МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проект

здобувача освіти денної форми навчання
РП.07.18.000.ДП

РЯБОШАПКИ ДАРІЇ ЄВГЕНІВНИ

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка 2D-гри у жанрі платформеру на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 76 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах)

Дипломник _____ (Рябошапка Д.Є)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Іванченков В.С.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «27» 06 2024 р. Протокол ЕК № 3

Оцінка ЕК 5 (відмінно) 90 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 04 2024 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Рябошапці Дарії Євгенівні*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 2D-гри у жанрі платформеру на програмному рушії Unity*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-АД-ОД

2. Термін здачі закінченого проекту 10.06.2024

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Використання принципів ООП у проектуванні та розробці ігрових проектів; Реалізація основних ігрових механік комп'ютерної 2D-гри в жанрі платформер; Гра повинна мати основні признаки жанру 2D платформер; Гра повинна мати інтерфейс; Гравець повинен мати змогу пересуватись по ігровому простору та взаємодіяти з неігровими персонажами.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз ігрового жанру 2D платформер; Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом; Формування концепту ігрового процесу 2D-гри в жанрі платформер; Проектування основних елементів ігрового процесу та принципи їх роботи; Реалізація основних елементів ігрового процесу; Тестування працездатності ігрових елементів.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Особливості ігрового жанру 2D платформер; Особливості програмного рушія Unity та причини його вибору для розробки проекту; Особливості ігрових механік розроблюємого проекту; Огляд основних елементів ігрового процесу; Принцип роботи основних елементів ігрового процесу; Етапи реалізації основних елементів ігрового процесу; Хід тестування гри.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Іванченко В.С.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Рябошапка Д.Є.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	*20.05.2024*	Виконано
2	*Аналіз ігрового жанру 2D платформер*	*23.05.2024*	Виконано
3	*Аналітичний огляд та вибір програмного рушія*	*25.05.2024*	Виконано
4	*Формування концепту ігрового процесу гри*	*28.05.2024*	Виконано
5	*Проектування основних елементів ігрового процесу*	*30.05.2024*	Виконано
6	*Проектування графічної частини гри*	*01.06.2024*	Виконано
7	*Реалізація графічної частини гри*	*03.06.2024*	Виконано
8	*Реалізація основних елементів ігрового процесу*	*05.06.2024*	Виконано
9	*Імплементація та відлагодження ігровіх елементів*	*07.06.2024*	Виконано
10	*Тестування працездатності елементів гри*	*09.06.2024*	Виконано
11	*Виправлення виявлених помилок*	*10.06.2024*	Виконано
12	*Аналіз результатів, підготовка слайдів презентації*	*11.06.2024*	Виконано
13	*Економічні розрахунки та питання з охорони праці*	*12.06.2024*	Виконано
14	*Підготовка графічної частини проекту*	*13.06.2024*	Виконано
15	*Підготовка проекту до захисту та тестування ПП*	*14.06.2024*	Виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 18 000. 00 ДП	Дипломний проект		
А4			РП 07. 18 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 18 000. 00 ДП		
Розробив		Рябошапка Д.Є		10.06	Розробка 2D-гри у жанрі платформеру на програмному рушії Unity	Літ.	Аркуш 5 / Аркушів 76
Перевірив		Джабраілов Д.В.		10.06		Н Д П	
Н. контр.		Петрашова В.І.		10.06	ВСП "ОТФК ОНТУ" гр.4РП-07		
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП

Ігрова індустрія стрімко розвивається, постійно з’являються нові технології та жанри. Проте, жанр платформерів залишається одним із найпопулярніших та затребуваних, як серед гравців, так і серед розробників. Це пов'язано з його простотою, динамічністю та можливістю реалізувати різноманітні ігрові механіки.

Окрім цього, самі по собі відеоігри займають величезну частину ринку IT і користуються шаленим попитом, порівняно з минулими роками. Це пов’язано з появою мультимедійних пристроїв майже у кожної людини на планеті. Тепер, для того, щоб пограти в улюблену гру не обов’язково витрачати гроші та час, щоб піти в ігровий клуб, можна просто дістати телефон з карману і почати грати тут і зараз. Головне, щоб був доступ до мережі інтернет.

Так як створення ігор давно перестало бути загадкою і ним займаються не лише компанії, а і інді розробники, моя тема «Розробка 2D-гри у жанрі платформеру на програмному рушії Unity» ставить перед собою мету – насамперед показати алгоритм створення таких ігор, тому що на основі одного алгоритму можна створити безліч ігор.

Сам жанр платформеру передбачає величезну кількість різних версій майже однакових за суттю проектів. Вони можуть відрізнятись використанням різних рівнів, складніших платформ та перешкод, щоб кожен гравець міг знайти для себе оптимальну складність, або це може бути просто не складна гра, основною ідеєю якої є показати історію персонажа, що дуже легко зробити у грі жанру платформер.

Також, можна сміливо заявити, що дана тема є актуальною технологічно, оскільки вона використовує сучасні мови програмування та засоби розробки, тому на прикладі готового проекту я покажу наскільки легшим стає його створення за умов використання сучасних інструментів розробки.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз ігрового жанру 2D платформер

Для початку розробки будь якої гри розробнику потрібно вирзначитись з концептом майбутнього проекту. Основною темою мого дипломного проекту є «Розробка 2D-гри у жанрі платформеру на програмному рушії Unity», з цього можна зрозуміти, що мені необхідно створити гру у жанрі платформер, з використанням характерних цьому жанру ознак. Для того, щоб зрозуміти, які ж ознаки характерні жанру потрібно детальніше розібратись, що ж таке платформери взагалі.

Ігри у цьому жанрі представляють з себе проекти у яких персонаж переміщається платформами, звідти і назва. Такі проекти можуть бути виконані як у двовимірній графіці так і у тривирімній. Основна суть полягає в тому, що гравець долає певні перешкоди на рівні. Переміщення у іграх-платформерах може відбуватись як знизу вверх по горизонтальних платформах, так і зліва-направо.

Жанр платформер є одним з перших жанрів комп’ютерних ігор, через це він має велику історію і налічує величезну кількість ігор. Проте, через це також з’являється просто неймовірно велика конкуренція. Особливо в епоху консолей, коли платформери заполонили величезну частину ринку ігор і займали 15% прибутку від всіх ігрових проектів. Велику популярність такі ігри здобули через свою простоту, різнобарвність та цікавих персонажів. Як наприклад головний герой гри Super Mario Bros (1985 року) став популярним далеко за межами простої гри платформеру і став головним символом-маскотом компанії Nintendo.

Першими платформерами були: Frogs (1978) де були стрибки, але не було платформ і Space Panic (1980) де були платформи, але не було стрибків. В Donkey Kong (1981) вперше з'явився скроллінг. Першим справжнім платформером стала гра Pitfall.

Рисунок 1.1. Скріншот з гри Pitfall

Як можна помітити, найперший платформер виглядає дуже примітивно і зовсім не нагадує нинішні, але треба пам’ятати, що з моменту його появи пройшло більше 40 років і з тих пір технології пішли далеко вперед.

Однак легендарною ця гра не стала. Першу славу платформерам приніс вже згаданий вище Super Mario Bros. Його суть також була проста, потрібно пройти до грибного королівства, вціліти під час зустрічі з ворогами та звичайно ж врятувати принцесу. У процесі було необхідно збирати пауер-апи (посилення), досягати чекпоінтів (точок збереження) на рівнях та переходити на наступний рівень. А також долати ворогів незвичним методом, а саме – стрибаючи їм на голови.

Ця гра стала настільки популярною, що потрапила у Книгу рекордів Гіннеса, розійшовшись тиражем у 40 мільйонів копій. Цей рекорд не було побито аж до 2009 року, грою тієї ж компанії.

Рисунок 1.2. Скріншот з гри Super Mario Bros

Після Super Mario Bros почали виходити і нові версії гри про двох братів. У них змінювався стиль, покращувались елементи гри, проте вони так само залишатись платформерами, вивівши цей жанр у топи.

Іншою, трохи менш популярною, але не менш легендарною грою-платформером стала гра про блакитного їжачка, а саме Sonic the Headhog, від компанії Sega. Гру було випущено у 1991 рокі, але вона є актуальної і досі. В порівнянні з минулими іграми, ця була найшвидшою, була написана для 16 бітної консолі і мала відповідну графіку.

Але звичайно, з тих пір пройшло багато часу і вже не потрібно використовувати лише 16 біт і вміщати гру у декілька кілобайт місця на диску, проте це не означає що жанр двовимірних платформерів у стилі піксельної анімації залишився у минулому. Візьмемо до уваги декілька сучасних ігор такого формату.

Першою я хочу розповісти про гру Dead Cells. Вийшовши у 2017 році, ця

гра використовує технології, схожі на ті, що використовувались для створення умовного Super Mario Bros. Проте, звичайно, можна помітити наскільки плавнішою стала анімація, наскільки багато додаткових опцій має ця гра і звичайно кидається у очі те, що рівні цієї гри генеруються рандомно кожен раз і майже неможливо отримати такий самий рівень вдруге, що робить гру легкою для повторного проходження багато разів, адже гравець не втрачає інтерес роблячи одне і те саме, на одних і тих самих локаціях.

Рисунок 1.3. Скріншот з гри Dead Cells

Наостанок, я хочу розповісти про Celeste. Це відеогра розроблена інді-студією у 2018 році. Гравець керує Медлін, яка має здатність бігати, стрибати, лазити по стінах і робити ривок в одному з восьми напрямків. У гри присутній глибокий сюжет, який і є її основою. Головною метою персонажа є – дістатися до вершини гори Селест, проходячи при цьому різноманітні рівні різної складності і спілкуючись з людьми, що заселяють цю гору. Не дивлячись на те, що розробкою не займалась велика студія і у розробку не було витрачено багато грошей – вона стала комерційно успішною і популярною у своєму жанрі, завдяки гарній історії та приємній картинці.

Рисунок 1.4. Скриншот з гри Celeste

Підводячи підсумки, можна побачити, що навіть схожі за класифікацією ігри одного і того ж жанру дуже відрізняються одна від одної, а також можна зазначити те, що вони всі є комерційно успішними, що ще раз показує актуальність жанру платформер. Такі ігри є досить простими у виготовлені, проте це не робить їх менш цікавими для гравця, якщо у грі також присутня цікава історія.

1.2 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом

Для розробки дипломного проекту було використано ігровий рушій Unity. Unity це багатоплатформенний інструмент для створення ігор, а також рушій на якому вони працюють. Він є одним з найпопулярніших інструментів для створення ігор, що є його великим плюсом. Завдяки своїй популярності Unity має величезну кількість матеріалів по ньому, що допомагає створювати їх розробникам-початківцям. Також, великим плюсом Unity є його клієнтоорієнтовність, його дизайн є зрозумілим і не вимагає багато часу на вивчення, а також інтерфейс можливо змінити, залежно від ваших потреб.

Unity використовує як основну мову програмування C#, а тому користується також і всіма перевагами цієї об’єктноорієнтованної мови програмування, як наприклад наслідування об’єктів.

На рисунку нижче показано головний екран Unity

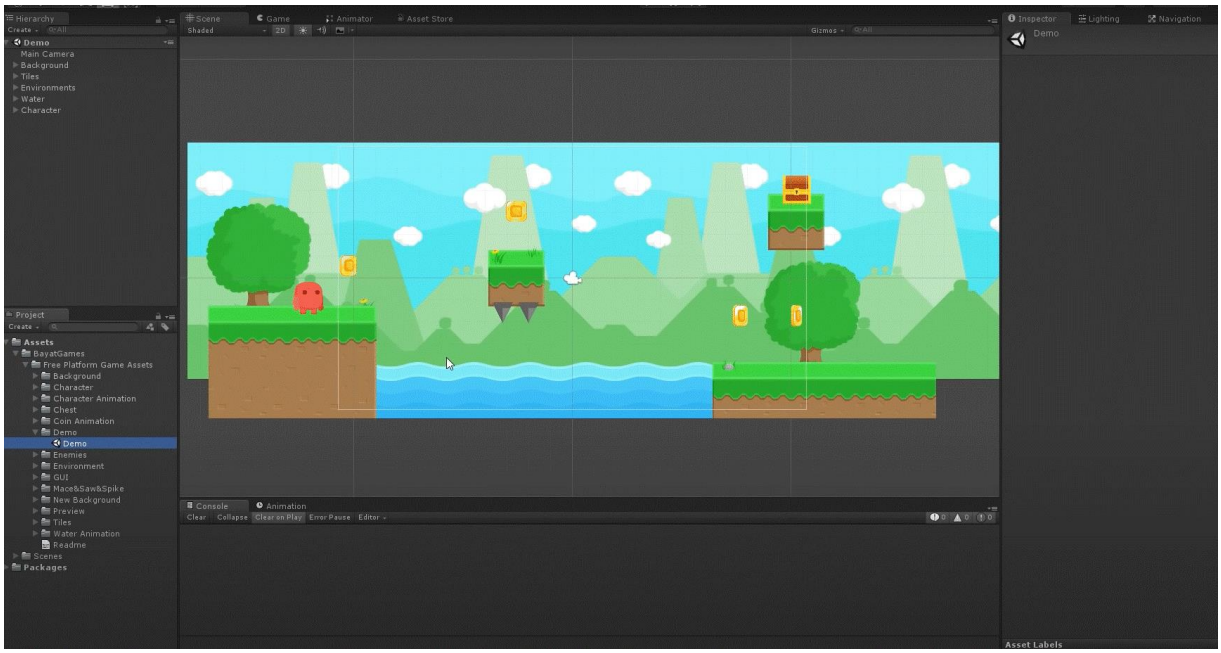


Рисунок 1.5. Скриншот інтерфейсу Unity

Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

Також, цей редактор підтримує як тривимірні, так і двовимірні проекти. Для 2D проектів здебільшого використовуються спрайти, що представляють з себе плоске зображення будь якого формату, який підтримується Unity. Для 3D ж об’єктів система відрізняється і для створення будь чого на сцені потрібні меші, які потрібно створювати у спеціальних застосунках, таких як наприклад Blender, Maya, ZBrush тощо.

На відміну від багатьох ігрових движків, Unity має дві основні переваги: наявність візуального середовища розробки і міжплатформну підтримку. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегроване середовище , ланцюжок складання , що спрямоване на підвищення продуктивності розробників, зокрема етапів створення прототипів та тестування. Під міжплатформною підтримкою надається не тільки місця розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі тощо), але й наявність інструментарію розробки (інтегроване середовище може

використовуватися під Windows та Mac OS).

Також, дуже важливим є ліцензія, що дозволяє використовувати Unity безкоштовно доти, доки проект не буде приносити прибуток більше ніж 200000 доларів. Фактично, це означає, що користувач може створювати необмежену кількість проектів, без жодних обмежень, допоки вони не будуть приносити значний комерційний успіх.

Unity має велику кількість бібліотек, як для роботи з кодом, так і для асетів. Одним з основних елементів додатку є Asset Store, у якому можна знайти як безкоштовні, так і платні рішення для гри, починаючи з зображень для двовимірних спрайтів, закінчуючи фоновими зображеннями та повноцінними тривимірними модельками персонажів.

За використанням Unity було написано майже половина мобільних ігор, більше 60 відсотків додатків віртуальної та доповненої реальності і значний відсоток комп'ютерних ігор. За допомогою Unity було написано такі ігри як Genshin Impact (Рисунок 1.6), Ori and the Blind Forest (Рисунок 1.7), як і всі подальші ігри цієї серії та Outer Wilds (Рисунок 1.8).

Рисунок 1.6. Скриншот з гри Genshin Impact

Рисунок 1.7. Скриншот з гри Ori and the Blind Forest

Рисунок 1.8. Скриншот з гри Outer Wilds.

Unreal Engine та Unity є двома найпопулярнішими ігровими рушіями, кожен з яких має свої унікальні переваги та особливості. Вибір між ними залежить від різноманітних факторів, включаючи технічні вимоги, специфіку проекту та рівень досвіду розробника. Однак, для новачків у сфері розробки ігор Unity є більш сприятливим варіантом з кількох причин.

По-перше, Unity відомий своєю доступністю та простотою використання. Інтерфейс Unity інтуїтивно зрозумілий, що дозволяє новачкам швидко освоїти основні функції та розпочати роботу над своїми проектами без значних

перешкод. Unity надає розробникам великий обсяг навчальних матеріалів, включаючи офіційну документацію, відеоуроки та інтерактивні курси. Це сприяє швидкому навчальному процесу та допомагає новачкам успішно долати початкові труднощі.

По-друге, Unity підтримує велику кількість платформ, включаючи мобільні пристрої, ПК, консолі та веб-браузери. Ця універсальність робить Unity ідеальним вибором для розробників, які бажають створювати крос-платформні ігри. Здатність Unity швидко адаптуватися до різних платформ є важливою перевагою, оскільки новачки можуть експериментувати з різними типами ігор та аудиторіями, не обмежуючись однією платформою.

Технічні вимоги Unity також є менш суворими порівняно з Unreal Engine. Unity може ефективно працювати на менш потужному апаратному забезпеченні, що робить його доступнішим для широкого кола розробників. Це дозволяє новачкам розпочати роботу без необхідності в дорогих комп'ютерах, що є важливим аспектом для тих, хто тільки починає свій шлях у розробці ігор.

Unreal Engine, хоча й відомий своїми потужними інструментами та високою якістю графіки, має більш складну криву навчання. Інтерфейс Unreal Engine багатший на функції, але також є більш складним для освоєння. Високі технічні вимоги та специфіка налаштувань можуть бути перешкодою для новачків, які ще не мають достатнього досвіду в розробці ігор. Крім того, Unreal Engine більше орієнтований на розробку високоякісних графічних проектів, що може вимагати від розробників додаткових знань у сфері графіки та оптимізації.

Загалом, Unity є кращим вибором для новачків завдяки своїй доступності, простоті використання, універсальності та менш суворим технічним вимогам. Ці характеристики роблять Unity ідеальною платформою для початківців, які хочуть швидко зануритися в процес розробки ігор та поступово розширювати свої знання і навички. Використання Unity дозволяє новачкам зосередитися на творчих аспектах розробки, не відволікаючись на складнощі технічного характеру, що є важливим для успішного старту у світі ігрової індустрії.

Не дивлячись на це, я хочу розповісти більше про проекти створені на

основі Unreal Engine і чому цей рушій славиться такою популярністю серед студій розробки.

Unreal Engine – це популярний ігровий двигун, який відомий своєю потужністю та графічною точністю. Він використовується для створення високоякісних AAA-ігор.

Останнє покоління, Unreal Engine 5, було запущено в квітні 2022 року. Його вихідний код доступний на GitHub , а комерційне використання надається на основі моделі роялті , при цьому Epic стягує 5% від доходу понад 1 мільйон доларів США, який не поширюється на ігри. опубліковано в Epic Games Store . Epic включила в движок функції від придбаних компаній, таких як Quixel, чому сприяли доходи Fortnite .

Однією з ключових особливостей Unreal Engine є його графічний рушій, який підтримує передові технології, такі як трасування променів (ray tracing), що дозволяє створювати неймовірно реалістичні графічні ефекти. Це робить Unreal Engine популярним вибором для розробників AAA-ігор, а також для створення вражаючих візуалізацій та симуляцій в інших галузях. На рисунку 1.8 можна побачити приклад графіки яку відтворює Unreal Engine.

Рисунок 1.9. Приклад графіки Unreal Engine

Unreal Engine також був використаний для створення серії Batman:

Arkham, зокрема Batman: Arkham Asylum та Batman: Arkham City. (Рисунок 1.10) Ці ігри продемонстрували можливості рушія у створенні складних, відкритих світів з багатошаровими деталями та інтерактивними елементами. Гра забезпечує високий рівень реалістичності та деталізації, що дозволяє гравцям повністю зануритися в атмосферу Готема.

Рисунок 1.10. Приклад графіки гри Batman: Arkham.

1.3 Вибір інструментів розробки

Наступним важливим кроком у підготовці до розробки ігрового проекту є обрання інструментарію. Програмне забезпечення для розробки має дуже велику роль у процесі роботи, адже кожне рішення має свої особливості та що не менш важливо, користувач може мати свої улюблені редактори, в яких має навички роботи та знає всі нюанси. Більшість крупних компаній розробників досить вільно відносяться до прикладного інструментарію розробки, наприклад, як редактори коду.

Редактор коду – це спеціалізоване програмне забезпечення, яке використовується для створення, модифікації та організації вихідного коду програм. Такі редактори зазвичай мають інтуїтивно зрозумілий інтерфейс, підтримують підсвічування синтаксису, автоматичне доповнення коду, пошук та заміну тексту, а також інтеграцію з іншими інструментами розробки, включаючи системи контролю версій, дебагери та компілятори. Серед відомих редакторів

коду можна виділити Visual Studio Code та Sublime Text.

Для написання самого коду було використано редактор коду Visual Studio 2019. Visual Studio є середою розробки компанії Microsotf Windows. Вона містить інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ця програма дає змогу розробляти як консольні програми, так і програми з графічним інтерфейсом.

Він приймає плагіни, які розширюють функціональні можливості майже на всіх рівнях, включаючи додавання підтримки систем керування джерелами (наприклад, Subversion і Git) і додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для доменно-спеціальних мов або наборів інструментів для інших аспектів розробки програмного забезпечення.

Visual Studio містить редактор коду, який підтримує підсвічування синтаксису та завершення коду за допомогою IntelliSense для змінних , функцій , методів , циклів. IntelliSense підтримується для включених мов, а також для XML, каскадних таблиць стилів. Пропозиції щодо автозавершення з’являються в немодальному списку над вікном редактора коду поблизу курсору редагування . У Visual Studio 2008 і новіших версіях його можна тимчасово зробити напівпрозорим, щоб побачити код, який він закриває. Редактор коду використовується для всіх підтримуваних мов.

Ліцензія Visual Studio є дуже гнучкою. Для використання середи розробки в цілях освіти, науки, персонального та некомерційного використання, вона розповсюджується безкоштовно. Окрім того між безоплатною та платною версією немає функціональних різниць.

Aseprite це редактор зображень, призначений для малювання та анімації у піксельному стилі. Деякі з його функцій включають:

- Шари та рамки з групуванням шарів і тегами анімації
- Трансформації та інструменти, специфічні для піксельного мистецтва (режими ідеального пікселя, спеціальні пензлі тощо)
- Попередній перегляд анімації в реальному часі та очищення цибулі
- Режими Tilemap і Tileset

- Керування палітрою кольорів , включаючи 65 палітр за замовчуванням

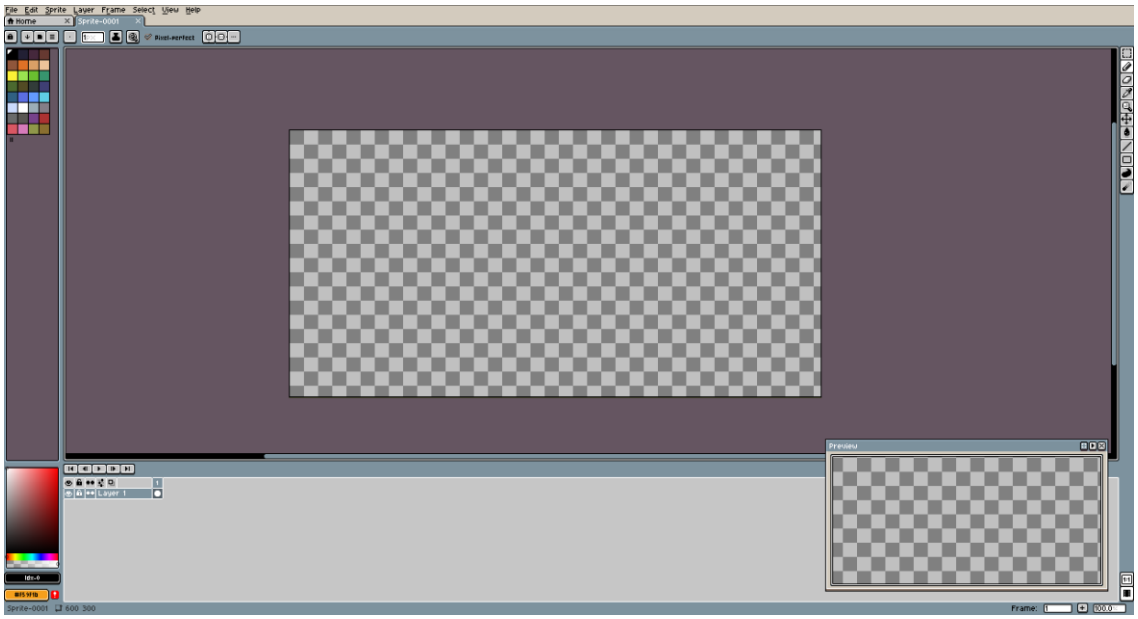


Рисунок 1.11. Скриншот з додатку Aseprite

У цій програмі у користувача є можливість створити зображення у жанрі малюнку по пікселям, анімувати цей малюнок і створити набір спрайтів для таких додатків як Unity.

Це програмне забезпечення розповсюджується за ліцензією, проте має безкоштовну демо-версію, яка хоч і має трошки менший функціонал, але так само придатна до використання і була використана мною задля створення спрайтів для гри.

1.4 Формування концепту ігрового процесу гри

Формування концепту ігрового процесу починається з технічного опису проекту. З теми зрозуміло, що жанр, а саме гра платформер вже обрано, тому варто врахувати особливості цього жанру і дотримуватись їх.

Найважливішим етапом створення платформеру є звичайно ж створення рівня та платфор по яких персонаж може рухатися у сцені. В платформерах може бути декілька сцен, або одна велика, поєднана на декілька екранів.

Також, у більшості ігор платформерів є вороги. У деяких з них персонаж помирає від першого ж торкання з ворогом, у інших випадках – втрачає одне життя. Також важливо передбачити чи буде у гравця можливість відбиватись від ворогів, або він буде просто перестрибувати їх чи уникати іншими способами.

Для ігор жанра платформер дуже важливо додавати історію, яка буде пояснювати мотивацію головного героя проходити рівні та розкривати його сюжет, тому що в еру сучасного геймінгу гравцям більше не цікаво грати у аби що лише заради механік стрибку чи ухилення від кулі.

Тож, розглянувши ці питання, стає зрозумілим, що гра має слідувати наступним правилам:

- У грі має бути створено хоча б декілька рівнів з перешкодами, або один довгий;
- У грі має бути присутня механіка смерті персонажа, яка може відбутись як від падіння з платформи, так і від зіткнення з ворогом;
- Гра не має бути занадто простою, тому що це не буде викликати зацікавленість
- Гравець завжди має стабільну швидкість, достатню для того щоб долати перешкоди
- На рівні можуть з’являтись пауер-апи (посилення) для гравця
- У грі має бути присутній хоча б найменший сюжет

Після того, як ми визначились з правилами і концептом гри – можна починати її реалізацію.

У моєму випадку, реалізація буде відбуватись на ігровому рушії Unity, тому виконувати всі поставлені вище завдання потрібно виходячи з особливостей цього додатку. Активна сцена визначається положенням камери, який у випадку гри платформера закріплюється і слідує за персонажем.

Рівні гри будуть промальовані одразу та не будуть генеруватись, що зробить гру простішою для реалізації та оптимізації. Однак на рівні будуть рандомно генеруватись моби-вороги, які будуть заважати персонажу перейти на наступний рівень.

Окрім цього, потрібно реалізувати схему, при якій гравець після виходу за межі мапи, злітання з платформи або зіткнення з платформи – починав все спочатку або ж з точки збереження (чекпоінту).

При створенні своєї гри я базувалась на двох улюблених іграх, це Dead Cells

(Рисунок 1.11), а також гра Spiritfarer (Рисунок 1.12)

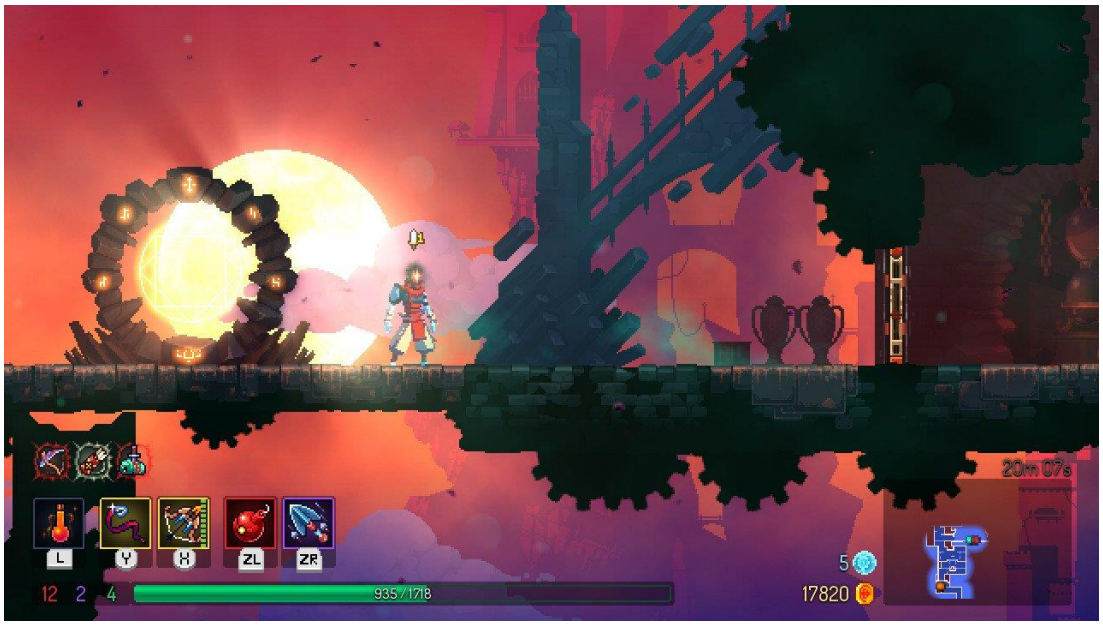


Рисунок 1.12. Скріншот з гри Dead Cells

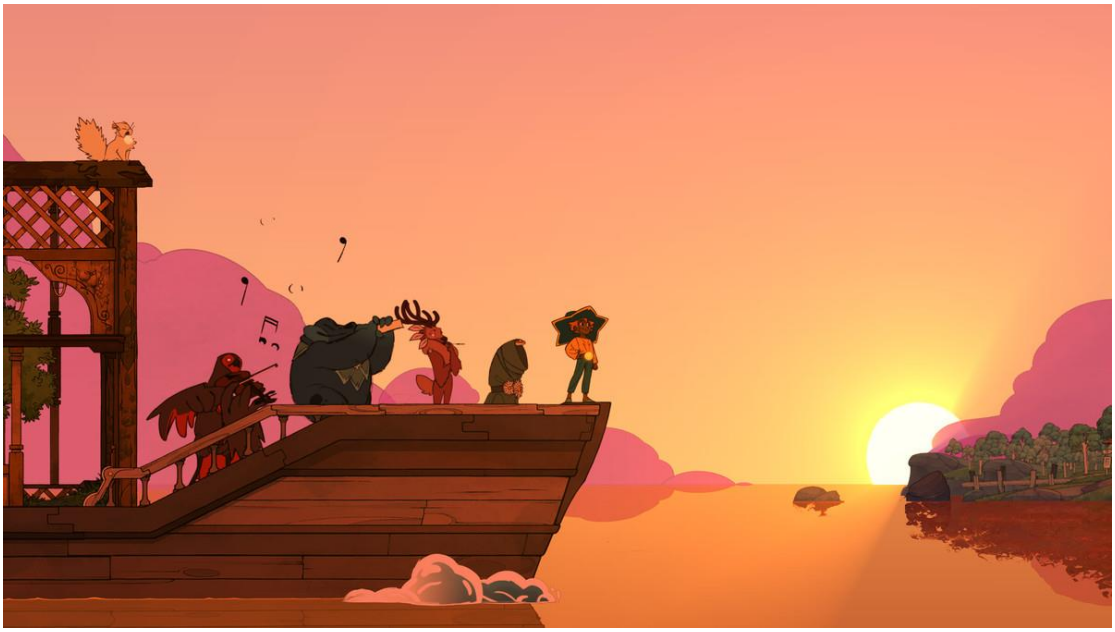


Рисунок 1.13. Скріншот з гри Spiritfarer

З першої гри було взято концерт піксельної гри, а також фізика персонажа на сцені. З другої гри в більшості було взято концепт простої платформеності та акценту на сюжеті.

1.5 Проектування основних модулів гри, принцип їх роботи

Проектування платформених 2D проектів на Unity є складним і багатогранним процесом, що включає концептуалізацію, технічну реалізацію та тестування різноманітних ігрових елементів для створення привабливого та

функціонального ігрового середовища. Основною метою цього процесу є створення захопливого ігрового досвіду, який включає цікаві рівні, чітке управління персонажем, виразні анімації та інтерактивні об'єкти.

Першим етапом проектування є концептуалізація ігрової ідеї та розробка дизайну гри. Це включає створення загального бачення гри, визначення її жанру, стилю та основних механік. У платформених 2D іграх особливу увагу приділяють дизайну рівнів, які повинні бути ретельно сплановані, щоб забезпечити плавний і цікавий геймплей. На цьому етапі створюються концептуальні малюнки та ескізи, що допомагають визначити основні елементи рівнів, такі як платформи, перешкоди, вороги та інтерактивні об'єкти.

Після завершення концептуального етапу розпочинається технічна реалізація проекту в Unity. Першим кроком є налаштування середовища розробки та імпорт необхідних графічних ресурсів, таких як спрайти персонажів, фонів, об'єктів та анімацій. У Unity для створення ігрового середовища часто використовується система Tilemap, яка дозволяє розміщувати плитки (Tiles) на сітці, створюючи таким чином рівні. Це значно спрощує процес розробки, дозволяючи легко змінювати та тестувати різні конфігурації рівнів.

Ключовим аспектом платформених ігор є управління персонажем. Для цього у Unity використовуються компоненти RigidBody2D та Collider2D, які забезпечують фізичні властивості та взаємодії об'єктів. Написання скриптів на мові C# дозволяє визначити логіку руху персонажа, включаючи стрибки, біг, зіткнення з об'єктами та ворогами. Важливо налаштувати параметри фізики, такі як гравітація, маса та сила тертя, щоб забезпечити природний рух персонажа та його взаємодію з платформами.

Анімація персонажа також відіграє важливу роль у створенні привабливого ігрового досвіду. У Unity використовується анімаційна система Animator, яка дозволяє створювати та керувати анімаціями персонажа. Анімаційні кліпи, такі як біг, стрибок, атака та інші, об'єднуються у Animation Controller, що забезпечує плавні переходи між різними станами персонажа на основі заданих умов.

Розробка рівнів є ще одним критично важливим аспектом проектування

платформених ігор. На цьому етапі створюються різноманітні платформи, перешкоди, вороги та інші елементи, що формують геймплей. Використання Tilemap Grid дозволяє легко додавати та змінювати розташування об'єктів на рівні, забезпечуючи гнучкість у дизайні. Важливо ретельно продумати розташування платформ та перешкод, щоб створити цікаві виклики для гравця, що стимулюватимуть його навички та реакції.

Інтерактивні об'єкти та вороги додають динаміки та різноманітності у геймплей. Скрипти на C# дозволяють визначити поведінку ворогів, включаючи їхні рухи, атаки та реакції на дії гравця. Інтерактивні об'єкти, такі як кнопки, двері, скрині та інші, також програмуються для взаємодії з персонажем, додаючи додаткові елементи загадок та завдань.

Після завершення основних етапів розробки розпочинається тестування гри. Це включає перевірку всіх ігрових механік, анімацій, рівнів та взаємодій, щоб виявити та виправити можливі помилки та недоліки. Тестування проводиться як вручну, так і автоматизовано, щоб забезпечити стабільну роботу гри та високу якість користувацького досвіду.

Для подальшої розробки проекту також важливою частиною є проектування його основних складових, елементів ігрового процесу та взагалі описання процесу роботи цих складових. Це важливо з тієї точки зору, що раніше спроектовані елементи набагато легше розробляти, оскільки під час процесу проектування більшість логічних та структурних проблем ігрових або логістичних механік.

Спочатку, перед безпосереднім створенням проекту, потрібно виконати проектування основних модулів гри. Для цього необхідно розділити гру на менші частини і виділити модулі концептуально. Всього я виділила три основні модулі:

- Код гравця;
- Код рівня;
- Код ворога

Кожен з цих модулів має певний код, який йому належить. Схема основних модулів зображена на малюнку нижче:

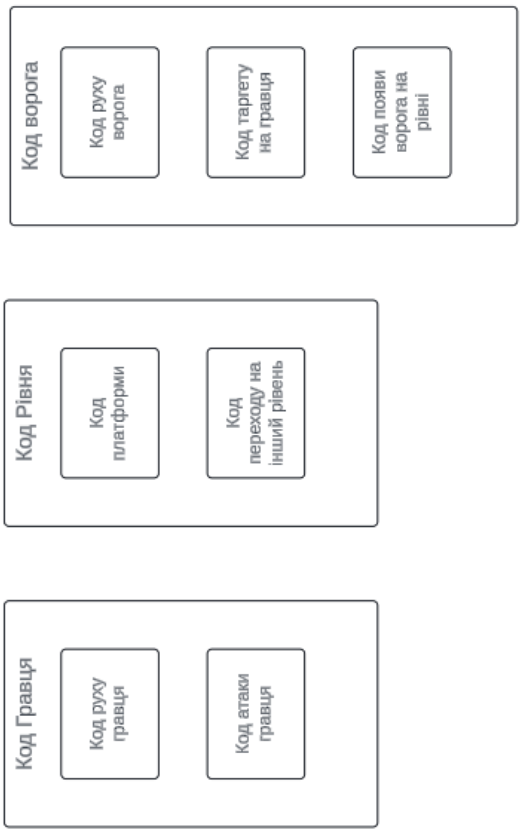


Рисунок 1.14. Схема модулів гри та коду, що їм належать.

Останнє про що потрібно зазначити, це про структуру платформ, оскільки в темі дипломного проекту зазначено використання фізики ігрового програмного рушія Unity, то необхідно більш детально розглянути питання фізики в Unity.

Фізична складова ігрового рушія Unity є однією з ключових компонентів, що забезпечує реалістичність та інтерактивність ігрового процесу. Цей аспект включає моделювання фізичних властивостей об'єктів, їх взаємодію та поведінку в ігровому середовищі, використовуючи механізми обчислення фізичних явищ.

Основою фізичної системи Unity є двигун фізики, який дозволяє імітувати реальні фізичні закони віртуального світу. Unity використовує два основні фізичні рушії: PhysX від NVIDIA для тривимірних ігор та Box2D для двовимірних. Обидва ці рушії надають можливість налаштовувати фізичні властивості об'єктів, такі як маса, гравітація, тертя та пружність, що дозволяє створювати реалістичну

взаємодію між об'єктами.

Компоненти RigidBody та RigidBody2D є центральними елементами фізичної системи. Вони додаються до GameObject-ів для того, щоб надати їм фізичні властивості і забезпечити їхню взаємодію з іншими об'єктами. RigidBody контролює такі параметри, як швидкість, прискорення та сили, що діють на об'єкт. Це дозволяє реалізувати різноманітні фізичні ефекти, такі як падіння, зіткнення та ковзання.

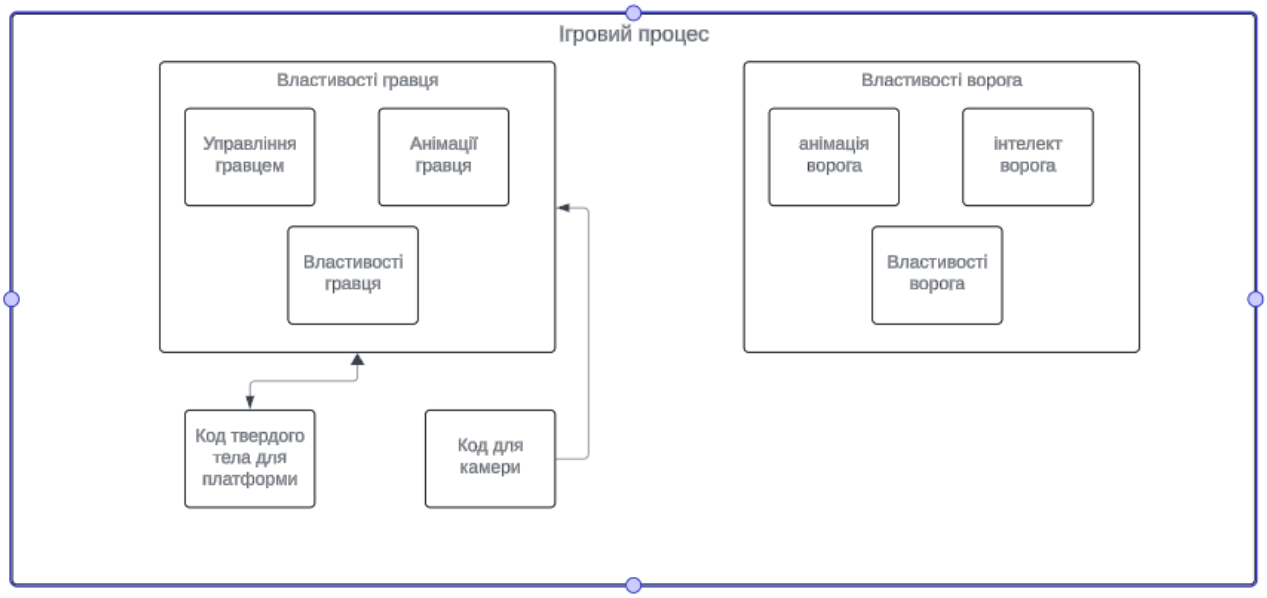


Рисунок 1.15. Схема проекту та взаємодії об’єктів у ньому

Схема показує, яким чином взаємодіють між собою об’єкти у проекті, а також визначають основні його процеси. З неї зрозуміло які об’єкти є головними у проекті і з чого вони мають складатись. На основі цього і буде побудовано програмний код для кожного з зазначених модулів.

1.6 Реалізація основних модулів гри

Для виконання завдання дипломного проектування необхідно реалізувати ряд модулів, що були визначені вище. Реалізація гри буде відбуватись у Unity, для основи свого проекту я обрала двовимірну сцену, так як такою ж буде і гра. Для цього у головному меню Unity Hub, у меню створення проекту я обрала двовимірну (2D) сцену. Кожна з показаних на рисунку 1.15 заготовок має свої параметри, тому важливо одразу визначитись з тим, який проект буде створено з переліку.

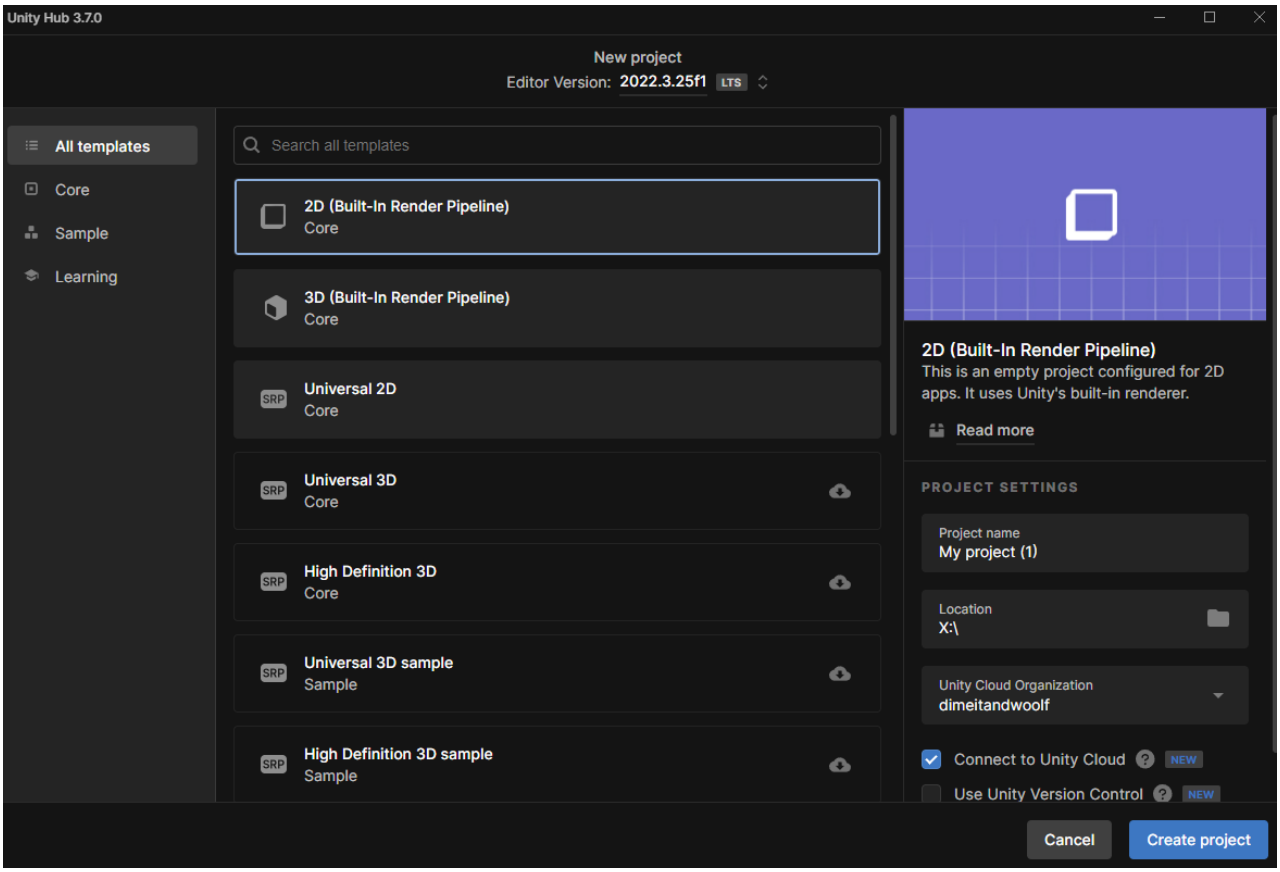


Рисунок 1.16. Створення 2D проекту у Unity

Наступним кроком було створення у проекті ігрових моделей для реалізації функціоналу гри. Для цього потрібно створити заздалегідь та перенести у сцену спрайт гравця та всіх інших об’єктів на сцену. Для свого проекту я використала спрайти, що намалювала у редакторі Aseprite.

Намалювавши декілька різних варіантів поз для головного персонажа я анімувала його рухи, за допомогою покадрової анімації. Це я зробила створивши копії одного і того самого персонажа, змінивши кожну копію зовсім трохи, буквально на декілька пікселів аби при запуску програвалась анімація (Рисунок 1.17). За таким самим принципом, як створюється анімація для мультфільмів або інших анімованих проектів.

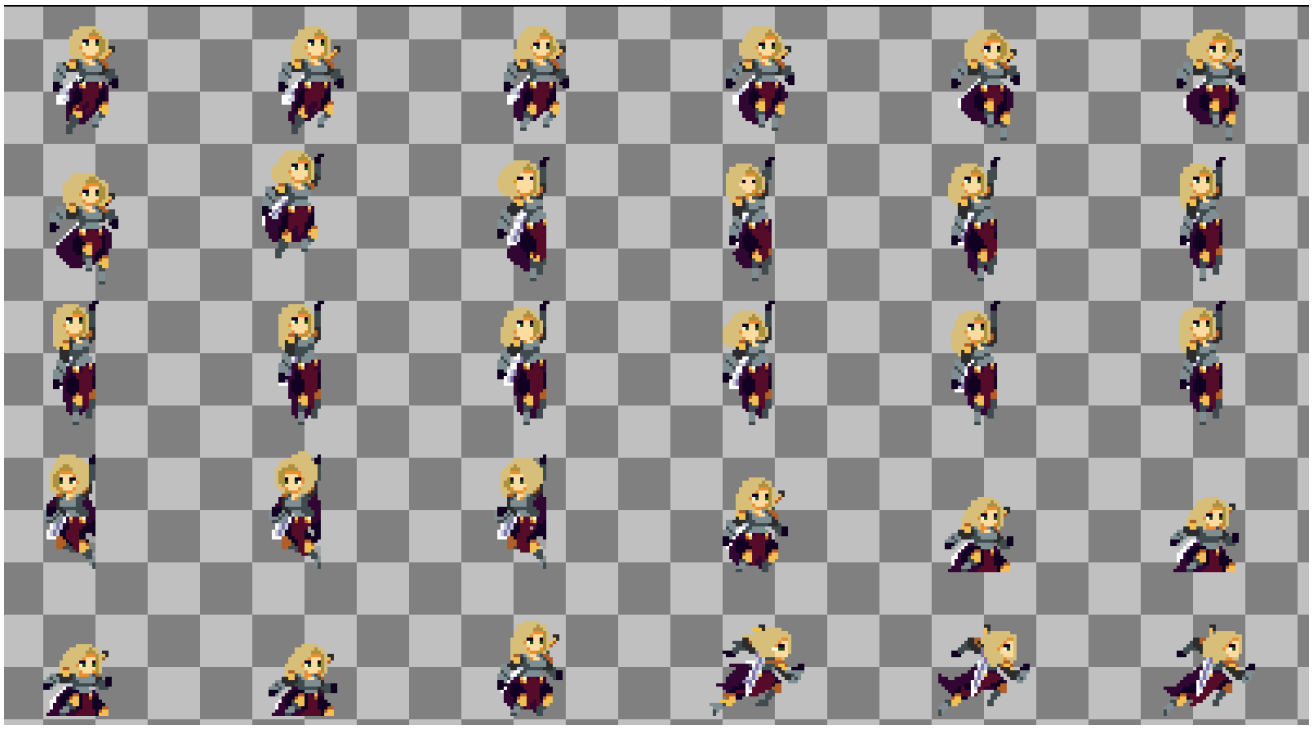


Рисунок 1.17. Спрайти гравця

Animator у Unity є потужним інструментом для створення та керування анімаціями об'єктів у грі. Він дозволяє розробникам створювати складні анімаційні системи, що включають різноманітні стани, переходи та умови, які контролюють поведінку персонажів і об'єктів.

Основною складовою Animator є анімаційний контролер (Animator Controller), який служить для управління анімаціями об'єкта. Animator Controller складається з множини станів (states), кожен з яких представляє конкретну анімацію, наприклад, біг, стрибок або атака. Стани можуть бути пов'язані переходами (transitions), які визначають, коли і як об'єкт має змінювати свій стан. Переходи можуть базуватися на різних умовах, таких як значення параметрів, тригери або завершення попередньої анімації.

Щоб створити анімацію в Unity, розробник спершу імпортує спрайти або 3D-моделі, які будуть використовуватися для анімації. Потім, за допомогою вікна Animation, можна створити анімаційні кліпи (animation clips), що визначають послідовність кадрів і тривалість кожного кадру. У цьому вікні також можна налаштувати ключові кадри (keyframes) для різних властивостей об'єкта, таких як позиція, обертання, масштаб і навіть матеріали.

Animator підтримує роботу з параметрами (parameters), які

використовуються для управління переходами між станами. Параметри можуть бути різних типів: числові (float), цілі (int), логічні (bool) і тригери (trigger). Ці параметри можуть змінюватися через скрипти під час виконання гри, дозволяючи створювати динамічні анімації, що реагують на дії гравця або інші події у грі.

Ще однією важливою функцією Animator є можливість створення дерев станів (state machines), що дозволяють структурувати анімації в ієрархічний порядок. Це особливо корисно для складних персонажів або об'єктів з великою кількістю різноманітних анімацій. Наприклад, можна створити окремі піддерева для анімацій руху, бойових дій і взаємодії з оточенням, що спрощує управління анімаційними переходами і їх налаштування.

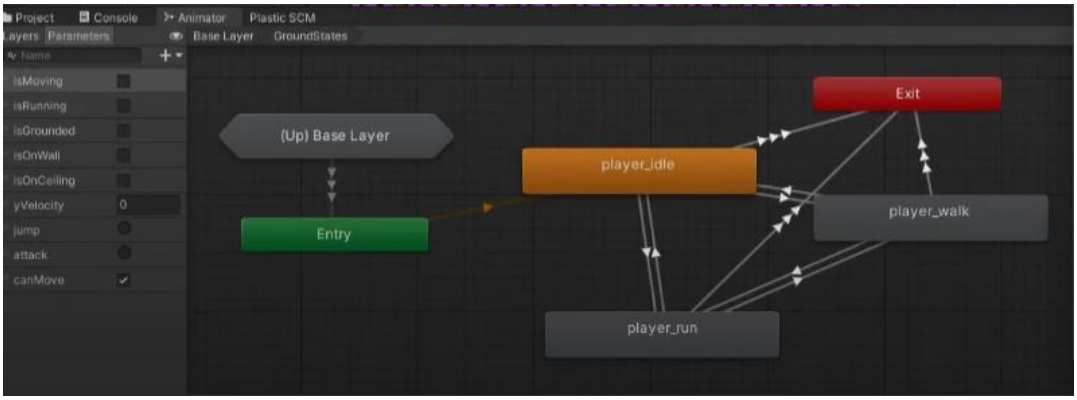


Рисунок 1.18. Скріншот дерева анімації персонажа

Animator також підтримує використання Blend Trees, які дозволяють плавно поєднувати кілька анімацій на основі значень параметрів. Це особливо корисно для створення реалістичних рухів персонажів, наприклад, при переході між ходьбою і бігом або при зміні напрямку руху. Blend Trees можуть поєднувати анімації з різною інтенсивністю, забезпечуючи плавні і природні переходи.

Крім того, Animator інтегрований з системою скриптів Unity, що дозволяє програмно контролювати анімації. За допомогою скриптів на C# можна змінювати параметри Animator, запускати певні анімації або реагувати на події анімаційних кліпів. Це надає розробникам велику гнучкість у створенні складних анімаційних систем, які можуть динамічно змінюватися під час гри.

Сама ж анімація відбувається за рахунок розміщення спрайтів покадрово на шкалу часу і їхнього відтворення. На рисунку 1.18 показано анімацію для бігу персонажу, яка зображена у сімох кадрах, які програються циклічно.

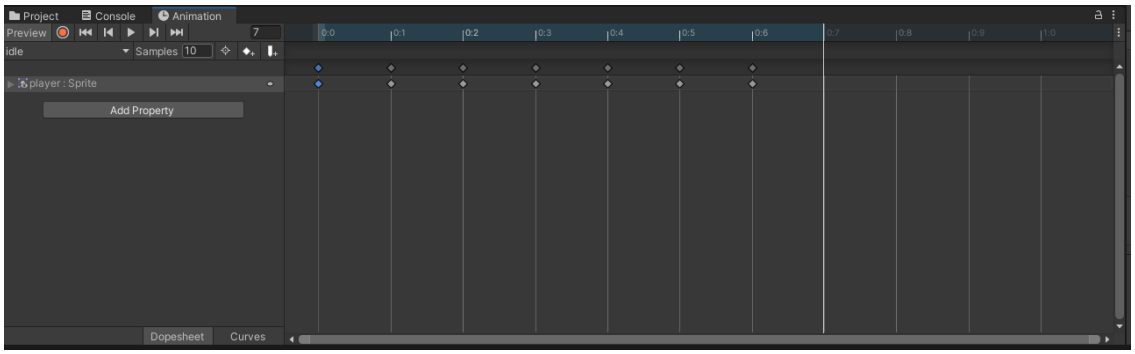


Рисунок 1.19 Лінія часу проекту.

На рисунку також можна помітити, що за секунду відбувається 10 кадрів, цей параметр можливо змінити і від цього буде залежати швидкість анімації. Чим більше кадрів тим, відповідно, швидша анімація. Проте, не варто використовувати більше 24 кадрів на секунду для піксельної анімації, адже для цього потрібно буде використати забагато кадрів у проміжках, які все одно непомітні для людського ока.

У самій же грі персонаж виглядає трохи більшим, ніж здається з малюнків, тому як це є головною особливістю піксельної анімації.

Рисунок 1.20. Спрайт гравця у проекті.

Рисунок 1.21 . Спрайт ворога

Цих об’єктів буде достатньо для реалізації основних модулів для ігрового процесу, а також написання для нього коду скриптів.

Простий код ворога, який рухається туди-назад по платформі показано нижче:

```
using UnityEngine;


public class EnemyPatrol : MonoBehaviour
{
    public float speed = 2f; // Speed of the enemy
    public Transform pointA; // First point of the patrol path
    public Transform pointB; // Second point of the patrol path
    private Vector3 targetPosition;
    private void Start()
```

```
        {
            targetPosition = pointB.position; // Start moving towards pointB
        }
        private void Update()
        {
            Move();
        }
        private void Move()
        {
            // Move the enemy towards the target position
            transform.position = Vector3.MoveTowards(transform.position, targetPosition, speed * Time.deltaTime);
            // Check if the enemy has reached the target position
            if (Vector3.Distance(transform.position, targetPosition) < 0.1f)
            {
                // Switch the target position to the other point
                if (targetPosition == pointB.position)
                {
                    targetPosition = pointA.position;
                }
                else
                {
                    targetPosition = pointB.position;
                }
            }
        }
    }
```

Префаби в Unity - це важливий інструмент для розробників ігор, який дозволяє створювати, налаштовувати та повторно використовувати об’єкти в ігровому середовищі. Префаби можуть включати будь-яку комбінацію об’єктів

гри, скриптів, анімацій, звуків, фізичних властивостей та інших компонентів, які визначають поведінку та вигляд об’єкта в ігрі.

Використання префабів в Unity має кілька переваг. По-перше, вони дозволяють легко створювати та налаштовувати об’єкти в ігрі. Префаби можна створити в редакторі Unity, налаштувати їх властивості та поведінку, а потім зберегти для подальшого використання. По-друге, префаби сприяють повторному використанню коду та ресурсів. Якщо об’єкт в ігрі повторюється кілька разів, його можна створити як префаб і потім клонувати в ігровому середовищі. По-третє, префаби полегшують управління змінами. Якщо потрібно змінити властивості або поведінку об’єкта, це можна зробити в одному місці, а зміни автоматично застосуються до всіх екземплярів префаба в ігрі.

Використання префабів в Unity включає створення префаба, налаштування його властивостей та компонентів, збереження префаба для подальшого використання, а потім клонування префаба в ігровому середовищі. Префаби можна також вкладати один в один, що дозволяє створювати складні ієрархії об’єктів. Крім того, Unity надає можливість перевизначення властивостей префаба для конкретних екземплярів в ігровому середовищі, що дозволяє розробникам легко налаштовувати поведінку та вигляд об’єктів на основі контексту.

В Ієрархії префаб об’єкт позначається синім кольором, а на сцені спеціальним знаком, щоб його можна було відлічити від не префаб об’єктів.

Розміщення об’єктів у Unity відбувається за допомогою панелі Ієрархії (Hierarchy), яка зображена на рисунку 1.22. Ця панель забезпечує структурування та організацію всіх об'єктів сцени, надаючи розробникам можливість ефективно керувати ієрархією об'єктів та їх взаємодією.

Основна функція панелі Hierarchy полягає у відображенні всіх GameObject-ів, що присутні на поточній сцені. Кожен GameObject, незалежно від його типу та призначення, буде відображений у вигляді списку, який можна розгортати та згортати для зручності навігації. Це дозволяє розробникам швидко знайти потрібний об'єкт, перевірити його стан та взаємозв'язки з іншими об'єктами. Панель Hierarchy відображає об'єкти у вигляді деревовидної структури, де

вкладені об'єкти (діти) знаходяться під батьківськими об'єктами, що полегшує управління складними сценами з великою кількістю взаємопов'язаних елементів.

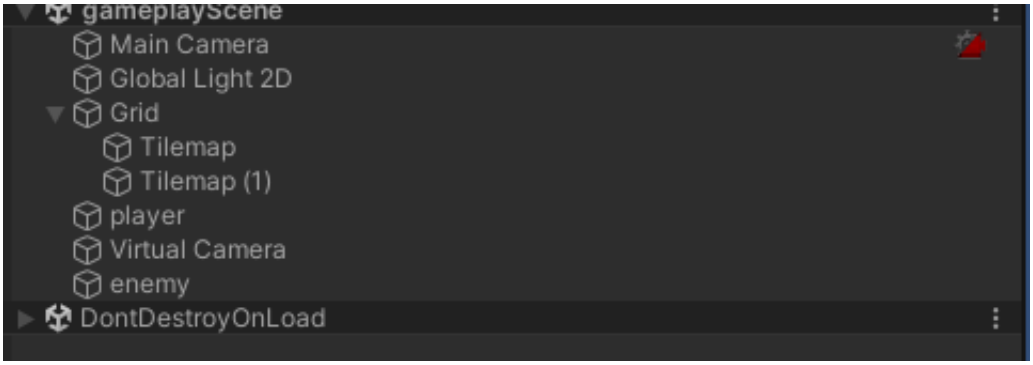


Рисунок 1.22. Зовнішній вигляд панелі Ієрархії

Панель Hierarchy також тісно інтегрована з іншими компонентами Unity, такими як панель Inspector та редактор сцени. Вибір об'єкта у панелі Hierarchy автоматично відображає його властивості у панелі Inspector, де можна редагувати компоненти та параметри об'єкта. Це забезпечує зручний та інтуїтивно зрозумілий робочий процес, де всі необхідні інструменти знаходяться під рукою.

Після розміщення основних об’єктів на сцені, потрібно також створити ігровий інтерфейс. Для цього у своєму проекті я додала головну сцену, рівень, за допомогою tile, вони ж «плитки» інтерфейсу.

Tile (плитка) є важливим інструментом для створення рівнів у розробці ігор, особливо у 2D-проектах. В Unity, система Tilemap пропонує зручні засоби для роботи з плитками, що дозволяють швидко та ефективно створювати детальні та різноманітні ігрові рівні.

Система Tilemap включає кілька основних компонентів:

Tilemap Grid – це базовий компонент, що визначає сітку, на яку будуть розміщуватися плитки. Сітка може бути налаштована для різних типів плиток, таких як квадратні або шестикутні, що дозволяє створювати різноманітні стилі рівнів. За допомогою наприклад ізометричної сітки створюються ігри у форматі 2.5D, що по суті представляє із себе тривимірне зображення створене у двовимітрному просторі.

Tilemap – це компонент, що зберігає інформацію про розташування плиток на сітці. Кожна плитка (Tile) розміщується на певній позиції в сітці, що дозволяє

легко керувати рівнем та змінювати його структуру.

Tile – це основний елемент, що представляє собою окрему плитку. Плитка може бути статичним зображенням або анімованою, що дозволяє створювати динамічні та привабливі рівні. Плитки можуть мати різні властивості, такі як можливість проходження, тип поверхні, або взаємодія з персонажами.

Процес створення рівня з використанням плиток включає кілька етапів. Спочатку розробник створює або імпортує набір плиток (Tileset), що включає всі необхідні графічні елементи для побудови рівня. Після цього налаштовується Tilemap Grid, яка визначає розміри та форму сітки. Далі розробник використовує компонент Tilemap для розміщення плиток на сітці, створюючи таким чином структуру рівня.

Однією з ключових переваг використання плиток для створення рівнів є їх висока ефективність. Плитки дозволяють швидко створювати великі рівні, використовуючи невелику кількість графічних елементів, що значно знижує обсяг пам'яті, необхідної для зберігання графіки. Крім того, сітка Tilemap дозволяє легко редагувати рівні, змінюючи лише окремі плитки, а не переробляючи всю сцену.

Ще однією важливою перевагою є можливість використання Tilemap для створення складних ігрових механік. Наприклад, розробник може використовувати різні типи плиток для позначення різних властивостей поверхонь, таких як лід, трава або вода, що впливатимуть на рух персонажів. Крім того, плитки можуть бути використані для створення інтерактивних об'єктів, таких як двері, пастки або кнопки, що додають глибини ігровому процесу.

У моєму випадку всі об’єкти зображені на сцені є плитками, разом з усіма елементами, такими як скриня, самі платформи тощо.

Задній фон, створений з набору плиток можна побачити на рисунку 1.23.

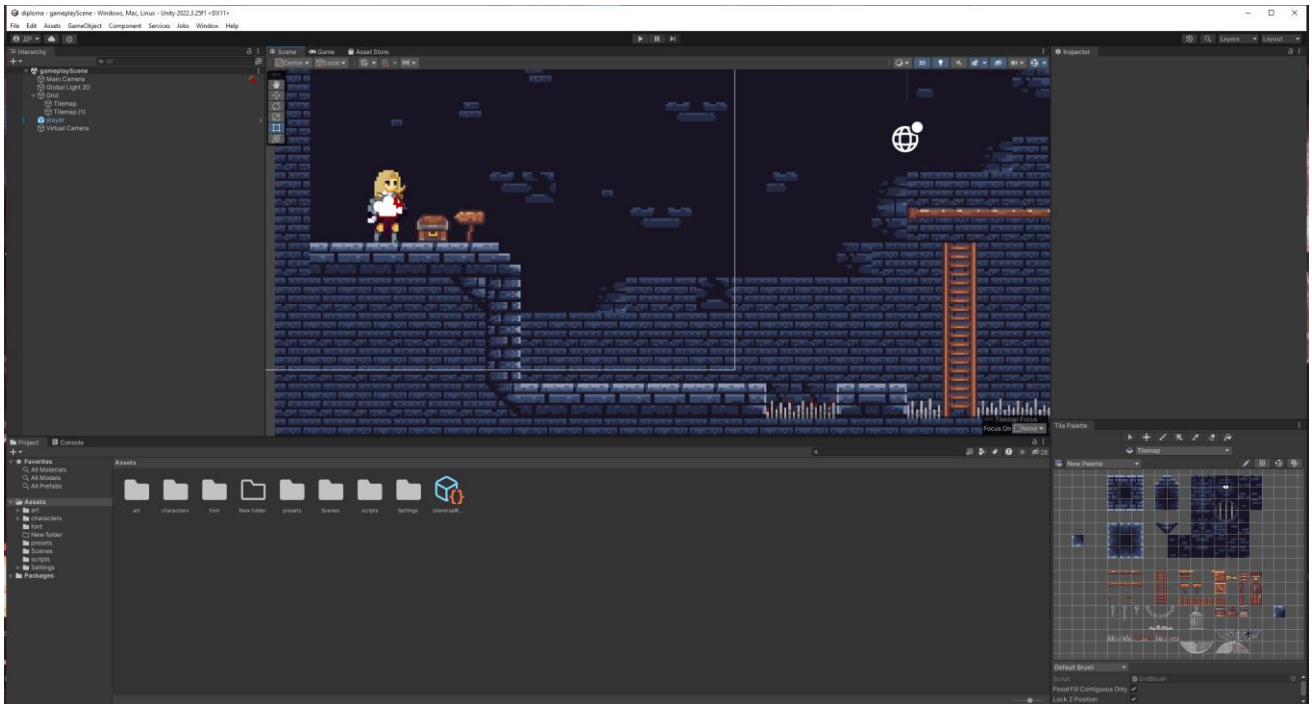


Рисунок 1.23. Скріншот зовнішнього виду рівня

Сам же набір плиток, завдяки було створено об’єкти на сцені виглядає так:

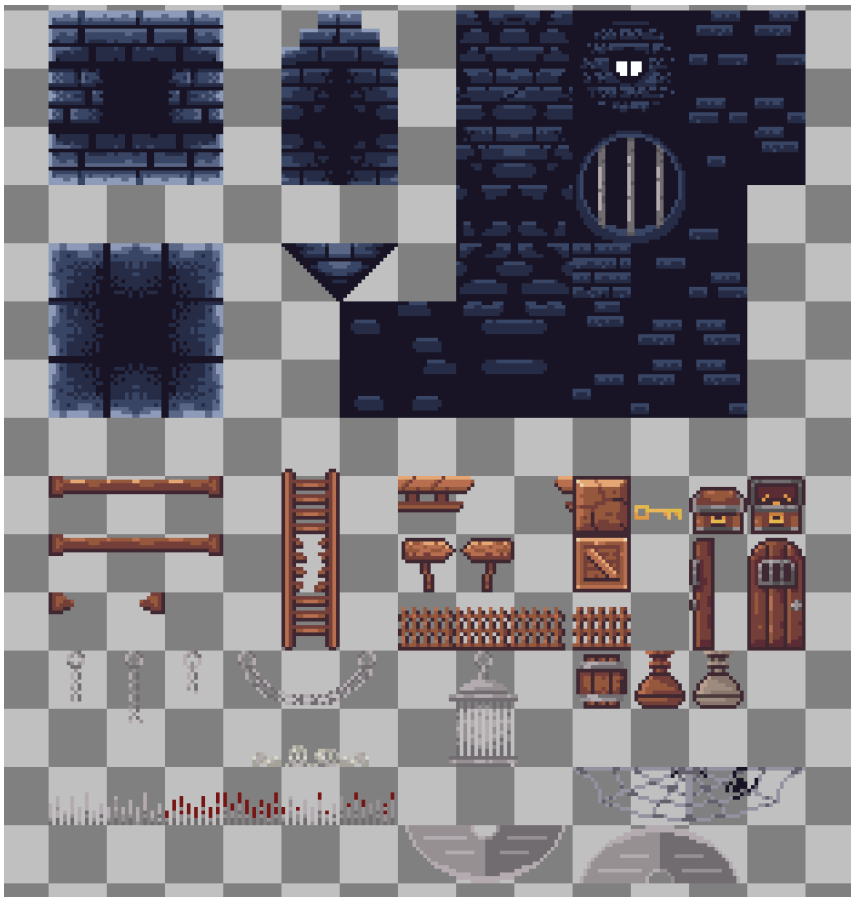


Рисунок 1.24. Структура плиток у додатку Aseprite

Як можна побачити, весь рівень був створений за допомогою плиток, що повторюються, за рахунок цього не потрібно створювати нове зображення для

кожного рівня, а також це допомагає грі виглядати більш цілісно.

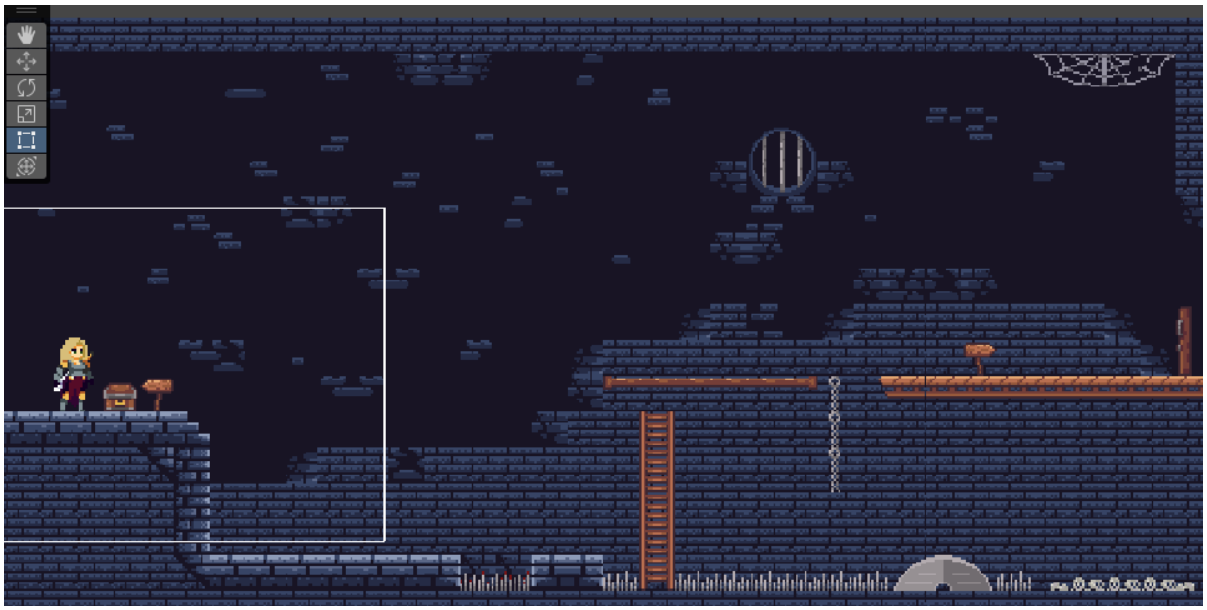


Рисунок 1.25 Скриншот рівня у редакторі Unity

Для виконання будь яких рухів, а також анімацій – персонажу необхідно додати скріпт. Будь який скріпт з використанням мови програмування C# починається з:

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

Це точка входу в код і додається для імпортування бібліотек у проект.

```
[RequireComponent(typeof(Rigidbody2D))]
```

Цей фрагмент коду гарантує, що будь-який GameObject, до якого приєднано цей сценарій, також повинен мати компонент Rigidbody2D. Якщо Rigidbody2D ще не приєднано, Unity автоматично додасть його, коли цей сценарій буде додано до GameObject.

```
 public class PlayerController : MonoBehaviour
{
    public float walkSpeed = 5f;
    Vector2 moveInput;
```

Ця частина визначає новий клас під назвою PlayerController, який успадковує MonoBehaviour, роблячи його компонентом, який можна приєднати до GameObjects в Unity. Також, він визначає швидкість персонажа та напрям його руху.

```
public float CurrentMoveSpeed
{
   get
   {
      if (IsMoving)
      {
         return walkSpeed;
      }
      else
      {
         return 0;
      }
   }
```

У цій частині коду задається швидкість руху персонажа і перевіряється код на правильність. Якщо дія «IsMoving» виконується, то персонаж набуває швидкість 5, а якщо дія не виконується, то швидкість персонажа залишається 0.

```
} [SerializeField]
   private bool _isMoving = false;
public bool IsMoving
{
   get
   {
      return _isMoving;
   }
   private set
   {
```

```
            _isMoving = value;
            animator.SetBool(AnimationStrings.isMoving, value);
        }
    }
```

У останній частині коду на персонажа накладається анімація руху за допомогою оператора animator.

Готовий інспектор персонажа наразі виглядає так:

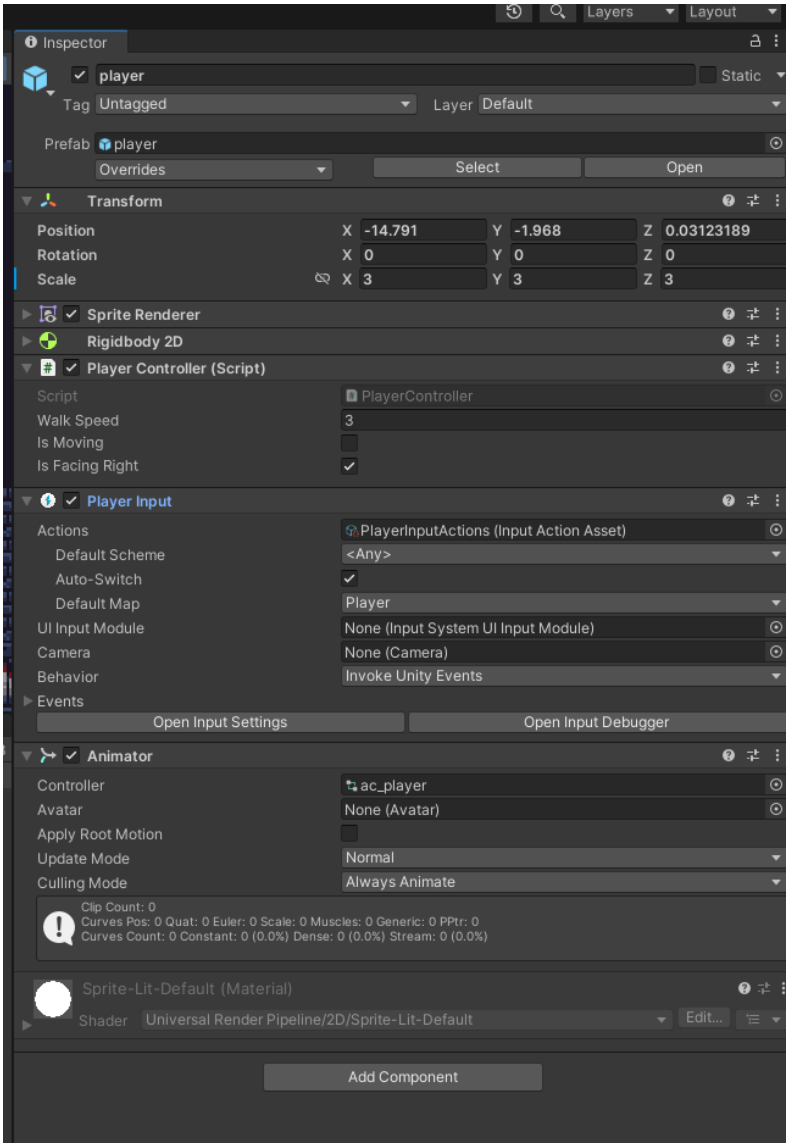


Рисунок 1.26. Інспектор персонажа

На цьому знімку можна побачити всі компоненти з яких складається логіка персонажа, його дії, колайдер та аніматор. Безпосередньо в інспекторі є можливість змінити швидкість персонажа, завдяки публічній змінній у коді.

Як можна бачити всі основні файли, з якими проводилась робота розміщені

у підкаталогах. Також до складу проекту входять файли ігрового програмного рушія Unity. Всі файли проекту було розподіллено по підкаталогам розділу проекту Assets.

Рисунок 1.27. Структура проекту гри.

У свій проект я також вирішила додати функцію атаки, для того, щоб захищатись від мобів, не зважаючи на те, що цей пункт не є обов'язковим у жанрі платформер і більшість персонажів залишаються «пацифістами». На рисунку 1.28 можна побачити замах мечем персонажа, перед атакою.

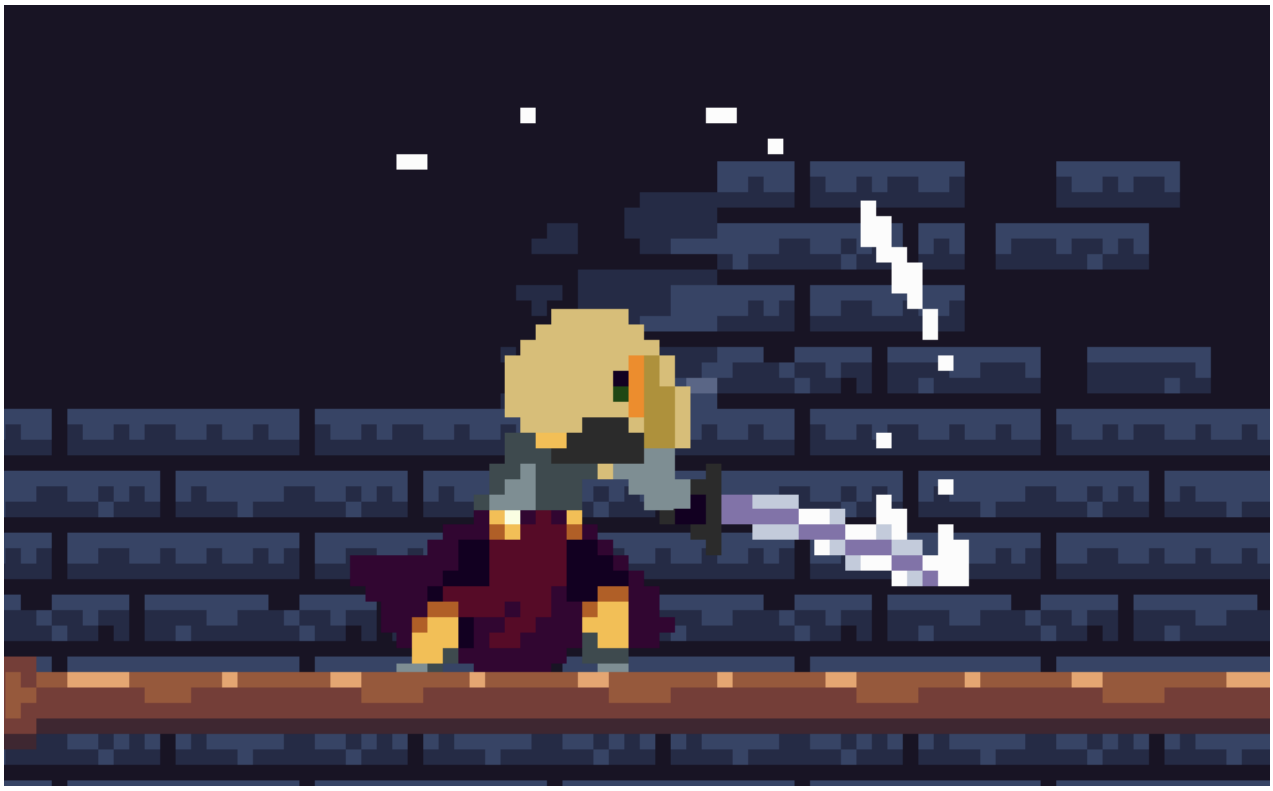


Рисунок 1.28. Скриншот персонажа в атаці

На сцені є драбина, тому, щоб показати її застосування я також додала до проекту лазіння. Це може стати в нагоді, коли далі в проекті драбини та цепи стануть єдиним способом добратись до кінця рівня.

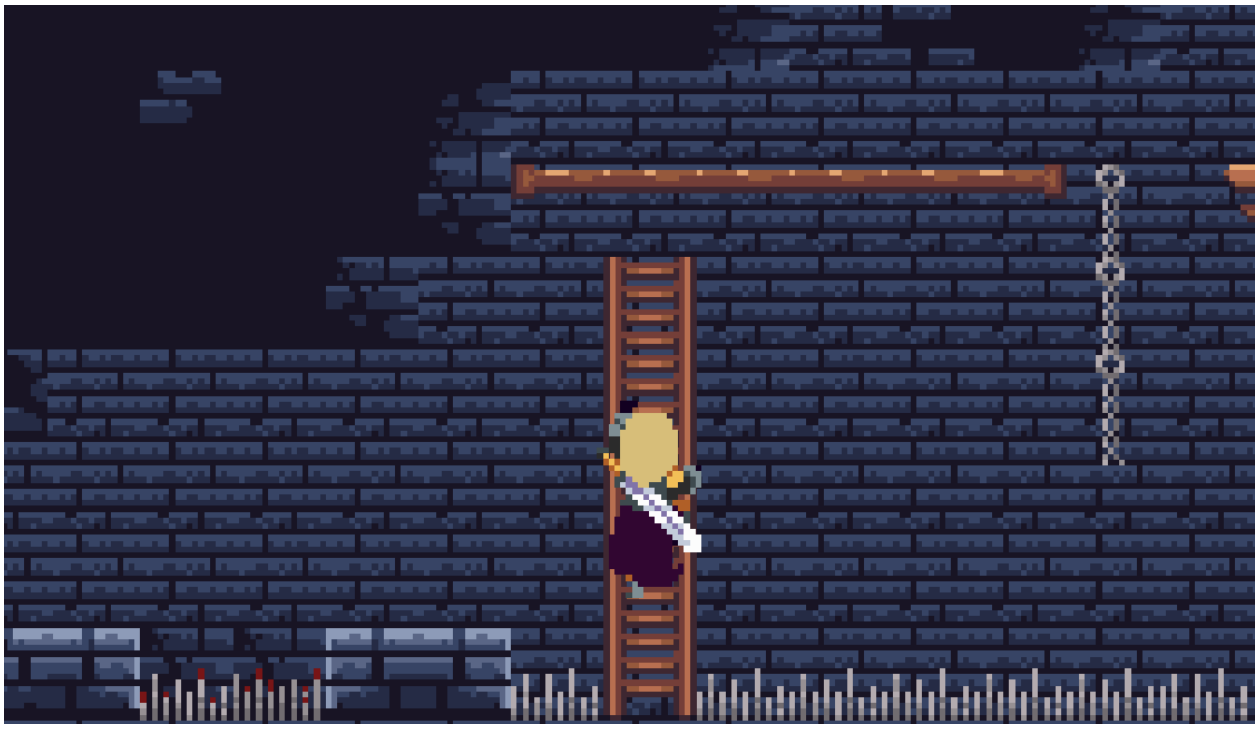


Рисунок 1.29. Анімація підіймання вверх драбиною

Як можна було помітити, у проекті встановлено не вид на всю сцену, а слідкування за гравцем. Це зроблено для того, щоб вороги могли з'являтись за межами сцени, щоб гравець не відволікався, а також щоб гра виглядала цікавішою, з безпосереднім фокусом на гравці.

Цей ефект було зроблено за допомогою «віртуальної камери» (Virtual Camera). Віртуальна камера, зокрема в 2D-іграх, зазвичай слідує за гравцем, щоб тримати його в центрі екрану або в межах видимого простору, що дозволяє гравцеві завжди бачити свого персонажа і мати адекватний огляд навколишнього середовища. Це важливо для платформерів, де гравець часто стрибає, бігає та уникає перешкод. Без належного стеження за гравцем, камера може залишити його поза полем зору, що негативно вплине на ігровий процес і може призвести до втрати контролю або дезорієнтації.

Щоб налаштувати камеру, яка слідує за гравцем у 2D-грі, я використовувала інструменти з пакету Cinemachine, доступного в Unity. Cinemachine надає розширені можливості для управління камерами, дозволяючи легко створювати кінематографічні ефекти без необхідності написання складного коду. Віртуальна камера з Cinemachine може бути налаштована для слідування за гравцем з певним відставанням або з певним вікном слідування,

що дозволяє гравцеві дещо рухатися по екрану без миттєвого пересування камери.

Для налаштування віртуальної камери з Cinemachine, спочатку необхідно додати пакет Cinemachine до проекту через менеджер пакетів Unity. Після цього слід додати компонент Cinemachine Virtual Camera до сцени та налаштувати її параметри. Важливим параметром є Follow, який потрібно встановити на об’єкт гравця. Це дозволить камері автоматично слідувати за гравцем у режимі реального часу. Додаткові налаштування, такі як Dead Zone, дозволяють визначити область на екрані, де гравець може рухатися без переміщення камери, створюючи більш плавний і природний рух.

Використання віртуальної камери в платформері дозволяє забезпечити постійний огляд гравця на ігровий світ, що сприяє кращому контролю персонажа і запобігає втраті орієнтації. Це особливо важливо в іграх, де точність і швидкість реакції гравця мають вирішальне значення для успіху. Крім того, віртуальна камера допомагає створити більш привабливий візуальний досвід, оскільки розробники можуть налаштувати камеру для створення певних кінематографічних ефектів, підвищуючи загальну естетичну привабливість гри.

Налаштування віртуальної камери в моєму проекті показані на рисунку 1.29. На ньому видно, що має пройти деякий час, перш ніж камера піде за персонажем, що створює гарний ефект плавної картинки.

Поточна реалізація проекту дає змогу зіграти у платформер із використанням фізики ігрового програмного рушія Unity, створити більш складні рівні та ігрові ситуації. Втім така реалізація також дає можливість розширювати ігровий процес.

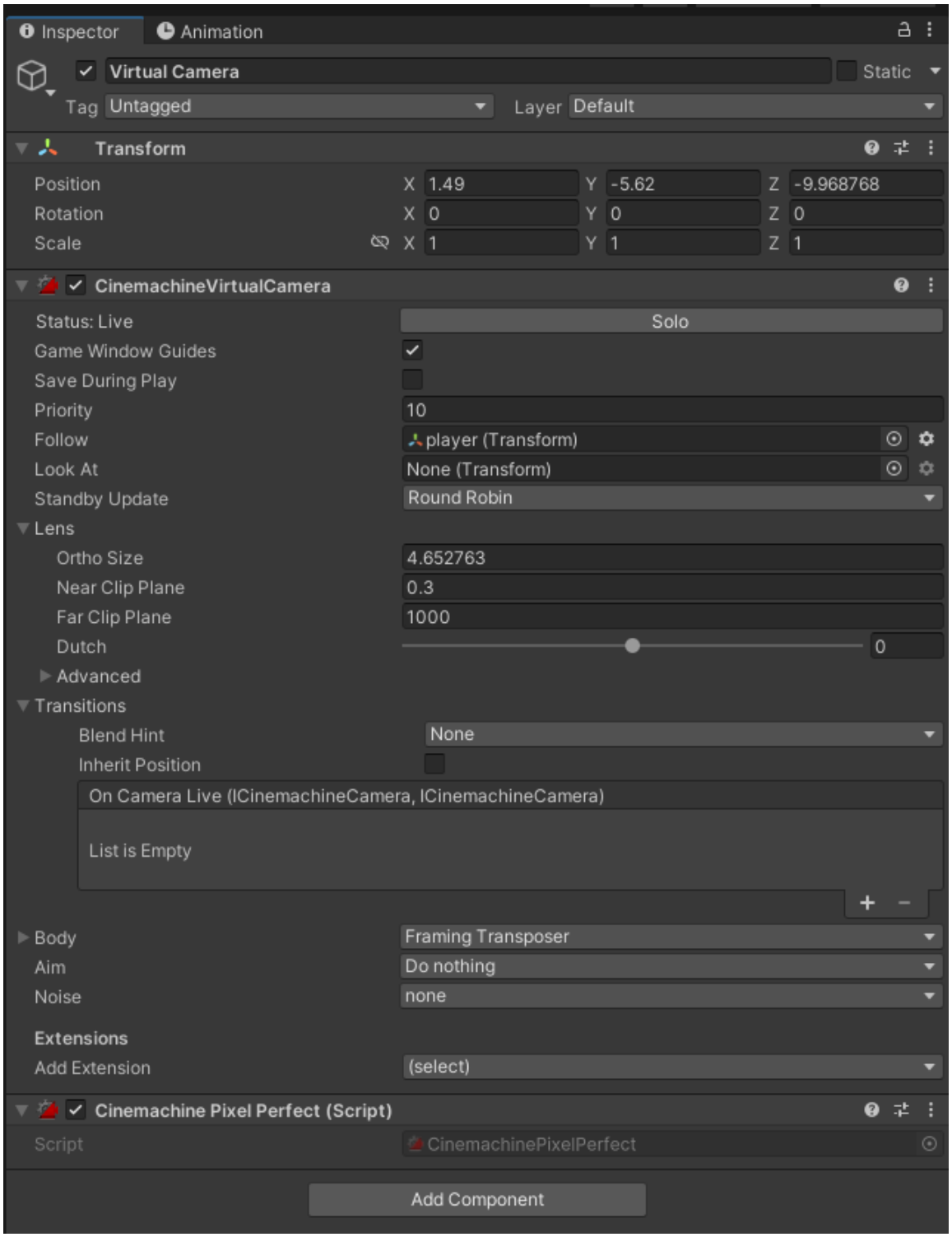


Рисунок 1.30. Налаштування віртуальної камери

Додавати нові типи платформ, або способи взаємодії із ними, а також у подальшому додавати нові режими гри, наприклад для гри на виділений час, або навпаки для створення рекордів та змагання між гравцями у мережі Інтернет.

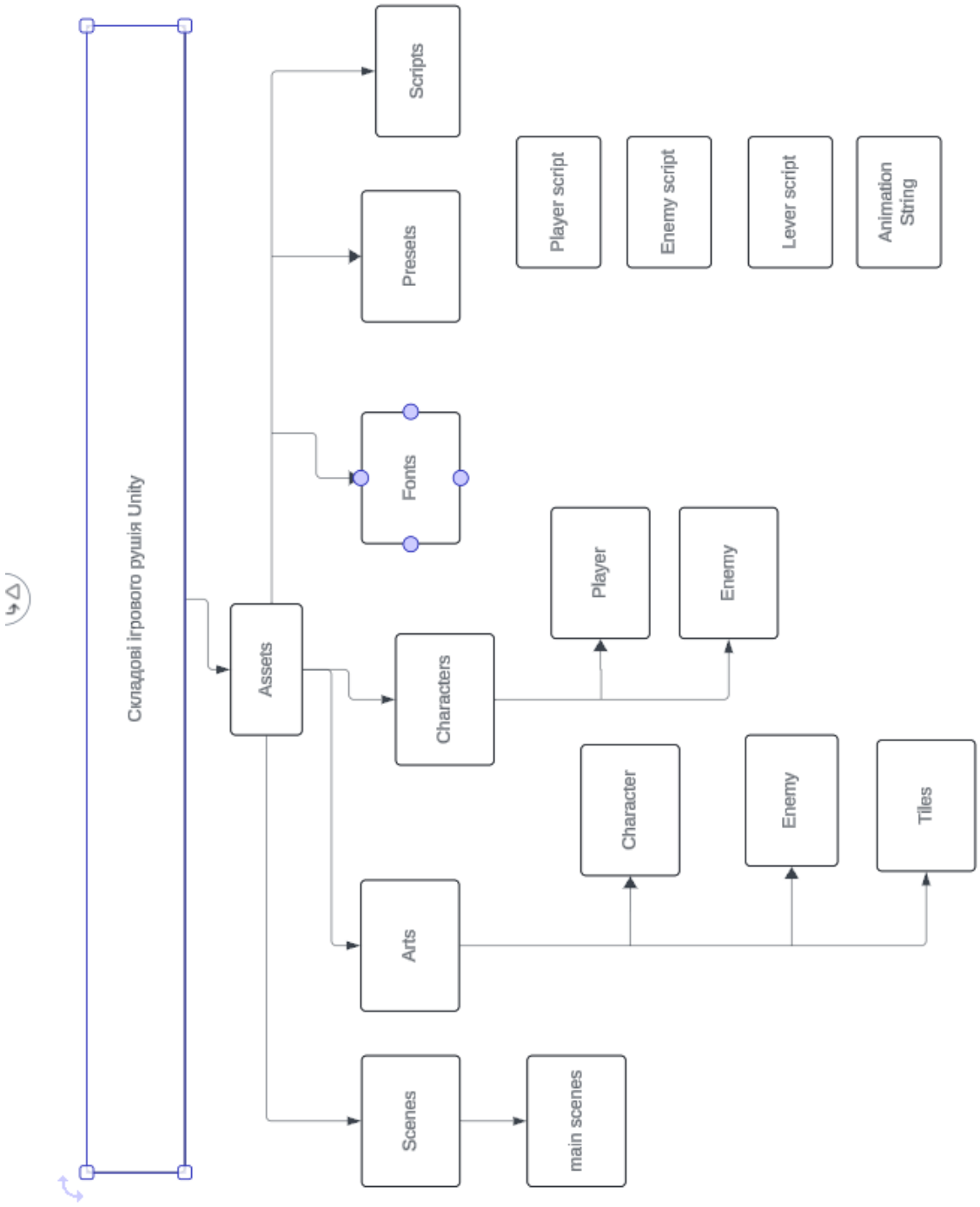


Рисунок 1.31. Розміщення файлів проекту по підкаталогам розділу Assets

1.7 Тестування працездатності ігрових елементів

Тестування проектів на платформі Unity є ключовим етапом у розробці ігор та інтерактивних додатків, що забезпечує їх високу якість та надійність. Важливість цього процесу можна розглядати через призму кількох аспектів, які включають виявлення та виправлення помилок, оптимізацію продуктивності,

забезпечення користувацького досвіду та дотримання стандартів якості.

По-перше, тестування дозволяє виявити і виправити помилки на ранніх етапах розробки, що значно знижує витрати на їх виправлення у майбутньому. Помилки, що залишаються непоміченими, можуть призвести до серйозних проблем у роботі готового продукту, включаючи падіння програми, некоректну роботу ігрової логіки, графічні артефакти та інші технічні несправності. Виявлення та виправлення таких проблем на ранніх етапах розробки допомагає забезпечити стабільну та безперебійну роботу кінцевого продукту.

По-друге, тестування сприяє оптимізації продуктивності проектів на Unity. Ігри та додатки, створені на цій платформі, часто потребують високої продуктивності для забезпечення плавного ігрового процесу та якісної графіки. Тестування дозволяє виявити вузькі місця у продуктивності, такі як занадто високе навантаження на процесор або графічний процесор, та знайти шляхи їх усунення. Це особливо важливо для забезпечення гарної роботи на різних апаратних платформах, включаючи мобільні пристрої, консолі та персональні комп'ютери.

Третім аспектом є забезпечення якісного користувацького досвіду. Високоякісна гра або додаток повинні бути інтуїтивно зрозумілими, зручними у використанні та привабливими для користувачів. Тестування користувацького інтерфейсу, геймплею та загальної взаємодії з програмою допомагає виявити недоліки у дизайні та реалізації, які можуть негативно вплинути на задоволення користувачів. Це включає перевірку логіки ігрових механік, забезпечення коректної роботи елементів управління та перевірку відповідності очікуванням користувачів.

Окрім того, тестування є необхідним для дотримання стандартів якості, які можуть бути визначені як внутрішніми політиками розробника, так і зовнішніми вимогами платформи. Unity, як платформа для розробки, часто використовується для створення комерційних продуктів, що вимагає дотримання певних стандартів якості для досягнення успіху на ринку. Тестування забезпечує відповідність проекту цим стандартам, що включає як технічні, так і естетичні аспекти.

Таким чином, тестування проектів на Unity є невід'ємною частиною процесу розробки, що забезпечує їх стабільність, продуктивність, привабливість для користувачів та відповідність стандартам якості. Ігнорування цього етапу може призвести до значних проблем у роботі готового продукту, що негативно вплине на його успіх на ринку та задоволення кінцевих користувачів.

При тестуванні свого проекту я виявила помилку з тим, що персонаж іноді провалюється у проміжки між платформами, але я оперативно виправила цю помилку і у фінальній версії проекту персонаж рухається правильно.

Для цього першим кроком я ретельно проаналізувала проблеми. Це включає відтворення помилки у різних сценаріях, щоб зрозуміти умови, за яких вона виникає. Зокрема, важливо було визначити, чи проблема проявляється у певних місцях рівня, чи вона носить загальний характер. Для цього можна використовувати різні інструменти дебагінгу, наявні в Unity, такі як логування подій або візуалізація колайдерів.

Після виявлення конкретних умов виникнення помилки, я перевірила налаштування фізичних колайдерів для платформ та персонажа. У Unity 2D платформи зазвичай використовують Box Collider 2D, який визначає межі зіткнень об'єктів. Важливо переконатися, що колайдери платформ щільно прилягають один до одного і не мають проміжків, через які персонаж може провалитися. Це може бути досягнуто шляхом коригування розмірів та положення колайдерів у редакторі Unity.

Наступним кроком є перевірка колайдера персонажа. Персонажі часто використовують Capsule Collider 2D або Box Collider 2D для визначення меж своїх зіткнень. Важливо переконатися, що колайдер персонажа має адекватні розміри і правильно налаштований центр мас, щоб уникнути непередбачуваних взаємодій з платформами. Можливо, знадобиться змінити розміри колайдера або налаштувати його офсет для досягнення стабільної взаємодії з поверхнями.

Окрім фізичних налаштувань, важливо було перевірити і код, який відповідає за рух персонажа. Управління персонажем у 2D грі часто реалізується через скрипти, що контролюють його положення, швидкість та анімацію. Важливо

переконатися, що код корректно обробляє зіткнення з платформами та належним чином реагує на зміни поверхонь. Це може включати перевірку логіки обробки подій зіткнень (OnCollisionEnter2D, OnCollisionStay2D, OnCollisionExit2D), а також коду, що відповідає за гравітацію та переміщення персонажа.

У деяких випадках проблему можуть вирішити додаткові механізми захисту від провалювання. Наприклад, можна додати допоміжний скрипт, який перевіряє положення персонажа і коригує його, якщо він знаходиться занадто близько до краю платформи або на межі провалювання. Це може бути реалізовано через невеликі коригування положення персонажа або шляхом додавання невидимих колайдерів, які запобігають провалюванню.

Нарешті, слід провести ретельне тестування після внесення змін, щоб переконатися, що проблема вирішена і більше не виникає у різних сценаріях гри. Це включає тестування на різних рівнях, з різними швидкостями руху персонажа та при різних умовах зіткнення з платформами.

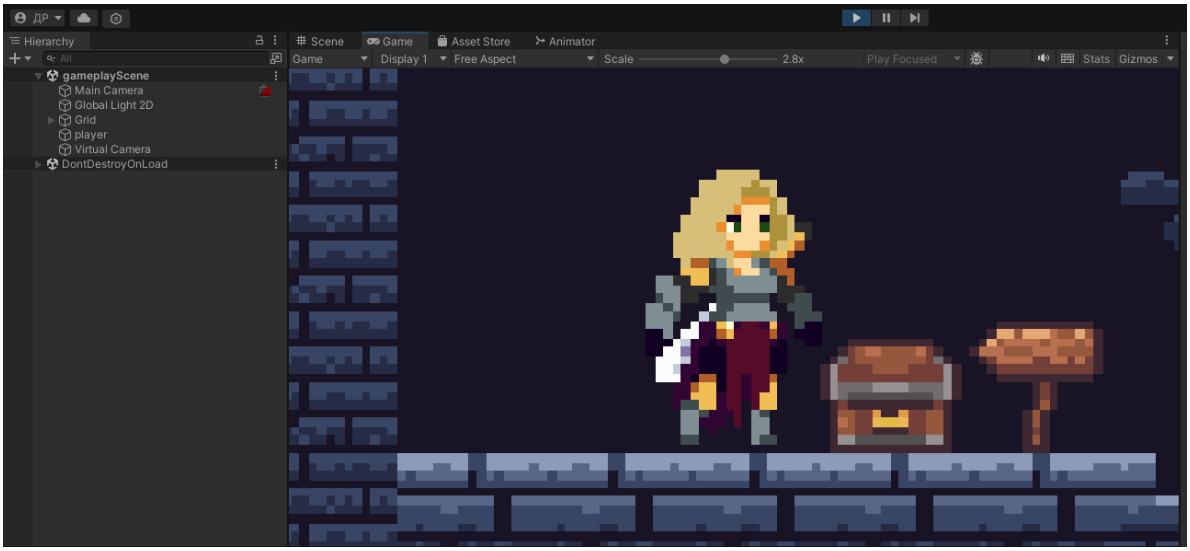


Рисунок 1.32. Фінальне тестування запуску гри

Персонаж більше не провалюється під платформи, а гра запускається без жодних інших помилок будь якого разу. При цьому, проект все ще може потребувати повторного тестування, на елемент знаходження у грі схованих помилок.

2 ЕКОНОМІЧНА ЧАСТИНА

2.1 Резюме

В рамках цього дипломного проекту була розроблена рання версія 2D-гри жанру платформер, а також повністю завершена робота над програмною частиною базового ігрового інтерфейсу, характерного для даного жанру ігор, і взаємодією з ігровим процесом. Використовуючи принципи модульної розробки, ми змогли внести цікаві аспекти в гру, додавши ігрові елементи, механізми або режими.

Розробка відеоігор є процесом, що вимагає значних фінансових інвестицій. Для успішного втілення проекту необхідно враховувати різноманітні економічні аспекти, включаючи витрати на розробку, маркетинг, а також прогнозовані доходи.

2.2 Визначення трудомісткості розробки програмного забезпечення

Визначення трудомісткості розробки програмного забезпечення є критичним етапом планування проекту, що дозволяє оцінити необхідні ресурси, строки виконання завдань та загальні витрати. Тривалість розробки програмного продукту залежить від його кількості, складності розробки, кваліфікації виконавця і планованого терміну, який визначається ринковими умовами.

У таблиці 2.1 наведені аналоги програмного забезпечення, і їх функції в більшій чи меншій мірі виконуються розробленими програмними продуктами. У нашій версії це виділено.

Таблиця 2.1 Каталог аналогів програмного забезпечення

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП імітаційного моделювання	7800 – 8800

При виборі аналога ПП з Vo в умовній машинної команді складність визначається на підставі таблиці 2.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманих значень встановлюється розширена норма часу на розробку програмних аналогів (коригується з поправочним коефіцієнтом з урахуванням умов розробки програмного забезпечення, тобто Кк =0,7÷0,8 в комп'ютерних умовах): Тар=229х0,7= 160,3 (люд/год).

Складність програмного продукту заснована на складності аналога з урахуванням складності розробки, ступеня новизни, ступеня використання при розробці стандартних модулів на основі формул:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.3)$$

Для розрахунку потрібні наступні коефіцієнти:

Li - питома вага на етапі розробки збірок (Таблиця 2.3); Кн - поправочний коефіцієнт з урахуванням ступеня новизни (таблиця 2.4); Кт-поправочний коефіцієнт з урахуванням ступеня його використання при розробці типових програм (таблиця 2.5).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

У нашій версії він виділений сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта з урахуванням ступеня новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

У нашій версії він виділений сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання при розробці стандартних програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

У нашій версії він виділений сірим кольором.

Тепер розрахуйте трудомісткість для кожного етапу окремо:

Сила робочої сили в технічних завданнях

$Т_{тз}=Та*L_1*К_н=160,3 *0,12*0,8= 15,3$(люд/годин)

Трудомісткість розробки технічного проекту

$Т_{тп}=Та*L_2*К_н=160,3 *0,11*0,8 = 14,1$ (люд/годин)

Трудомісткість розробки робочого проекту

$Т_{рп}=Та*L_3*К_н*К_т=160,3 *0,61*0,8*0,8= 62,5$ (люд/годин)

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}=2$ (стр), розробка ТП $N_{тп}= 22$ (стр), розробка робочого проекту $N_{рп}=18$ (стр), пояснювальна записка відповідно $N_{пз}=40$ (стр). Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=15,38	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}=0,7*N_{тп}$=0,7*22= 15,4	$T_{нк}=0,15*N_{тп}$=0,15*22=3,3
3.Розробка РП	$T_{Ррп}$= 62,58	$T_{кк}=0,7*N_{рп}$=0,7*18=12, 6	$T_{нк}=0,15*N_{рп}$=0,15*18=2,7
4.Розробка ПЗ	$T_{пз}=1,5**N_{пз}$ = 1,5*40 =60	$T_{кк}=0,7*N_{тз}$=0,7*40 =28	$T_{нк}=0,15*N_{пз}$=0,15*40 =6
Усього, в т.ч.:	227,37		
на розробку	$\Sigma T_р$=152,07		
контроль керівника		$\Sigma T_{кк}$=57,4	
нормконтроль			$\Sigma T_{нк}$=12,3

2.3 Розрахунок ціни програмного продукту

Визначення ціни програмного продукту є ключовим етапом у процесі його комерціалізації. Розрахунок базової заробітної плати виконавця наведено в таблиці 2.7. Стаття 8 Закону Про державний бюджет України на 2024 рік. Відповідно до статті про декомунізацію мінімальна щомісячна заробітна плата встановлена на рівні 8000 гривень в період з 2024; мінімальна погодинна тарифна ставка становить 48 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	152,07	48,00	7299,36
2.Контроль керівника	57,4	68,10	3908,94
3.Нормоконтроль	12,3	68,10	837,63
Усього	-	-	ΣЗо= 12045,93

Розрахувати матеріальні витрати на розробку ПП. Розрахунки наведені в таблиці 2.8:

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПП

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	76	3	228
Разом	-	-	-	$В_{м\,і}$=228,0
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1*189=22,80
Усього				$В_м=В_{мі}+В_{тр\_з}$= 250,80

На підставі даних, отриманих за окремими статтями витрат, розрахунок планових витрат за ПП в цілому узагальнено у форматі, наведеному в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	250,80	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	12045,93	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	1204,59	$Зд = 0{,}1 \times Зо$ = 12045,93*0,1
4.Відрахування до єдиного фонду соціального внеску	1204,59	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (12045,93+1204,59)
5. Накладні витрати	4818,37	$Внак. = 0{,}4 \times Зо$ =0,4 * 12045,93
6. Повна собівартість	19524,28	$С_{пов}= В_м +З_о + З_д +$Вє.с.в. + Внак.= 250,80+12045,93+1204,59+1204,59+4818,37

Розмір включеної в ціну прибутку визначається за такою формулою:

$П=(С_п*Р)/100=(19524,28*10)/100=1952,42$грн

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_п+П$ = 19524,28+1952,42= 21476,7 грн;

Виходячи з отриманих даних, ціна продажу продукту, буде:

$Ц_р=Ц_о+$ПДВ = 21476,7 + (21476,7 *0.2) = 25772,04 грн;

3 ОХОРОНА ПРАЦІ

3.1. Вступ

Охорона праці є важливим чинником на підприємстві. Вона відіграє важливу роль у безпечному перебуванню працівників на їх робочому місці, бо наскільки б не були важливими здобутки працівників, вони ніколи не зможуть повернути втраченого здоров’я, або життя. Тому ми повинні пам’ятати, що нещасні випадки стаються всюди, навіть, з людьми, які проходять особливу підготовку.

Охорона праці програмісту потрібна для забезпечення безпечних та здорових умов праці, що сприяє підвищенню продуктивності та запобіганню професійним захворюванням та травмам.

В сучасному світі розробка комп'ютерних ігор стає все більш популярною сферою діяльності. Програмісти проводять багато часу за комп'ютером, що може негативно впливати на їхнє здоров'я та працездатність. Також така робота спричиняє значну розумову та нервово-емоційну напругу, навантаження на зоровий апарат і м’язів кістки зап’ястка, п’ястка та фаланги пальців. Важливо мати конструкцію для роботи, яка розташовує всі елементи робочого місця так, щоб робоча поза людини-оператора не була викривлена чи приносила дискомфорт. Тому важливо створювати оптимальні умови праці для програмістів, аби мінімізувати ризики, пов'язані з їхньою професією, а згідно зі статтею 13 Закону України «Про охорону праці» роботодавець зобов’язаний створити на кожному робочому місці у всіх структурних підрозділах умови праці відповідно до нормативно-правових актів, також забезпечити додержання і підтримання вимог законодавства щодо прав працівників.

Цей розділ присвячено визначенню оптимальних умов праці програміста(оператора) , а також його обов'язків з охорони праці при розробці комп'ютерної 2D-гри у жанрі платформер на програмному рушії Unity.

Охорона праці для програміста є важливою складовою створення безпечного та здорового робочого середовища. Цей розділ містить рекомендації, які допомогли забезпечити належні умови праці, включаючи аспекти пожежної

безпеки, під час створення даного проекту дипломного проектування.

3.2 Пожежна безпека

3.2.1 Загальні положення

Пожежна безпека є ключовим аспектом охорони праці, що передбачає запобігання виникненню пожеж, забезпечення ефективної евакуації та мінімізацію наслідків у разі їх виникнення. Основні правила пожежної безпеки для програмістів включають:

3.2.2 Профілактичні заходи

Контроль за електроприладами: Перевіряти справність електроприладів та кабелів. Забороняється використовувати пошкоджені або несправні прилади.

Організація робочого простору: Уникати накопичення горючих матеріалів поблизу робочого місця. Використовувати спеціальні стелажі та контейнери для зберігання паперів та інших матеріалів.

Безпечне використання електрики: Не перевантажувати електричні розетки. Використовувати подовжувачі та розгалужувачі, що відповідають технічним стандартам.

3.2.3. Засоби пожежогасіння

Наявність вогнегасників: У приміщенні повинні бути встановлені та розміщені у доступних місцях вогнегасники. Вони мають регулярно перевірятися на справність.

Навчання персоналу: Кожен співробітник має бути обізнаний із місцем розташування та правилами користування вогнегасниками, а також іншими засобами пожежогасіння.

3.2.4. Евакуація

Плани евакуації: У кожному приміщенні повинні бути розміщені схеми евакуації, що вказують на маршрути виходу у разі пожежі.

Відкриті евакуаційні виходи: Всі евакуаційні виходи повинні бути вільними від перешкод і чітко позначеними. Двері на шляхах евакуації мають легко відчинятися зсередини.

Регулярні тренування: Організовувати регулярні тренування з евакуації

для всіх співробітників, щоб забезпечити швидку та безпечну евакуацію у разі пожежі.

3.2.5. Дії у разі пожежі

Сигналізація: Негайно повідомити про пожежу за допомогою системи оповіщення або іншими доступними засобами.

Евакуація: Організовано залишити приміщення згідно з планом евакуації, не користуючись ліфтами.

Захист документів та обладнання: За можливості, вимкнути електричні прилади та забезпечити захист важливих документів.

Особливу увагу треба приділити освітленню так як, саме воно впливає не тільки на збереження здоров'я, а і на якість праці та продуктивність працівників. Відповідно до ДБН В.2.5-28:2018 «Природне і штучне освітлення» приміщення має мати, що природнє, що штучне освітлення, що рівномірно розподілено. Природне освітлення має забезпечувати коефіцієнт природної освітленості (КПО) не нижче ніж 1,5%. Для регулювання рівня освітлення природним світлом бажано застосовувати жалюзі. Також обладнене робоче місце має бути розміщено так, щоб уникнути попадання прямого сонячного світла у очі. Застосування світильників екрануючих сіток, або без розсіювачів забороняється. В свою чергу рівень освітленості на робочому столі та в зоні зберігання документів має становити 300-500 лк.

3.3 Мікроклімат

У виробничих приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату, а саме: температура, відносна вологість та рухливість повітря за ДСН 3.3.6.042-99 «і норми мікроклімату виробничих приміщень». Робочі приміщення мають бути обладнені системою опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. Також кількість теплоти, що генерується системою опалення, иає бути відповідною до втрат теплоти у приміщенні.

При недотриманні вказаних показників мікроклімату в офісних приміщеннях робочий день для робітників повинен бути скорочений мінімум на

10%.

3.4 Електромагнітні випромінювання

Рівень неіонізуючих електромагнітних випромінювань, електростатичних і магнітних полів, а також інтенсивність потоків інфрачервоного та ультрафіолетового випромінювань на робочих місцях повинні відповідати санітарно-гігієнічним вимогам, встановленим у ДСанПіН 3.3.2.007-98 "Державні санітарні норми та правила гігієни праці при роботі з візуальними дисплейними терміналами" та ДСанПіН 3.3.6.096-2002 "Державні санітарні норми та правила щодо фізичних факторів у робочому середовищі ". Важливо зазначити, що дані норми встановлюють гранично допустимі рівні (ГДР) та допустимі рівні (ДР) впливу неіонізуючого випромінювання на людину. Контроль за дотриманням санітарно-гігієнічних вимог до неіонізуючого випромінювання на робочих місцях здійснюється органами державного санітарно-епідеміологічного нагляду.

3.5 Вимоги до робочого місця працівника та його організації

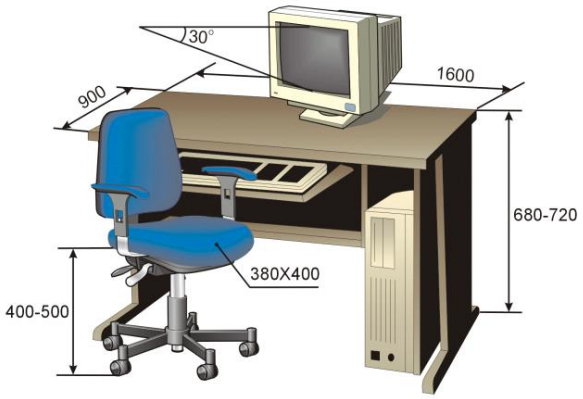
Робоча зона: Робоче місце програміста має бути розташоване у добре освітленому та вентильованому приміщенні з достатньою кількістю простору для зручного розміщення обладнання та виконання робіт.

Ергономіка: Робочий стіл, стілець та комп'ютерне обладнання повинні бути налаштовані відповідно до ергономічних вимог, щоб мінімізувати ризик виникнення м'язово-скелетних захворювань.

Працівник, який допускається до роботи з комп’ютером має пройти медичний огляд, навчання за фахом згідно якого хоче працювати, вступний та первинний інструктаж з охорони праці на робочому місці. Надалі кожні пів року вони проходять повторні інструктажі з охорони праці на робочому місці та медичні огляди раз на два роки.

Робочі місця мають бути організовані так, щоб у поле зору працівника не потрапляли відбиваючі поверхні, вікна та освітлювальні прилади. Неприпустимо розташовувати відео-дисплейні термінали так, щоб працівник був повернутий обличчям або спиною до вікон, незалежно від способу освітлення, будь то пряме чи відбите світло. На рисунку 3.1 представлена схема регулювання робочого столу програміста.

Рисунок 3.1. Схема регулювання робочого столу програміста.



Робочий стіл має регулюватися по висоті в межах 68-80 см, а ширина столу повинна забезпечувати можливість виконання операцій в зоні досяжності моторного поля.

Рекомендовані розміри столу: висота 725 мм, ширина 600-1400 мм, глибина 800-1000 мм. Кожен параметр регулювання має бути легко налаштованим, незалежним і надійно фіксуватися.

Розташування екрана повинно забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30° до нормальної лінії погляду працівника. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до працівника.

Не слід також забувати про «спеціальне харчування» для очей. Варто вживати продукти, що зміцнюють судини сітківки ока: чорниці, чорну смородину, моркву. Корисні для очей також вітаміни (особливо комплексні полівітаміни, у яких вітаміни сполучаються з мікроелементами: цинком, кальцієм).

ВИСНОВКИ

Для виконання дипломного проекту було обрано жанр 2D платформер Розробка 2D-гри у жанрі платформеру на програмному рушії Unity є важливим проектом з кількох причин. По-перше, цей проект демонструє здатність застосовувати сучасні інструменти та технології у створенні інтерактивних програмних продуктів, що є актуальним для індустрії розробки ігор. По-друге, робота над проектом дозволяє розширити практичні навички у програмуванні, дизайні графіки та управлінні проектами, що сприяє моєму професійному зростанню.

У процесі розробки гри було використано кілька важливих інструментів. Aseprite став ключовим інструментом для створення і редагування піксельної графіки, що є основним візуальним елементом у 2D-платформерах. Завдяки Aseprite було можливо створити якісні спрайти, що додають грі унікальний художній стиль.

Visual Studio було використано як основне середовище розробки. Цей інструмент забезпечив зручність написання коду, налагодження та тестування проекту. Можливості Visual Studio, включаючи інтелектуальне завершення коду та інтеграцію з Unity, дозволили підвищити продуктивність та якість розробки.

Unity виступив основною платформою для розробки гри. Цей програмний рушій забезпечив широкі можливості для реалізації ігрової логіки, фізики та анімації. Unity надає потужні інструменти для управління проектом, що включають редактор сцен, систему компонентів та модуль для обробки подій, що дозволило створити комплексну та інтерактивну гру.

Проект був написаний мовою програмування C#, яка є основною мовою для розробки в Unity. C# забезпечує високу продуктивність та багаті можливості для об’єктно-орієнтованого програмування. Використання цієї мови дозволило ефективно реалізувати складні ігрові механіки. Цей проект став важливим кроком у професійному розвитку, надавши цінний досвід та знання, необхідні для подальшої кар'єри у сфері розробки ігор.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Дікінсон К. Оптимізація ігор у UNITY 5. BIRMINGHAM – MUMBAI, 2017. 306 с.
2. Мэннинг Д. , Батфилд-эддисон П. Unity для розробника. Мобільні мультиплатформенні ігри. Бостон, 2018. 352 с.
3. Wikipedia: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Unity_(ігровий_рушій) (дата звернення: 01.06.2024).
4. Arstechnica: [Веб-сайт]. URL: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/ (дата звернення: 01.06.2024).
5. Unity: [Веб-сайт]. URL: https://unity.com/solutions/government-aerospace (дата звернення: 01.06.2024).
6. Албахарі Б. , Албахарі Д. C# 10 Pocket Reference. Бостон. 270 с.
7. Wikipedia: [Веб-сайт]. URL: https://en.wikipedia.org/wiki/Unreal_Engine (дата звернення: 01.06.2024).
8. Guinnessworldrecords: [Веб-сайт]. URL: https://www.guinnessworldrecords.com/world-records/most-successful-game-engine (дата звернення: 01.06.2024).
9. Gamespot: [Веб-сайт]. URL: https://www.gamespot.com/articles/unreal-engine-5-gets-stunning-demo-with-incredible-graphics-enters-early-access/1100-6491998/ (дата звернення: 01.06.2024).

ДОДАТОК А. Лістинг коду основних модулів гри мовою C#

Скрипт AnimationString.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

internal class AnimationStrings
{
    internal static string isMoving = "isMoving";
    internal static string isGrounded = "isGrounded";
    internal static string yVelocity = "yVelocity";
    internal static string jumpTrigger = "jump";
    internal static string isOnWall = "isOnWall";
    internal static string isOnCeiling = "isOnCeiling";
    internal static string attackTrigger = "attack";
    internal static string canMove = "canMove";
    internal static string hasTarget = "hasTarget";
    internal static string isAlive = "isAlive";
    internal static string isHit = "isHit";
    internal static string hitTrigger = "hit";
    internal static string lockVelocity = "lockVelocity";
    internal static string attackCooldown = "attackCooldown";
    internal static string rangedAttackTrigger = "rangedAttack";
}
```

Скрипт Attack.cs

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

public class Attack : MonoBehaviour
{
    public int attackDamage = 10;
    public Vector2 knockback = Vector2.zero;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // See if it can be hit
        Damageable damageable = collision.GetComponent<Damageable>();

        if(damageable != null)
        {
            // If parent is facing the left by localscale, our knockback x flips its value to face the left as
well
            Vector2 deliveredKnockback = transform.parent.localScale.x > 0 ? knockback : new
Vector2(-knockback.x, knockback.y);
```

```
            // Hit the target
            bool gotHit = damageable.Hit(attackDamage, deliveredKnockback);

            if(gotHit)
                Debug.Log(collision.name + " hit for " + attackDamage);
        }
    }
}
```

Скрипт Damageable.cs

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

public class Damageable : MonoBehaviour
{
    public UnityEvent<int, Vector2> damageableHit;
    public UnityEvent damageableDeath;
    public UnityEvent<int, int> healthChanged;

    Animator animator;

    [SerializeField]
    private int _maxHealth = 100;

    public int MaxHealth
    {
        get
        {
            return _maxHealth;
        }
        set
        {
            _maxHealth = value;
        }
    }

    [SerializeField]
    private int _health = 100;

    public int Health
    {
        get
        {
            return _health;
        }
        set
        {
            _health = value;
            healthChanged?.Invoke(_health, MaxHealth);
```

```
            // If health drops below 0, character is no longer alive
            if(_health <= 0)
            {
                IsAlive = false;
            }
        }
    }

    [SerializeField]
    private bool _isAlive = true;

    [SerializeField]
    private bool isInvincible = false;

    private float timeSinceHit = 0;
    public float invincibilityTime = 0.25f;

    public bool IsAlive {
        get
        {
            return _isAlive;
        }
        set
        {
            _isAlive = value;
            animator.SetBool(AnimationStrings.isAlive, value);
            Debug.Log("IsAlive set " + value);

            if(value == false)
            {
                damageableDeath.Invoke();
            }
        }
    }

    // The velocity should not be changed while this is true but needs to be respected by other physics components like
    // the player controller
    public bool LockVelocity
    {
        get
        {
            return animator.GetBool(AnimationStrings.lockVelocity);
        }
        set
        {
            animator.SetBool(AnimationStrings.lockVelocity, value);
        }
    }
```

```
    private void Awake()
    {
        animator = GetComponent<Animator>();
    }

    private void Update()
    {
        if(isInvincible)
        {
            if(timeSinceHit > invincibilityTime)
            {
                // Remove invincibility
                isInvincible = false;
                timeSinceHit = 0;
            }

            timeSinceHit += Time.deltaTime;
        }
    }

    // Returns whether the damageable took damage or not
    public bool Hit(int damage, Vector2 knockback)
    {
        if(IsAlive && !isInvincible)
        {
            Health -= damage;
            isInvincible = true;

            // Notify other subscribed components that the damageable was hit to handle the knockback
and such
            animator.SetTrigger(AnimationStrings.hitTrigger);
            LockVelocity = true;
            damageableHit?.Invoke(damage, knockback);
            CharacterEvents.characterDamaged.Invoke(gameObject, damage);

            return true;
        }

        // Unable to be hit
        return false;
    }

    // Returns whether the character was healed or not
    public bool Heal(int healthRestore)
    {
        if(IsAlive && Health < MaxHealth)
        {
            int maxHeal = Mathf.Max(MaxHealth - Health, 0);
            int actualHeal = Mathf.Min(maxHeal, healthRestore);
```

```
            Health += actualHeal;
            CharacterEvents.characterHealed(gameObject, actualHeal);
            return true;
        }

        return false;
    }
}
```

Скрипт Detection.cs

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

public class Detection : MonoBehaviour
{
    public UnityEvent noCollidersRemain;

    public List<Collider2D> detectedColliders;
    Collider2D col;

    private void Awake()
    {
        col = GetComponent<Collider2D>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        detectedColliders.Add(collision);
    }


    private void OnTriggerExit2D(Collider2D collision)
    {
        detectedColliders.Remove(collision);

        if(detectedColliders.Count <= 0)
        {
            noCollidersRemain.Invoke();
        }
    }
}
```

Скрипт PlayerController.cs

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

[RequireComponent(typeof(Rigidbody2D), typeof(TouchingDirections), typeof(Damageable))]
```

```
public class PlayerController : MonoBehaviour
{
    public float walkSpeed = 5f;
    public float runSpeed = 8f;
    public float airWalkSpeed = 3f;
    public float jumpImpulse = 10f;
    Vector2 moveInput;
    TouchingDirections touchingDirections;
    Damageable damageable;

    public float CurrentMoveSpeed { get
        {
            if(CanMove)
            {
                if (IsMoving && !touchingDirections.IsOnWall)
                {
                    if (touchingDirections.IsGrounded)
                    {
                        if (IsRunning)
                        {
                            return runSpeed;
                        }
                        else
                        {
                            return walkSpeed;
                        }
                    }
                    else
                    {
                        // Air Move
                        return airWalkSpeed;
                    }
                }
                else
                {
                    // Idle speed is 0
                    return 0;
                }
            } else
            {
                // Movement locked
                return 0;
            }

        } }

    [SerializeField]
    private bool _isMoving = false;

    public bool IsMoving { get
```

```
    {
        return _isMoving;
    }
    private set
    {
        _isMoving = value;
        animator.SetBool(AnimationStrings.isMoving, value);
    }
}

[SerializeField]
private bool _isRunning = false;

public bool IsRunning
{
    get
    {
        return _isRunning;
    }
    set
    {
        _isRunning = value;
        animator.SetBool(AnimationStrings.isRunning, value);
    }
}

public bool _isFacingRight = true;

public bool IsFacingRight { get { return _isFacingRight;  } private set {
        // Flip only if value is new
        if(_isFacingRight != value)
        {
            // Flip the local scale to make the player face the opposite directino
            transform.localScale *= new Vector2(-1, 1);
        }

        _isFacingRight = value;
    } }

public bool CanMove { get
    {
        return animator.GetBool(AnimationStrings.canMove);
    }
}

public bool IsAlive {
    get
    {
        return animator.GetBool(AnimationStrings.isAlive);
    }
```

```
}

Rigidbody2D rb;
Animator animator;

private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    touchingDirections = GetComponent<TouchingDirections>();
    damageable = GetComponent<Damageable>();
}

private void FixedUpdate()
{
    if(!damageable.LockVelocity)
        rb.velocity = new Vector2(moveInput.x * CurrentMoveSpeed, rb.velocity.y);

    animator.SetFloat(AnimationStrings.yVelocity, rb.velocity.y);
}

public void OnMove(InputAction.CallbackContext context)
{
    moveInput = context.ReadValue<Vector2>();

    if(IsAlive)
    {
        IsMoving = moveInput != Vector2.zero;

        SetFacingDirection(moveInput);
    }
    else
    {
        IsMoving = false;
    }

}

private void SetFacingDirection(Vector2 moveInput)
{
    if(moveInput.x > 0 && !IsFacingRight)
    {
        // Face the right
        IsFacingRight = true;
    }
    else if (moveInput.x < 0 && IsFacingRight)
    {
        // Face the left
        IsFacingRight = false;
    }
```

```
    }

    public void OnRun(InputAction.CallbackContext context)
    {
        if (context.started)
        {
            IsRunning = true;
        } else if(context.canceled)
        {
            IsRunning = false;
        }
    }

    public void OnJump(InputAction.CallbackContext context)
    {
        // TODO Check if alive as well
        if(context.started && touchingDirections.IsGrounded && CanMove)
        {
            animator.SetTrigger(AnimationStrings.jumpTrigger);
            rb.velocity = new Vector2(rb.velocity.x, jumpImpulse);
        }
    }

    public void OnAttack(InputAction.CallbackContext context)
    {
        if(context.started)
        {
            animator.SetTrigger(AnimationStrings.attackTrigger);
        }
    }

    public void OnRangedAttack(InputAction.CallbackContext context)
    {
        if (context.started)
        {
            animator.SetTrigger(AnimationStrings.rangedAttackTrigger);
        }
    }

    public void OnHit(int damage, Vector2 knockback)
    {
        rb.velocity = new Vector2(knockback.x, rb.velocity.y + knockback.y);
    }
}
```

Скрипт Projectile.cs

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

public class Projectile : MonoBehaviour
```

```
{
    public int damage = 10;
    public Vector2 moveSpeed = new Vector2(3f, 0);
    public Vector2 knockback = new Vector2(0, 0);

    Rigidbody2D rb;

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

    // Start is called before the first frame update
    void Start()
    {
        // If you want the projectile to be effected by gravity by default, make it dynamic mode rigidbody
        rb.velocity = new Vector2(moveSpeed.x * transform.localScale.x, moveSpeed.y);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Damageable damageable = collision.GetComponent<Damageable>();

        if(damageable != null)
        {
            // If parent is facing the left by localscale, our knockback x flips its value to face the left as well
            Vector2 deliveredKnockback = transform.localScale.x > 0 ? knockback : new Vector2(-knockback.x, knockback.y);

            // Hit the target
            bool gotHit = damageable.Hit(damage, deliveredKnockback);

            if (gotHit)
                Debug.Log(collision.name + " hit for " + damage);
                Destroy(gameObject);
        }
    }
}
```

Скрипт TouchingDirections.cs

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

// Uses the collider to check directions to see if the object is currently on the ground, touching the wall, or touching the ceiling
public class TouchingDirections : MonoBehaviour
{
```

```
    public ContactFilter2D castFilter;
    public float groundDistance = 0.05f;
    public float wallDistance = 0.2f;
    public float ceilingDistance = 0.05f;

    CapsuleCollider2D touchingCol;
    Animator animator;

    RaycastHit2D[] groundHits = new RaycastHit2D[5];
    RaycastHit2D[] wallHits = new RaycastHit2D[5];
    RaycastHit2D[] ceilingHits = new RaycastHit2D[5];

    [SerializeField]
    private bool _isGrounded;

    public bool IsGrounded { get {
            return _isGrounded;
        } private set {
            _isGrounded = value;
            animator.SetBool(AnimationStrings.isGrounded, value);
        } }

    [SerializeField]
    private bool _isOnWall;

    public bool IsOnWall
    {
        get
        {
            return _isOnWall;
        }
        private set
        {
            _isOnWall = value;
            animator.SetBool(AnimationStrings.isOnWall, value);
        }
    }

    [SerializeField]
    private bool _isOnCeiling;
    private Vector2 wallCheckDirection => gameObject.transform.localScale.x > 0 ? Vector2.right :
Vector2.left;

    public bool IsOnCeiling
    {
        get
        {
            return _isOnCeiling;
        }
        private set
```

```
        {
            _isOnCeiling = value;
            animator.SetBool(AnimationStrings.isOnCeiling, value);
        }
    }

    private void Awake()
    {
        touchingCol = GetComponent<CapsuleCollider2D>();
        animator = GetComponent<Animator>();
    }

    void FixedUpdate()
    {
        IsGrounded = touchingCol.Cast(Vector2.down, castFilter, groundHits, groundDistance) > 0;
        IsOnWall = touchingCol.Cast(wallCheckDirection, castFilter, wallHits, wallDistance) > 0;
        IsOnCeiling = touchingCol.Cast(Vector2.up, castFilter, ceilingHits, ceilingDistance) > 0;
    }
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка 2D-гри у жанрі платформеру на програмному рушії Unity

Дипломний проект студентки групи 4РП-07: Рябошапки Дарії Євгенівни

Керівник: Джабраілов Д.В.

Особливості ігрового жанру 2D платформер;

- Рівні з перешкодами;
- Зростаюча поступово складність;
- Просте керування;
- Простий та яскравий дизайн персонажів;
- Не завжди розумні вороги, які часто просто ходять по рівню назад-вперед
- Відсутність шкали здоров'я у головного персонажа

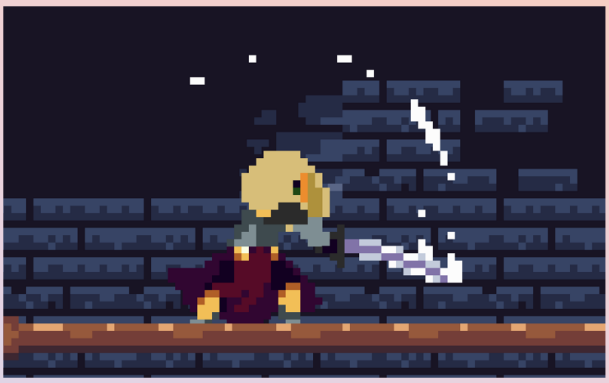
Особливості програмного рушія Unity та причини його вибору для розробки проекту;

Unity це ігровий рушій та інструмент для розробки. Unity є відносно простим у вивчені та користуванні.

Його особливостями є велика користувацька база, можливість кастомізації та мультимедійний інтерфейс

Особливості ігрових механік розроблюємого проекту

Проаналізувавши інші проекти цього ж жанру, я вирішила додати можливість атаки для персонажа, також у проекті реалізовані прості анімації, створені за допомогою аніматора Unity.



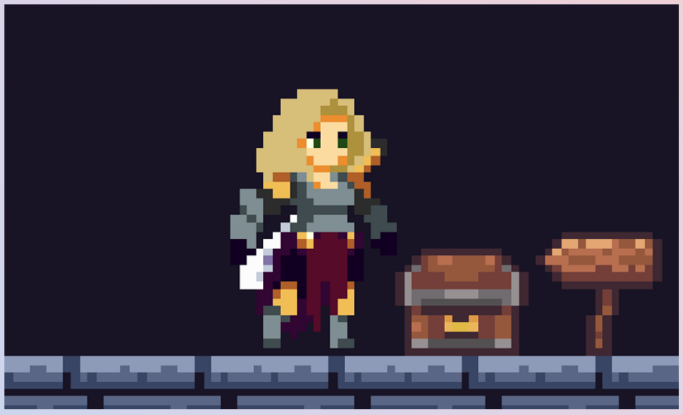
Особливості розробки проекту

Для створення проекту було використано велику кількість функцій та скриптів. До прикладу, для того, щоб не плутатись у назвах для анімацій, та для того, щоб використовувати однакові класи без помилок, я створила «зовнішній» клас, у якому було узагальнено назви всіх анімацій, для того, щоб посилатись на нього у інших скриптах.

Огляд основних елементів ігрового процесу;

Основними елементами мого проекту є безпосередньо рівень, гравець та ворог. Всі ці елементи були створені за допомогою додатку Aseprite та маю наступний вигляд:

Принцип роботи основних елементів ігрового процесу



Головний персонаж рухається платформами за допомогою box collider'у, а також rigidbody, встановленому як на персонажі, так і на самій платформі. Це дозволяє двом об'єктам не провалюватись, але при цьому потрібно слідкувати за розмірами колайдеру, щоб не виникало помилок.

Етапи реалізації основних елементів ігрового процесу

Для реалізації проекту я:

- Розробила ідею гри
- Продумала основний концепт проекту
- Намалювала спрайти для майбутнього проекту
- Створила проект в Unity
- Написала скрипт та анімації персонажу та ворогам
- Розробила та створила сцену гри
- Протестувала гру та виправила всі виявлені помилки

Хід тестування гри

На етапі тестування було помічено помилку, при якій персонаж провалювався між плитки платформ, від чого не мав можливості рухатись. Цю помилку було виправлено зміненням розміну колайдера платформи.

Після цього помилок більше не виникало.

Висновки

За час створення цього проекту я підвищила свої навички використання Unity для розробки мультимедійних проектів, а також провела аналіз того, для чого потрібні ігри і яка їх актуальність зараз.

Та визначила що робить ігри комерційно вдалими.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Рябошапки Дарії Євгенівни

(прізвище, ім'я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка 2D-гри у жанрі платформеру на програмному рушії Unity

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 76 сторінок. У пояснювальній записці виконано опис предметної області, способи реалізації 2D-гри в жанрі платформер. Проведено проектування та модифікація реалізація оточення, персонажів гри та інших систем. Графічна частина складається з 10 слайдів мультимедійної презентації, які передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом: Протягом всього строку дипломного проектування та переддипломної практики здобувачка освіти Рябошапка Д.Є. поступово та послідовно виконувала всі етапи розробки. Всі роботи здобувачка освіти виконувала самостійно, з оглядом на рекомендації керівника.

в) теоретична підготовка випускника (випускниці): Здобувачка освіти Рябошапка Д.Є. під час роботи над дипломним проектом вивчила достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка дипломника добра і вона готова до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувачка освіти Рябошапка Д.Є. мала змогу самостійно приймати рішення з реалізації елементів та систем гри платформеру, та показала вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові керівника дипломного проекту

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проекту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач комісії комп'ютерних технологій та програмної інженерії

Підпис

«10» 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Рябошапки Дарії Євгенівни

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення "

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка 2D-гри у жанрі платформеру на програмному рушії Unity*

Обсяг розрахунково-пояснювальної записки 76 сторінок

Обсяг графічної (презентаційної) частини 10 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity, в цілому.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У технологічному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано вимоги до гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляемої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені зміни до ігрового процесу

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Виконано докладний аналіз ігрових рушіїв та ігрових проектів;
2. Яскрава візуальна складова ігрового процесу.

д) основні недоліки дипломного проекту (роботи)

1. Безпосередньо етапам розробки гри у пояснювальній записці приділено недостатньо уваги.
2. Графічну частину проекту недостатньо пропрацьовано.
3. У роботі присутні деякі помилки оформлення.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

Підпис:

« 18 » 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016340013

Дата перевірки:
09.06.2024 23:48:01 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.06.2024 08:07:29 EEST

ID користувача:
100011688

Назва документа: **4РП-07_Рябошапка**

Кількість сторінок: **42** Кількість слів: **7795** Кількість символів: **54776** Розмір файлу: **2.24 MB** ID файлу: **1016141143**

9.67%
Схожість

Найбільша схожість: **3.9%** з Інтернет-джерелом (https://ela.kpi.ua/handle/123456789/49671)

9.67% Джерела з Інтернету	259	Сторінка 44

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНГО ДИПЛОМНОГО ПРОЕКТА
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Рябошапка Дарія Євгенівна,
здобувач освіти гр. 4РП-07, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускного дипломного проекту фахового молодшого бакалавра на тему:

«Розробка 2D-гри у жанрі платформеру на програмному рушії Unity» (автор роботи – Рябошапка Д.Є., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Рябошапка Д.Є. /

Керівник _____ / Джабраілов Д.В./

«10» 06 2024 р.