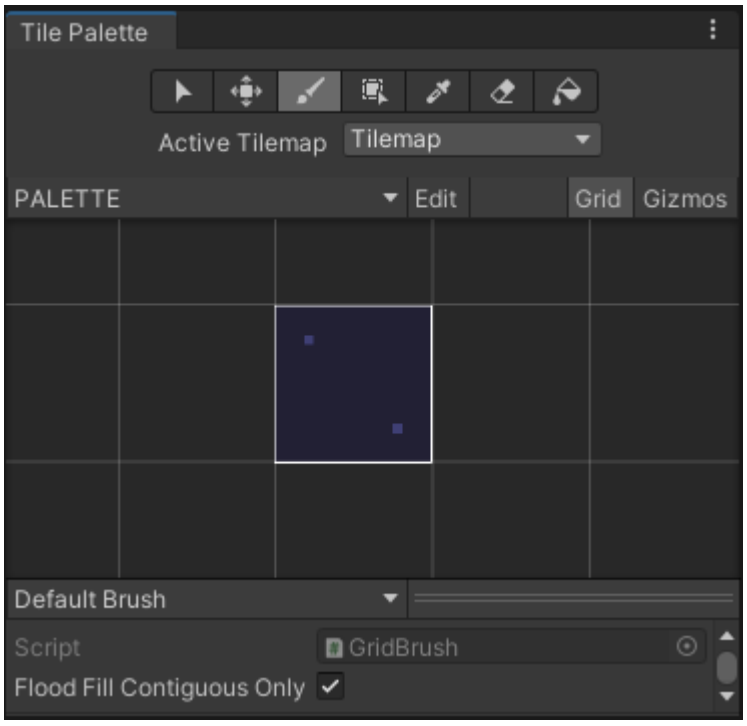


Рис. 3.10 Вікно палітри тайлів з тайлсетом з однієї плитки

В палітру можна додавати безліч варіантів тайлів, їх комбінацій в залежності від розміщення сусідніх тайлів чи особливостей певного рівня, але

іноді рівні складаються з використанням одного тайлу для всієї мапи плиток. Приклад побудови рівня наведено на Рис. 3.11

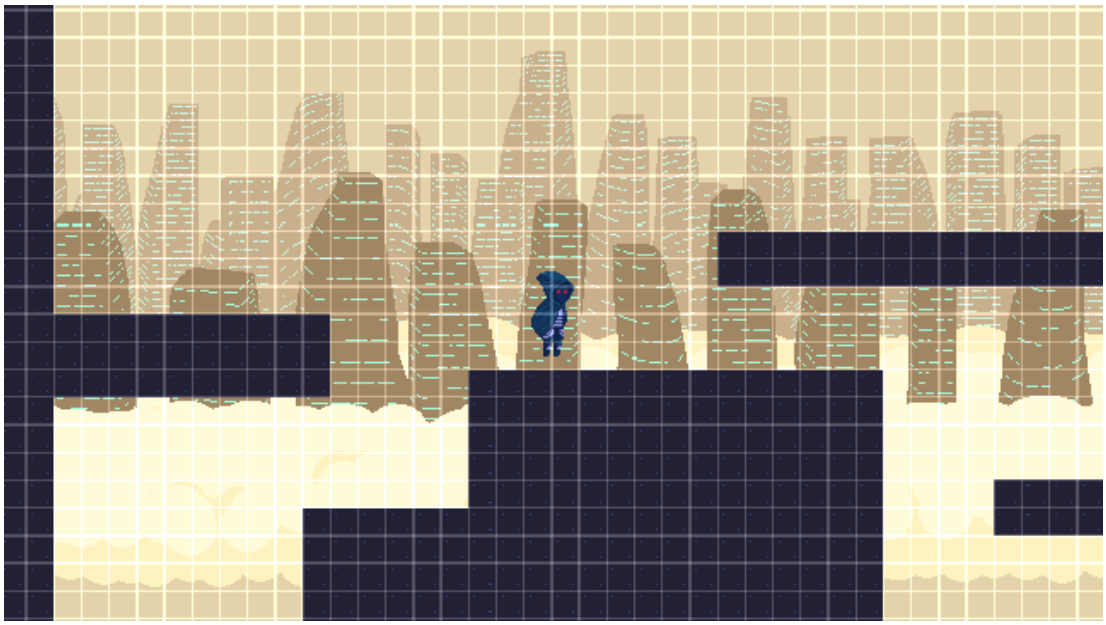


Рис. 3.11 Приклад побудови рівня з використанням мапи плиток

Для ініціалізації ігрових поверхонь рівня, їм потрібно надати колізію, аби інші об’єкти з колізією могли з ними взаємодіяти, в тому числі персонаж. Тому наступний крок це створення компоненту Tilemap Collider 2D, який автоматично застосує колізію до всіх наявних тайлів в сцені. Вікно заданого компоненту і його параметри зображено на Рис. 3.12.

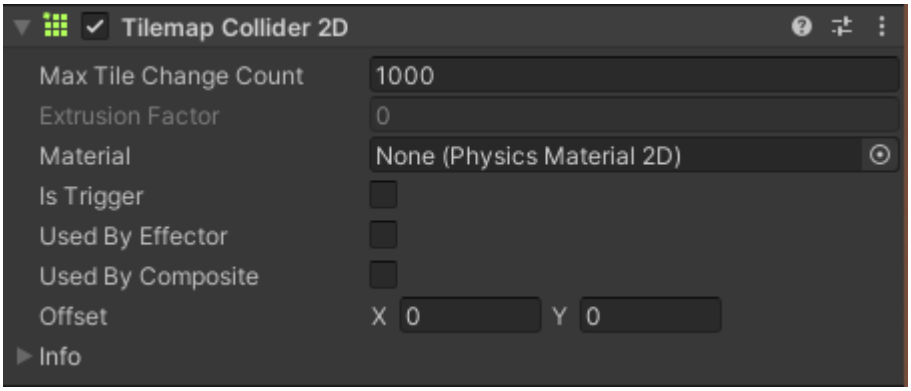


Рис. 3.12 Вікно компоненту колізії рівню

Отже, було створено приклад побудови рівню з використанням технології тайлів.

3.4. Рух персонажу

Для додавання руху персонажу необхідно насамперед розробити скрипт, який буде додавати необхідні параметри та з їх допомогою контролювати рух персонажу і його взаємодію з іншими об’єктами.

Скрипт Character Controller 2D – передбачає створення низки функцій для переміщення персонажу, серед яких:

- плавний рух
- стрибки
- присідання та скрадання
- події для налаштування анімацій
- 2D фізика

Згенероване редактором Unity вікно з параметрами скрипту зображене на Рис. 3.13.

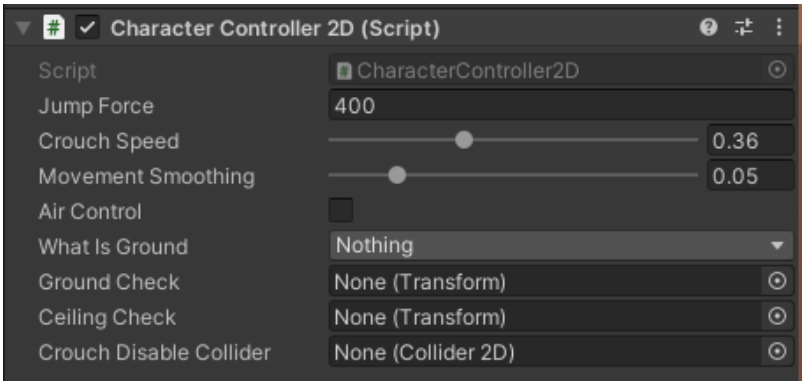


Рис. 3.13 Вікно скрипту для контролю персонажу

З допомогою параметрів у вікні скрипту можна регулювати наступні аспекти переміщення:

- Jump Force – сила стрибка;
- Crouch Speed – швидкість скрадання;
- Movement Smoothing – плавність переміщення;
- Air Control – увімкнути або вимкнути контроль персонажа в повітрі під час стрибка;
- What Is Ground – обрати варіанти поверхонь, якими може переміщуватись персонаж;
- Ground Check та Ceiling Check – призначені для розпізнавання моментів, коли персонаж не може припинити скрадання;
- Crouch Disable Collider – налаштування колізії під час скрадання.

Надалі слід додати персонажу фізичні межі, оскільки зараз у нього вбракує колізії і він може проходити крізь об’єкти. Для цього необхідно додати компонент Box Collider 2D (Рис. 3.14) і підлаштувати утворений об’єкт під межі нашого персонажу таким чином, яким він має межувати з колізіями інших об’єктів у грі (Рис. 3.15).

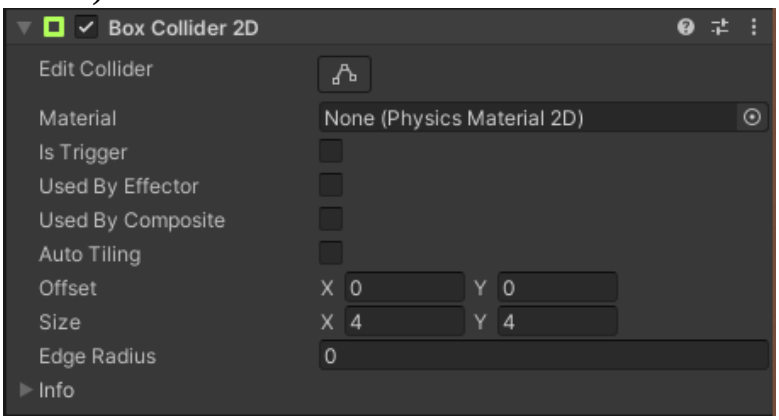


Рис. 3.14 Вікно компонента колізії

Рис. 3.15 Межі колізії персонажу

Також необхідно додати компонент Rigidbody 2D для налаштування фізики персонажу, зокрема гравітації, вікно компоненту зображено на Рис. 3.16.

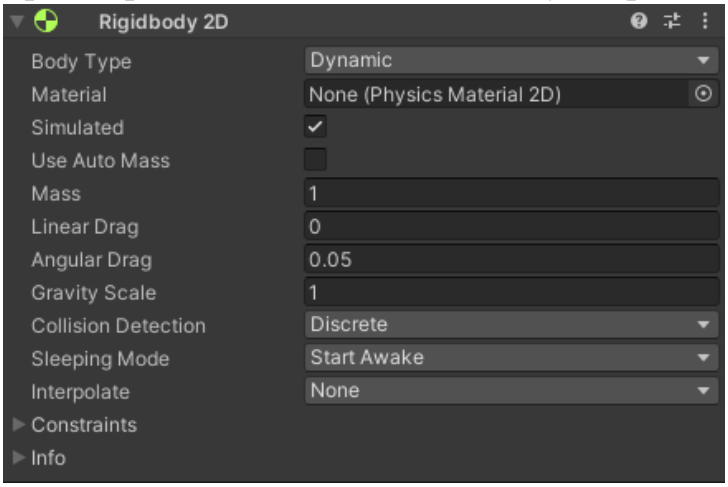


Рис. 3.16 Вікно компоненту Rigidbody 2D

Тепер можна додати новий скрипт, який буде відповідати за сам рух персонажу, використовуючи параметри зі скрипту Character Controller 2D та контролювати процес руху, вікно скрипту всередині редактора, після написання коду зображено на Рис. 3.17.

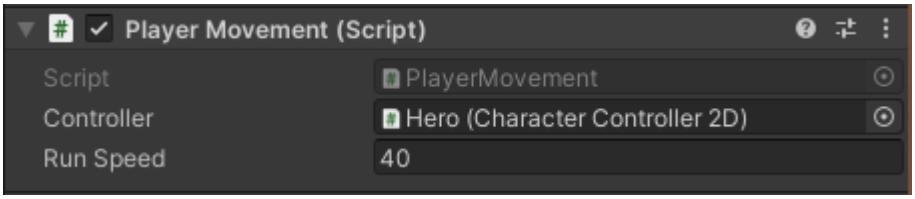


Рис. 3.17 Вікно скрипту для руху з ключовими параметрами

Після всіх виконаних дій з налаштування об’єктів та компонентів і програмування скриптів, головний персонаж може рухатись, але поки без будь-яких анімацій.

3.5. Додавання анімацій

Для додавання анімацій для ігрових об’єктів насамперед необхідно розробити самі анімації у вигляді спрайтів з набору кадрів і додати їх до проєкту. Приклади спрайтів анімацій для головного персонажа, розроблених в редакторі Aseprite зображені на Рис. 3.18 та Рис. 3.19.

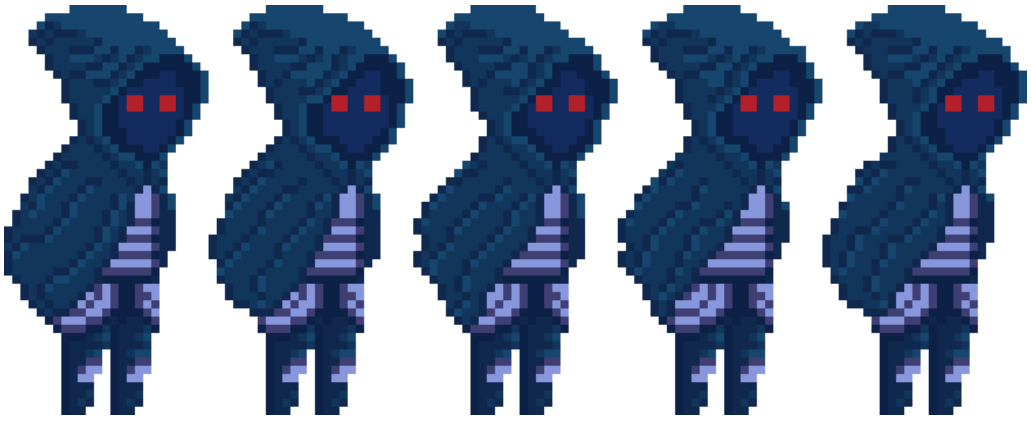


Рис. 3.18 Спрайт анімації очікування головного персонажу

Рис. 3.19 Спрайт анімації бігу головного персонажу

Після цього необхідно створити файли анімацій в кореневій папці проєкту (Рис. 3.20) і з допомогою редактору налаштувати анімацію відповідних кадрів.

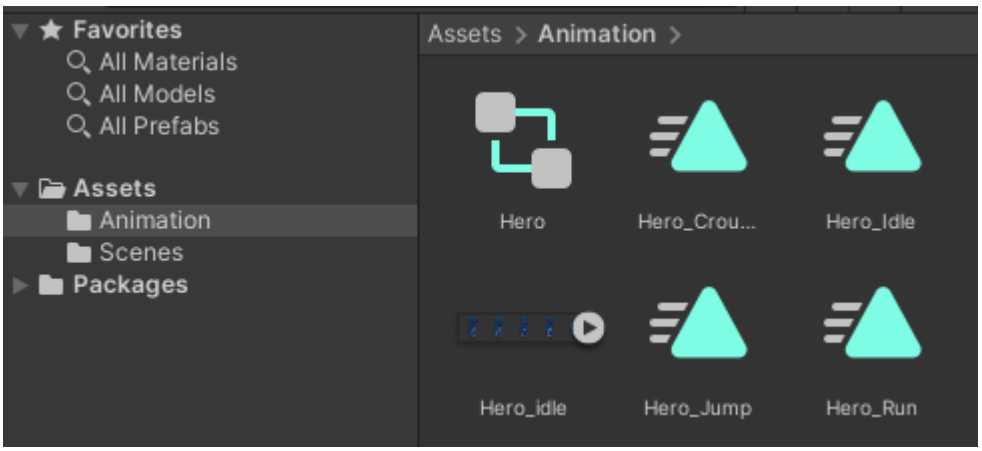


Рис. 3.20 Вікно проєкту з файлами анімацій та контролером анімацій

Коли будуть налаштовані всі анімації, створюється контролер анімацій, який з допомогою візуального програмування налаштовує логіку програвання анімацій. Отже персонаж буде у відповідні моменти програвати потрібні анімації, з відповідною швидкістю, напрямком та поправками на інші параметри. Логіка анімацій головного героя зображена на Рис. 3.21.

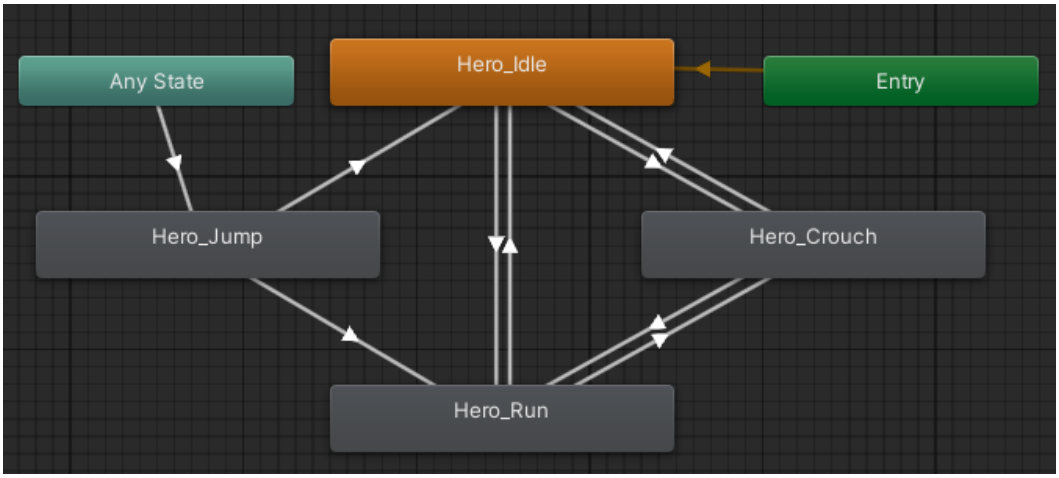


Рис. 3.21 Схема логіки анімацій головного персонажа

Логіка анімацій демонструє, що персонаж за замовчуванням програє анімацію очікування, може програвати анімацію стрибку з будь-якого положення, після чого вона змінюється на анімацію очікування або бігу, а анімації скрадання, бігу та очікування можуть змінюватись одна на одну. Більш детальні умови для кожної анімації програмуються всередині скрипту Character Controller 2D та з допомогою редактора Unity у відповідному вікні параметрів.

3.6. Додавання ближнього бою

Для створення бойової системи ближнього бою необхідно розробити скрипт Player Combat, який буде виконувати кілька основних функцій:

- програвати анімацію атаки
- шукати ворогів в зоні ураження
- фіксувати шкоду ворогам

Також необхідно розробити скрипт Enemy, який буде контролювати відповідні дії ворогів, а саме:

- починати фіксувати шкоду
- програвати анімацію отримання шкоди
- програвати анімацію знищення
- вимикати ворога після знищення

Розроблений в Aseprite асет ворога зображений на Рис. 3.22.

Рис. 3.22 Асет ворога Potbot

Дизайн ворога Potbot передбачає зображення базового слабкого ворога – робота, який нагадує вазон і назву має відповідну.

Логіка анімацій головного персонажа не сильно змінюється від появи нової анімації (Рис. 3.23).

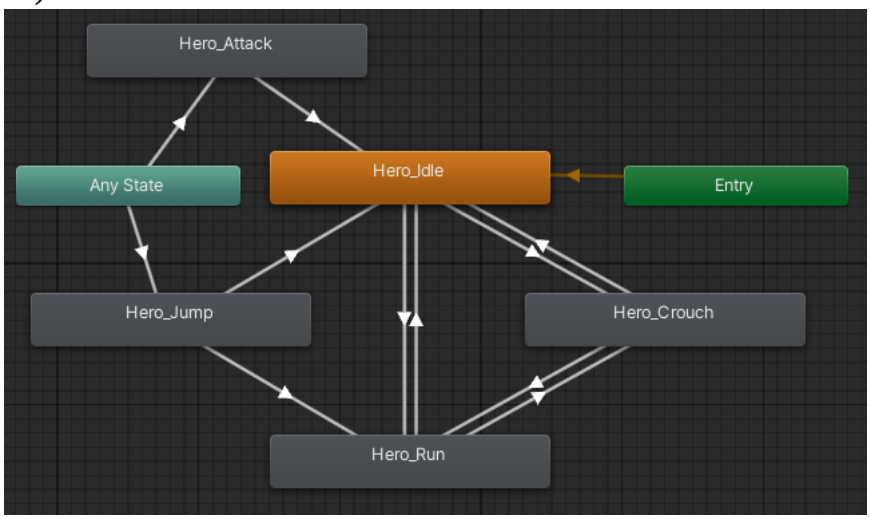


Рис. 3.23 Схема логіки анімації головного персонажа з атакою

Логіка анімацій ворога буде виглядати як вказано на Рис. 3.24

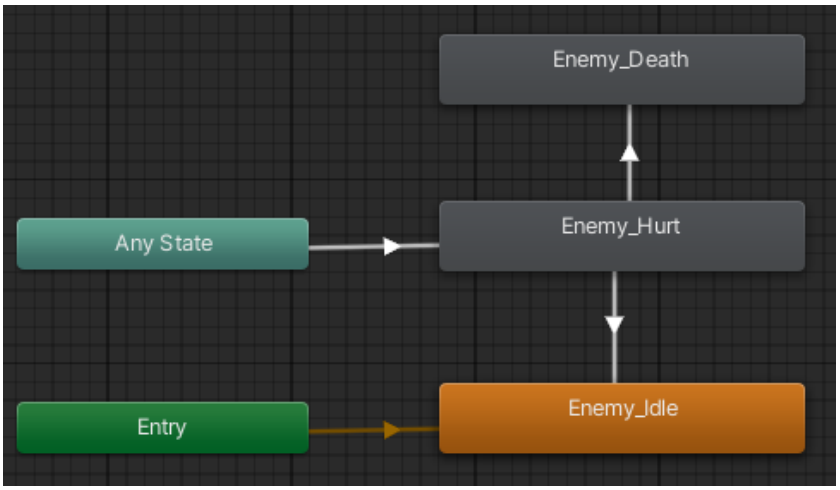


Рис. 3.24 Схема логіки анімацій ворога

Таким чином була реалізована система ближнього бою головного персонажа проти ворогів, які в свою чергу отримують шкоду і знищуються. На Рис. 3.25 зображений спрайт анімації атаки головного персонажу.

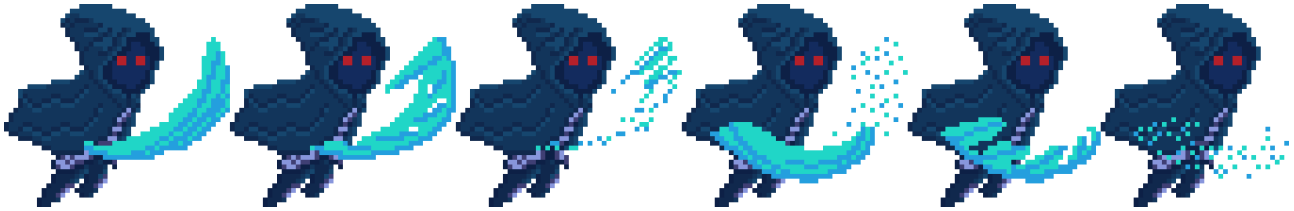


Рис. 3.25 Спрайт анімації атаки головного персонажу

Спрайти анімацій отримання шкоди і знищення ворога зображені на Рис. 3.26 та 3.27 відповідно.

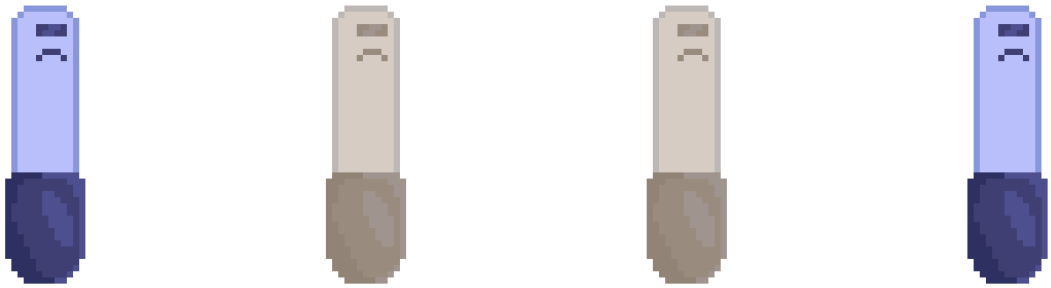


Рис. 3.26 Спрайт анімації отримання шкоди ворога

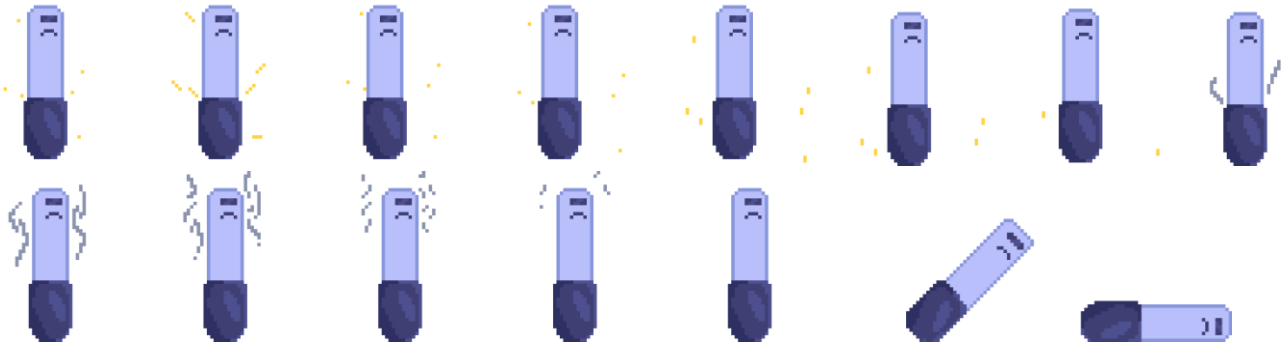


Рис. 3.26 Спрайт анімації знищення ворога

ВИСНОВКИ

В даній дипломній роботі досліджено особливості створення відеоігор, їх вплив на суспільство та викликаний ними технологічний розвиток периферійних пристроїв і комп'ютерних систем. Вплив ігор на людство не слід недооцінювати, оскільки чим більше людей стають споживачами, тим сильніше це впливає на індустрію, і тим більше з'являється можливостей впливати на суспільство через цей продукт.

В процесі роботи з'ясовано, що індустрія розробки відеоігор має безліч складових, залучає дуже багато людей та професій, це виправдовується великим об'ємом і різним характером роботи. Для розробки гри, в ході дипломної роботи, довелось:

- працювати над концептом майбутнього продукту;
- складати поетапний план проєкта;
- малювати комп'ютерну графіку;
- створювати анімації;
- писати програмний код;
- досліджувати нові галузі та методи розробки;
- використовувати різні програми.

У підсумку була розроблена демонстраційна версія комп'ютерної гри жанру платформер на базі ігрового рушія Unity, із застосуванням стилізованої двовимірної графіки стилю Pixel art.

В процесі розробки гри можна зрозуміти, наскільки це складний і багатогранний процес, скільки різноманітних технологій і здобутків використовується для виготовлення сучасної комп'ютерної гри. Розробка об'єднує в собі різні напрямки роботи і сфери досліджень.

Відеоігри створюють роботу, можливості, технології, вплив, емоції, мрії, уявлення, розуміння, культуру, історію, створюють нас. Тому треба ретельно слідкувати за процесом їх створення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Harrison Ferrone (2021). Learning C# by Developing Games with Unity 2020 (5th edition).

2. Джейсон Шраєр. Кров, піт і пікселі. По той бік створення відеоігор. Book Chef, 2020. 336 с.

3. Paris Buttfield-Addison, Jonathon Manning, Tim Nugent. (2019) Unity Game Development Cookbook: Essentials for Every Game (1st edition).

4. Jeffrey Richter (2012) CLR via C# (Developer Reference) (4th edition).

5. Jesse Shell (2018) The Art of Game Design: A Book of Lenses (3rd edition).

6. Добірка рушіїв для розробки ігор [Електронний ресурс]/DOU: 2021 – Посилання на ресурс: https://gamedev.dou.ua/blogs/game-engines-collection/?from=similar_posts

7. Хто такий геймдизайнер в ігровій індустрії [Електронний ресурс]/DOU: 2022 – Посилання на ресурс: https://gamedev.dou.ua/articles/game-designer-in-gamedev/

8. Хто такий артдиректор в ігровій індустрії [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/art-director-in-gamedev/

9. Хто такий наративний дизайнер в ігровій індустрії [Електронний ресурс]/DOU: 2022 – Посилання на ресурс: https://gamedev.dou.ua/articles/narrative-designer-in-gamedev/

10. Хто такий UI/UX Designer в ігровій індустрії [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/ui-ux-designer-in-gamedev/

11. Хто такий Technical Artist в ігровій індустрії [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/technical-artist-developer-in-gamedev/

12. Як Naughty Dog розповідають сюжет гри через оточення [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/environmental-storytelling-in-tlou/

13. Огляд Valiant Hearts: Coming Home [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/valiant-hearts-coming-home-review/

14. Як працювати з ігровими механіками й чому не всі з них вдалі: поради геймдизайнерів [Електронний ресурс]/DOU: 2023 – Посилання на ресурс: https://gamedev.dou.ua/articles/how-to-work-with-game-mechanics/

15. Game Engines, Frameworks for building games across multiple platforms. [Електронний ресурс]/GitHub: 2023 – Посилання на ресурс: https://github.com/collections/game-engines

16. Games Market Reports and Forecasts [Електронний ресурс]/Newzoo: 2023 – Посилання на ресурс: https://newzoo.com/games-market-reports-forecasts

17. Global Games Market Report [Електронний ресурс]/Newzoo: 2023 – Посилання на ресурс: https://resources.newzoo.com/hubfs/Factsheets/Newzoo_Games_Market_Reports_Forecasts_Factsheet.pdf?utm_referrer=https%3A%2F%2Fnewzoo.com%2F

18. Blender About [Електронний ресурс] – Посилання на ресурс: https://www.blender.org/about/

19. Blender Features [Електронний ресурс]/Blender: 2023 – Посилання на ресурс: https://www.blender.org/features/

20. Blender Manual [Електронний ресурс]/Blender: 2023 – Посилання на ресурс: https://docs.blender.org/manual/en/latest/

21. Unreal Engine 5 [Електронний ресурс]/Epic games: 2023 – Посилання на ресурс: https://www.unrealengine.com/en-US/unreal-engine-5

22. MetaHuman [Електронний ресурс]/Epic games: 2023 – Посилання на ресурс: https://www.unrealengine.com/en-US/metahuman

23. Bridge by Quixel [Електронний ресурс]/Epic games: 2023 – Посилання на ресурс: https://www.unrealengine.com/en-US/bridge

24. Unity Documentation [Електронний ресурс]/Unity: 2023 – Посилання на ресурс: https://docs.unity3d.com/2021.3/Documentation/Manual/index.html

25. Unity Products [Електронний ресурс]/Unity: 2023 – Посилання на

ресурс: https://unity.com/products

26. Unity Learn [Електронний ресурс]/Unity: 2023 – Посилання на ресурс: https://learn.unity.com/

27. Pixel Art Questions & Answers [Електронний ресурс]/Reddit: 2021 – Посилання на ресурс: https://www.reddit.com/r/PixelArt/comments/hsbdfo/pixel_art_vs_digital_art_which_would_you_rather/

28. Pixel art – od czego zacząć i jak tworzyć? Poradnik! [Електронний ресурс]/IndieGamer: 2022 – Посилання на ресурс: https://indie-gamer.pl/pixel-art-od-czego-zaczac-i-jak-tworzyc-poradnik/

29. Aseprite Documentation [Електронний ресурс]/Igara Studio: 2023 – Посилання на ресурс: https://www.aseprite.org/docs/

30. Aseprite [Електронний ресурс]/GitHub: 2023 – Посилання на ресурс: https://github.com/aseprite/aseprite

ДОДАТОК А

Програмні скрипти

CharacterController2D.cs

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

public class CharacterController2D : MonoBehaviour
{
	[SerializeField] private float m_JumpForce = 400f;							// Amount of force added when the player jumps.
	[Range(0, 1)] [SerializeField] private float m_CrouchSpeed = .36f;			// Amount of maxSpeed applied to crouching movement. 1 = 100%
	[Range(0, .3f)] [SerializeField] private float m_MovementSmoothing = .05f;	// How much to smooth out the movement
	[SerializeField] private bool m_AirControl = false;							// Whether or not a player can steer while jumping;
	[SerializeField] private LayerMask m_WhatIsGround;							// A mask determining what is ground to the character
	[SerializeField] private Transform m_GroundCheck;							// A position marking where to check if the player is grounded.
	[SerializeField] private Transform m_CeilingCheck;							// A position marking where to check for ceilings
	[SerializeField] private Collider2D m_CrouchDisableCollider;				// A collider that will be disabled when crouching

	const float k_GroundedRadius = .2f; // Radius of the overlap circle to determine if grounded
	private bool m_Grounded;            // Whether or not the player is grounded.
	const float k_CeilingRadius = .2f; // Radius of the overlap circle to determine if the player can stand up
	private Rigidbody2D m_Rigidbody2D;
	private bool m_FacingRight = true;  // For determining which way the player is currently facing.
	private Vector3 m_Velocity = Vector3.zero;

	[Header("Events")]
	[Space]

	public UnityEvent OnLandEvent;

	[System.Serializable]
	public class BoolEvent : UnityEvent<bool> { }

	public BoolEvent OnCrouchEvent;
	private bool m_wasCrouching = false;

	private void Awake()
	{
		m_Rigidbody2D = GetComponent<Rigidbody2D>();

		if (OnLandEvent == null)
			OnLandEvent = new UnityEvent();

		if (OnCrouchEvent == null)
			OnCrouchEvent = new BoolEvent();
	}

	private void FixedUpdate()
	{
```

```
            bool wasGrounded = m_Grounded;
            m_Grounded = false;

            // The player is grounded if a circlecast to the groundcheck position hits
anything designated as ground
            // This can be done using layers instead but Sample Assets will not overwrite
your project settings.
            Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck.position,
k_GroundedRadius, m_WhatIsGround);
            for (int i = 0; i < colliders.Length; i++)
            {
                  if (colliders[i].gameObject != gameObject)
                  {
                        m_Grounded = true;
                        if (!wasGrounded)
                              OnLandEvent.Invoke();
                  }
            }
      }

      public void Move(float move, bool crouch, bool jump)
      {
            // If crouching, check to see if the character can stand up
            if (crouch)
            {
                  // If the character has a ceiling preventing them from standing up,
keep them crouching
                  if (Physics2D.OverlapCircle(m_CeilingCheck.position, k_CeilingRadius,
m_WhatIsGround))
                  {
                        crouch = true;
                  }
            }

            //only control the player if grounded or airControl is turned on
            if (m_Grounded || m_AirControl)
            {

                  // If crouching
                  if (crouch)
                  {
                        if (!m_wasCrouching)
                        {
                              m_wasCrouching = true;
                              OnCrouchEvent.Invoke(true);
                        }

                        // Reduce the speed by the crouchSpeed multiplier
                        move *= m_CrouchSpeed;

                        // Disable one of the colliders when crouching
                        if (m_CrouchDisableCollider != null)
                              m_CrouchDisableCollider.enabled = false;
                  }
                  else
                  {
                        // Enable the collider when not crouching
                        if (m_CrouchDisableCollider != null)
                              m_CrouchDisableCollider.enabled = true;

                        if (m_wasCrouching)
                        {
                              m_wasCrouching = false;
                              OnCrouchEvent.Invoke(false);
                        }
```

```
                    }

                    // Move the character by finding the target velocity
                    Vector3 targetVelocity = new Vector2(move * 10f,
m_Rigidbody2D.velocity.y);
                    // And then smoothing it out and applying it to the character
                    m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity,
targetVelocity, ref m_Velocity, m_MovementSmoothing);

                    // If the input is moving the player right and the player is facing
left...
                    if (move > 0 && !m_FacingRight)
                    {
                           // ... flip the player.
                           Flip();
                    }
                    // Otherwise if the input is moving the player left and the player is
facing right...
                    else if (move < 0 && m_FacingRight)
                    {
                           // ... flip the player.
                           Flip();
                    }
             }
             // If the player should jump...
             if (m_Grounded && jump)
             {
                    // Add a vertical force to the player.
                    m_Grounded = false;
                    m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
             }
      }

      private void Flip()
      {
             // Switch the way the player is labelled as facing.
             m_FacingRight = !m_FacingRight;

             // Multiply the player's x local scale by -1.
             Vector3 theScale = transform.localScale;
             theScale.x *= -1;
             transform.localScale = theScale;
      }
      }
```

PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour
{
    public CharacterController2D controller;
    public Animator animator;

    public float runSpeed = 40f;

    float horizontalMove = 0f;
    bool jump = false;
    bool crouch = false;
```

```
    void Update()
    {
        horizontalMove = Input.GetAxisRaw("Horizontal") * runSpeed;

        animator.SetFloat("Speed", Mathf.Abs(horizontalMove));

        if (Input.GetButtonDown("Jump"))
        {
            jump = true;
            animator.SetBool("IsJumping", true);
        }

        if (Input.GetButtonDown("Crouch"))
        {
            crouch = true;
        }
        else if (Input.GetButtonUp("Crouch"))
        {
            crouch = false;
        }
    }

    public void OnLanding ()
    {
        animator.SetBool("IsJumping", false);
    }

    public void OnCrouching(bool isCrouching)
    {
        animator.SetBool("IsCrouching", isCrouching);
    }

    void FixedUpdate()
    {
        // Move our character
        controller.Move(horizontalMove * Time.fixedDeltaTime, crouch, jump);
        jump = false;


    }
}
```

PlayerCombat.cs

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

public class PlayerCombat : MonoBehaviour
{
    public Animator animator;

    public Transform attackPoint;
    public LayerMask enemyLayers;

    public float attackRange = 0.5f;
    public int attackDamage = 40;

    // Update is called once per frame
    void Update()
    {
         if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            Attack();
        }
    }
```

```
    void Attack()
    {
        // play an attack animation
        animator.SetTrigger("Attack");

        // detect enemies in range of attack
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position,
attackRange, enemyLayers);

        // damage them
        foreach(Collider2D enemy in hitEnemies)
        {
            enemy.GetComponent<Enemy>().TakeDamage(attackDamage);
        }
    }

    void OnDrawGizmosSelected()
    {
        if (attackPoint == null)
            return;

            Gizmos.DrawWireSphere(attackPoint.position, attackRange);

    }
}
```

Enemy.cs

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

public class Enemy : MonoBehaviour
{
    public Animator animator;

    public int maxHealth = 100;
    int currentHealth;

    // Start is called before the first frame update
    void Start()
    {
        currentHealth = maxHealth;
    }

    public void TakeDamage(int damage)
    {
        currentHealth -= damage;

        //play hurt animation
        animator.SetTrigger("Hurt");

        if (currentHealth <= 0)
        {
            GetComponent<Rigidbody2D>().gravityScale = 0;
            Die();
        }
    }
    // Update is called once per frame
    void Die()
    {
        Debug.Log("Enemy died!");

        // die animation
        animator.SetBool("IsDead", true);
```

```
        //disable the enemy
        GetComponent<Collider2D>().enabled = false;
        this.enabled = false;

    }

}
```

DeadZone.cs

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

public class DeadZone : MonoBehaviour
{
    private UIManager uiManager;

    private void Awake()
    {
        uiManager = FindObjectOfType<UIManager>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            Debug.Log("fall");
            Fall();
        }
    }

    void Fall()
    {
        uiManager.GameOver();
    }
}
```

Parallax.cs

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

public class Parallax : MonoBehaviour
{
    public Camera cam;
    public Transform subject;

    Vector2 startPosision;

    float startZ;

    Vector2 travel => (Vector2)cam.transform.position - startPosision;

    float distanceFromSubject => transform.position.z - subject.position.z;

    float clippingPlane => (cam.transform.position.z + (distanceFromSubject > 0 ?
cam.farClipPlane : cam.nearClipPlane));

    float parallaxFactor => Mathf.Abs(distanceFromSubject) / clippingPlane;

    public void Start()
    {
        startPosision = transform.position;
        startZ = transform.position.z;
```

```
    }

    public void Update()
    {
        Vector2 newPos = startPosision + travel * parallaxFactor;
        transform.position = new Vector3(newPos.x, newPos.y, startZ);
    }

}
```

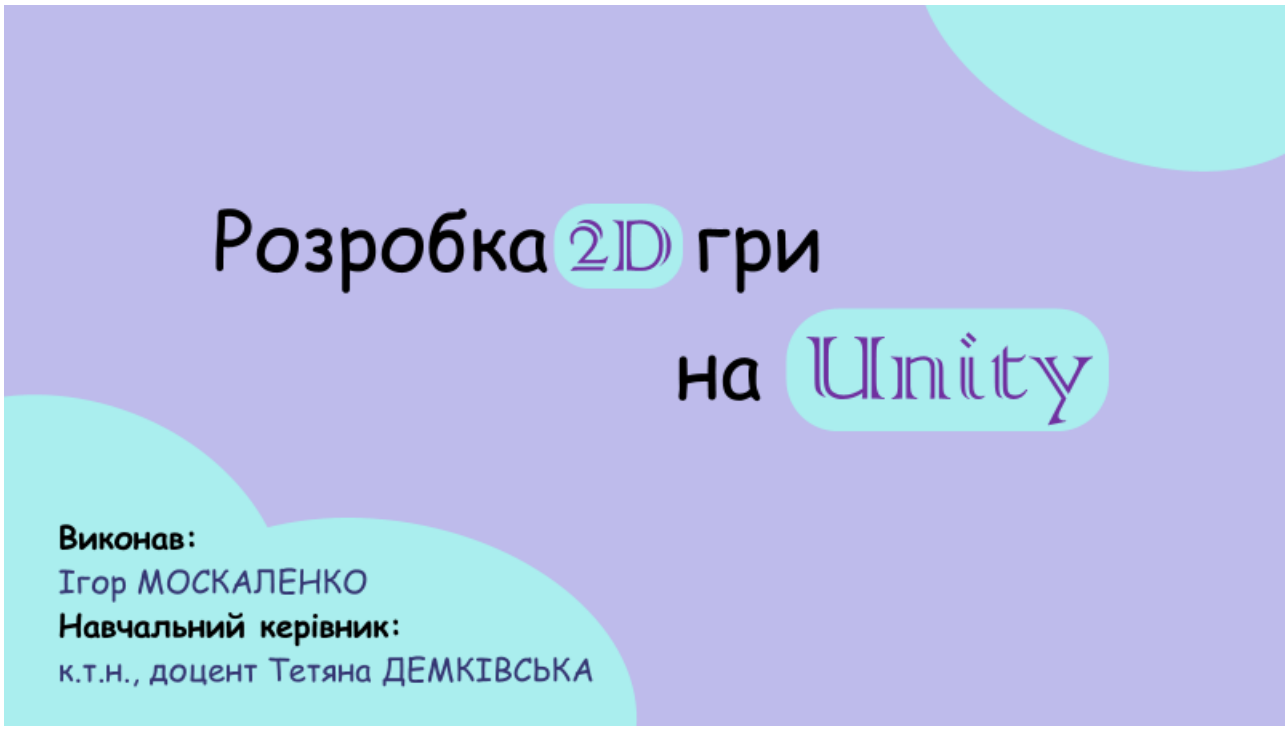
UIManager.cs

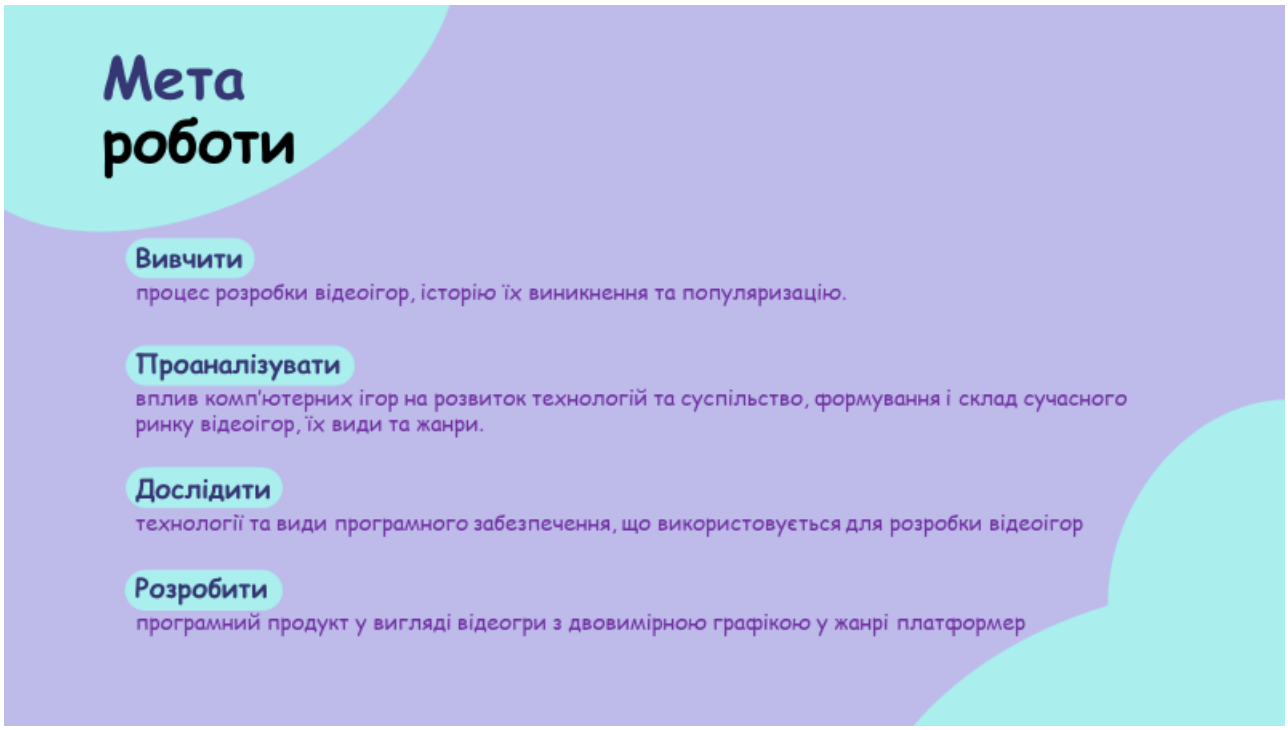
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class UIManager : MonoBehaviour
{
    [SerializeField] private GameObject gameOverScreen;
    [SerializeField] private GameObject pauseScreen;
    private void Awake()
    {
        gameOverScreen.SetActive(false);
        pauseScreen.SetActive(false);
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            //If pause screen already active unpause and viceversa
            PauseGame(!pauseScreen.activeInHierarchy);
        }
    }

    public void GameOver()
    {
        gameOverScreen.SetActive(true);
        Time.timeScale = 0;
    }

    public void PlayGame()
    {
        SceneManager.LoadScene(1);

        if (Time.timeScale == 0)
        {
            Time.timeScale = 1;
        }
    }

    public void MainMenu()
    {
        SceneManager.LoadScene(0);
    }

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

    public void PauseGame(bool status)
    {
        //If status == true pause | if status == false unpause
        pauseScreen.SetActive(status);
```

```
        //When pause status is true change timescale to 0 (time stops)
        //when it's false change it back to 1 (time goes by normally)
        if (status)
            Time.timeScale = 0;
        else
            Time.timeScale = 1;
    }

}
```

ДОДАТОК Б

Презентація до роботи





Відеоігри

Взаємодія з грою:

Пристрої вводу інформації
Геймпад, джойстик, клавіатура та миша

Обчислювальні пристрої
Комп'ютер або ігрова консоль з вбудованими графічними процесорами

Пристрої виводу інформації
Монітори, навушники, тактильна віддача геймпадів

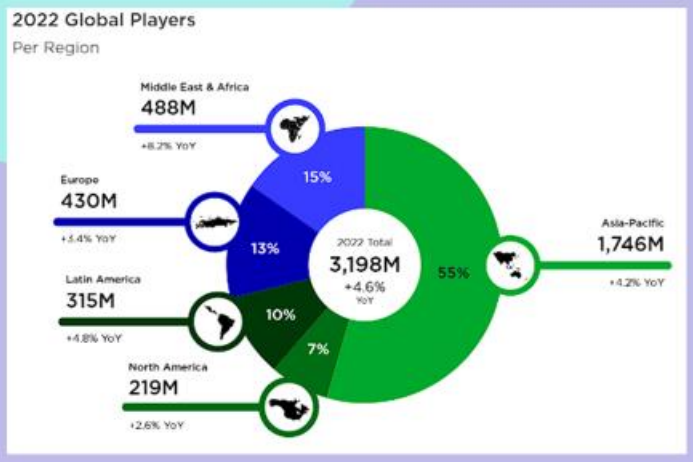
Аркадні автомати

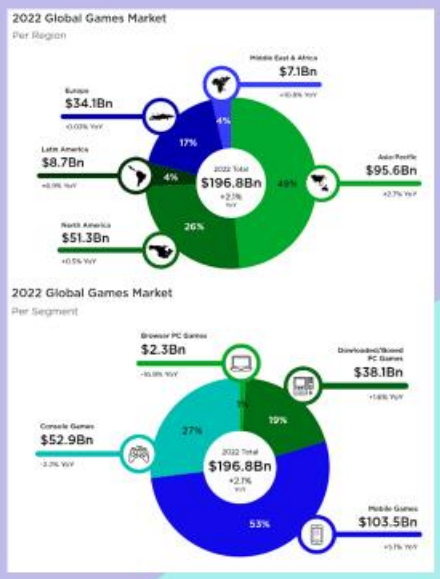
Портативні консолі

Мультиплеєрні ігри

Кінематографічні бестселери

Актуальність





Gamedev

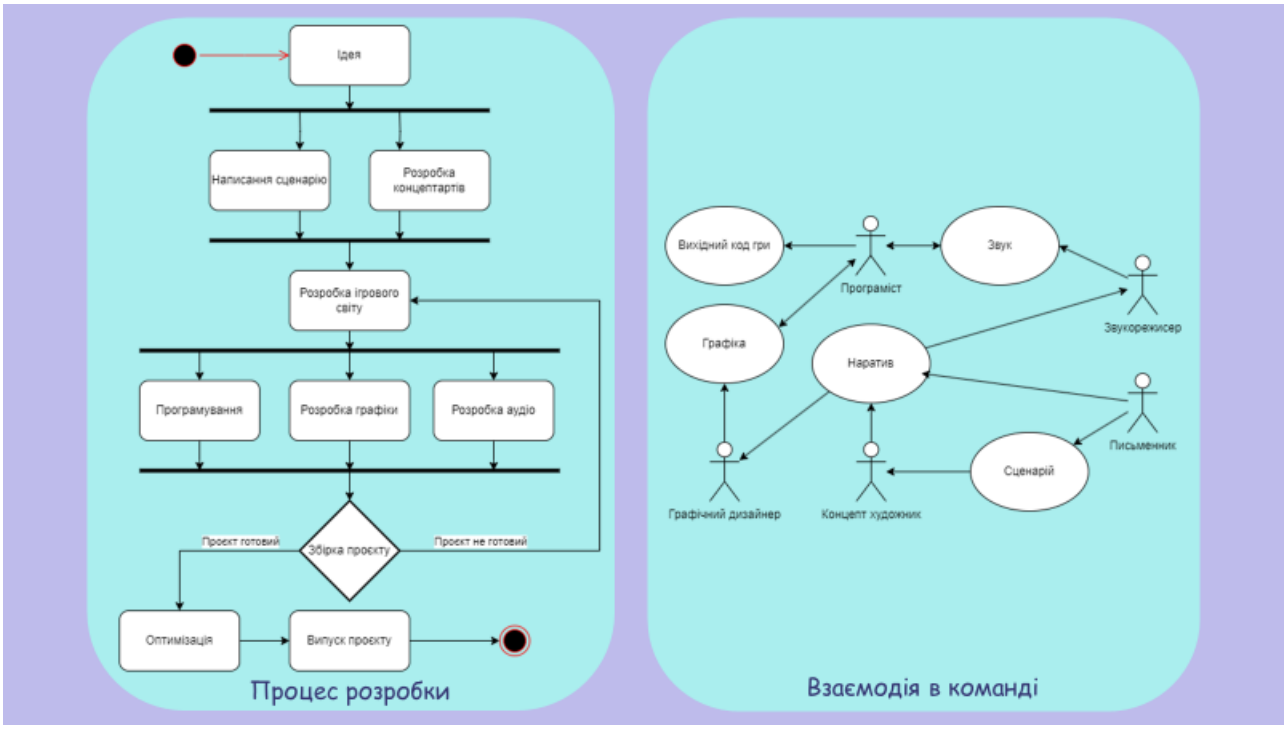
Основні напрями:

Концепція
Ідея, концепт арти, сюжет і наратив

Дизайн
Графіка, анімації, будова рівнів

Код
Скрипти, програмування ефектів, об'єднання всіх складових проєкту, оптимізація

Все інше
Звук, підтримка, зворотній зв'яок, реклама



Проєкт

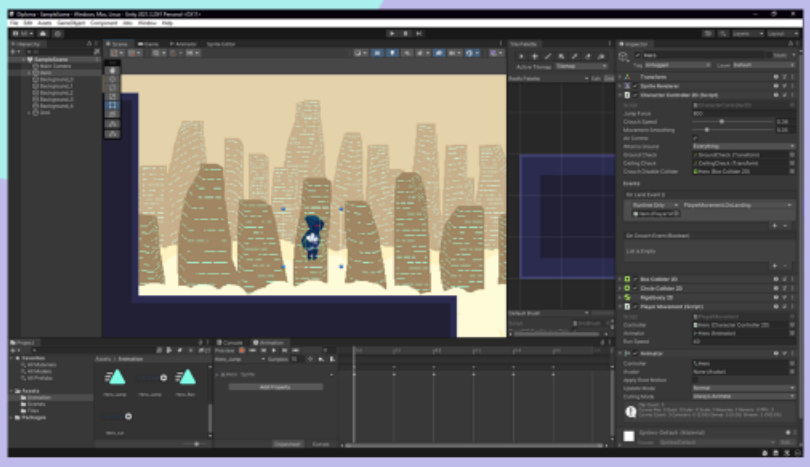
Характеристики:

Unity
Ігральний рушій

Платформер
Ігровий жанр

2D, Pixel Art
Вид графічної складової

Unity



Переваги:

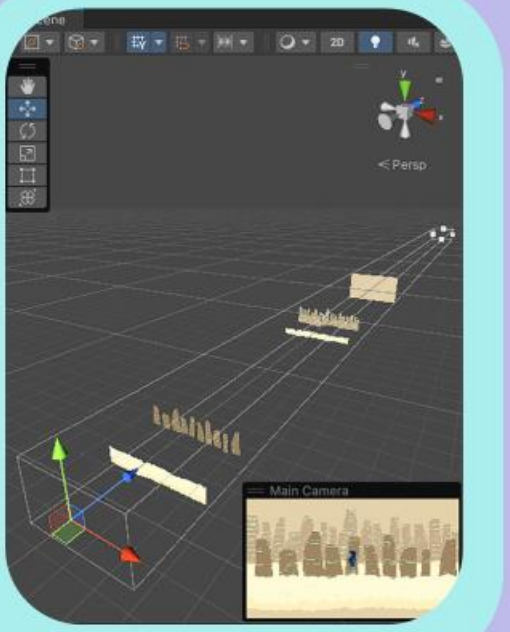
2D і 3D
Підтримка різних типів графіки

C#
Безпечніше оперування ресурсами та кросплатформність

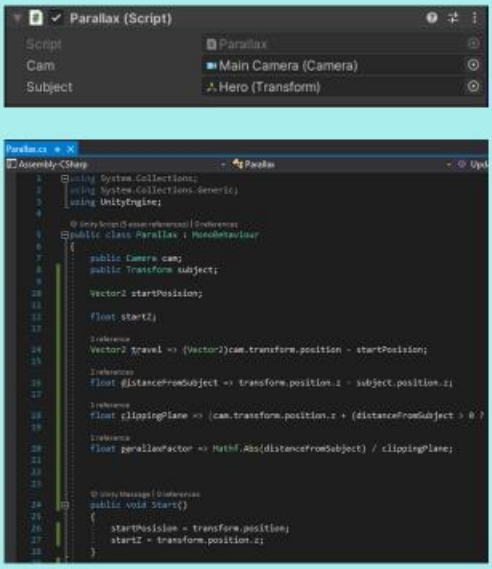
User friendly
Легкий в освоєнні, інтегровані туторіали

Функціональність
Можливість виконувати безліч специфічних задач прямо в редакторі

Мультиплатформність
Сумісний з всіма ОС та може відтворювати додатки на більшості пристроїв



Розміщення елементів



Скрип та параметри в редакторі

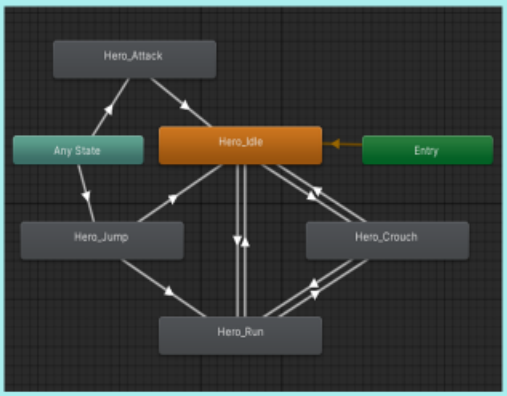
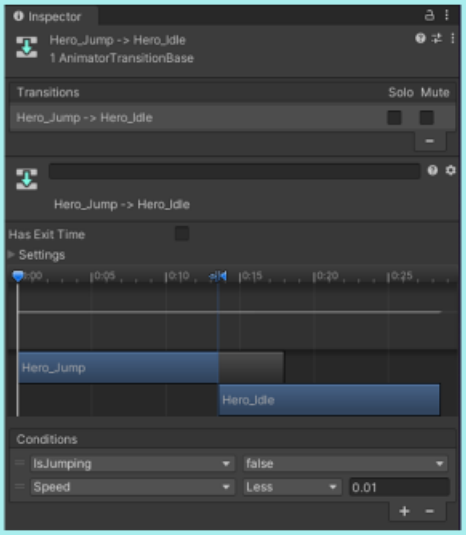
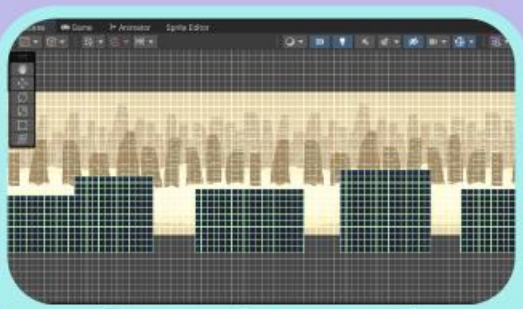


Схема логіки анімацій

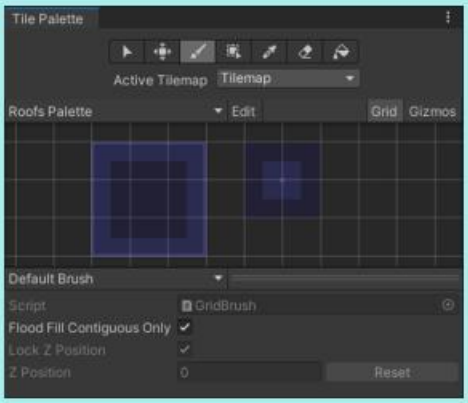


Налаштування окремого з'єднання

Набір плиток (Tile set)

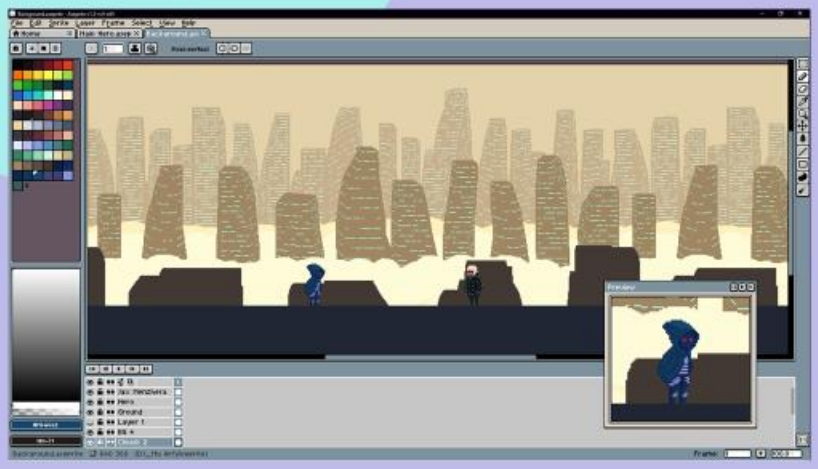


Створення рівня в редакторі



Палітра плиток

Aseprite



Переваги:

Pixel art
Націлений виключно на піксельну графіку

Малювання
Підтримка безлічі налаштувань палітр та функцій малювання пікселями

Анімації
Широкий функціонал для створення анімацій та шарів

Імпорт та експорт
Зручні функції та спеціальні налаштування імпорту/експорту файлів

Підтримка
Часті оновлення та зворотній зв'язок

Розробка персонажу та анімацій

Готові спрайти анімації

Дизайн персонажів

Загальний вигляд фону

Спрайти фону для паралаксу

Висновки

В процесі роботи з'ясовано
що індустрія розробки відеоігор має безліч складових, залучає дуже багато людей та професій, це виправдовується великим об'ємом і різним характером роботи.

Для розробки гри, в ході дипломної роботи, довелось:

- працювати над концептом майбутнього продукту;
- складати поетапний план проєкта;
- малювати комп'ютерну графіку;
- створювати анімації;
- писати програмний код;
- досліджувати нові галузі та методи розробки;
- використовувати різні програми.

Висновки

У підсумку

була розроблена демонстраційна версія комп'ютерної гри жанру **платформер** на базі ігрового рушія **Unity**, із застосуванням стилізованої двовимірної графіки стилю **Pixel art**