МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання
та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: **«РОЗРОБКА ІГРОВОГО ЗАСТОСУНКУ В ЖАНРІ ACTION-ПЛАТФОРМЕР ПІД МОБІЛЬНУ СИСТЕМУ ANDROID НА РУШІЮ UNITY: ІГРОВА ЛОГІКА»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Котула Владислав Ігорович

Керівник:
Гаврильчик Леонід Сергійович

Рецензент:
Місай Володимир Віталійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № ____ від « ___ » _____ 2022 р.

Острог, 2022

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки

Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА

«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу/проєкт студента

Котули Владислава Ігоровича

1. Тема роботи: Розробка ігрового застосунку в жанрі action-платформер під мобільну систему Android на рушію Unity: ігрова логіка

керівник роботи/проєкту: Гаврильчик Леонід Сергійович.

Затверджено наказом ректора НаУОА від 29 жовтня 2021 року №110

2. Термін здачі студентом закінченої роботи/проєкту: 03 червня 2022 року

3. Вихідні дані до роботи/проєкту: постановка задачі, аналіз аналогів, вихідні дані.

4. Перелік завдань, які належить виконати: опис предметного середовища; огляд наявних аналогів; постановку задачі; опис архітектури рішення, що розроблюється; опис коду та інтерфейсу програми; тестування.

5. Перелік графічного матеріалу: діаграма прецедентів функціональних вимог, діаграма станів ігрового циклу, діаграма станів системи меню, діаграма станів додаткової сцени – пауза.

6. Консультанти розділів роботи/проєкту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.
2	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.
3	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.

7. Дата видачі завдання: 01.12.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи/проєкту	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	до 01.11.2021 р.	
2	Постановка завдання	до 01.12. 2021 р.	
3	Розробка архітектури та загальної структури системи	до 01.02.2022 р.	
4	Розробка структур окремих підсистем	до 01.03. 2022 р.	
5	Програмна реалізація системи	до 01.05.2022 р.	
6	Попередній захист кваліфікаційної роботи/проєкту	до 01.06.2022р.	
7	Здача кваліфікаційної роботи/проєкту на кафедру	03.06.2022 р.	

Студент: _____ Владислав КОТУЛА

Керівник кваліфікаційної роботи/проєкту: _____ Леонід ГАВРИЛЬЧИК

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *Розробка ігрового застосунку в жанрі action-платформер під мобільну систему Android на рушію Unity: ігрова логіка*

Автор: *Котула Владислав Ігорович*

Науковий керівник: *Гаврильчик Леонід Сергійович*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *62 с., 60 рис., 12 джерел.*

Ключові слова: *ігровий застосунок, Unity, 2D*

Короткий зміст праці:

У кваліфікаційній роботі/проєкті метою було створення 2D ігрового додатку в жанрі action-платформер під мобільну систему Android на рушію Unity. Для досягнення поставленої мети я проаналізував жанрову класифікацію, представників даного жанру. Різні середовища розробки та двигуни, на яких можна виконати це завдання. У результаті роботи були описані компоненти, методи, використані реалізації проекту. Програмним середовищем написання коду є Visual Studio 2022, мовою програмування – C#.

In the bachelor's thesis the task was to create a 2D game application in the genre of action platformer for the Android mobile system on the Unity engine. To achieve this goal, I analyzed the genre classification of this genre. Different development environments and engines on which you can perform this task. In the course of the work the components and methods used for the project implementation were described. Software environment for writing code - Visual Studio 2022, programming language - C #.

_____

ЗМІСТ

РОЗДІЛ 1.ПЛАТФОРМЕР ЯК ЖАНР ВІДЕОІГОР. ІНСТРУМЕНТИ РОЗРОБКИ ТА ОСНОВА ХУДОЖНЬОГО ОФОРМЛЕННЯ.

Починаючи опис робочого процесу, потрібно зазначити певні особливості поточного проекту. Проект являє собою ігровий застосунок в жанрі *Action platformer* під мобільну систему Android на рушію Unity у сетингу – *Steampunk* (укр. *стімпанк* або *паропанк)*.

1.1 Поняття відеоігор та історія їхнього виникнення

Відеоігри разом з людством вже більше 60 років. По міркам існування людства це невеликий термін. Але при цьому відеоігри розвивались і змінювались до невпізнанності.

Відеогра – це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотну інформацію з відеопристрою. Електронні пристрої, які використовуються для того щоб грати, називаються ***ігровими платформами***. До таких платформ належать персональний комп'ютер, смартфони, гральні консолі. Пристрій введення, який використовується для керування грою, називається ***ігровим контролером***. Це може бути, наприклад, джойстик, клавіатура та мишка, геймпад або сенсорний екран.

Різноманітні електронні та механічні ігрові пристрої існували ще в першій половині XX століття, але не мали досить значного поширення. Попередниками відеоігор є пристрій «Cathode ray tube Amusement Device» (укр. Розважальний пристрій з електронно-променевою трубкою), патент на яку Томас Голдсміт Молодший та Істл рей Менн отримали 14 грудня 1948 року, і шахова комп'ютерна програма, розроблена у 1947 Аланом Тьюрінгом. Початково ігрові програми, як шахи чи хрестики-нулики, розроблялися в рамках військової програми США у прагненні створити комп'ютер, здатний передбачати дії противника.

Перша успішна спроба створити розважальний пристрій, який використовує для зворотного зв'язку із гравцем, належить Вільяму Гігінботаму. У 1958 він розробив *Tennis For Two*, однак не розглядав гру як щось важливе і зрештою розібрав обладнання для інших, наукових, проектів.

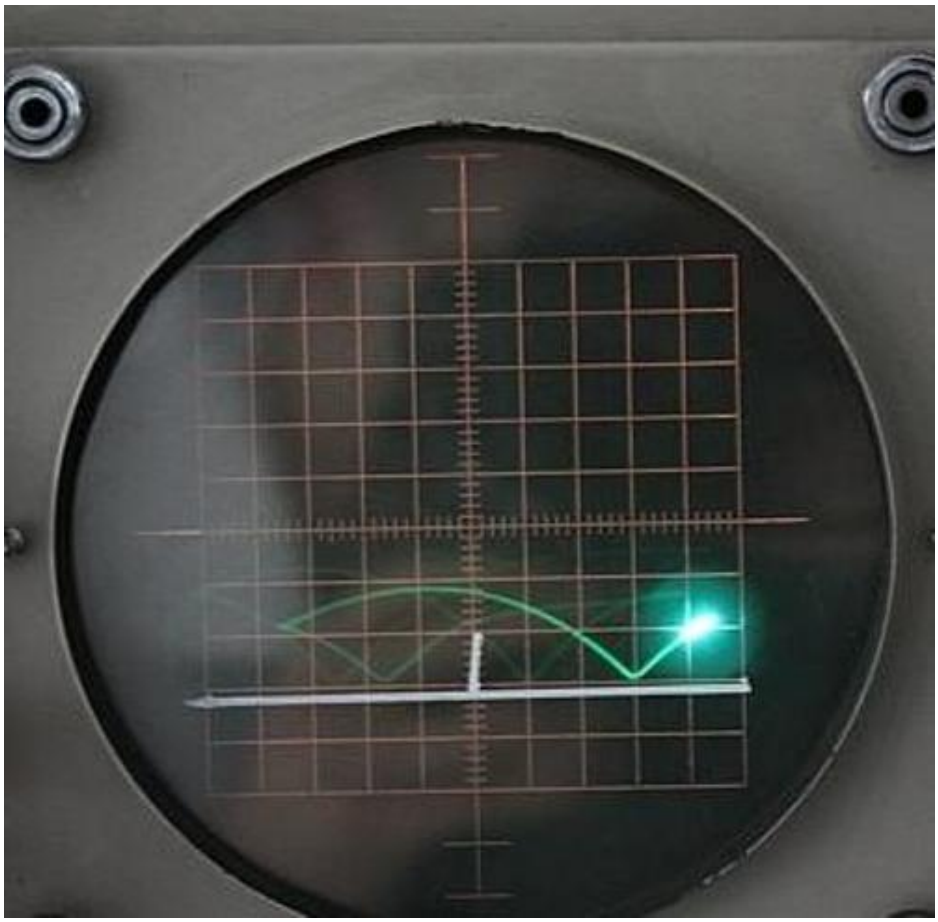

Рис. 1.1 Гра *Tennis For Two*

В 1960-ті студентами Массачусетському технологічному інституту було написано гру Spacewar!, яка у 1966 підштовхнула Sanders Associates на думку про можливість створення грального пристрою, що під'єднувався б до домашніх телевізорів. Коли комп'ютери стали порівняно дешевими і могли використовуватися не лише науковими установами, студент Стенфорду Білл Піттс, вражений Spacewar!, вирішив, що на основі комп'ютера PDP-11 реально створити пристрій спеціально для ігор – *Galaxy Game*. Разом з другом Х'ю Таком він розробив ігровий автомат, що за монети надавав можливість пограти і таким чином окупив би себе. За підтримки Нолана Бушнелла з компанії Nutting Associates цей автомат, *Computer Space* і однойменна гра на ньому 1971

року, став першим комерційним пристроєм з відеогрою.

Рис. 1.2 Гра *Computer Space*

Вже у 1972 році з'явилась перша домашня гральна консоль - *«Magnavox Odyssey»*, яка підключалася до звичайного телевізора. На старті консоль мала з собою в коплекті 12 ігор, а саме *Table Tennis, Ski, Simon Says, Tennis, Analogic, Hockey, Submarine, Football, Cat and Mouse, Haunted House, Roulette, States.*

Рис. 1.3 Ігрова консоль *«Magnavox Odyssey»*

Поштовхом до виникнення індустрії відеоігор стала висока популярність спочатку аркадної, а потім і домашньої відеогри *«Pong»* 1975 року, оскільки її комерційний успіх призвів до появи великої кількості клонів від інших компаній.

Рис. 1.4 Гра *«Pong»*

Перенасичення ринку однотипними консолями, що настало згодом, призвело до першого обвалу ринку відеоігор у 1977 році, який завершився у 1978 році з виходом гри *«Space Invaders»* від компанії *«Taito»*, яка отримала велику популярність і призвела до початку так званої «золотої епохи аркадних відеоігор» та надихнула багатьох компаній вийти чи повернутися на ринок.

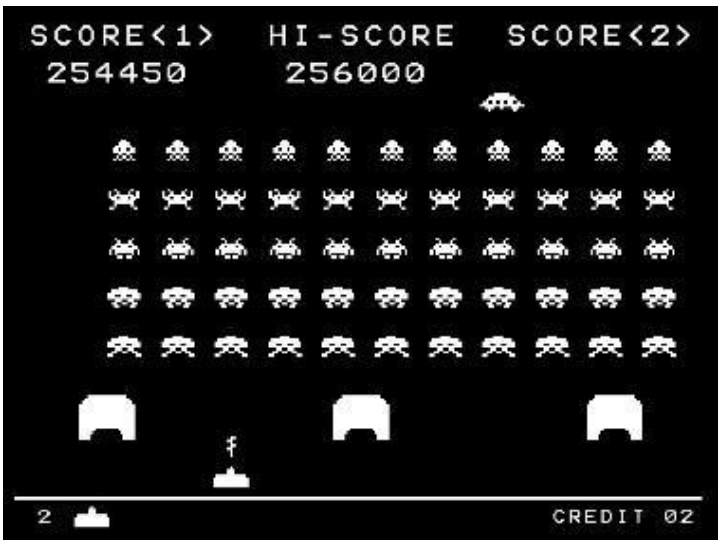

Рис. 1.5 Гра *«Space Invaders»*

Незабаром гра була ліцензована для грального консолі *«Atari VCS»* (пізніше відома як *«Atari 2600»*), продажі якої збільшились у чотири рази. Комерційний успіх компанії дозволив їй відновитися після втрат минулих років та повернув довіру до ринку домашніх грального консолей, почавши їхнє друге покоління.

Рис. 1.6 Ігрова консоль *«Atari 2600»*

Але з часом ринок ігор був знову перенасичений. Як наслідок почався новий кризис в ігровій індустрії. В цей період **Atari** зіткнулися з обмеженістю тодішніх ігрових платформ і браком оригінальних ідей. Ринок консольних ігор різко впав з 3,2 млрд. доларів у 1983 році до 100 млн. у 1986 році.

Рис. 1.7 Знищення та утилізація великої кількості не проданих ігор для «*Atari 2600*» під час кризису

Цей кризис не задів платформу домашніх комп'ютерів. Також на цей період лідерство на ігровому ринку забрали японці. Компанія Nintendo випустила першу свою гральну консоль під назвою «*Nintendo Entertainment System*». Японці не тільки нормалізували ринок, а й значно його збільшили. Їх успіх почав третє покоління гральних консолей, під час якого в індустрії домінували японські розробники.

Рис. 1.8 Ігрова консоль «*Nintendo Entertainment System*»

Із поширення CD-дисків та впровадження CD-приводів як стандартної складової ПК, домашні комп'ютери стали розглядатися як повноцінна ігрова платформа і конкурент приставок. Стрімкий розвиток ПК дозволив створювати все більш видовищні та сюжетно наповнені ігри, залучати до їх створення професійних акторів і сценаристів. До використання дисків поступово переходили і приставки. В гонитві за видовищністю за кілька років відеоігри еволюціонували від двовимірних і псевдо тривимірних (Doom, Duke Nukem, Hexen, System Shock) до повноцінних тривимірних (Half-Life, Quake, Unreal, Thief: The Dark Project), бюджети яких дорівнювали бюджетам фільмів. Компанія «*Id Software*» започаткувала новий підхід до створення інтерактивних творів, продаючи ліцензії на використання ігрових рушіїв та інструменти створення відеоігор. Це дозволило робити ігри простіше і вийти на ринок незалежним студіям.

У 2000-ні роки відеоігри ставали все реалістичними в плані графіки, розвинулися онлайн ігри та мережеві режими, що дозволили багатьом гравцям взаємодіяти в реальному часі. Наприкінці десятиліття відбулася інтеграція з ігровими сервісами та соціальними мережами. Ігри на фізичних носіях стали відходити у минуле, поступаючись цифровій дистрибуції. Це обумовило поширення інді-ігор. Гравці стали активними учасниками створення й

модифікації відеоігор шляхом моддингу, конструювання інтерактивних творів у спеціальних редакторах.

У 2010-і стрімко підвищилися стандарти якості графіки, зокрема після виходу восьмого покоління ігрових систем, що позначає поширення 4K-дисплеїв. Значно зросла кількість ігор і пристроїв з підтримкою віртуальної реальності. Сильно розвинулася індустрія ігор для мобільних пристроїв, зокрема смартфонів.

1.2 Ігрові жанри платформер та action. Їх особливості.

Платформер (англ. *platformer*) - жанр комп'ютерних ігор, в основу ігрового процесу яких становлять стрибки платформами, пересування сходами, збирання предметів, які необхідні, щоб перемогти ворогів чи завершити рівень.

Цей ігровий жанр з’явився ще з ранніх часів ігрової історії, тому на цей час має багату історію, яка ділиться на декілька етапів:

1. ***Ера двовимірної графіки*** - платформери з'явилися на початку 1980-х, коли ігрові консолі були не досить потужними, щоб зображати тривимірну графіку чи відео. Вони були обмежені статичними ігровими світами, які містилися в одному екрані, а ігрового героя було видно у профіль. Персонаж міг пересуватися вгору та вниз за допомогою драбини або стрибав платформами, борючись із противниками та збираючи предмети.
2. ***Ера тривимірної графіки*** - термін «тривимірний платформер» може позначати або геймплей, що включає всі три виміри, або використання тривимірних полігонів у реальному часі для малювання рівнів та героїв, або те й інше. Із-за розвитку технологій, ігри цього жанру змогли змінити цілі проходження рівня. Наприклад щоб пройти рівень треба було вирішити головоломку, а для цього було необхідно обшукувати всі частини мапи, щоб знайти підказку. Приклади: «Banjo-Kazooie» або «Super Mario 64». Це дало можливість ефективного використання великих тривимірних областей і винагороджувати гравця за ретельне дослідження рівня.
3. ***Ізометричні ігри*** - ізометричні платформери є піджанром двовимірних та тривимірних платформерів. Вони зображують тривимірну сцену за допомогою двовимірної графіки, яка показує світ із жорстко орієнтованою камерою без урахування перспективи. І хоча ізометричні

платформери не були першими ізометричними іграми. Приклади: *«Congo Bongo»* та *«3D Ant Attack для ZX Spectrum»*.

На сьогоднішній день здебільшого найпопулярнішим жанром серед платформерів є *action*. Тому саме даний кваліфікаційний проект проектувався як один із представників даного жанру, за стиль художнього оформлення був обраний жанр *steampunk*.

Action (*укр. екшен*) - жанр комп'ютерних ігор, в якому наголошується на експлуатацію фізичних можливостей гравця, у тому числі координації очей, рук і швидкості реакції.

Історію цього жанру важко описати тому, що він є важливою частиною всіх інших жанрів. Перша гра, яка підпадає під цей жанр з'явилася в 1978 році, та називалась - *«Space Invaders»*. Після масового успіху Space Invaders в індустрії стали домінувати бойовики, які залишаються домінуючим жанром у відео-аркадах та на ігрових консолях до сьогодні. *Space Invaders* створили шаблон для наступних ігор у піджанрі шутера, і вона вважається однією з найвпливовіших ігор усіх часів.

Саме ця гра повернула рух індустрії комп'ютерних ігор у бік активного ігрового процесу *(action).*Термін *«екшн»* почав використовуватися на початку 1980-х років у зв'язку з новим жанром екшн-ігор, що виникли від японських розробників аркадних ігор, черпаючи натхнення з культури манги та аніме.

Популярна гра-лабіринт від Namco *Pac-Man* (1980) популяризувала жанр екшн-ігор з не зовсім бойовим геймплеєм. Це була одна з перших популярних екшн-ігор без стрільби, яка визначала ключові елементи жанру, такі як «паралельна візуальна обробка», яка вимагає одночасного відстеження кількох сутностей, включаючи персонажа гравця, місцезнаходження персонажа, ворогів та активізатори. Інші класичні приклади екшн-ігор із персонажами включають *Donkey Kong* (1981) від Nintendo, який створив шаблон для піджанру платформних ігор , а також *Frogger* від Konami (1981). Ігри про бойові мистецтва зрештою з'явилися в середині 1980-х років, коли

Data East's Karate Champ (1984) заснував піджанр файтингів один на один.

Тенденцією, яка була популяризована для екшн-ігор на початку 1990-х років, була змагальна багатокористувацька гра, включаючи те, що пізніше буде відоме як кіберспортивні турніри. Аркадний файтинг *Street Fighter II* (1991) від Capcom популяризував концепцію прямого змагання на турнірному рівні між двома гравцями.

У 1990-х відбулася «3D-революція», коли екшн-ігри здійснили перехід від 2D-і псевдо-3D-графіки до 3D-графіки в режимі реального часу. Системні плати 3D-аркади, які спочатку були розроблені для 3D-гоночних ігор наприкінці 1980-х - початку 1990-х років, такі як Namco System 21, Sega Model 1 і Sega Model 2, використовувалися для створення 3D-аркадних ігор на початку 1990-х, включаючи 3D шутери, такі як *Galaxian 3* від Namco (1990) і *Solvalou* (1991), тривимірні файтинги, такі як *Virtua Fighter* від Sega AM2 (1993) і Tekken (1994), а також тривимірні шутери з використанням світлового пістолету, такі як *Virtua Cop* від Sega AM2. На персональних комп'ютерах жанр шутера від першої особи (FPS) популяризував Doom; це також вважається, незважаючи на відсутність використання 3D-полігонів, великим стрибком вперед для тривимірного середовища в іграх. 3D-полігонне відображення текстур з'явилося в іграх-екшенах приблизно в середині 1990-х років, увійшли до файтингів від Sega AM2 *Virtua Fighter 2* (1994) та до ігор FPS від Parallax Software's *Descent* (1995).

Steampunk (*укр. стімпанк, стимпанк або паропанк*) - підвид фантастики, дія якого відбувається у світі, де широко використовуються технології парових машин, що замінюють електроніку. Класичні всесвіти стімпанку стилізуються авторами під Америку або Англію другої половини XIX століття в епоху раннього капіталізму з характерним фабрично-міським пейзажем і різким соціальним розшаруванням. Звичайно, що події можуть відбуватися не тільки в цих рамках. Це може бути, наприклад інший фантастичний світ, інша планета, або взагалі альтернативне минуле чи

альтернативне майбутнє, де парові технології являються основними.

Термін *«стімпанк»* вперше використали два письменники, Джеймс Блейлок і Кевін Джетер, під час дискусії на сторінках журналу *«Locus»* в 1987 році. Сам термін виник як сатира на жанр *«кіберпанк»*. Тому перші історії та сюжети використовували саме з цього жанру, замінюючи все що можливо паровими технологіями. Депресивне ставлення головних героїв до навколишньої дійсності зберігалося без яких-небудь істотних змін. Видання роману Вільяма Гібсона і Брюса Стерлінга *«Різницева машина»* привернуло велику увагу до цього жанру, і саме після цього роману *стімпанк* почав набувати великої популярності.

В даній роботі було використано стилістичне оформлення яке є не зовсім *стімпанком,* а одним із його підвид - *фентезійний стімпанк*. Його особливості полягають в тому, що світ, в якому відбуваються події є повністю вигаданим. Тобто в поточному випадку можуть бути інші закони фізики, інша хімічна складова та присутня магія, але при цьому світ населяють не тільки люди, а й інші вигадані істоти.

1.3 Постановка задачі

У розробці сучасних комп’ютерних ігор найбільш продуктивною є технологія ігрового движка, яка спрямована на спрощення процесу розробки шляхом уніфікації та систематизації внутрішньої структури гри.

Ігровий рушій (анг***. game engine)*** - це основне ядро, яке відповідає за реалізацію основних функцій гри: візуалізацію ігрової сцени, моделювання фізичних законів реального світу у віртуальному, відтворення звуку, що створює ілюзію інтелекту (ігровий движок), штучний інтелект, анімація.

Розглянемо декілька ігрових рушій, які сьогодні є найбільш популярні на ринку:

Unity - ігровий рушій, який був створений для того, щоб надати більшій кількості розробників доступ до інструментів розробки ігор у 2D та 3D вимірах. Ігровий движок зосереджений на тому, щоб забезпечити найнадійніший набір інструментів для індустрії розробки ігор, а також зробити його якомога легшим для розробників ігор будь-якого рівня кваліфікації (включаючи початківців). Коли Unity з'явився на ринку, більшість серйозних двигунів для розробки ігор були платними. Безкоштовні програми, такі як *RPG Maker*, пропонували лише частину функцій. Тому, коли світ побачив повноцінну та безкоштовну альтернативу, багато хто вирішив зламати свій страх і зайнятися розробкою. Unity розвивався, пропонуючи все більше цікавих рішень як для 3D, так і 2D-ігор.

Unreal Engine – ігровий рушій, який містить різні механізми, які надають користувачам платформу та допомагають їм запускати гру. Крім того, він має величезний систематичний набір інструментів і редакторів, які допомагають користувачам керувати своїми властивостями та змінювати їх. Ігри, розроблені на Unreal Engine працюють на майже будь-якій платформі (Mac, iOS, Android, PC, VR). Освоїти Unreal Engine складніше, ніж Unity, проте рушій має велику кількість інструментів, від створення ландшафтів до роботи

з 3D-моделями.

Godot - це кросплатформний ігровий движок, орієнтований на розробку 2D і 3D ігор. Ігровий движок зосереджений на наданні всеохоплюючого набору інструментів для розробки, включаючи вбудований редактор коду, механізм візуалізації графіки, інструменти відтворення аудіо, інструменти анімації тощо. На відміну від інших двигунів, де 2D досягається простим вирівнюванням однієї осі, Godot пропонує справжній 2D-движок. Це означає, що движок може ефективно обробляти 2D-серверні обчислення, а також належним чином працювати з одиницями на основі пікселів. Крім того, пакет для розробки 2D-ігор містить ряд спеціальних інструментів, таких як редактори карт плиток, підтримка фізики 2D та системи 2D освітлення.

В даній кваліфікаційній роботі планується використовувати ігровий рушій - Unity, тому що його великою перевагою є те, що для початківців він є більш зрозумілішим та простішим в освоєнні. Для прикладу, Unreal Engine є більш вибагливим до системних вимог і він дещо поступається Unity в мобільній розробці як в 2D так і в 3D. Здебільшого наразі на ринку мобільних ігор не вимагається найтехнологічніша найсучаснішої графіка, зокрема це обумовлюється технічними характеристиками самих мобільних пристроїв що зазвичай є далеко не найпотужнішими, тому даний проект який розроблявся в межах поточної кваліфікаційної роботи старається охопити якомога більшу кількість мобільних пристроїв з огляду на їх характеристики, саме тому ігровим рушієм був обраний Unity. Також сам рушій має нижчі ніж у конкурентів системні вимоги, сам рушій і проекти на ньому займають менше місця на диску. Варто зауважити, що велика кількість ігор в App Store та Play Market працюють саме на Unity.

1.3.1 Ціль та завдання проекту

В даній роботі було реалізовано ігровий застосунок в жанрі «Action Platformer» під мобільну систему Android на рушію Unity у сетингу – *steampunk*. Для досягнення поставленої цілі нам необхідно було вирішити такі моменти:

- провести аналіз аналогічних ігор;
- провести аналіз та вибір засобів реалізації гри;
- описати концепцію гри;
- намалювати персонажів для гри
- спроектувати програмну систему;
- реалізувати гру;
- провести тестування реалізованої гри.

1.3.2 Знайомство з світом. Сюжет

Події гри відбуваються у світі, який схожий на наш, але з деякими відмінностями. У цьому світі парові технології прийшли набагато раніше, ніж у нашому. Це сталось десь на кінець XV століття. У XVI столітті нові технології заполонили майже всі сфери життя людей. У цей період змогли реалізувати технологію, яка дала можливість створювати автоматони на паровому двигуні для різних робіт, протези які майже повністю заміняли втрачені кінцівки за рахунок парових технологій, парові станки, парові карети, дережаблі, автономні летючі платформи.

Сама гра відбуваються у невеличкому промисловому містечку Берн, який знаходився на окраїні королівства Славмакія. Данемістечко відіграє значнуроль в королівстві за рахунок двох великих підприємств, які займаються видобуванням кам'яного вугілля та виробництвом деревного вугілля. Цінність цього товару обумовлена тим, що вони є основним паливом для парових рушіїв.

Головного героя також звати Берн. Він колишній професійний

військовий на пенсії через поранення (втратив руку). Але хороший протез і кваліфікація механіка дала йому можливість знайти роботу у лісозаготівельній компанії, яка знаходилась у містечку.

Події гри починаються з звичайного робочого дня. Берн снідав зі своєю родиною, як прийшла телеграма з викликом на місце останньої лісозаготівлі. Сталася поломка майже зі всіма робочими автоматонами типу «Лісоруб».

Рис. 1.9 Головний герой з сином

Головний герой відправляється на виклик. Коли він дібрався до місця, нікого не було крім одного з’їхавшого з глузду автоматона, який на нього напав, але Берн знищив його. Після цього інциденту головний герой відправляється до невеличкої промислової зони в лісі, яка належить цій компанії. Прибувши туди, Берн розуміє, що всі автоматони з’їхали з глузду, і почали нападати на людей. Тепер його ціль добратися додому, і вивезти сім’ю подалі звідси.

Висновок до розділу 1

У даному розділі було розглянуто історію відеоігор, жанр «платформер», його конкуренти, ключові особливості та механіки. Також розглянули що таке

ігровий рушій та які є популярні на ринку. В якості ігрового рушія, який використовується для розробки 2D гри є Unity.

РОЗДІЛ 2. ПОБУДОВА КОНЦЕПЦІЇ ГРИ ТА ПРОЕКТУВАННЯ ІГРОВИХ МЕХАНІК. СТВОРЕННЯ ПОКАДРОВОЇ АНІМАЦІЇ ІГРОВИХ ТА НЕ ІГРОВИХ ПЕРСОНАЖІВ.

2.1 Анімація спрайтів

Перед описом робочого процесу варто пояснити, що таке Photoshop, анімація та покадрова анімація.

Adobe Photoshop - графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей редактор є одним із найпопулярніших на ринку в галузі комерційних засобів редагування растрових зображень. Він доступний на платформах Mac OS і Windows.

Анімація - вид образотворчого та кіномистецтва, яка створюються шляхом знімання послідовних фаз руху 2D або 3D об'єктів.

Покадрова анімація - це анімація, яка використовує покадрову технологію.

Покадрова технологія - це технологія, за якою кожен кадр малюється окремо. Ця технологія вважається найбільш складною та тривалою, вона вимагає високої майстерності, досвіду та інтуїції. Проте ця технологія дозволяє здійснити практично будь-які зміни об'єкта.

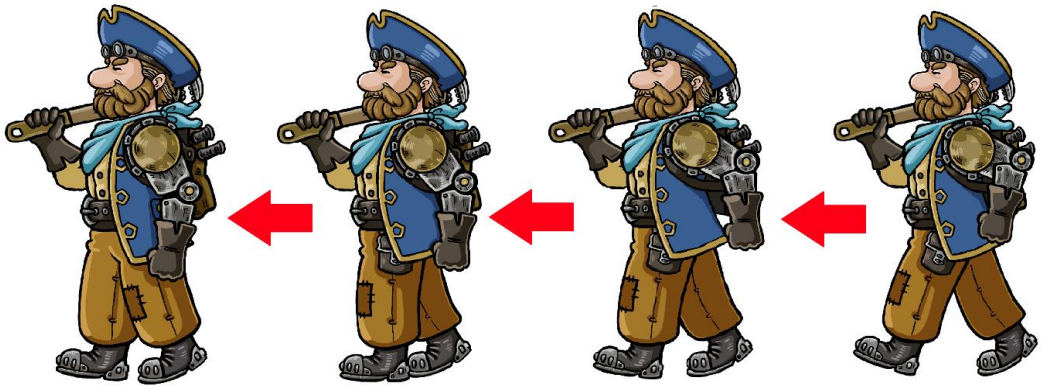

Рис. 2.1 Приклад анімації по покадрової технології.

Для початку був створений простий проект розміром 3000 * 3000 пікселів.

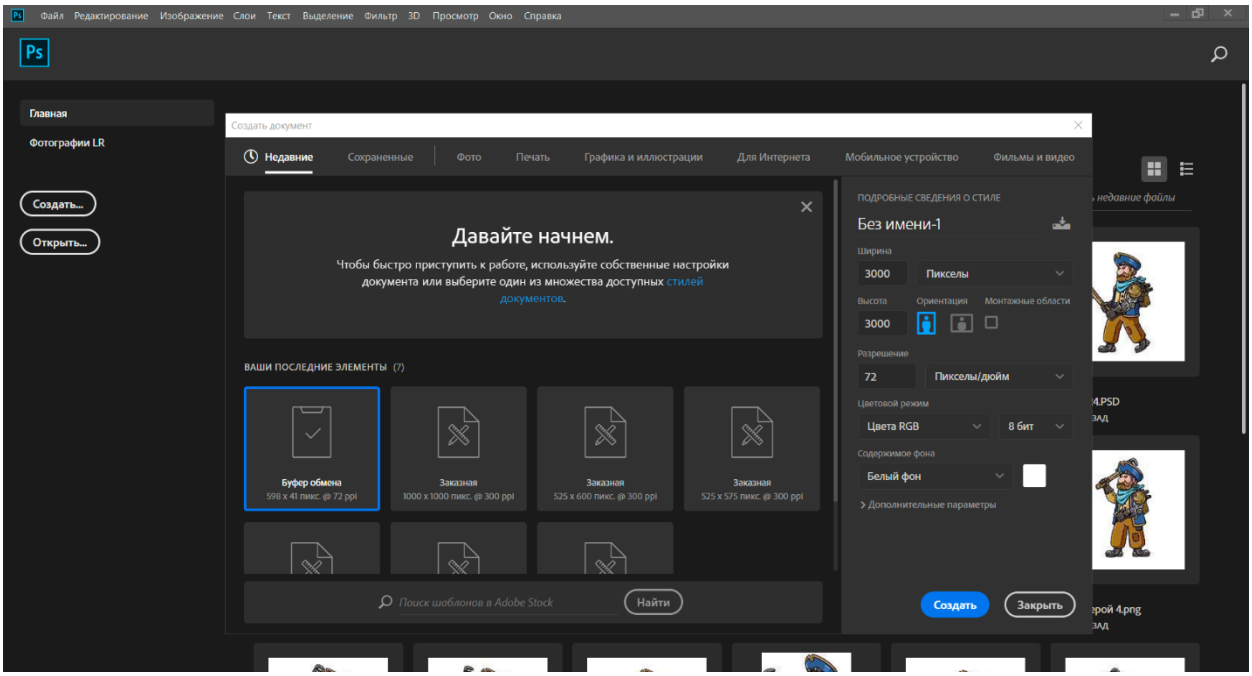

Рис. 2.2 Створення нового проекта в Adobe Photoshop

Далі командою клавіш Ctrl + Shift + N створюється новий пустий шар. Для першого кадру буде достатньо трьох, а саме:

- Повністю білий (для заднього фону).
- Шар з лінією(англ. line).
- Шар з пігментом.

Цей розподіл буде більш наглядно продемонстровано на Рис. 2.3

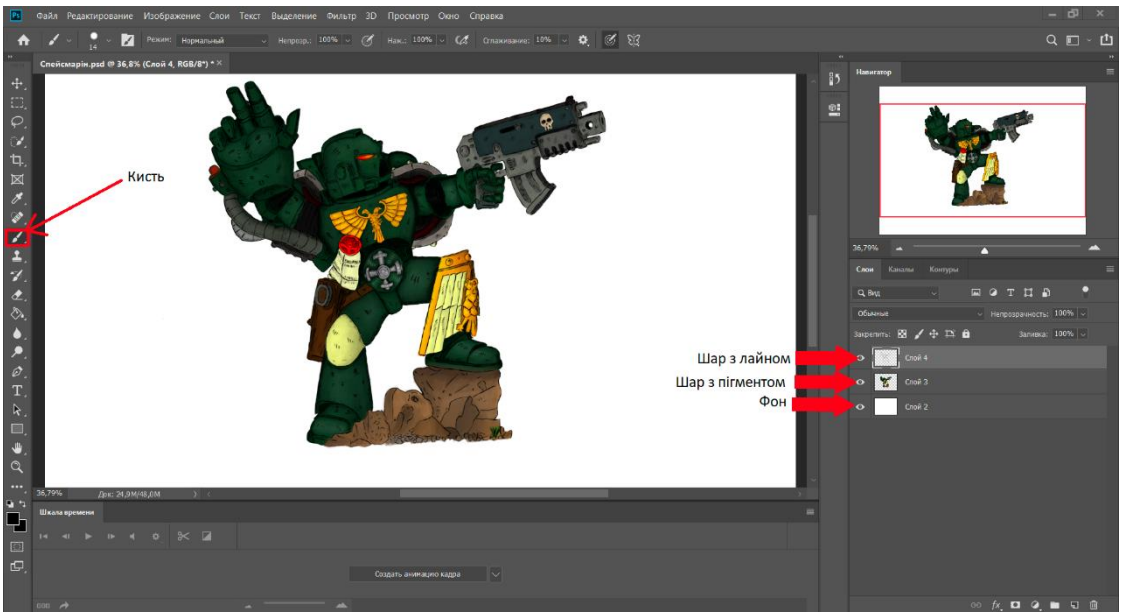

Рис. 2.3 Демонстрація розподілу шарів у Adobe Photoshop

Для створення першого кадру не використовував ніяких особливих інструментів. Просто робиться ескіз за допомогою інструмента *Кисть*. Потім задається пігментація на іншому шарові використовуючи також цей інструмент.

Фінальний результат потрібно зберегти у .PNG форматі. Для цього натискаємо клавішу *Файл*, а потім *Зберегти як…* і вибираємо потрібний формат.

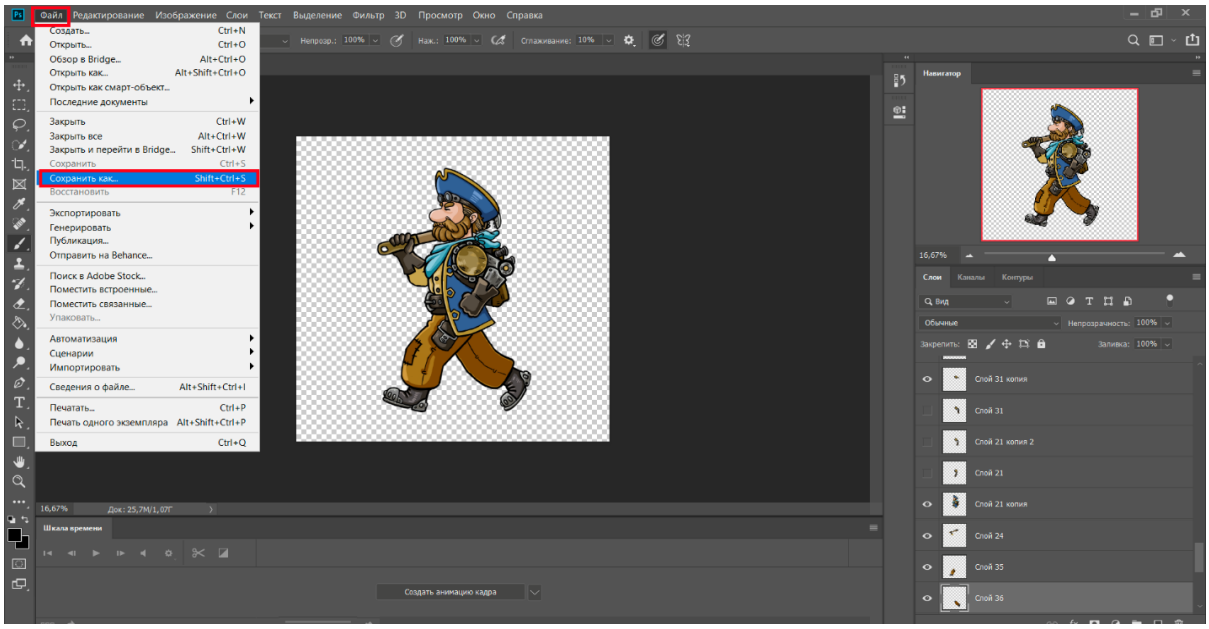

Рис. 2.4 Зберігання 1 кадру в Adobe Photoshop

В роботі з наступними кадрами користуємось однією хитрістю. Наступний кадр не робимо повністю з нуля. Замість цього скористався таким інструментом як *Лассо*.

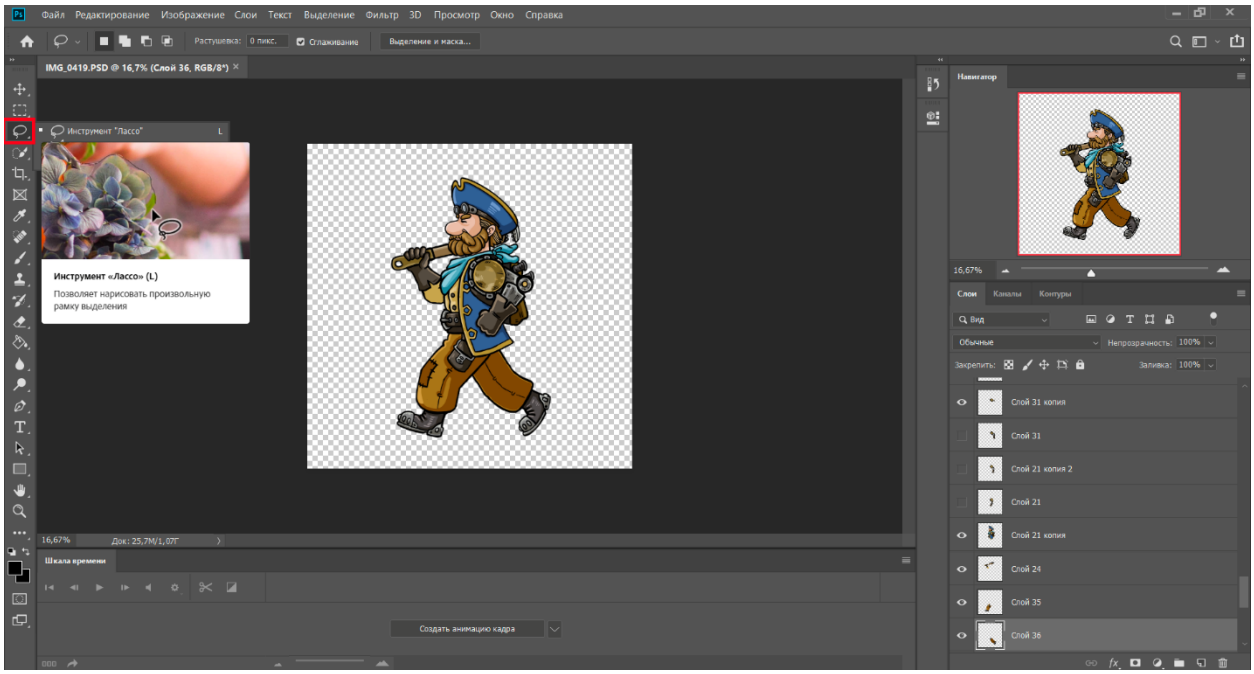

Рис. 2.5 Інструмент *Лассо* Adobe Photoshop

Воно використовується для виділення частини зображення для того, щоб його скопіювати на новий шар, вирізання даного сегмента зображення. В даному випадку воно використовував для створення дублікатів сегмента зображення на новому шарі.

Потім змінюючи положення даного сегмента, можна отримати нову позу головного героя. Для цього використовується комбінацію клавіш Ctrl + T.

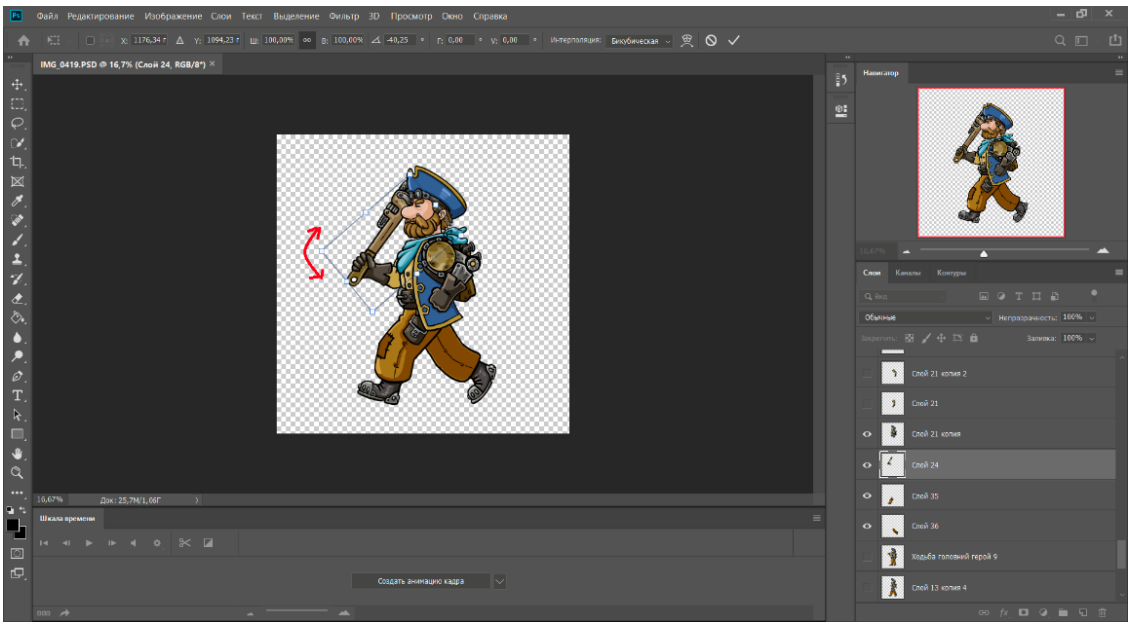

Рис. 2.6 Коригування зображення за допомогою *Ласо* та комбінацію клавіш Ctrl + T в Adobe Photoshop

Звичайно для цього використовується не тільки *Лассо*. В цьому процесі необхідні були такі інструменти як *Кисть* та *Гумка*, для корегування зображення. Якщо точніше, то корекції підлягають: тіні, лінії, зайві шматки зображення, відсутні шматки зображення.

Після цього зберігаємо кадр. І цей процес повторюється до останнього кадру. На фіналі отримується ланцюг з 36 кадрів у форматі PNG. В самій анімації демонструється ходьба головного героя. Особливими є лише тільки перші кадри, на них головний герой паралельно кроку закидає гайковий ключ собі на спину, а на наступних продовжує свій крок уже з закинутим ключем. Інші ланцюги анімації створюються абсолютно за схожим принципом.

Після створення інших ланцюгів анімації виникає нова потреба, а саме підігнати спрайти під потрібні розміри. Для цього також необхідно використовувати додаток Adobe Photoshop. У даному випадку необхідний розмір кадру 300*300 пікселів. Для цього відкриваємо кадр у форматі PNG через Adobe Photoshop. Після цього натискаємо на клавішу *Файл*, далі на клавішу *Автоматизація*, далі на клавішу *Змінити розміри*, і вже там задаємо

потрібні габарити.

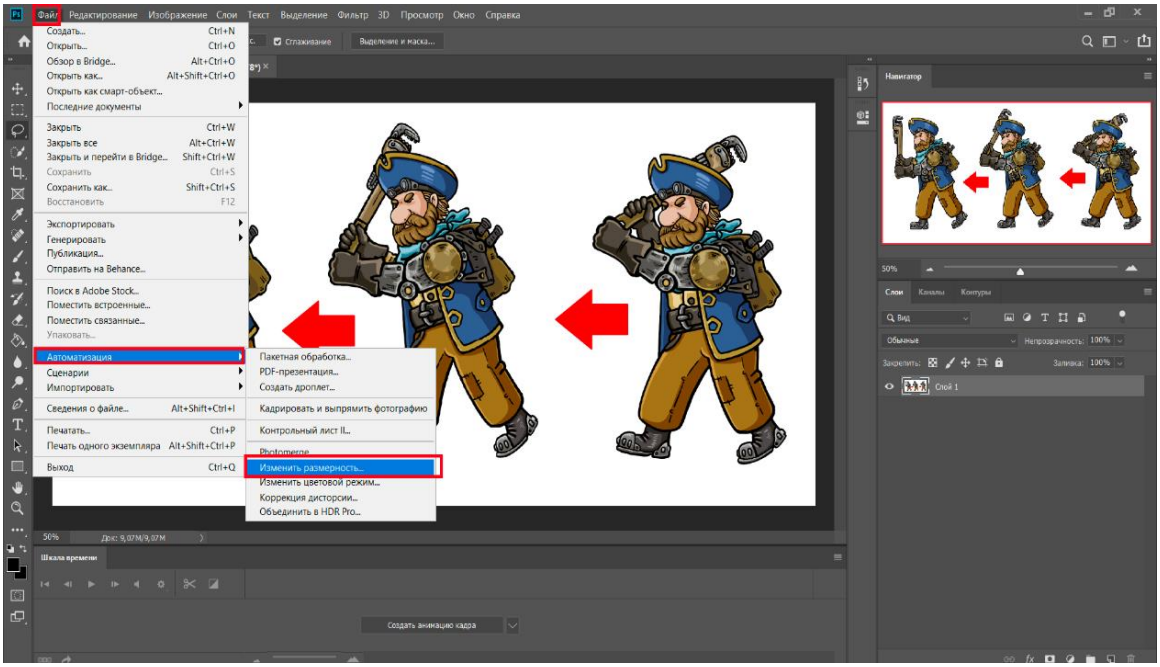

Рис. 2.7 Зміна розміру кадра

Після цього все зберігаємо. Тепер кадри потрібного розміру. Наступне, що буде необхідно зробити, це створити загальний PNG файл з усіма ланцюгами анімації персонажа. Для виконання даного етапу також використовується додаток Adobe Photoshop. Створюється пустий PNG файл, де буде розтановлені створенні кадрові ланцюги. Ці кадри програма відобразить у вигляді слоїв. Далі необхідно розмістити ці слої у потрібному положенні. Після чого зберігаємо файл у форматі PNG. Також ці операції будуть аналогічні і для ворогів.

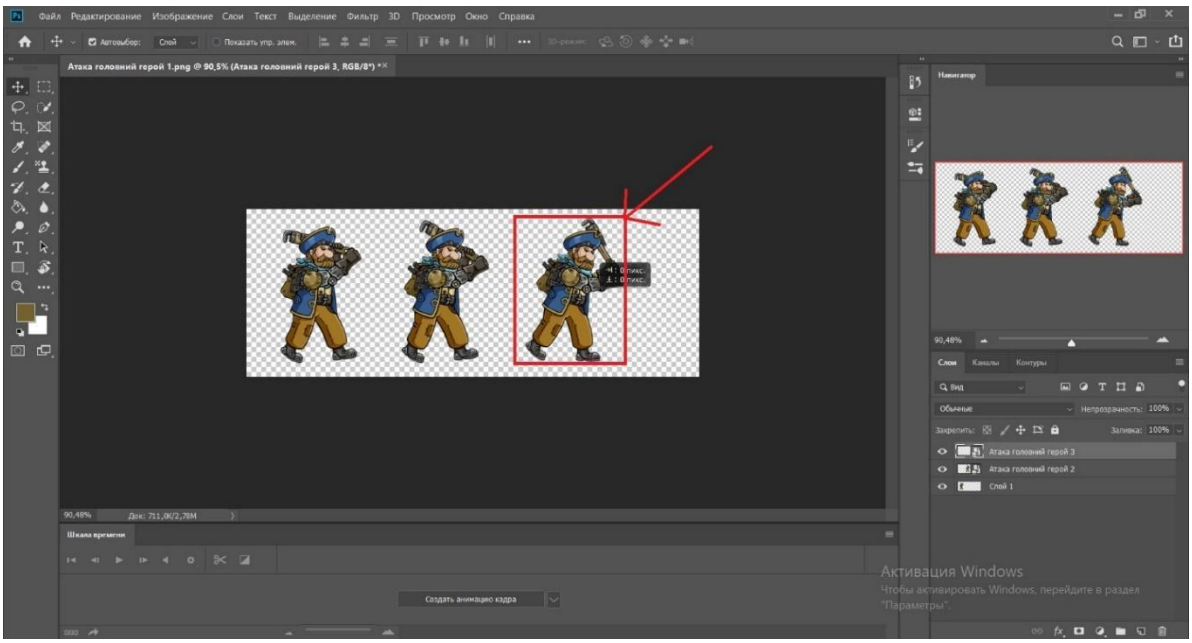

Рис. 2.8 Розстановка кадрів на полотні

Рис. 2.9 Розставлені кадри на полотні

2.2 Дизайн та філософія рівнів

Дизайн рівнів (англ. level design) – дисципліна в розробці відеоігор, яка включає в себе створення рівнів – локацій, оточення, місій на них, завдань. Зазвичай це робиться за допомогою редактора рівнів – програми, спеціально призначеної для таких цілей.

Сам дизайн рівнів визначається загальною концепцією відеогри, візуальним стилем і доступними технологіями. Перед самим процесом виконання роботи над дизайном рівня передує робота художників і дизайнерів з створення концепт-артів, спрайтів об’єктів та персонажів, з-поміж різних варіантів яких затверджуються ті, що найбільше відповідають задуманому рівню та грі.

Концепт-арт – напрям у мистецтві, призначений для того, щоб візуально передати ідею твору, але не форму чи зовнішні атрибути. Як правило, створюється на стадії розробки проекту і призначається для використання у фільмах, комп'ютерних іграх, коміксах до створення фінальної версії.

Спрайт – двомірне зображення, що застосовується в комп'ютерній графіці, частіше за все являє собою растрове зображення, що вільно переміщується по екрану.

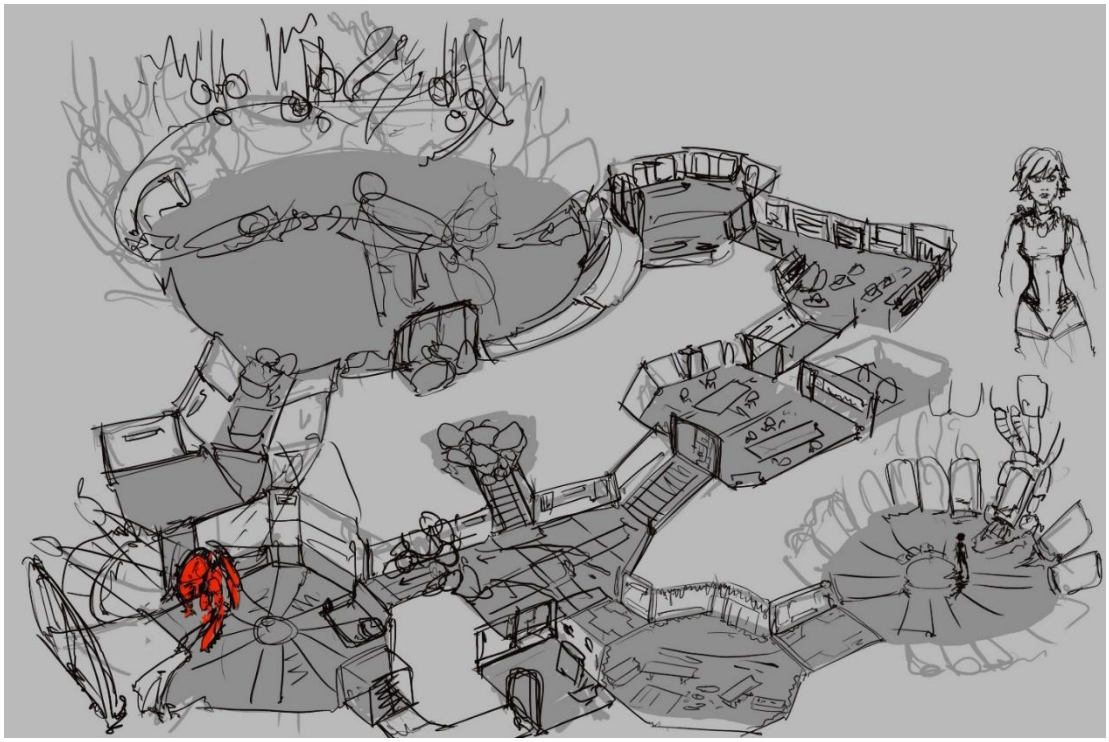

Рис. 2.10 Приклад концепт-артів рівня до гри

При роботі над рівнем творці відеогри виокремлюють із загального ігрового процесу ігровий процес даного рівня. Звичайно по механікам можуть бути схожі рівні. Але також є рівні з повністю унікальним геймплеем. Прикладом можуть слугувати рівні з переслідуванням противника або втечею у платформері, враховуючи, що майже весь час на інших рівнях була задача просто дійти з точки А до точки В.

Але дизайнер рівнів не може змінити основні правила гри, встановлені геймдизайнером, тому, оперуючи налаштуваннями ігрового світу, оточенням і персонажами, їх можливостями та властивостями, змінює ігровий процес за рахунок створення нових ігрових ситуацій. Він може обрати переважний тип діяльності персонажа на конкретному рівні, накласти обмеження, відповідно до специфіки жанру гри.

Коли стають відомими основні риси рівня, дизайнери беруться за створення робочого прототипу. Він складається з простих геометричних форм, наприклад, замість будинків ставляться куби без текстур або графічні заглушки.

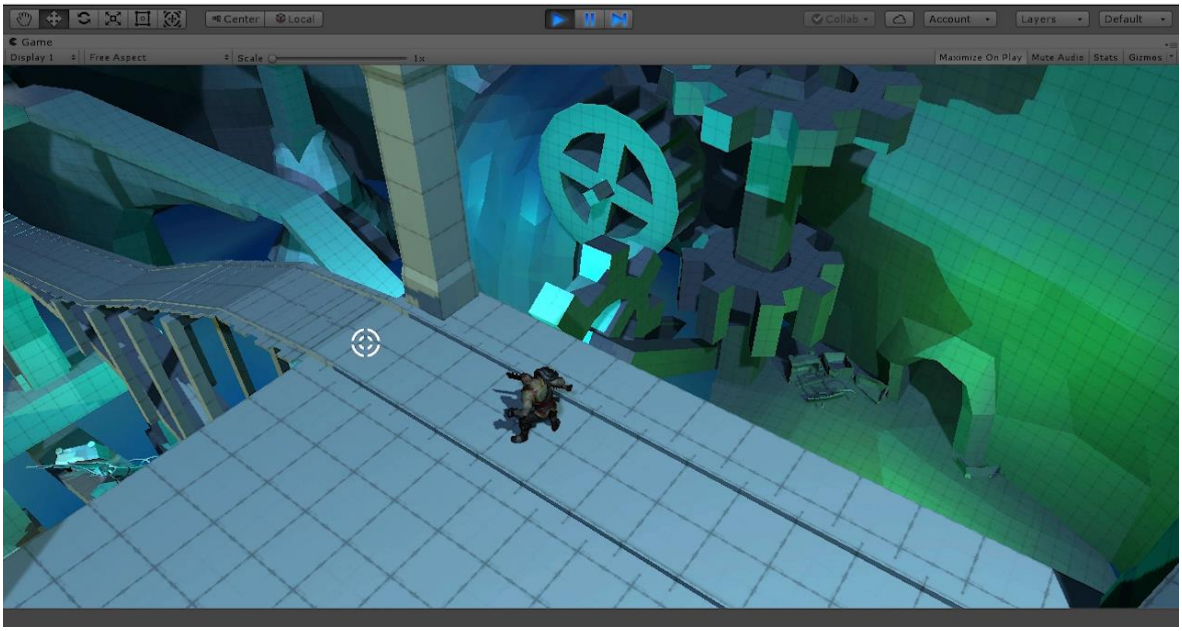

Рис. 2.11 Приклад робочого прототипу

На фінальних етапах роботи над рівнем уже ведеться робота над різними дрібницями, наприклад: налаштовується освітлення, спецефекти, озвучення. Під час виконання роботи над самим проектом концепт-арти рівня не використовувались.

Геометрія рівня в проекті складається з тайлів. **Тайлова 2D-гра** – це будь-яка гра, в якій рівні або ігрові області складаються з безлічі невеликих плиток (тайлів), що утворюють сітку тайлів. Іноді відмінності між тайлами можуть бути очевидними, а іноді вони здаються гравцям суцільними та невиразними.

Для початку роботи над створенням рівня необхідно створити пару «тайлсетів» для палітри тайлів. Колекція тайлів, що є в грі, називається «тайлсетом», і кожен тайл зазвичай є спрайтом, частиною листа спрайтів (spritesheet).

Рис. 2.12 Приклад листа спрайтів (spritesheet)

У проекті використовується вже нарізані тайли. Вони були нарізані за допомогою додатку *Photoshop*. Нарізання тайлів виконувалось за допомогою інструменту *прямокутна область*, вибравши стиль *задання пропорцій* вказуючи їх у розмірі 259 * 259 пікселів. Після цього виділяємо потрібну область і робимо дублікат на інший шар. Далі цей процес повторюємо, поки не наріжемо всі потрібні елементи з листа спрайтів.

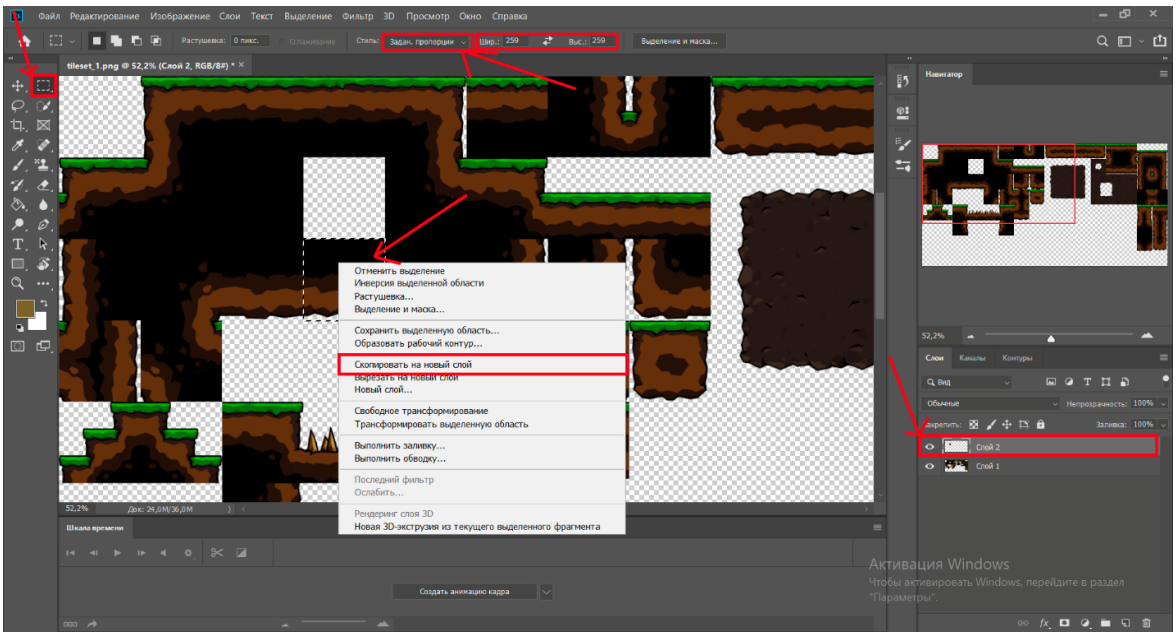

Рис. 2.13 Нарізання тайлів за допомогою Photoshop

Далі необхідно вибрати потрібний нам шар і зберегти його у форматі PNG. Цю процедуру повторити з усіма вирізаними тайлами. Тепер повертаємося до проекту. Після нарізання тайлів, додаємо їх в наш проект. Pixels Per Unit (пер. Пікселі на одиницю) на всіх тайлах виставляємо на позначці 250. Також створимо папку Tiles, куди буде зберігатися наші тайлсети. Далі відкриваємо «Tile palette»(пер. Палітра тайлів). Для цього необхідно натиснути клавішу Window, потім 2D, після цього Tile palette.

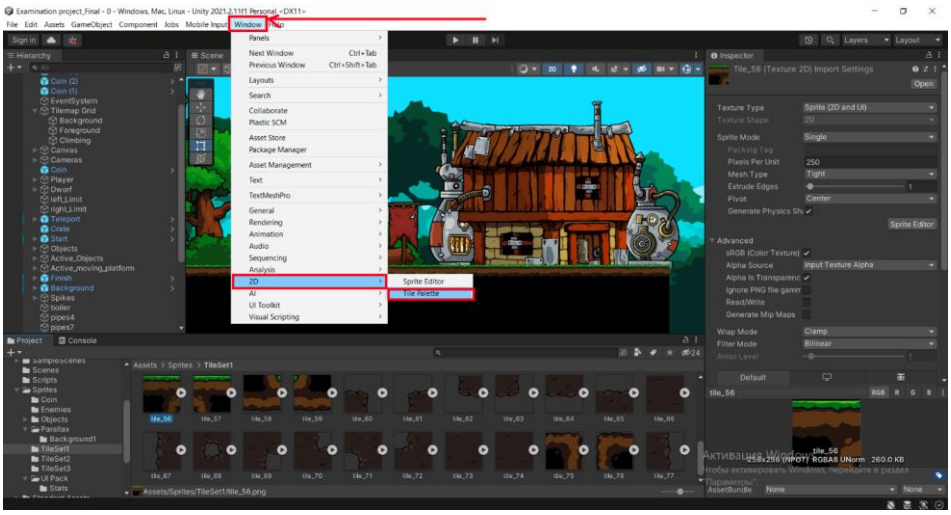

Рис. 2.14 Відкриття Tile palette

Після цих операцій відкриється вікно, де створюємо нову палітру. Для цього натискаємо клавішу вибору палітри, там натискаємо клавішу Create new palette, далі надаємо назву для палітри, яку нам необхідно, після чого обираємо потрібну папку, де збережеться палітра.

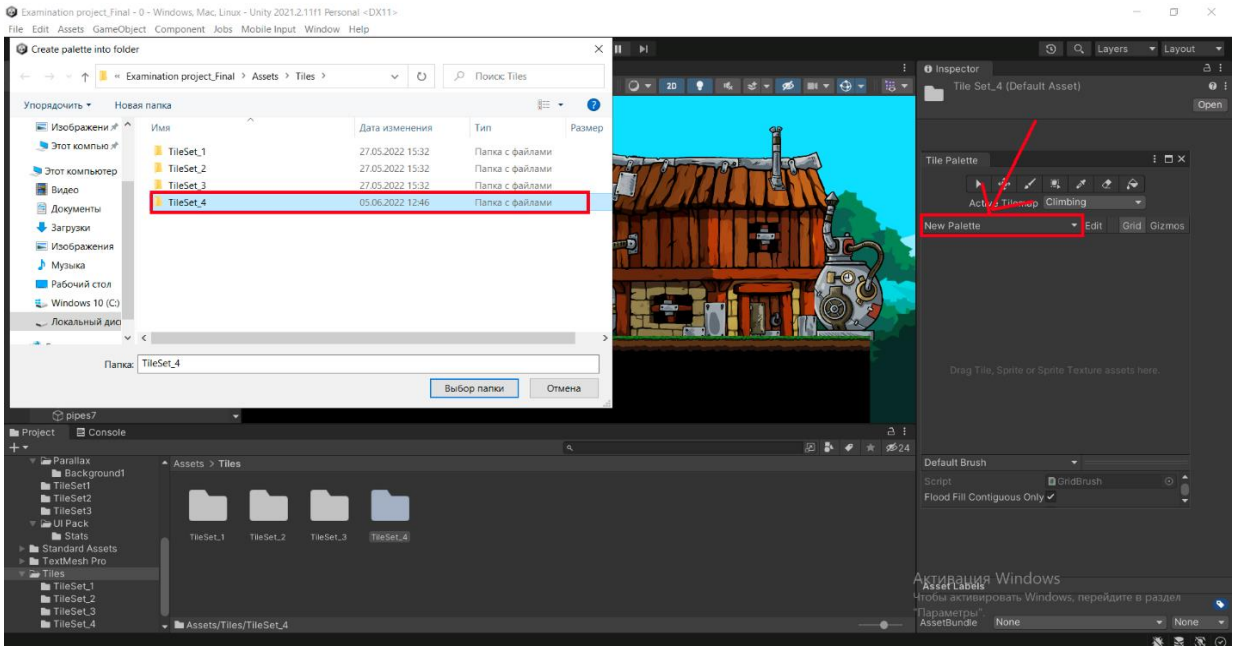

Рис. 2.15 Створення нової палітри тайлів.

Далі, щоб додати тайли до палітри необхідно виділити всі необхідні тайли і перетягти їх на пусту палітру методом Drag & Drop.

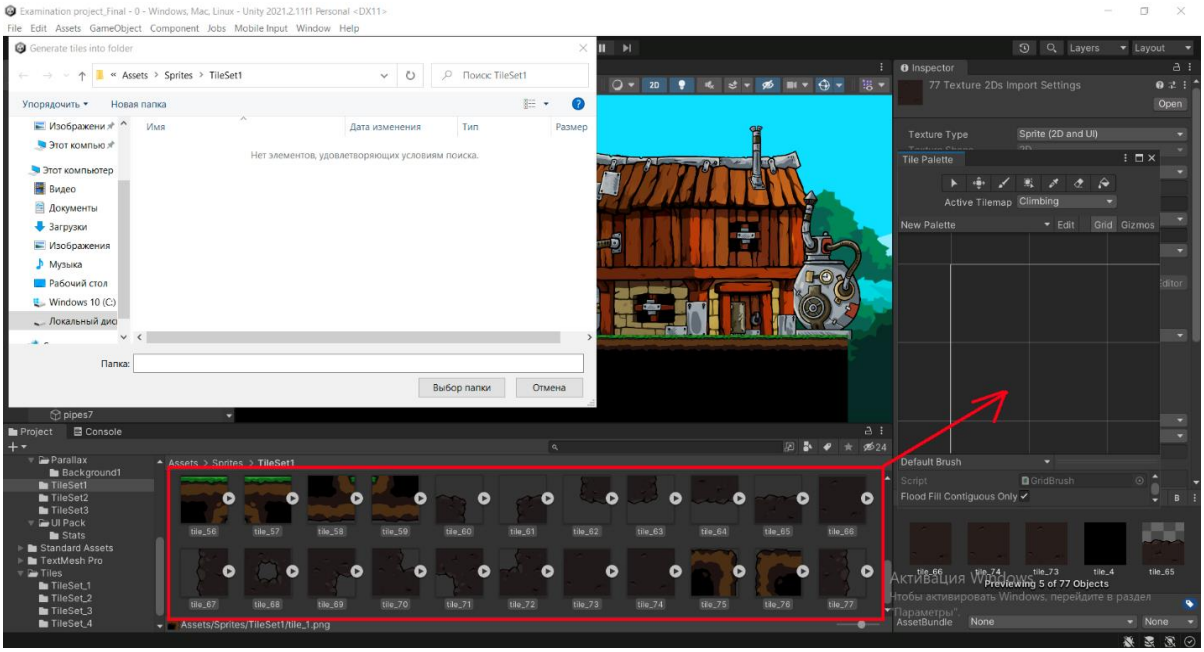

Рис. 2.16 Перенесення тайлів на палітру.

Далі обираємо потрібну папку, де ми збережемо тайли з посиланням для палітри. Після цього отримуємо нову палітру тайлів, за допомогою якої можна

вже створювати рівні. Далі необхідно створити Tilemap. Для цього створюємо 2D об’єкт, вибираємо Tilemap.

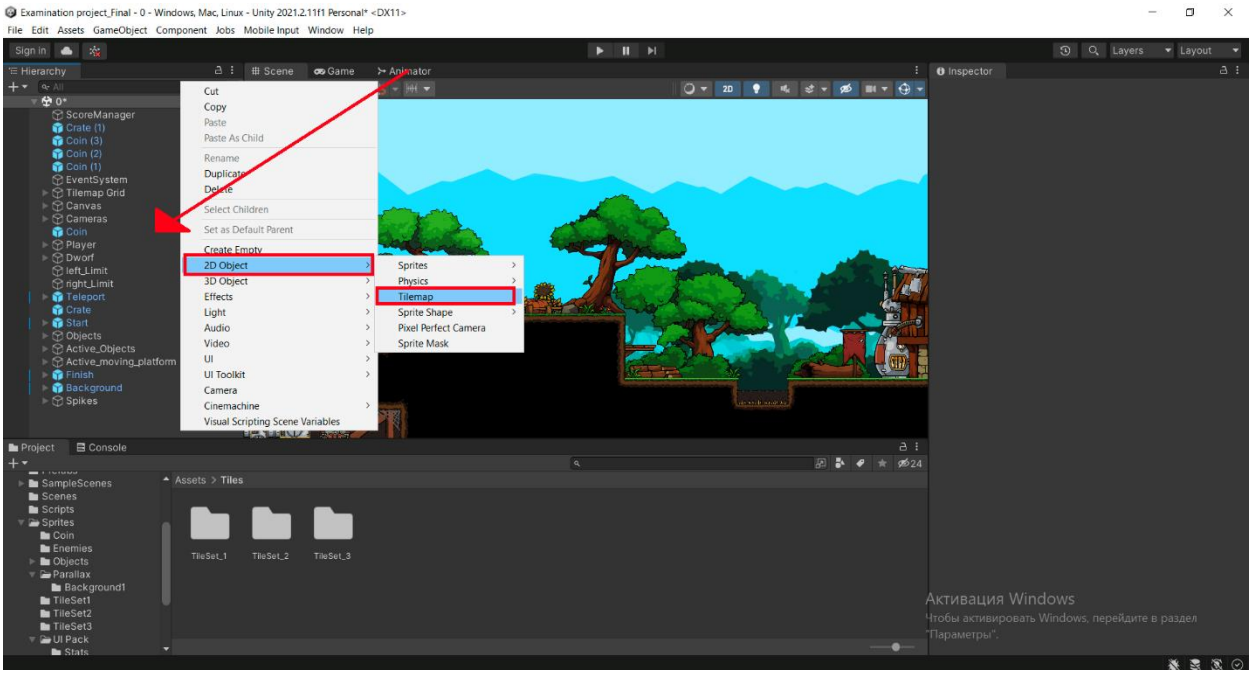

Рис. 2.17 Створення Tilemap.

Назвемо цей тайлмап «Background». У цьому тайлмапі буде створюватись геометрія рівня. Також до «Background» необхідно додати Rigidbody2D і 2 колайдери, а саме: Composite Collider 2D, Tilemap Collider 2D.

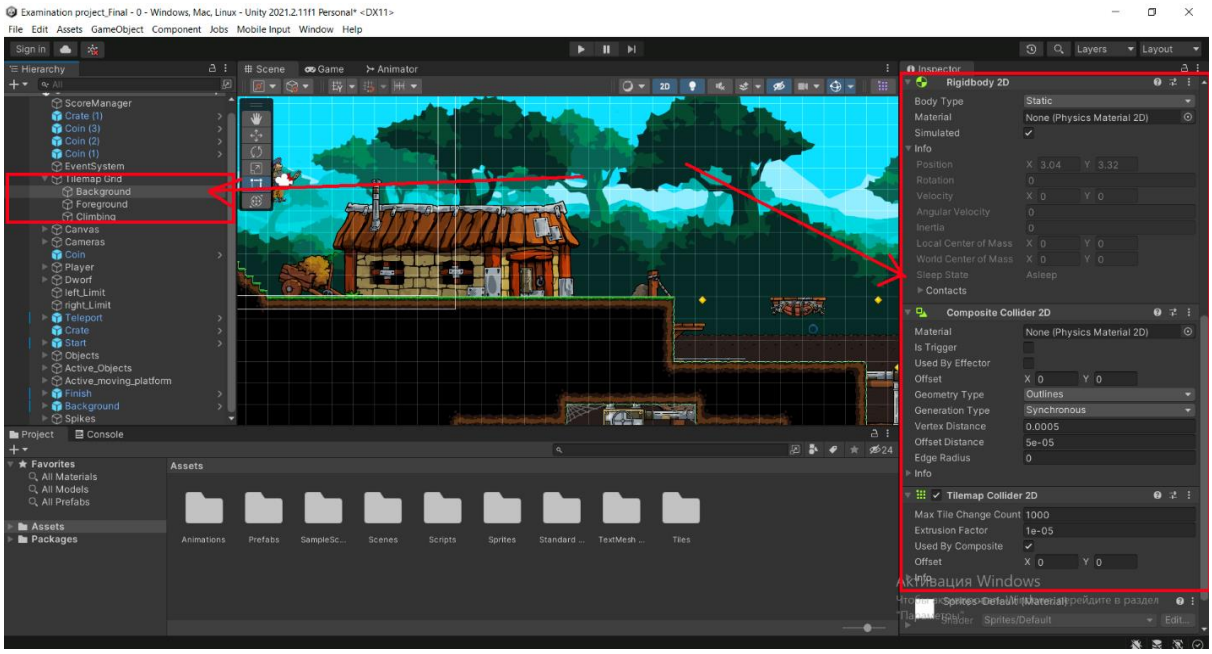

Рис. 2.18 Налаштування «Background»

Також необхідно за схожим принципом створити ще один тайлмап, під назвою «Foreground». У цьому тайлмапі буде створюватись задній фон у печерах, та закритих областях. Після створення Tilemap, можна приступити до роботи над створенням геометрії рівня. Відкриваємо палітру тайлів. В пункті активіні тайлмапи вибираємо «Background», після цього можемо починати формувати ландшафт. Обираємо потрібний для нас тайл, і відображаємо його на сцені.

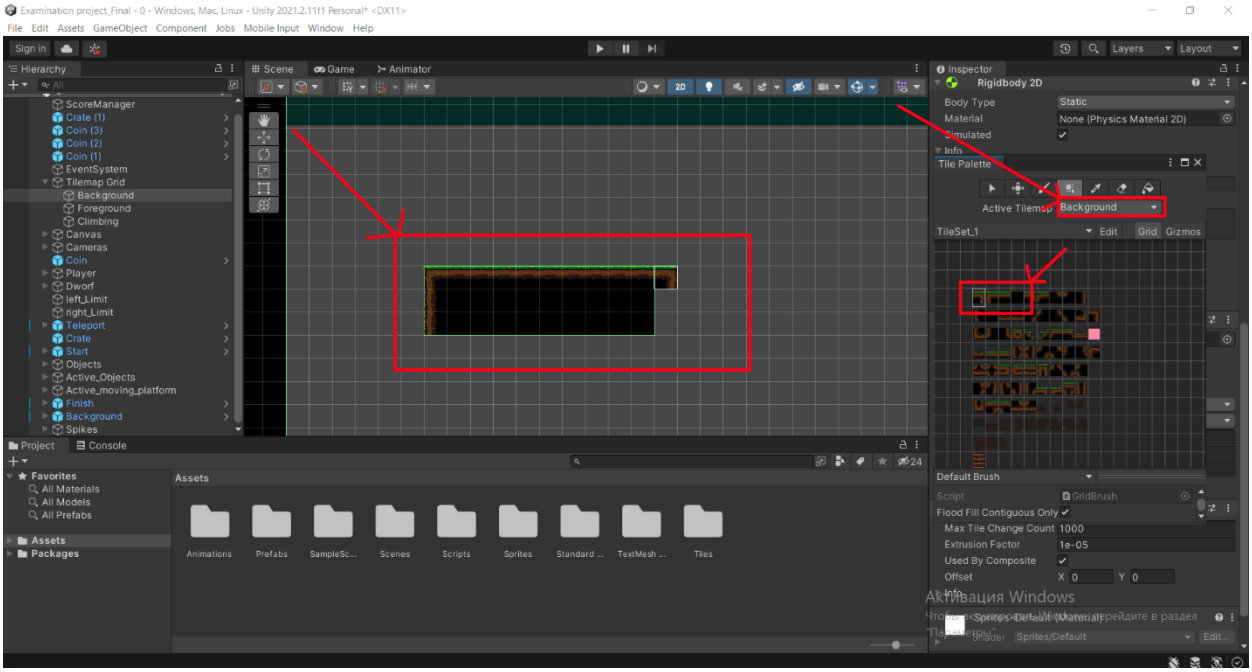

Рис. 2.19 Приклад використання Tile palette у тайлмапі «Background»

Аналогічно працюємо з «Foreground».

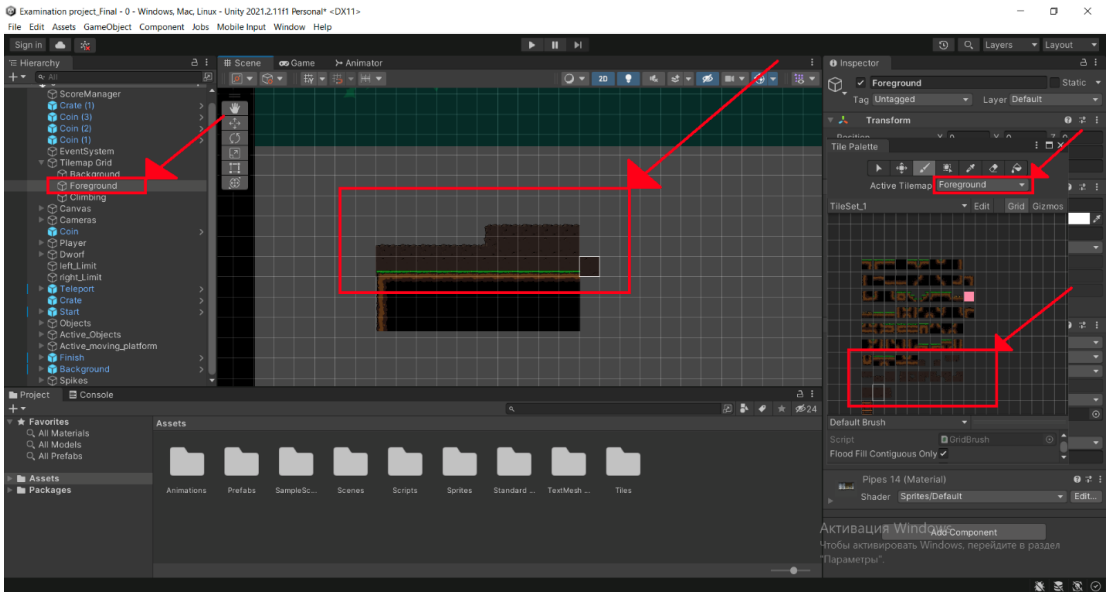

Рис. 2.20 Приклад використання Tile palette у тайлмапі «Foreground»

Але не всі елементи рівня відображаються за допомогою тайлів. Є ще об’єкти, які розташовуються на сцені для художнього збагачення картинки. Наприклад: будівлі, дерева, кущі, пучки трави, ящики.

Для того, щоб розмістити цей об’єкт на сцені достатньо перетягнути спрайт цього об’єкта.

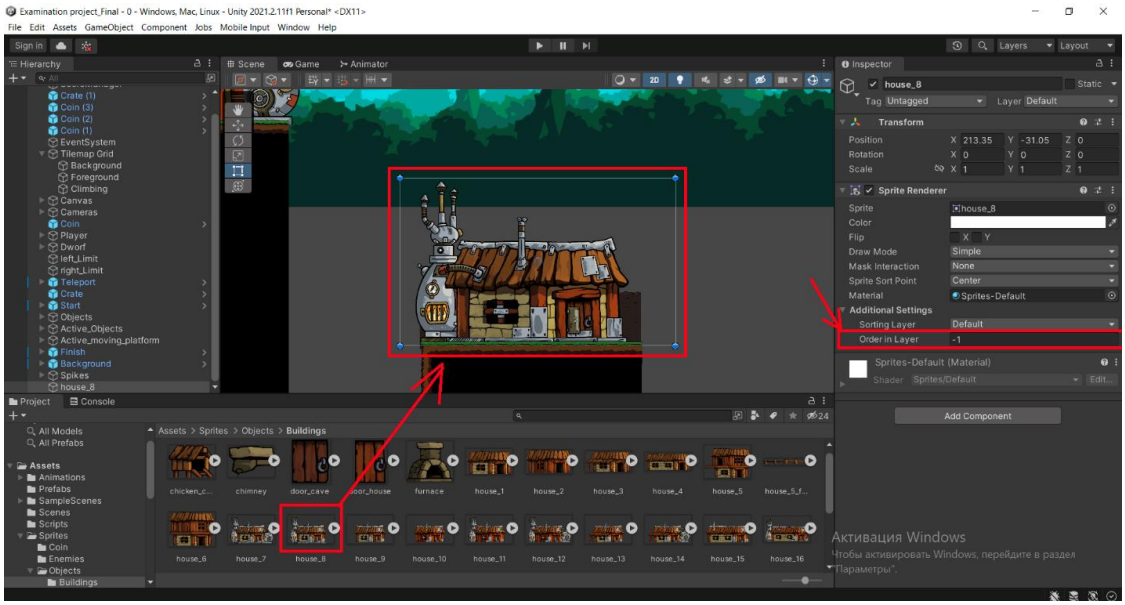

Рис. 2.21 Приклад розміщення об’єкта

2.3. Проектування ігрових механік

У грі реалізовано такі механіки:

- Переміщення героя
- Прижок героя
- Атака героя
- Смерть героя
- Систему здоров’я героя
- Патрулювання противника
- Смерть противника
- Атака противника
- Взаємодія з деякими проходами
- Руйнування ящиків

Ціль гравця, схожа майже на всі платформери, а саме дойти з точки А до точки В. Ускладнювати цю задачу для гравця будуть пастки, противники, його цікавість (відхилення від основного маршруту, заради додаткових бонусів).

Висновок до розділу 2

У даному розділі було розглянуто процес побудови концепції гри та проектування ігрових механік, а також описано процес створення покадрової анімації ігрових та не ігрових персонажів. Також був описаний процес роботи над створенням ландшафту та дизайном рівнів.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Так як ми реалізовуємо ігровий додаток, було вирішено використовувати ігровий рушій *«Unity 2021.2.11.f1»* у двовимірному просторі (2D). Для написання сценаріїв і опису логіки використовувалась мова програмування C# у програмному середовищі *«Visual Studio 2022»*.

Для створення спрайтів використовувався *«Adobe Photoshop CC 2019»* платної версії та графічний планшет Huion h640p. Також використовувались спрайти, які були куплені на офіційному сайті *«Unity Store»* БЕЗ авторського права, тобто ми могли видозмінювати чи перемальовувати.

Проект розроблявся на двух комп'ютерах:

Dream Machines

- Персональний комп'ютер з операційною системою Windows 10 64x;
- Процессор Intel Core i5-9 (2.4GHz);
- Оперативна пам'ять 8 GB;
- Відеокарта NVIDIA GeForce 1650 TI 6GB

HP Pavilion Gaming

- Персональний комп'ютер з операційною системою Windows 10 64x;
- Процесор Intel Core i5-8 (2.4GHz);
- Оперативна пам'ять 8 GB;
- Відеокарта NVIDIA GeForce GTX 1050 2GB

3.2. Опис програмної реалізації

3.2.1 Створення спрайт-анімацій головного героя/ворога (Sprite Animations)

Слово ***спрайт*** (анг. *sprite*) являє собою нерухому картинку. Отже, ***спрайт-анімаціями*** (анг. sprite animations) називають не рухомі картинки, які збираються в анімаційний фрагмент, який відтворюватиметься по кадрово, щоб створити анімацію. Чим більше спрайтів в анімаційному кліпі, тим краще та чіткіше відтворення даної анімації.

Рис. 3.1 Спрайт

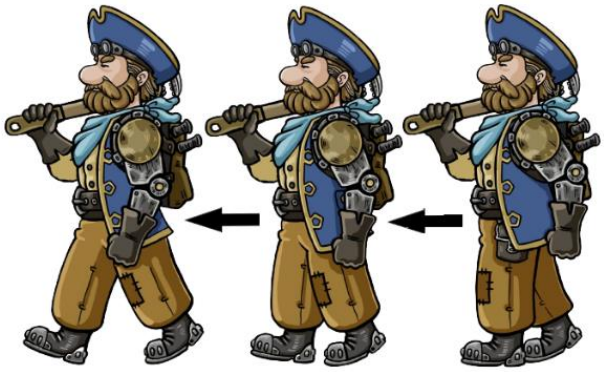

Рис. 3.2 Спрайт-анімація

Також до нашого об’єкту потрібно додавати компонент, який буде репрезентувати персонажа - ***спрайт рендер*** (анг. *sprite renderer*). Він дозволяє показувати зображення на сцені у вигляді спрайтів.

Для створення анімації потрібен компонент – Animator. ***Аніматор*** (анг. Animator) – це компонент в Unity, який анімує рухи нашого персонажу і “диктує” йому коли він буде бігати, стрибати чи ходити.

У полі *Controller* ми призначаємо об’єкт головного героя. Контролер аніматора дозволяє нам створювати взаємодії між анімаціями (бігу, стрибків, ходьби і т.д.) Всі ці анімації об’єднуються в єдину ієрархію. Цей компонент може бути присутній у любого об’єкту, якому ми створюємо анімації, у нашому випадку це об’єкти: ***Player*** та ***Enemy***. У кожного з цих об’єктів свій аніматор, зі своїми анімаціями.

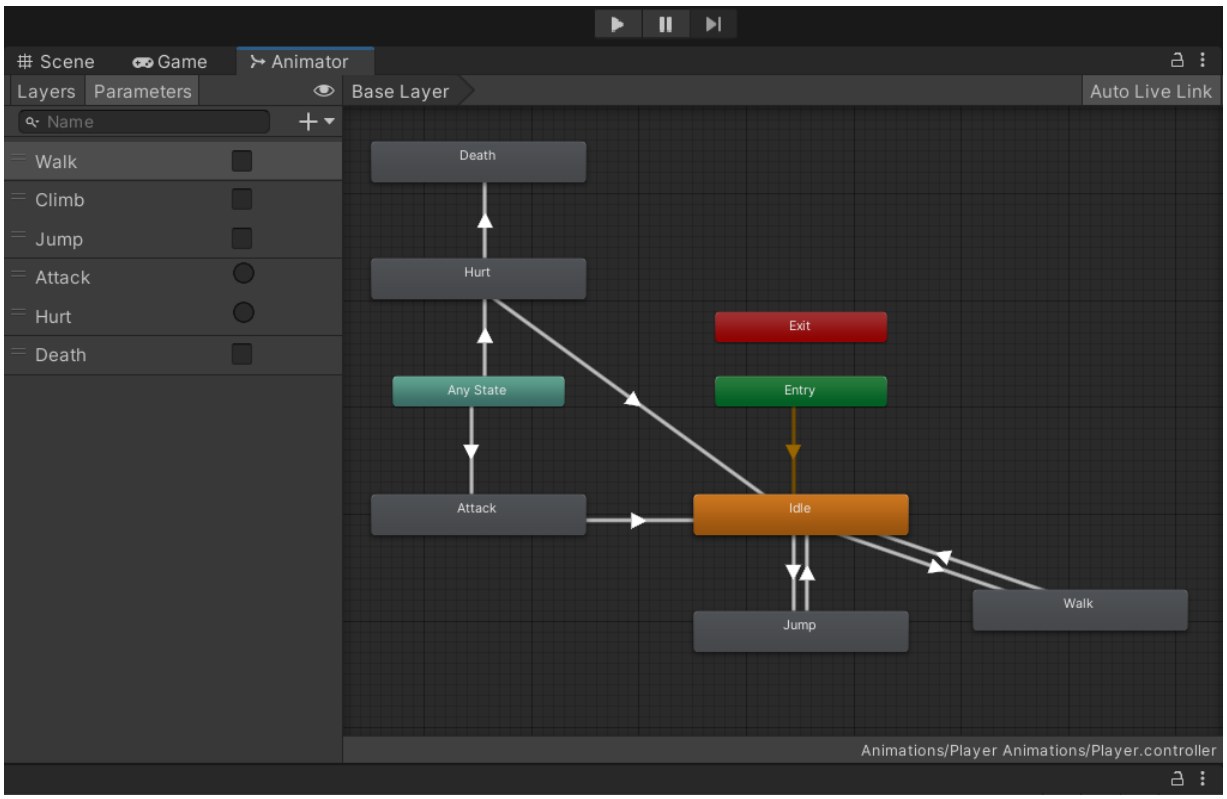

Рис. 3.3 Візуальне відображення анімацій головного героя у інструменті *Animator*

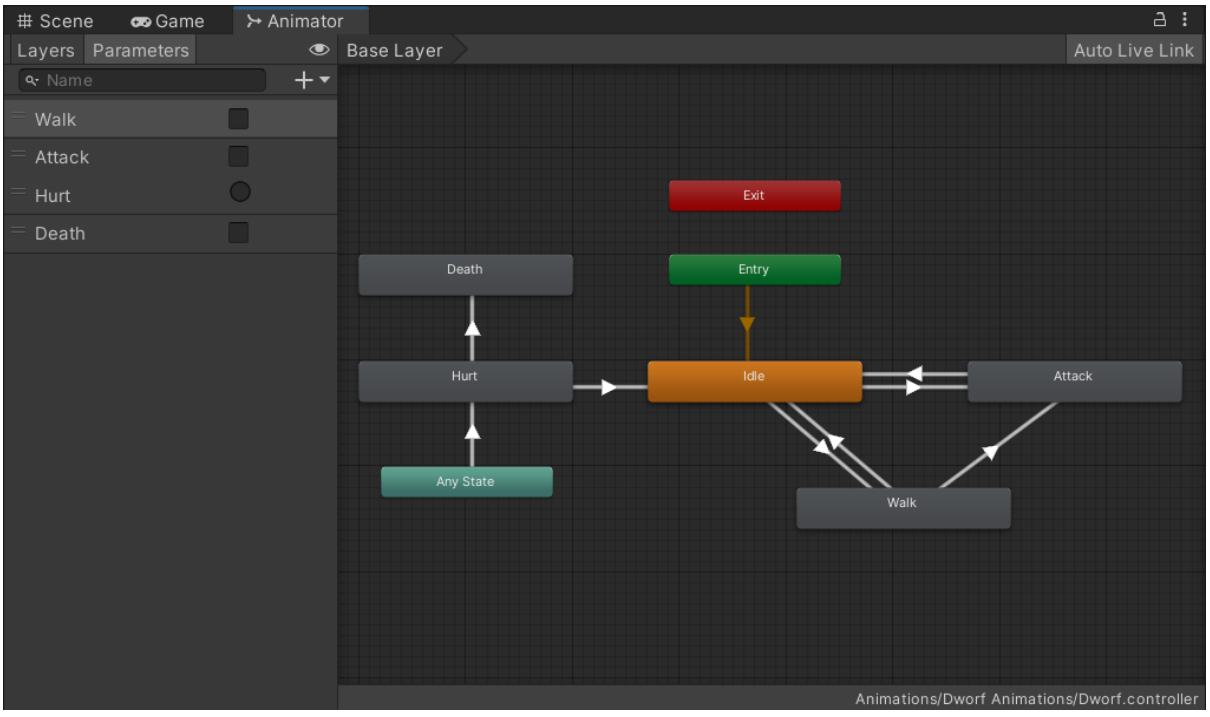

Рис. 3.4 Візуальне відображення анімацій ворога у інструменті *Animator*

Всі наші анімації зберігаються тут, готові для того, щоб мати можливість виконувати структурні взаємодії між ними. Кожен стан містить

індивідуальну анімаційну послідовність (анг. *blend tree*), яка програватиметься, коли персонаж перебуває у цьому стані.

Щоб створити будь яку анімацію, нам потрібно зайти у вкладку - Анімація (Animation), вибрати об’єкт, якому хочемо створити анімацію і натискаємо кнопку – Create New Clip. Для створення анімації використовуємо спрайти, які автоматично додаються в ієрархію аніматора.

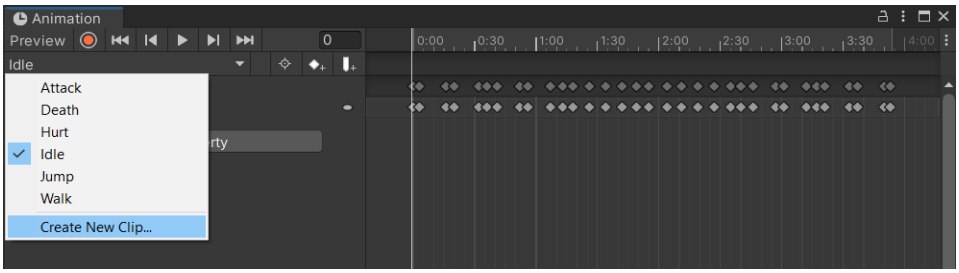

Рис. 3.5 Редактор анімації

Якщо ми подивимось на Рис. 3.3-3.4, то видно що Аніматори об’єктів мають певні параметри. В Unity існують різні типи параметрів (*float, int, bool* і *trigger*), і всі вони використовуються для того, щоб ми могли отримати доступ до станів (наших анімацій) через скрипти. Наприклад розглянемо зв’язок між станом «спокою» та станом «ходьби».

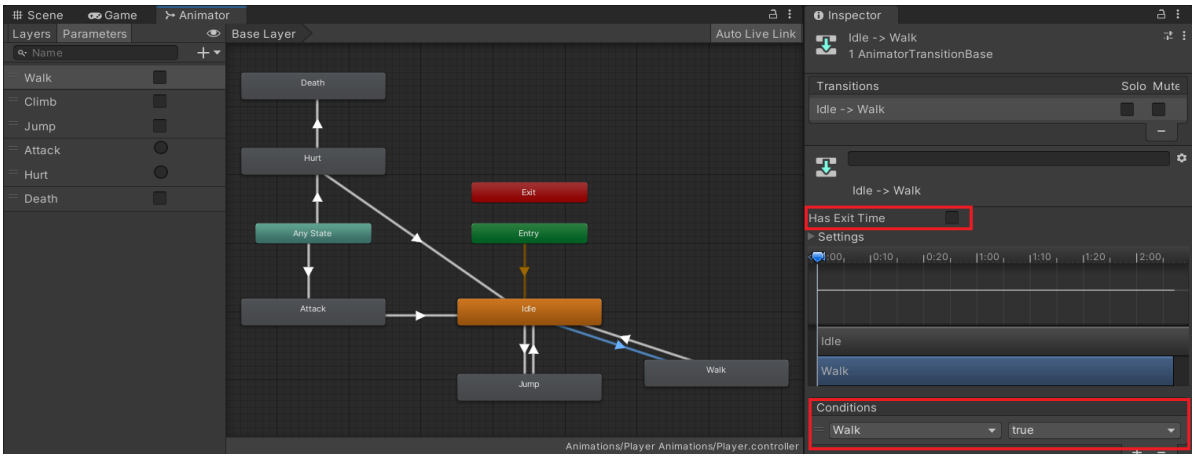

Рис. 3.6 Інструмент Animator

Тут ми використовуємо параметр «Walk», типу *bool*. Ми вказуємо аніматору перевіряти певні параметри за сценарієм «правда чи неправда (True or False)». Якщо всі умови дотримані, то перехід між анімаціями відтвориться (умови задаємо у скрипті).

Виділимо параметр Has Exit Time, який надалі будемо

використовувати. Налаштування цього параметру прості. Він дає можливість запускати наступну дію, не чекаючи завершення попередньої.

3.2.2 Рух персонажу та ворога/зв'язка з анімаціями

У грі ми повинні керувати персонажем за допомогою клавіатури, тому у подальшій роботі ми будемо застосовувати - Сценарії (анг. scripts), або їх ще називають – скриптами.

Сценарії (анг. *scripts*) – це послідовність дій, які можна застосувати до об'єктів в Unity, які описані за допомогою скриптової мови програмування, які виконуються під час гри.

Але перед тим як створювати сценарій для наших героїв, нам потрібно додати їм фізичні властивості та додати форму об'єкту, так як зараз це тільки набір спрайтів на сцені. Для цього будемо використовувати Rigidbody2D та Collider2D.

Rigidbody2D – компонент, який надає фізичні властивості для об'єкта. Якщо цей компонент доданий для нашого герою, він може взаємодіяти, стикатися з іншими фізичними об'єктами на сцені.

Collider2D – компонент, який визначає форму об'єкта.

Отож перейдемо до пояснення кожної функції головного героя. На Рис.1.4 можна бачити, що у головного героя є певні функції, які він може виконувати, тобто: ходьба (Walk), стрибок(Jump), атака(Attack), отримання шкоди(Hurt), спокій(Idle), смерть(Death).

```
//--------------------Walk--------------------//
ссылка: 1
private void Walk()
{
    float controlThrow = CrossPlatformInputManager.GetAxis("Horizontal");
    Vector2 playerVelocity = new Vector2(controlThrow * runSpeed, myRigidBody.velocity.y);
    myRigidBody.velocity = playerVelocity;

    bool playerHasHorizontalSpeed = Mathf.Abs(myRigidBody.velocity.x) > Mathf.Epsilon;
    myAnimator.SetBool("Walk", playerHasHorizontalSpeed);


    isMoving = playerHasHorizontalSpeed;
}
```

Рис. 3.7 Метод *Walk*

```
//--------------------------Jump-----------------------------//
ссылка: 1
private void Jump()
{
    bool playerHasVerticalSpeed = Mathf.Abs(myRigidBody.velocity.y) > Mathf.Epsilon;

    if (myFeet.IsTouchingLayers(LayerMask.GetMask("Ground")))
    {
        myAnimator.SetBool("Jump", playerHasVerticalSpeed);
    }

    if (!myFeet.IsTouchingLayers(LayerMask.GetMask("Ground")))
    {
        return;
    }

    if (CrossPlatformInputManager.GetButtonDown("Jump"))
    {
        Vector2 jumpVelocityToAdd = new Vector2(0f, jumpSpeed);
        myRigidBody.velocity += jumpVelocityToAdd;
        myAnimator.SetBool("Jump", playerHasVerticalSpeed);
    }
}
```

Рис. 3.8 Метод *Jump*

```
if (Time.time >= nextAttackTime)
{
    if (Input.GetKeyDown(KeyCode.LeftShift))
    {
        Attack();
        nextAttackTime = Time.time + 1f / attackRate;
    }
}

void Attack()
{
    animator.SetTrigger("Attack");
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);


    foreach (Collider2D enemy in hitEnemies)
    {
        //Debug.Log("Attack animation is played");
        enemy.GetComponent<Enemy_Controller>().TakeDamage(attackDamage);
        enemy.GetComponent<Crate_Break>().TakeDamage(attackDamage);
    }
}
```

Рис.3.9 Метод *Attack*

```
ссылка: 1
void Death()
{
    Debug.Log("Player died!");
    animator.SetBool("Death", true);

    GetComponent<Collider2D>().enabled = false;
    this.enabled = false;

    Destroy(gameObject, 1);
}
//------------------------------------------------------------//
```

Рис. 3.10 Метод *Death*

```
//--------------------TakeDamage/Death--------------------//
Ссылок: 2
public void TakeDamage(int damage)
{
    currentHealth -= damage;

    animator.SetTrigger("Hurt");

    if (currentHealth <= 0)
    {
        Death();
    }
    health_Bar.SetHealth(currentHealth);
}
```

Рис. 3.11 Метод *Hurt*

Важливо відзначити, як у сценаріях ми отримуємо доступ до параметрів з контролера аніматора. У коді ми використовуємо параметр *Animator* і отримуємо доступ до його параметрів (*float*, *int*, *bool* і *trigger*) у нашому об’єкті, щоб надалі при програванні, наприклад функції стрибку, ми могли посилатися саме на анімацію стрибку у аніматорі.

Отож перейдемо до пояснення кожної функції ворога. На Рис 3.5 можна бачити, що ворога є певні функції, які він може виконувати, тобто: ходьба (Walk), атака(Attack), отримання шкоди(Hurt), спокій(Idle), смерть(Death).

Якщо у головного героя ми використовували управління власноруч, то для ворога нам потрібно зробити ігрову логіку, щоб він міг сам пересуватися по заданим координатам. Все це реалізовано у сценарії – *EnemyLogic*.

```
void EnemyLogic()
{
    distance = Vector2.Distance(transform.position, target.position);


    Debug.Log(target.gameObject.name);

    if (distance > attackDistance)
    {
        StopAttack();
    }
    else if (attackDistance > distance && cooling == false)
    {
        Attack();
    }

    if (cooling)
    {
        Cooldown();
        anim.SetBool("Attack", false);
    }
}
```

Рис. 3.12 Метод *EnemyLogic*

```
void Update()
{
    if (!attackMode)
    {
        Move();
    }

    if (!InsideOfLimits() && !inRange && !anim.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
    {
        SelectTarget();
    }

    if (inRange)
    {
         EnemyLogic();
    }
}
```

Рис.3.13 Метод *Attack*

```
//--------------------Move--------------------//
ссылка: 1
void Move()
{
    anim.SetBool("Walk", true);

    if (!anim.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
    {
        Vector2 targetPosition = new Vector2(target.position.x, transform.position.y);

        transform.position = Vector2.MoveTowards(transform.position, targetPosition, moveSpeed * Time.deltaTime);
    }
}
```

Рис.3.14 Метод *Move*

```
//-----------Attack/Cooldown/Coolingtrigger-----------//
ссылка: 1
void Attack()
{
    timer = intTimer; //Скидання таймера, коли гравець входить у діапазон атаки
    attackMode = true; // Щоб перевірити, чи може ворог все ще атакувати чи ні

    anim.SetBool("Walk", false);
    anim.SetBool("Attack", true);


    Collider2D[] hitPlayer = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);
    foreach (Collider2D player in hitPlayer)
    {
        player.GetComponent<PlayerController>().TakeDamage(attackDamage);
    }
}
```

Рис. 3.15 Метод *Attack*

```
void Cooldown()
{
    timer -= Time.deltaTime;

    if (timer <= 0 && cooling && attackMode)
    {
        cooling = false;
        timer = intTimer;
    }
}

ссылка: 1
void StopAttack()
{
    cooling = false;
    attackMode = false;
    anim.SetBool("Attack", false);
}

Ссылок: 0
public void TriggerCooling()
{
    cooling = true;
}
```

Рис. 3.16 Методи *Cooldown/StopAttack/TriggerCooling*

```
public void TakeDamage(int damage)
{
    Debug.Log("Damage is taken");
    currentHealth -= damage;

    animator.SetTrigger("Hurt");

    if (currentHealth <= 0)
    {
        Death();
    }
}
```

Рис. 3.17 Методи *TakeDamage*

```
void Death()
{
    Debug.Log("Enemy died!");

    animator.SetBool("Death", true);


    GetComponent<Collider2D>().enabled = false;
    this.enabled = false;

    Destroy(gameObject, 1);

}
```

Рис. 3.18 Методи *Death*

Зв’язка заключається у тому, що у методах ми посилаємось на *Animator* того, чи іншого об'єкта, який у свою чергу посилається саме на ту анімацію, яку ми вказали. В аніматорі використовуються параметри типу (*float*, *int*, *bool*, *trigger*) з певними властивостями, які ми використовуємо для зв’язку між анімаціями. Наприклад, на Рис.1.18 використовується посилання на анімацію «Death» з значенням *bool***.** Тобто якщо виконується метод *Death*, то безпосередньо з ним виконується і анімація «Death».

3.2.3 Головна камера

У рушії Unity камери є пристроями, які захоплюють та відображають світ гравцю. Можна зробити камеру статичною, яка охоплює всю сцену, або зробити камеру дочірньою по відношенню до гравця та розмістити її на рівні очей персонажа. Але для нашої гри потрібно закріпити камеру за гравцем і змусити її слідувати за ним. Для цього в Unity використовується набір інструментів - *Cinemachine*.

Cinemachine - набір компонентів, який дозволяє мати кілька віртуальних камер для однієї сцени. Без Cinemachine розробник чи оператор міг би місяцями встановлювати та налаштовувати дані типи кадрів вручну.

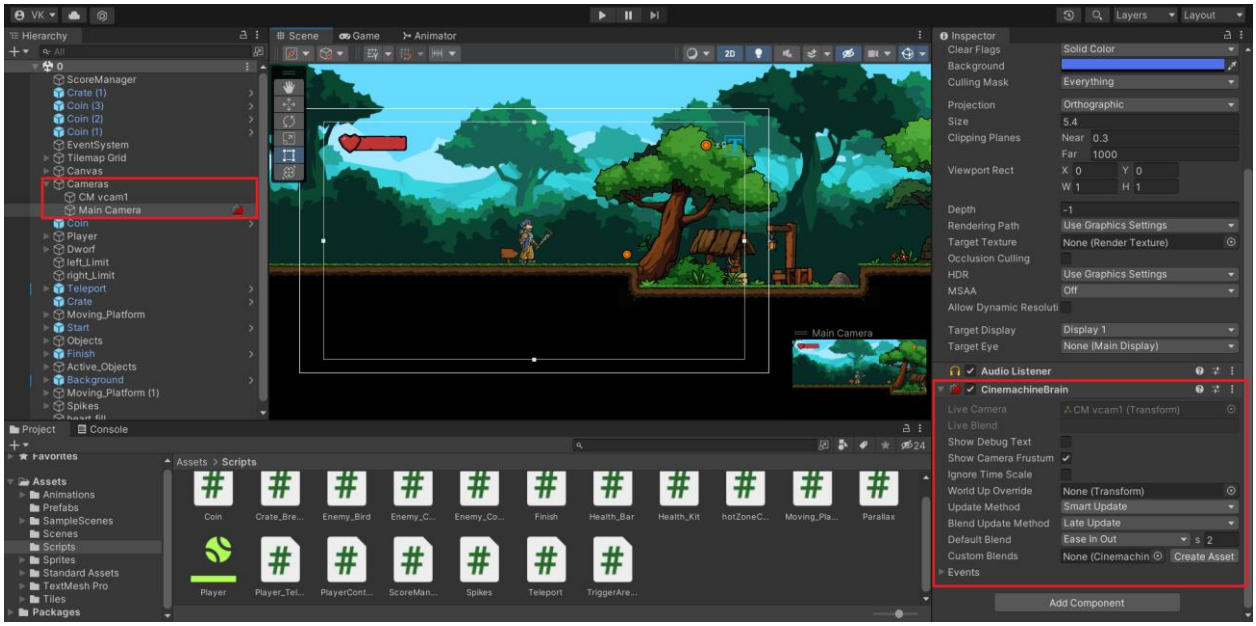

Рис. 3.19 Відображення ***Cinemachine*** у головного героя

Створення віртуальної камери в сцені створить компонент під назвою *Cinemachine Brain*. Це реальний зв'язок між головною та віртуальною камерами у сцені. Cinemachine Brain відстежуватиме активну зараз віртуальну камеру та застосовувати її стан до камери Unity. У цьому компоненті є властивість *Follow*, у якому стоїть наш об'єкт *Player*. Unity посилається на компонент *Transform* з об'єкта *Player* і використовує Transform як мету Target

для проходження.

3.2.4 Шкала здоров’я

Перш ніж ми зможемо створити панель здоров’я, нам потрібно спочатку створити очки здоров’я. Наприклад, якщо на даний момент гравець має 50% здоров’я, наша панель здоров’я повинна розрахувати його на основі цих даних і відображати половинну смугу здоров’я.

```
public Health_Bar health_Bar;
public int maxHealth = 100;
int currentHealth;
```

Рис. 3.20 Очки здоров’я головного героя

У наведеному вище коді ми створюємо набір функцій, які максимально ініціалізують значення поточного здоров’я гравця. Після того, як налаштували очки здоров’я гравця, створюємо інтерфейс користувача (анг. Canvas), у якому буде створений об’єкт інтерфейсу - Image (зображення), який буде виступати за картинку шкали здоров’я.

Інтерфейс користувача Unity - це набір інструментів для розробки інтерфейсу користувача для ігор і програм. Ця структура заснована на об’єктах гри, які впорядковують, розташовують і стилюють інтерфейси користувача за допомогою компонентів і перегляду гри.

Створюємо порожній об’єкт всередині ігрового об’єкта Canvas і змінюємо його розмір до того ж розміру, що й *Image*. Змінимо колір цього зображення на червоний, оскільки цей об’єкт буде представляти собою фактичну смугу точки здоров’я гравця.

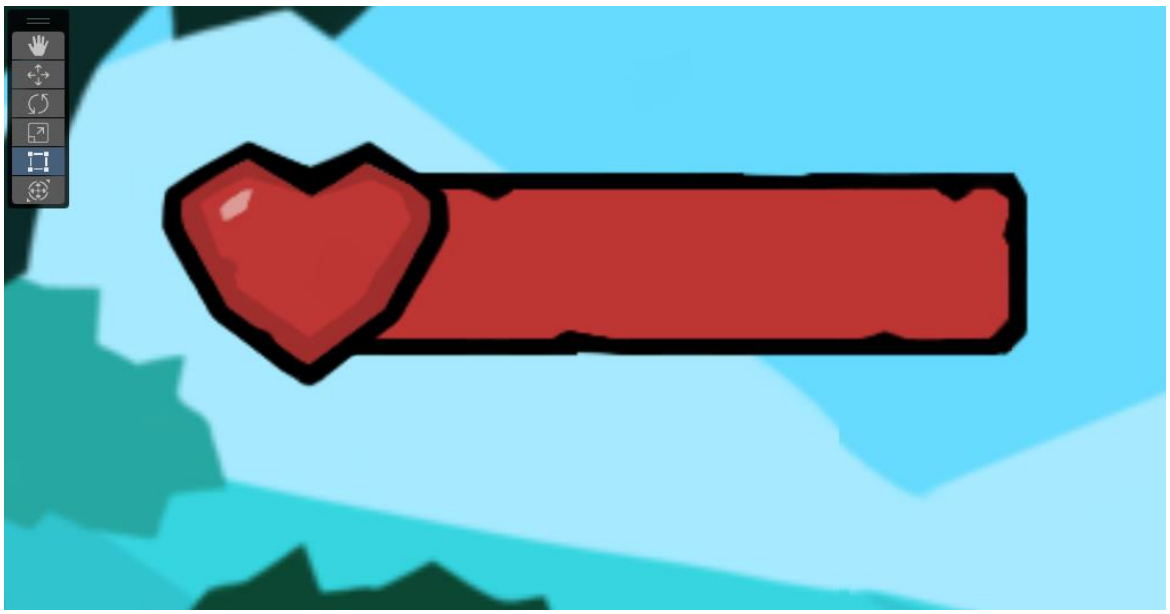

Рис. 3.21 Відображення шкали здоров’я у головного героя

Для того щоб панель здоров’я зменшувалась або збільшувалась, використовується компонент - ***Slider*** та сценарій для шкали здоров’я.

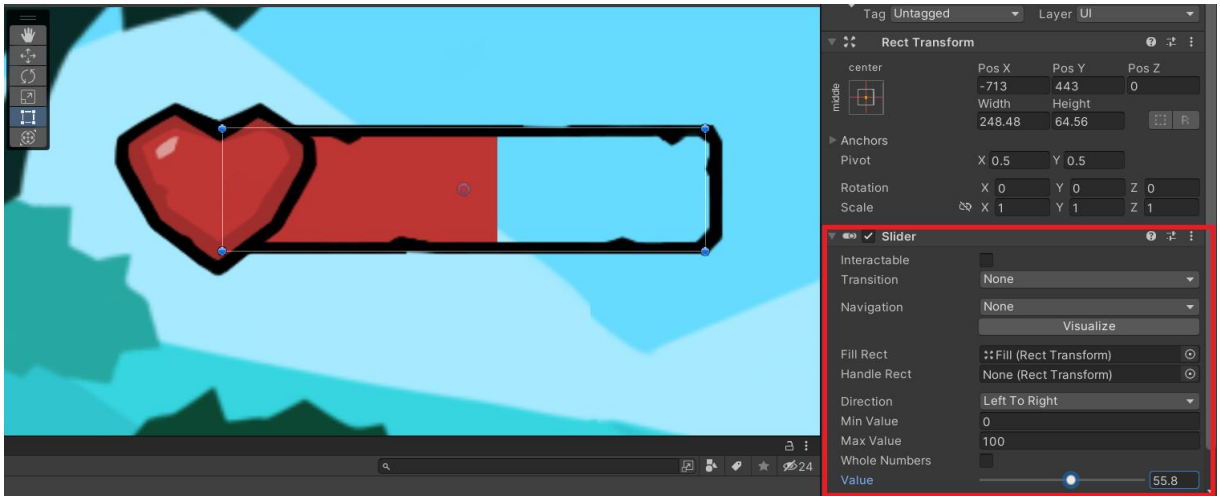

Рис. 3.22 Компонент *Slider*

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


public class Health_Bar : MonoBehaviour
{
    public Slider slider;


    public void SetMaxHealth(int health)
    {
        slider.maxValue = health;
        slider.value = health;
    }



    public void SetHealth(int health)
    {
        slider.value = health;
    }
}
```

Рис. 3.23 Сценарій шкали здоров'я

3.2.5 Збір монет та шкала рахунку

Спочатку потрібно створити збірні елементи (coins), які головний герой буде збирати. В ієрархії проекту додамо новий об'єкт з компонентами: ***Sprite Renderer***, ***Box Collider 2D***, ***Rigidbody 2D*** та ***Animator*** на панель Inspector. Вони будуть відповідати за відтворення графіки, анімації, а також за фіксацію фізичних зіткнень між персонажем та предметом колекціонування.

Сценарій для монет, заключається у тому, що як тільки персонаж торкається чітко вираженого спрайта, що представляє предмет, гравець отримує якусь «перевагу». В проекті це реалізовано через *теги (анг. tag)*.

Тег (анг. tag) - це слово, яке ви посилаєте на один або кілька об'єктів гри. Наприклад, ви можете визначити тег «Coin» для персонажів, керованих гравцем, який може бути призначений для предметів, які гравець може збирати на Сцені.

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

Скрипт Unity (ссылок на ресурсы: 4) | Ссылок: 0
public class Coin : MonoBehaviour
{
    public int coinValue = 1;

    Сообщение Unity | Ссылок: 0
    private void OnTriggerEnter2D(Collider2D coin)
    {
        if (coin.gameObject.CompareTag("Player"))
        {
            ScoreManager.instance.ChangeScore(coinValue);
        }
    }
}
```

Рис. 3.24 Сценарій монети

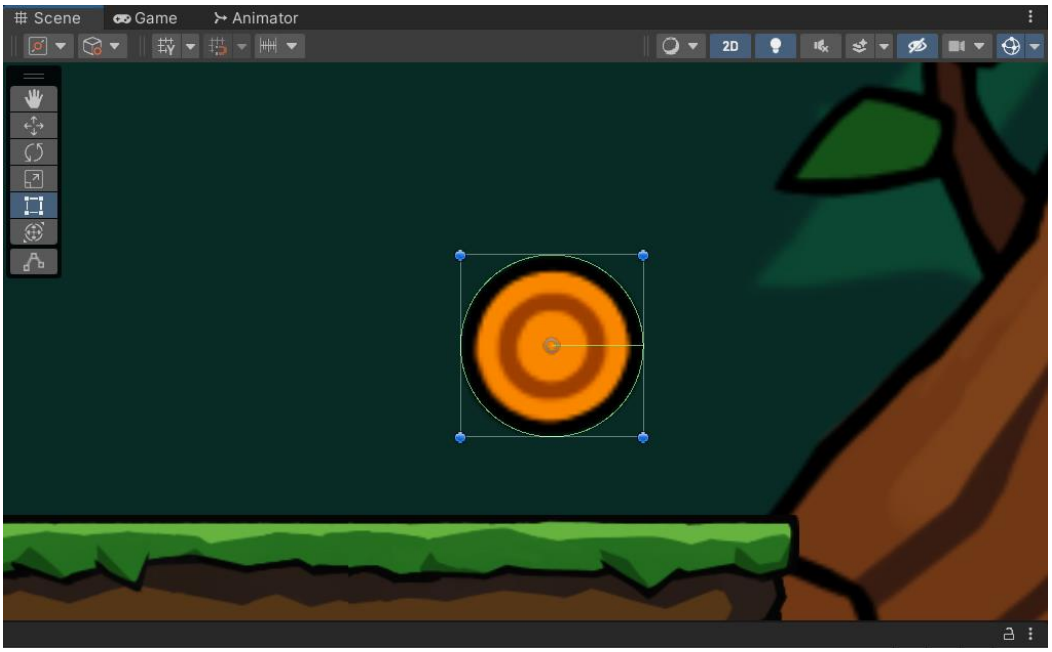

Рис. 3.25 Відображення монети на сцені

Створюємо текстовий об’єкт в *інтерфейсі користувача* (в компоненті - *Canvas*), який буде використовуватись як лічильник.

```
public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance;
    public TMP_Text text;
    int score;


    Сообщение Unity | Ссылок: 0
    void Start()
    {
        if(instance == null)
        {
            instance = this;
        }
    }

    ссылка: 1
    public void ChangeScore(int coinValue)
    {
        score += coinValue;
        text.text = "x" + score.ToString();
    }
}
```

Рис. 3.26 Сценарій шкали рахунку

Рис. 3.27 Інтерфейс гри

3.2.6 Перехід між рівнями (сценами)

Після створення різних сцен варто звернути увагу на відсутності нормального завершення рівня. Для цього створюється об'єкт «Finish».
Цей об’єкт може бути реалізований як невидима стіна з тригером, як об'єкт з тригером, який спрацьовує при взаємодії чи дотику, як невидима зона, заходячи в яку спрацьовує тригер. У проекті використовується варіант з невидимою стіною з тригером.

Для реалізації об'єкта «Finish», необхідно створити пустий об'єкт, який назвемо «Finish». Далі необхідно перемістити сам об’єкт на кінець рівня. Також до об’єкта додаємо «Box Collider 2D», а також в колайдері вказуємо галочку, що це тригер.

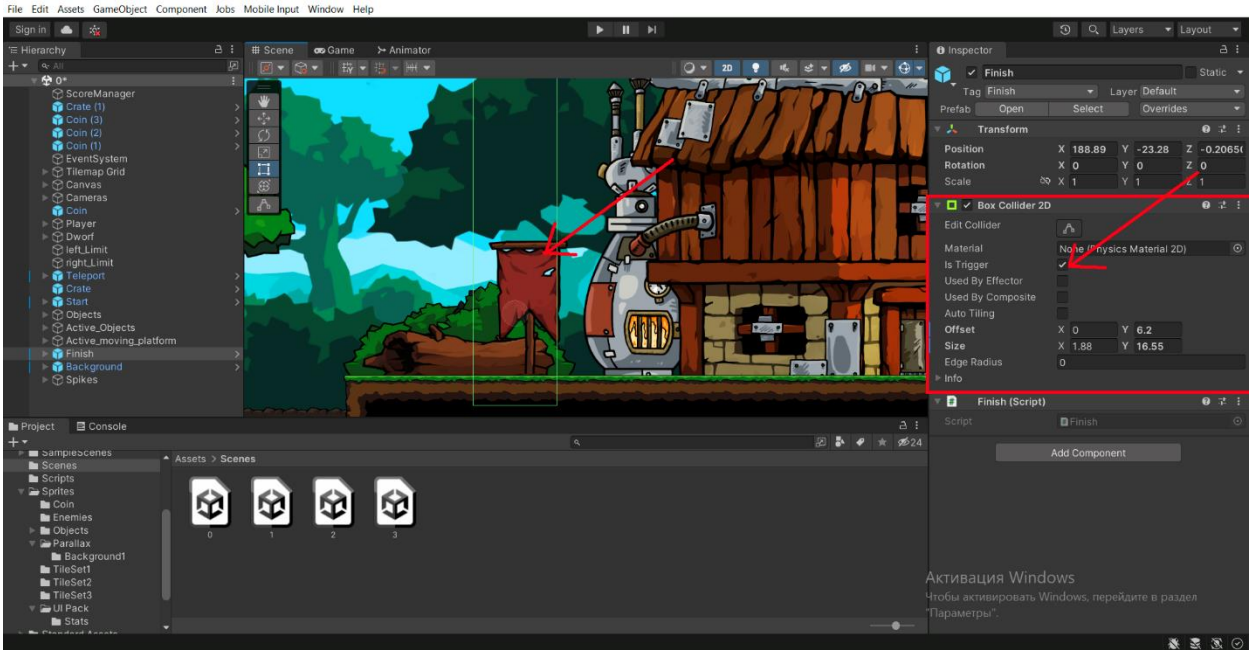

Рис. 3.28 Налаштування «Box Collider 2D» у об’єкті «Finish».

Далі необхідно надати індекси для всіх наших сцен. Для цього набираємо комбінацію клавіш Ctrl + Shift + B. Відкривається панель «Build Settings». Для того, щоб надати індекс для першої сцени необхідно відкрити дану сцену, а після цього натиснути клавішу «Add Open Scenes», ця клавіша додасть відкриту сцену до списку і надасть порядковий індекс.

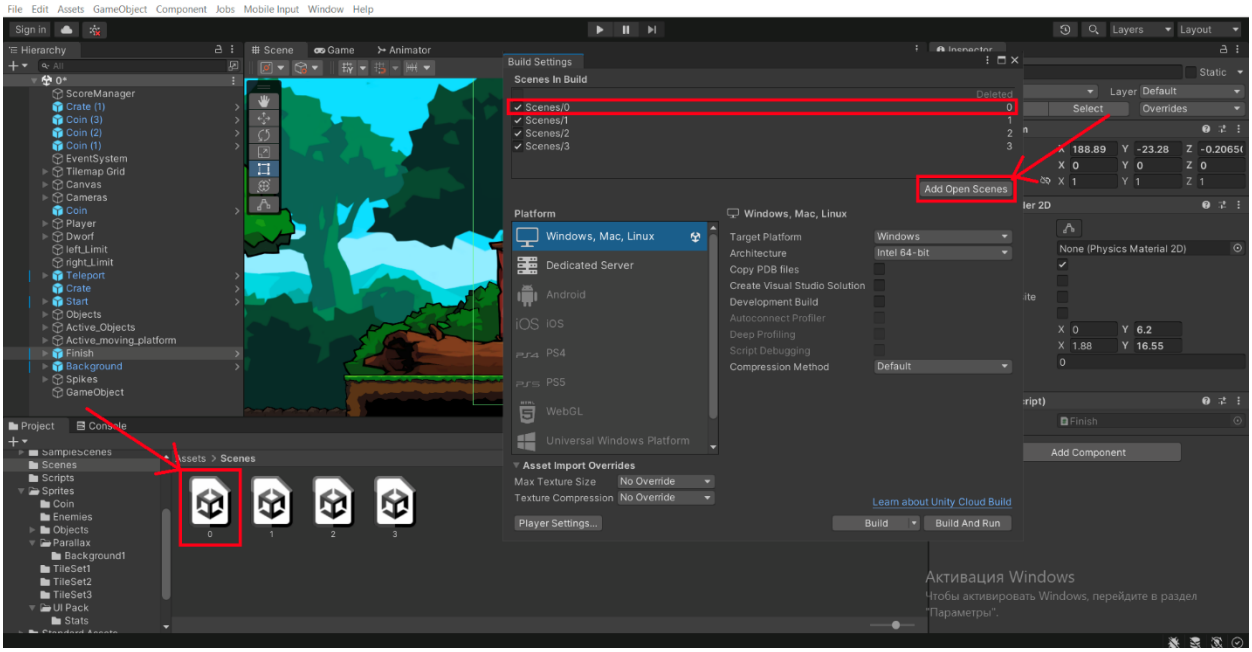

Рис. 3.29 Надання індекса сцені

Після цього, об’єкту «Finish» додається скрипт, який буде відповідати за виконання переходу від однієї сцени до іншої, скрипт також можемо назвати «Finish».

Скрипт має такий вигляд:

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

public class Finish : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            if (SceneManager.GetActiveScene().buildIndex < 3)
            {
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
            }
            else
            {
                SceneManager.LoadScene(0);
            }
        }
    }
}
```

Рис. 3.30 Код скрипта «Finish»

Якщо коротко, то даний код можна розтлумачити так: «Якщо гравець

стикається з даним об’єктом, то запускається така умова, а саме якщо номер активної сцени менший останньої (в даному випадку 3), то менеджер сцени завантажить наступну сцену, в іншому випадку завантажується перша сцена, зазвичай це меню. Також об’єкт «Finish» можна додати в папку Prefabs, а потім розмістити на інших сценах.

Висновок до розділу 3

У даному розділі було розглянуто програмне та технічне забезпечення, а також описано процес реалізації ігрових механік, таких як: ходьба, стрибок, атака, отримання шкоди головного героя. Також була описана ігрова логіка для ворога, який має можливість - патрулювання по заданим координатам, атаки головного героя, як тільки той з’явиться у його полі зору. Створили анімації та підв’язали їх до наших сценаріїв.

Була проблема у написанні ігрової логіки для ворога. Вона заключалась у тому, що ворог не міг атакувати героя в спину, через те що спрайт був неправильно нарізаний в графічному редакторі *Adobe Photoshop* і через це, ігровий рушій *Unity* неправильно виставив центральну точку на спрайті (не посередині зображення). Цю проблему ми вирішили, перемалювавши спрайти за потрібним розміром, та власноруч виставили центральні точки на зображеннях.

ВИСНОВКИ

У ході цього кваліфікаційного проекту було досліджено особливості створення відеоігор з двовимірною графікою за допомогою ігрового рушія Unity для операційної системи Android та на його основі було розроблено ігровий застосунок. Найбільша перевага цього рушія в тому, що цей інструмент є один із найдоступніших інструментів для початківців.

Робота поділена на три розділи. У першому розділі було розглянуто історію відеоігор, такий жанр як «платформер», його конкуренти, ключові особливості та механіки. Також проаналізували що таке ігровий рушій *Unity* та його конкуренти.

У другому розділі було розглянуто вже концепцію проекту. Була проведена робота зі створенням спрайтів анімацій для героїв, та побудова дизайну для рівнів з використанням графічного редактору ***Adobe Photoshop.*** Також важливо зазначити що у проекті використовувались власно намальовані асети.

У третьому розділі була описана технічна реалізація проекту. Розповідалось про засоби розробки, які були використані для написання роботи. Для написання сценаріїв і опису ігрової логіки використовувалась мова програмування C# у програмному середовищі Visual Studio 2022.

Отож згідно всіх перечислених програм, ми маємо ігровий застосунок, який у подальшому ми плануємо розвивати як більш масштабний проект у комерційних цілях.

Для досягнення поставленої цілі нам необхідно було вирішити такі моменти:

- провести аналіз аналогічних ігор;

- провести аналіз та вибір засобів реалізації гри;
- описати концепцію гри;
- намалювати персонажів для гри
- спроектувати програмну систему;
- реалізувати гру;
- провести тестування реалізованої гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Огляд жанру – Паропанк. Стаття на порталі OpenGamer присвячена комплексному огляду жанру: http://www.opengamer.com.ua/ohljad-zhanru-paropank-na-vsih-parah/(дата звернення: 09.02.2022).
2. Жанри: Стімпанк. Стаття в журналі Світ Фантастики: https://www.mirf.ru/worlds/steampunk-chto-eto-takoe(дата звернення: 09.02.2022).
3. Ernest Adams, Andrew Rollings. Fundamentals of Game Design. – 1st edition. – Prentice Hall, 2006. – 600 p. – ISBN 0131687476. – ISBN 978-0131687479.
4. Тимур Хорев. История жанра: боевики от первого лица // Лучшие компьютерные игры : журнал. – 2007. – Сентябрь (№ 9 (70)).
5. Evolution of Platformers (англ.) // Retro Gamer : journal. – Live Publishing, 2013. – No. 114. – P. 66-75.
6. Wolf M. J. P. Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming / Ed. by Mark J. P. Wolf. Santa Barbara, – CA: Greenwood, 2012. – 740 p.
7. Wolf, Mark J. P.The Video Game Explosion: A History from PONG to Playstation and Beyond Архівовано 18 квітня 2016 у Wayback Machine./ Mark J. P. Wolf. ABC-CLIO, 2008. – 380 p.
8. Донован Т. Грай! Історія відеоігор / Трістан Донован; пров. І. Вороніна. - М.: Біле Яблуко, 2014. - 648 с.
9. Bates, Bob. Game Design – 2nd. – Thomson Course Technology, 2004. – ISBN 1592004938.
10. Brathwaite, Brenda; Schreiber, Ian. Challenges for Game Designers – Charles River Media, 2009. – ISBN 158450580X.
11. Moore, Michael E.; Novak, Jeannie. Game Industry Career Guide – Delmar: Cengage Learning, 2010. – ISBN 978-1-4283-7647-2.
12. Oxland, Kevin. Gameplay and design – Addison Wesley, 2004. – ISBN