МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційне та програмне забезпечення ігрового додатку "Втеча з Раю"»

здобувача групи ІН – 93 Шолох Данило Костянтинович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Данило Шолох

(підпис)

Керівник,

старший викладач Артем Коробов

(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН-93 Шолох Данило Костянтинович

1. Тема роботи: «Інформаційне та програмне забезпечення ігрового додатку "Втеча з Раю"»

затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для прогнозування курсу валют. 3) Розробка інтелектуальної системи з прогнозування курсу валют. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Затвердження теми роботи		
2	Вивчення та аналіз завдання		
3	Написання вступної частини		
4	Огляд існуючих прототипів		
5	Програмна реалізація		
6	Оформлення документації дипломного проекту		

Здобувач вищої освіти _____ (підпис)

Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 85 стр., 34 рис., 16 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки є безліч маленьких інді-компаній, а також великих підприємств, які вкладають безліч ресурсів, щоб створити такі проекти в більшому масштабі для створення якісного продукту та мають мільйони проданих копій.

Об’єкт дослідження — процес створення прототипу гри в жанрі 3д платформер.

Мета роботи — розробка гри у 3д просторі в жанрі платформер, створена на ігровом рушії Unity

Методи дослідження — аналізі існуючих ігор жанру платформер з метою створення та протестування нових механік.

Результати — розроблений прототип гри, який надає можливість протестувати основні механіки. Цей прототип слугує демонстрацією концепції та основної ідеї проекту, які можуть бути подальше розвинуті.

Ігровий додаток, ігровий рушій Unity, 3д платформер, C#, Visual Studia

Зміст

Вступ

Робота присвячена розробці ігрового додатку в жанрі 3д платформер на ігровому рушії Unity. Щоб зробити хоча б прототип гри потрібно бути різнобічною особистістю, адже розробка гри це цілий процес з безліччю різних аспектів. Таких як створення сюжету та опрацювання цілісності світу, моделювання та анімація персонажів, програмування скриптів і навіть озвучування, розробка звуків та саундтрек гри. Це робить створення навіть маленької гри дуже складним процесом. Звісно Unity надає значну підтримку розробникам, особливо для інді-розробників, зручним інтерфейсом, навчальними матеріалами та інструментами для програмування. Вона автоматизує багато аспектів розробки, що дозволяє зосередитися на творчому процесі та швидко реалізовувати свої ідеї. Для створення захоплюючого 3D платформера необхідно розробити унікальні механіки та зручний контроль над персонажем. Головною метою є створення детально пророблено світу, який забирає гравця з самого початку гри та тримає його захопленим до самого кінця. Головним завданням є надання гравцеві позитивних емоцій та поглиблення відчуття побуття в віртуальній реальності. Важливо ретельно проектувати та реалізовувати різноманітні рівні, збалансовані за складністю, з прорисованим оточуючим середовищем, харизматичними персонажами, захоплюючими діалогами та сюжетом, що сприятиме більшому зануренню гравця в гру під час проходження. Також на початку важливо забезпечити проектування, що дає можливість пізніше оновлювати та вдосконалювати гру, додавати нові рівні, завдання та можливості, щоб зберегти інтерес гравців і запропонувати їм нові пригоди у захопливому віртуальному світі "Втеча з раю".

Розділ 1

"АНАЛІТИЧНИЙ ОГЛЯД ЖАНРУ 3D ПЛАТФОРМЕРІВ: ХАРАКТЕРИСТИКА, АКТУАЛЬНІСТЬ ТА МЕТА РОБОТИ"

1. Основні поняття та характеристики жанру 3D платформера

Гра в жанрі 3D-платформер це відеогра, у якій гравець керує персонажем у тривимірному світі, пересуваючись з однієї платформи на іншу, стрибаючи, бігаючи та взаємодіючи з різноманітними об'єктами. Головною метою гри є проходження різноманітних рівнів та подолання різноманітних перешкод, що постійно збільшуються у складності.

В цьому жанрі характерною є наявністю різноманітних видів завдань, таких як збирання предметів, битви з ворогами та розв'язування головоломок. Гра може мати лінійну або відкриту структуру, що дає гравцю свободу в дослідженні світу гри та виконанні завдань у будь-якому порядку. Головні елементи геймплею - це складні перешкоди, які гравець повинен подолати, щоб продовжити гру. Такі перешкоди можуть бути у вигляді лабіринтів, пасток, стрибків та інших завдань, що допомагають гравцю розвивати свої навички. Однією з ключових особливостей 3D платформерів є наявність головного героя, який збирає різні предмети, покращує свої характеристики та взаємодіє з оточенням, використовуючи різні знаряддя. Збір предметів та покращення характеристик дозволяють гравцеві продовжувати гру та долати складні перешкоди. Характеристики героя можуть включати параметри, такі як сила стрибку, швидкість, життєві запаси, міцність, здатність до польоту та інші. Взаємодія героя з оточенням може полягати в збиранні ключів для відкриття дверей, пересуванні різних об'єктів, підніманні на стіни та скелі та стрибанні по платформам.

Хоча і платформери - це старий жанр з устоялими правилами, це не означає, що він не може бути доповнений та змінений. Розробники ігор продовжують додавати нові елементи геймплею, щоб зробити платформери більш захопливими та різноманітними для гравців. Один із прикладів - гра "Краш Бандікут", яка вийшла в 1996 році. Ця гра стала однією з найпопулярніших платформерів на консолі PlayStation того часу. Її головний герой, Краш Бандікут, повинен був подолати безліч перешкод, збирати монети та кристали і врешті-решт врятувати свою дівчину від злих сил. Проте у наступних частинах гри були додані нові елементи геймплею, зокрема механіка гри "стрибки по стінах", яка з'явилася в грі "Crash Bandicoot: The Wrath of Cortex". Цей елемент дозволив гравцям переміщуватися вертикально та горизонтально, забезпечивши більше варіацій в проходженні рівнів.Іншим прикладом є гра "Super Mario Odyssey", яка вийшла в 2017 році для консолі Nintendo Switch. У цій грі розробники додали нові механіки гри, такі як здатність героя перетворюватися на різні об'єкти, включаючи вертольоти, автомобілі та танки. Це дозволило гравцям досліджувати величезний відкритий світ гри, розширюючи можливості геймплею.

Графіка у 3D платформерах

Графіка у 3D платформерах зазвичай характеризується яскравими кольорами та детальними текстурами. Деталізація дозволяє розглянути кожен об'єкт в ігровому світі, починаючи від дрібних деталей на головному герої до шуму листя в далекому лісі. Однією з найбільш помітних рис графіки у 3D платформерах є використання тривимірного простору. Це дозволяє гравцям рухатися в будь-якому напрямку та досліджувати віртуальний світ з усіх сторін. Завдяки цьому, графіка може бути набагато більш реалістичною та

деталізованою, адже гравці можуть бачити об’єкти з більш привабливої та реалістичної перспективи. У 3D платформерах часто використовуються анімаційні та кінематографічні ефекти, щоб зробити геймплей більш динамічним та ефектним. Такі ефекти можуть включати в себе вибухи, руйнування об’єктів, ефекти замедлення часу, що допомагають підкреслити дію та додати до неї більше енергії.

Перцепція

На мою думку о днією з ключових елементів 3D платформера є перспектива і сприйняття масштабу. Однак, гравці можуть мати проблеми з правильним сприйняттям масштабу і простору в грі, що може призвести до неправильних дій та нерозуміння. Особливо це стосується 3D ігор, де гравець має оцінювати відстань та глибину зображення щоб прорахувати силу стрибку з однієї платформи на іншу. Існують різні методи, які можна використовувати для допомоги гравцям краще орієнтуватись в просторі, такі як використання об'єктів з відомими розмірами, щоб дати загальне уявлення про відстань, або ефект паралаксу, який показує глибину простору за допомогою різних швидкостей руху об'єктів. Також важливо забезпечити гравців відповідними елементами, які допоможуть їм зорієнтуватись в просторі, такі як різні предмети, розміщені на різних відстанях від гравця, або елементи, які вказують на те, що далі продовжується шлях.

Проте, існують підходи, які допоможуть мінімізувати ці проблеми. Забезпечення гравця відповідними деталями та посиланнями на відстань допоможе зрозуміти глибину та відстань. Розміщення об'єктів, які гравець знає розмір, допоможе дати загальне уявлення про відстань. Але, для ефективного орієнтування в просторі, можна використовувати лінійні об'єкти з однаковими повторювальними елементами.

Рис 1.1 Візуалізація перцепції у 3D пространстві

Важливо визнати, що відповідне визначення та документування метрик у розробці ігрових рівнів є ключовим елементом для досягнення високої якості гри. Ці метрики допоможуть при розробцы відновити рівновагу між складністю та досяжністю, а також забезпечить гравцям більш точне сприйняття глибини та відстані в грі.

Метрика

У розробці ігор жанру 3D платформер важливо враховувати різні метрики, такі як час проходження рівня, кількість життів, вбивств та інші. Ці показники допомагають розробникам розуміти, які елементи гри потребують поліпшення та де можливі проблеми з балансуванням рівнів. Найважливішим аспектом розробки ігор жанру 3D платформер є створення гармонійного світу, який би відповідав задуму розробника та був зрозумілим та доступним для гравців. У цьому процесі метрики грають важливу роль, оскільки дозволяють контролювати всі фізичні характеристики світу гри. Наприклад, розмір перешкод у грі, таких як скелі, дерева або будівлі, важливо узгоджувати з розмірами персонажа, щоб гравець міг комфортно рухатися в світі гри. Під час розробки потрібно визначити розмір кімнат, локацій та інших об'єктів в грі, щоб вони були зручні для гравця. Занадто великі кімнати можуть змусити

гравця відчувати себе загубленим або неспроможним, а занадто малі можуть створювати відчуття стисненості та нестачі свободи дій. Крім того, важливо враховувати висоту та нахил скелі, щоб гравці могли рухатися по них без перешкод.

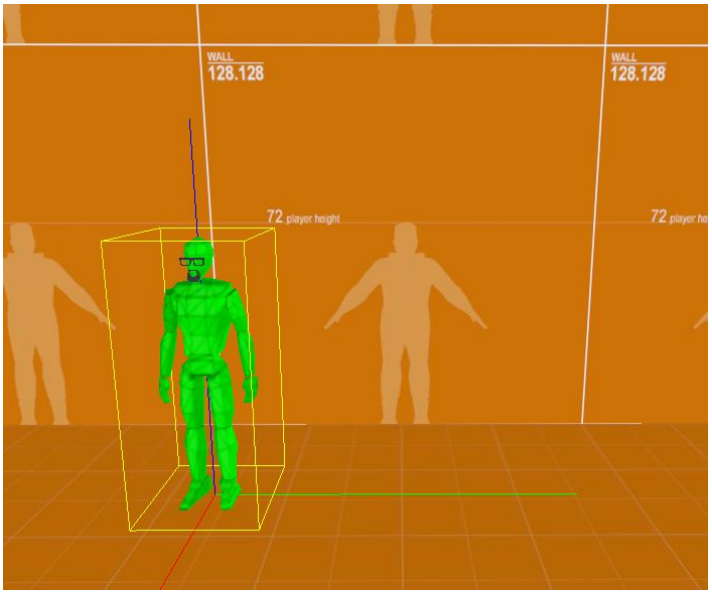


Рис 1.2 Тестування метрики

Ще однією важливою метрикою є балансування ворогів та вбивств. Розробники повинні забезпечувати, щоб кількість ворогів була достатньою, щоб створити виклик для гравця, але не настільки великою, щоб гра стала надто важкою та нездійсненною. Також важливо враховувати метрики часу та кількості життів, які гравець має на кожному рівні. Ці метрики допоможуть розробникам визначити, чи рівень є довгим чи складним настільки, щоб гравці витратили багато часу та життів на його проходження. Важливо забезпечувати, щоб гравці могли успішно пройти рівень, але не настільки великою, щоб гра стала надто простою та не цікавою. У загальному, метрики є важливим інструментом для розробників ігор жанру 3D платформер. Вони допомагають створювати гармонійний та добре збалансований світ гри, який би був доступним та зрозумілим для гравців. Якщо добре вивчити та

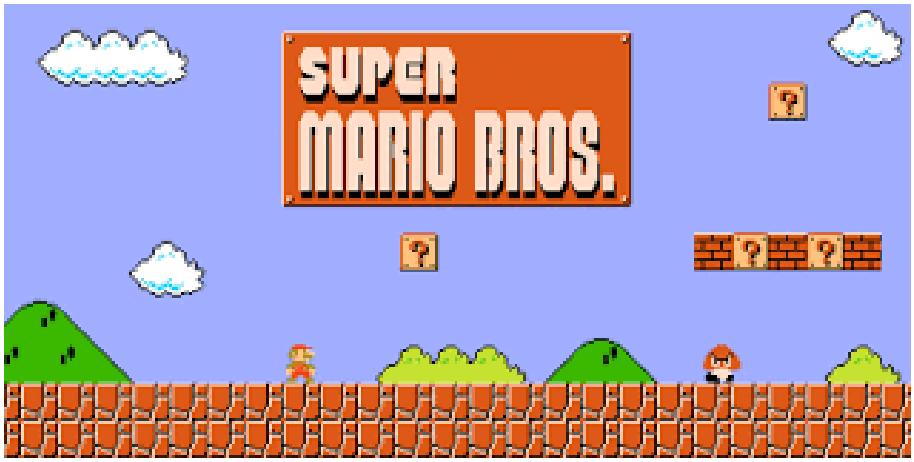
використовувати метрики правильно, то гравці зможуть насолоджуватися грою, так як вона і запланована надавати.

2. Актуальність 3д платформера в сучасному gamedev

Жанр 3D-платформерів з відкритим світом виник як результат еволюції платформерів у 3D-графіці. Першими платформерами були класичні 2D-ігри, такі як Super Mario Bros, Sonic the Hedgehog та Donkey Kong. Ці ігри мали лінійну структуру та вимагали від гравців проходження рівня за рівнем. З появою 3D-графіки, розробники стали експериментувати зі створенням тривимірних платформерів. Перші спроби не були дуже успішними через обмежену кількість полігонів, яку могли обробляти тодішні графічні процесори.

Однак з часом технології почали розвиватися, що дало розробникам можливість створювати все більш складні та реалістичні ігри. У 1996 році компанія Nintendo випустила Super Mario 64 - першу повнометражну 3D-ігру, яка стала насправді революційною у своєму жанрі. Гра забезпечила безліч

нових можливостей для взаємодії гравця з ігровим світом та встановила нові стандарти для розробки 3D-ігор.



2.1 гра Super Mario Bros

Еволюція жанру 3д платформера

3D-платформери з'явився в 90-х роках минулого століття. З тих пір вони стали одними з найпопулярніших ігрових жанрів, завдяки своїй відкритості та можливостям взаємодії з тривимірним світом. Один з найбільш відомих та популярних 3D-платформерів - Super Mario 64 - був випущений у 1996 році

для ігрової консолі Nintendo 64. Головним перевагою цієї гри було введення тривимірного світу, який значно розширив можливості геймплею.

Рис 3.1 Super Mario 64

Ця гра стала мегапопулярною завдяки своєму геймплею, зручному управлінню та інноваційному підходу до візуальної реалізації гри. Super Mario 64 відкрила двері для багатьох інших ігор, що працюють з тривимірним простором. Після випуску Super Mario 64, 3D-платформери стали набирати все більшої популярності, і до 2000-х років вони перетворилися на один з найпопулярніших жанрів відеоігор. Такі ігри, як Banjo-Kazooie, Spyro the Dragon, Rayman та Sonic Adventure, також зарекомендували себе в цьому жанрі. У 2001 році Nintendo випустила гру Luigi's Mansion, що була 3D-платформером зі зміщенням камери на більші відстані від персонажа. Це

дозволило гравцям більш повноцінно досліджувати великі світи та детально розглядати окремі елементи гри.

Рис 3.2 Luigi's Mansion 2001

Протягом наступних десятиліть 3D-платформери з відкритим світом стали все більш популярними. Нові ігри, такі як Ratchet and Clank, Jak and Daxter, і Sly Cooper, з'явилися на ринку, втілюючи сучасні ідеї та новітні технології. У 2010-х роках на передову сцену вийшли ігри, такі як Super Mario Odyssey, Ratchet & Clank (2016), та A Hat in Time, які змінили спосіб, яким гравці досліджують світи в іграх.

Однією з найбільших змін у жанрі була зміна відкритості рівнів. Починаючи зі старих 3D платформерів, рівні були зазвичай лінійними та одноразовими, але з часом розробники стали додавати більш відкриті рівні, які дають гравцям більше можливостей для дослідження та взаємодії з оточенням. Розширення відкритості рівнів в 3D платформерах було великим кроком вперед у розвитку жанру. Раніше, у старих 3D платформерах, гравець керував персонажем, який проходив рівні, складаючи завдання і збираючи різноманітні предмети, але рівні були лінійними та одноразовими. Такі рівні

не давали багато можливостей для взаємодії з оточенням, ігноруючи можливості, які надавала 3D-графіка.

Зі зростанням потужності графічних двигунів та розширенням можливостей розробників, стали з'являтись більш відкриті рівні з більшою кількістю дій. Гравці мали більші можливості для дослідження світу та взаємодії з ним. Нові 3D платформери дозволяють гравцям вільно переміщатись по світу та відкривати нові зони, які раніше були недоступними. Ця зміна відкритості рівнів також збільшила складність гри, оскільки гравці повинні знаходити більше сховищ предметів та прихованих секретів. Розширення відкритості рівнів також дозволяє розробникам створювати більш складні головоломки, що надають більшу глибину геймплею.

Особливості у сучасних 3D платформерах

Сучасні 3D платформери, це злиттям платформних ігор з 3D графікою та головоломками. Ці ігри вирізняються високим рівнем іммерсії та мають унікальні особливості, що роблять їх захоплюючими та незабутніми для гравців. Специфічні риси сучасних 3D платформерів включають глибокі історії з детально розробленими персонажами та локаціями, величезні мапи та складні головоломки. Гравці мають можливість виконувати завдання та розв'язувати головоломки на протязі всієї гри, що забезпечує збереження інтересу гравців протягом тривалого часу. Повторювані рівні відсутні в сучасних 3D платформерах, що дозволяє гравцям занурюватися в ігровий світ. Щодо геймплею, змінилося не тільки більш складними головоломками, а і значною різноманітністю геймплейних механік, які забезпечують багатошаровий підхід до ігрового досвіду. Гравці отримують доступ до широкого спектру можливостей для управління героєм, що включає різноманітність рухів, здібностей та взаємодії з навколишнім середовищем.

Стрибки, біг, підйоми та зміна розміру та форми персонажа є основними елементами геймплею в 3D платформерах. Вони забезпечують можливість для експлорації та дослідження гри. Додаткові здібності, такі як літання та телепортація, надають гравцям додаткові можливості для пройдення рівнів та виконання завдань. Такий різноманітний ігровий досвід дозволяє гравцям ефективно використовувати різні механіки, щоб розв'язати головоломки та дійти до кінцевої мети. Це сприяє зростанню креативності та стратегічного мислення гравців. Крім того, різноманітність геймплейних механік дозволяє гравцям насолоджуватися грою на довгий час, оскільки кожна гра проходиться унікальним способом, залежно від вибраних механік та стратегій гравця.

Оповідання гри та його оточення(Setting) також змінився в більш глибоку та багатоваріативну історію, який не просто потрібен для галочки як в іграх того ж жанру 90-х років і йде десь на фоні геймплею, а і в деяких випадках може бути частиною того ж геймплею. Прикладом такого феномену може слугувати дуже популярна гра Psychonauts2 кожен рівень виконаний у формі ментального ландшафту, який являє собою ментальний світ героя чи іншого персонажа. Гравець може досліджувати ці світи, збирати інформацію та взаємодіяти з персонажами. Крім того, в грі є багато діалогів та кат-сцен, які допомагають розкрити сюжет та персонажів.

Рис 4.2 Psychonauts 2 гра 2021 року

Перспективи розвитку жанру

Жанр 3D платформера має великий потенціал для подальшого розвитку, завдяки постійному прогресу в галузі технологій та зростаючим можливостям ігрових платформ. Раніше, платформери переважно були 2D і гралися на приставках з простими контролерами. Але з появою потужних процесорів та графічних чипів, цей жанр може бути адаптований для різних платформ, включаючи телефон та VR. Мобільні платформи забезпечують доступність і портативність, дозволяючи гравцям насолоджуватися платформерами у будь-якому місці та часі. Крім того, сенсорний екран та акселерометр дозволяють використовувати нові способи керування, які можуть бути більш інтуїтивними та забезпечувати нові можливості для геймплею. Але як для мене одним з найбільш захоплюючих напрямків розвитку жанру є використання віртуальної реальності (VR). VR дозволяє гравцям стати частиною гри, перебуваючи всередині вигаданого світу. Вони можуть бачити світ з першої особи, відчувати реалістичну просторову глибину та взаємодіяти з оточенням за

допомогою контролерів руху. Це створює неймовірно іммерсивний досвід та дозволяє розробникам експериментувати з новими механіками гри, які використовують можливості VR.

Творчі процеси у розробці платформерів не мають жодних суворих рамок, що дає велику свободу авторам. Цей жанр можна адаптувати до будь-якої тематики та створити гру на будь-який смак, що робить його актуальним у будь-який час. Основними елементами платформера є шлях з точки А до точки Б, елементи паркуру та перешкоди, які карають гравця за помилки, роблячи гру цікавішою. Основа платформера полягає у трьох ключових геймплейних елементах. По-перше, це шлях з точки А до точки Б, де гравець має пройти через різноманітні рівні та перешкоди. Другий елемент - елементи паркуру, такі як платформи, на які гравець може стрибати, лазити або підлазити. І нарешті, третій елемент - перешкоди, які карають гравця за помилки, роблячи гру цікавішою та вимагаючи від нього точності та реакції. Для створення більш цікавішої та більш глибокої гри, розробники можуть додавати додаткові елементи. Наприклад, систему винагороди за проходження рівнів, яка стимулює гравця до досягнень, або пасхалки, які винагороджують його за дослідження ігрового світу. Такі елементи розширюють ігровий досвід та заохочують гравця до більш активного взаємодії з грою.

Прикладом безмежної творчості в платформерах є гра дитинства “Підлога Лава", де потрібно перейти з однієї точки кімнати в іншу, уникаючи контакту з "лавою". Ця гра демонструє всі ключові елементи платформера, де лава виступає як перешкода, меблі - як платформи, по яких гравець може стрибати, щоб не доторкнутися до лави, та мета полягає в тому, щоб дістатися до фінішу. Прикладом що творчість немає обмежень у ігрової індустрії є концепт нової гри 2023 року у жанрі 3D платформер, Crash Team Rumble із серії ігор Crash Bandicoot. Вона особлива тим що це гра у жанрі 3D платформер у мультіплеєрі, тобто в неї можно стрибати по платформам разом з друзями. Цей концепт гри демонструє, що творчість у галузі створення ігор не має

обмежень. Розширення жанру 3D платформера на мультиплеєрний формат вносить свіжість і нові можливості для гравців, сприяючи соціальному взаємодії та спільним пригодам. Це лише один з прикладів того, як розробники можуть інновувати та створювати унікальні ігрові враження, розширюючи границі жанру.

Рисунок 4.3 Постер гри Crash Team Rumble

Усе це підкреслює, що жанр 3D платформерів має потенціал для постійного розвитку та відкриває безмежні можливості для творчості розробників. Завдяки своїй універсальності та здатності адаптуватися до нових технологій, платформери можуть надалі радувати гравців свіжими інноваціями та захоплюючим геймплеєм.

3. Інструменти для реалізації проекту

Unity

Звісно я буду використовувати ігровий рушій Юніті, я вважаю він ідеально підходить для розробки відеоігор, особливо для інді розробки. Його потужність і гнучкість дозволяють невеликим розробникам з власною візією створювати захоплюючі та якісні ігри. Unity володіє численними перевагами, які особливо корисні для інді розробки, особливо якщо хочеться розробити 3D платформер. Перевага Unity це його широкий спектр функціональності, яка дозволяє створювати різноманітні геймплеї та унікальні віртуальні світи. Завдяки вбудованому движку Unity, можно створювати складні 3D об'єкти, реалістичні фізичні ефекти та динамічну анімацію персонажів або просто скачати безплатні чи платні модельки та текстурки з Unity Asset Store. Це надає можливість швидко та якісно, а головне не потрібно шукати художника і можно всі аспекти розробки зробити самому, та створювати неповторні геймплейні механіки, які допомагають втілити все що завгодно в своєму проекті. Ще одна перевага Unity - це його дружній інтерфейс та легкість використання. Я можу швидко навчитися використовувати його і з легкістю розпочати розробку свого проекту. Unity також має велику спільноту розробників, де я можу знайти підтримку, поради та ресурси для покращення своїх навичок. Це робить процес розробки приємним і захоплюючим.

Таким чином, Unity - це найкращий інструмент для розробки відеоігор, який ідеально підходить для мене. Він надає мені всі необхідні засоби для втілення моїх творчих ідей, робить процес розробки ефективним і захоплюючим, а також дозволяє швидко створювати якісні ігрові досвіди,

включаючи 3D платформери. З Unity я можу реалізувати свої ігрові проекти та захопити гравців своїми творіннями.

Visual Studia і C#

Visual Studio, та мова C# - це незамінні інструменти для реалізації проектів розробки ігор. Ці потужні інструменти сприяють розробці захоплюючих ігор та дозволяють розробникам втілити свої найкращі ідеї. Unity, з його гнучкими можливостями створення як 2D, так і 3D ігор, дозволяє реалізувати унікальну графіку, фізику та управління персонажем. Мова C# - потужний і ефективний інструмент програмування, який забезпечує широкий набір функцій та інструментів, необхідних для створення складної логіки гри. Visual Studio, у свою чергу, забезпечує зручне середовище розробки, де розробники можуть легко писати, налагоджувати та візуалізувати свій код. Разом, ці інструменти створюють ідеальний набір для розробки ігор, дозволяючи мені творити захоплюючі проекти і здійснювати свої мрії у світі ігрової індустрії.

Unity Asset Store

Як я вже писав про Asset Store, він пропонує великий вибір різноманітних ресурсів, які можна використовувати для прискорення процесу розробки і поліпшення якості гри. Він включає готові 2D та 3D моделі персонажів, оточення, об'єкти, текстури, матеріали, ефекти частинок, звукові ефекти та музику. Крім того, Asset Store також надає доступ до плагінів, розширень, скриптів та інших інструментів, які розширюють можливості Unity і дозволяє реалізувати специфічні функції та механіки гри. Але як для мене

головною його фічею є те що завдяки йому, я як людина яка зовсім не вміє в моделювання, можу одним кліком скачати все що мені потрібно по візуалу і через пять хвилин користуватись цим в моєму проекті. Це дозволяє мені економити час і зусилля на створенні власних ресурсів з нуля, особливо для тих аспектів гри, які не є основними або можуть бути швидко вирішені за допомогою готових плагінів. Він дозволяє зосередитися на унікальних аспектах проекту і дуже прискорює процес розробки.

Asset Store є цінним ресурсом для розробки проекту, який сприяє прискоренню та поліпшенню процесу розробки гри. З його допомогою можно знайти готові активи, інструменти та розширення, що полегшують роботу, підвищують якість ігрового вмісту та дозволяють швидше досягнути бажаних результатів.

4. Висновок першого розділу

Жанр 3D платформерів продовжує бути актуальним і захоплюючим для гравців та розробників. Одна з ключових привабливостей цього жанру полягає у використанні тривимірної графіки, що дозволяє створювати реалістичні та деталізовані ігрові світи. Гравці можуть насолоджуватись візуальними ефектами, деталізованими об'єктами та просторовою перспективою, що підсилює іммерсивність геймплею. Гравці отримують можливість відчувати глибину простору та зміну перспективи, що робить гру більш реалістичною та цікавою. Наприклад, можливість пересуватись в тривимірному просторі, стрибати, рухатись вздовж та поперек платформ, а також взаємодіяти з об'єктами надають гравцям більшу свободу та контроль над головним

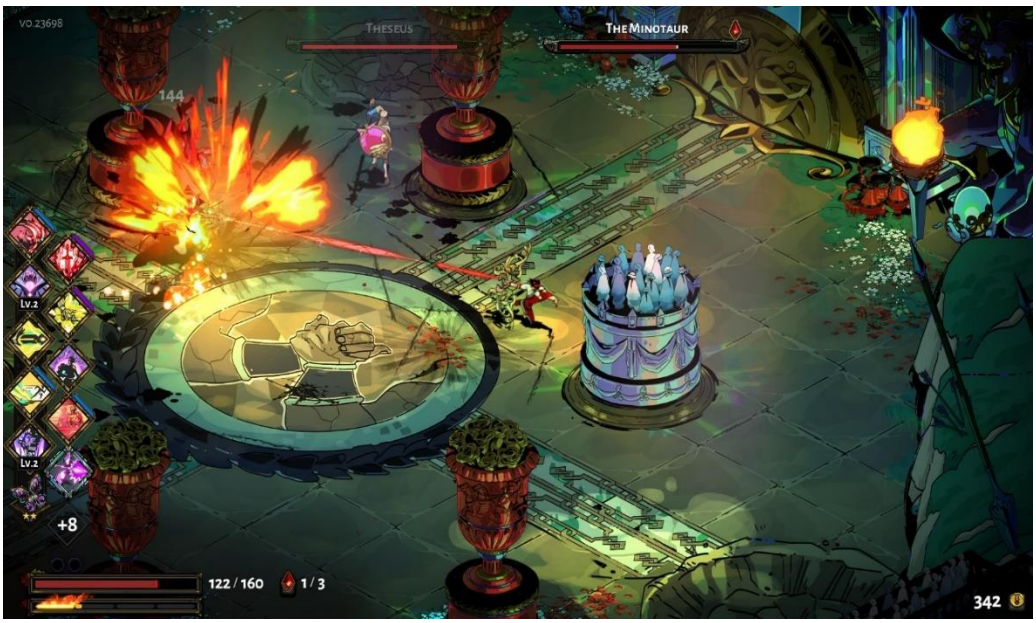
персонажем. Загалом, жанр 3D платформерів продовжує залишатись важливим та привабливим для гравців, а також стимулює розробників до вдосконалення та інновацій. Використання тривимірної графіки, перцепції та метрики допомагає створювати захоплюючі геймплеї, а постійний розвиток жанру відкриває нові можливості для майбутнього росту та розширення.

Розділ 2

ПРОЕКТУВАННЯ РОЗРОБКИ ГРИ

1. Створення сюжетної лінії та світу гри

При розробці сюжету «Escape Clouds» я надихався грою іншого жанру «Hades». Hades це гра в жанрі рогалайк, де гравець бере на себе роль сина Аїда, володаря підземного царства мертвих, з іменем Загрей. Головний герой бажає втекти з цього царства, що розкриває повний потенціал ключивої механіки жанру рогалайк “постійна смерть”. Коли герой помирає, вся гра починається заново, втрачаючи прогрес, здобуті предмети та покращення. Однак, деякі елементи прогресу можуть зберігатися, такі як знання, досвід чи розблоковані можливості, що дозволяє гравцю поступово покращувати свої навички і досягати кращих результатів. Сюжет гри Hades тісно переплітається з механікою смерті. Кожна нова спроба дозволяє гравцеві дізнаватись більше про персонажів, взаємини та історію підземного царства. Кожна смерть стає новою можливістю розкрити більше деталей та просунутися у сюжеті.



5.1 Геймплей гри Haders

Цей спосіб оповідання сюжету було взято в проект “Escape Clouds”. По сюжету гри, головний герой маленьке чертеня, яке випадково потрапило в місце де не повинне було опинитися, в раю. Щоб визволитися з незвичного для себе оточення, він повинен подолати безліч перешкод на своєму шляху. В цьому йому допомагає неігровий персонаж, він є коріним жителем раю і може допомогти демону, якщо виконати його завдання. За ці завдання можно отримати винагороду, яка допоможе легше долати перешкоди або просуватися далі по сюжету, щоб повернутися назад до пекал. Кожне завдання це окремий рівень в грі який потрібно подолати. В цьому допоможуть хмаринки, по ним можно стрибати тим самим пересуватися по рівню та долати його. В цьому світі є основна велика хмарка, кожень новий рівень починається з цієї хмаринки, це така собі безпечна зона. Там і опиняється чертеня на початку гри, і якщо невдало стрибнути на хмаринку і впасти, то в будь якому випадку потрапиш знову на цю основну хмаринку. На ній є можливість поспілкуватися з жителем раю, придбати в нього апгрейд характеристик персонажа, чи нові завдання.

2. Проектування персонажа та його основні вміння

Найголовніше щоб головний герой запамятовувався, щоб він мав унікальний зовнішний вигляд, та виділяти його серед інших персонажів гри, і зробити з нього легко впізнаваним для гравця. В моєму проекті головний герой - чертеня, яке опинилося в раю, що є чудовим прикладом поєднання протилежностей: рогатий, красний демон та білий, пухнастий рай. Мій персонаж, як головний герой платформера, вдало комбінує зовнішню привабливість, унікальні рухи та легку впізнаваність, що робить його незабутнім для гравця. Це втілення протилежностей створює інтригу та спонукає гравця відкривати світ гри, розкриваючи його таємниці та переживати захоплюючі пригоди. Але просто зовнішність та контрастом його не зробить цікавим, для цього потрібно додати деякі незвичні вміння герою, це робить персонажа не тільки цікавості, а і урізноманітнює геймплей гри. Унікальні вміння та характеристики персонажа створюють можливості для творчого геймплею та стратегічних рішень. Гравцеві дається свобода експериментувати з різними підходами та знаходити оптимальний шлях до досягнення мети. Це робить гру більш захоплюючою та стимулює гравця до дослідження можливостей свого персонажа.

Основне вміння це звісно стрибки, жоден платформер не може обійтись без стрибків. Щоб зробити стрибок більш цікавим це важлива складова геймплею, гравець повинен вміло керувати персонажем. Але просто нажимати на кнопку може швидко наскучити, щоб урізноманітнити геймплей та зробити так, щоб кожен стрибок був як своя окрема міні гра, потрібно зробити “контролюємий стрибок”. Це важлива механіка, яка дозволяє гравцю контролювати силу та дальність стрибка відповідно до ситуації. Гравець може використовувати легкий стрибок для точних маневрів або подолання невеликих прірв, а сильний стрибок для довгих перешкод чи досягнення

вищих платформ. Таким способом кожен рух з хмаринки на хмаринку це випробування бо якщо не розрахувати можно перестрибнути, або не дострибнути і втартити прогрес. Друге обовязкове для кожного платформера атрибут, це подвійний стрибок. Подвійний стрибок це звичайний стрибок, але він відбувається між тим коли закінчується перший стрибок, тобто коли персонаж стрибнув до найвищої точки сили свого стрибка, і землею. Подвійний стрибок також підлягає логіки контролюємого стрибока, це дає гравцеві більше варіативності та просто для маневрів і дослідження навколишнього світу.

3. Концепція алгоритмів

Дуже важливо до початку реалізації проекту продумати концепцію та будову створення скриптів, які є основною кодовою частиною проекта. За допомогою якою реалізується основні механіки гри та взаємодія з віртуальним світом. За допомогою продуманих алгоритмів можно реалізувати характеристики, стрибки та здібності героя, які є основою гри, та розробити фізику для гри з потрібними параметрами для неї, такі як сила гравітації, та продумати як вона буде поводити себе в різних ситуаціях. Щоб надати гравцю ідеальний відчуття контролю над гравцем та його взіємодія з навколишнім світом.

Алгоритми керуванням персонажу

Керування персонажем є одним з найважливіших аспектів гри в жанрі 3D платформер. Гравець повинен мати повний контроль над рухом персонажа, його стрибками та іншими взаємодіями з оточенням. Це вимагає створення алгоритмів, які забезпечують плавність та реалістичність руху, а також відтворити повне відчуття персонажа з оточенням.

Персонажу повинен вільно рухатися в тривимірному просторі, це переміщення вперед, назад, вліво, вправо, і повороти. Врахувати фізичні обмеження, такі як швидкість та максимальні значення обертання, щоб забезпечити реалістичність руху. Проектування сили стрибка та розрахунок траєкторії руху персонажа у повітрі. Особливу врахувати можливості виконання подвійних або багаторазових стрибків для забезпечення більшої маневреності. З врахуванням оточення, такі як платформи, сходи, перешкоди та інші об'єкти. Це включає розрахунок колізій та зіткнень персонажа з оточенням, щоб забезпечити правильну поведінку та реалістичність взаємодії з обєктами ігрового світу. Розробка цих елементів приведе до правильного сприйняття гравця откліку з його персонажем та взаємодія з оточуючими обєктами. Для візуального враження розробляєця анімація персонажа під час руху та взаємодії з оточенням. Це включає розрахунок та відтворення різних анімаційних станів, таких як ходьба, біг, стрибки та інші, що відповідають діям персонажа. Та вбудувати алгоритм анімацій в скрипти персонажа, щоб вони доповнювали один одного для кращего сприйняття віртуального світу з людиною яка в нього занурюється.

Проектування фізики гри

Реалістична фізика допомагає створити та забезпечити взаємодію персонажа з оточенням та іншими об'єктами в грі. Один з основних елементів фізика є гравітація, вона дозволяє персонажу падати, це одна з головних аспектів жанру платформера, і тільки цей аспект в моєму проекті дозволяє персонажу переместитися на початкову платформу, ішими словами померти. Потрібно правильно налаштувати розрахунки параметрів для визначення впливу гравітації на рух персонажа та інших об'єктів у грі. Це дозволяє адаптувати гравітацію до конкретного геймплею, щоб створити різні ефекти руху в залежності від частини гри та підлаштувати правильну взаємодію з обєктами їх реакцію на зіткнення з персонажем. За допомогою колізії можно

відповідати за взаємодію об'єктів у грі, включаючи персонажа, оточення та інші об'єкти, і гарантує правильну імітацію фізичних взаємодій, коли персонаж стикається з оточенням. Об'єкти повинні рухатися та відбиватися після зіткнення наприклад, при зіткненні персонажа зі стіною, він повинен відскакувати або зупинятися, залежно від параметрів колізії та фізичних властивостей об'єктів. Рухомі платформи при взаємодії зупиняють гравця або відталкують його зберігаючи свою швидкість та напрямок, це може суперечити фізиці реального світу, але саме такі умови створюються в жанрі платформер.

4. Висновок другого розділу

Проектування розробки гри є чи не найважливішим етапом у її створенні проекту. Вона дозволяє оцінити час на створення основних концепцій і ідей. Щоб оцінити та забезпечити відповідну організацію та планування процесу розробки гри, проектуванням персонажем його керуванням та фізики. Алгоритми забезпечать створення ефективних та реалістичну поведінку для керування персонажем у грі в жанрі 3D платформер. У фізиці було досягнуто реалістичності та природності взаємодії персонажа з оточенням у грі, а також забезпечує правильне функціонування різних геймплейних механік, пов'язаних з колізією.. Ці алгоритми дозволяють гравцю насолоджуватися плавним та точним керуванням персонажем, що сприяє покращенню геймплею та досвіду від гри.

Розділ 3

РЕАЛІЗАЦІЯ ПРОЕКТУ

1. Дизайн та анімація ігрових моделей

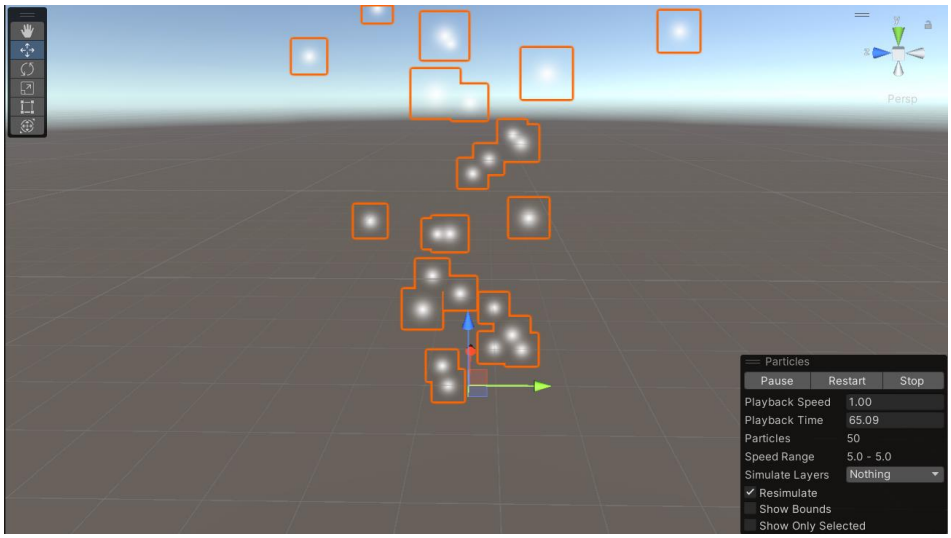
Дизайн і анімація ігрових моделей в Unity3D - це процес створення візуальних елементів та їх руху для використання в іграх, контролювати рухи персонажів, їх взаємодію з оточенням та іншими об'єктами. Це дозволяє створювати різноманітні і складні анімації, які зроблять гру більш живою та захоплюючою для гравців.

Дизайн об'єктів та оточення

При розробці дизайну ігрового світу своєї гри, я звісно надихався релігійною літературою. Основні атрибутики світу, такі як оточення чи персонажі. Світ гри “Escape Clouds” відбувається в раю, тобто основне що буде бачити гравець це хмаринки. Ці хмаринки я розділив на дві категорії, ігрові та неігрові, Неігрові хмаринки це звичайний елемент декору, який слугує тілки візуальним опціям, це просто 2д спрайти десь на фоні світу. Ігрові хмаринки це ігровий обєкт, з ним взаємодіє герой, тому вони мають більше візуальних деталей та деяку анімацію.

6.1 Скайбокс проекту "Clouds Escape"

Сам дизайн хмаринок я створив за допомогою внутрішної функції Юніті, системою частинок. Частинки - це дрібні графічні елементи, які можуть рухатися, змінювати кольори, міняти розміри та взаємодіяти з фізичним середовищем гри. За допомогою цієї системи можно створювати гарні та реалистичні ефекти, як вогонь, дим, вибухи, туман, сніжинки, в моєму випадку це хмаринки. Вони генеруюця в заданому місці і пролітають в випадкову сторону, їми можно керувати, задавати напрям та змінювати колір в тому числі прозрачність.



6.2 Система партікл в Юніті

Основні компоненти системи частинок, які можно коригувати для створення реалистичних ефектів це:

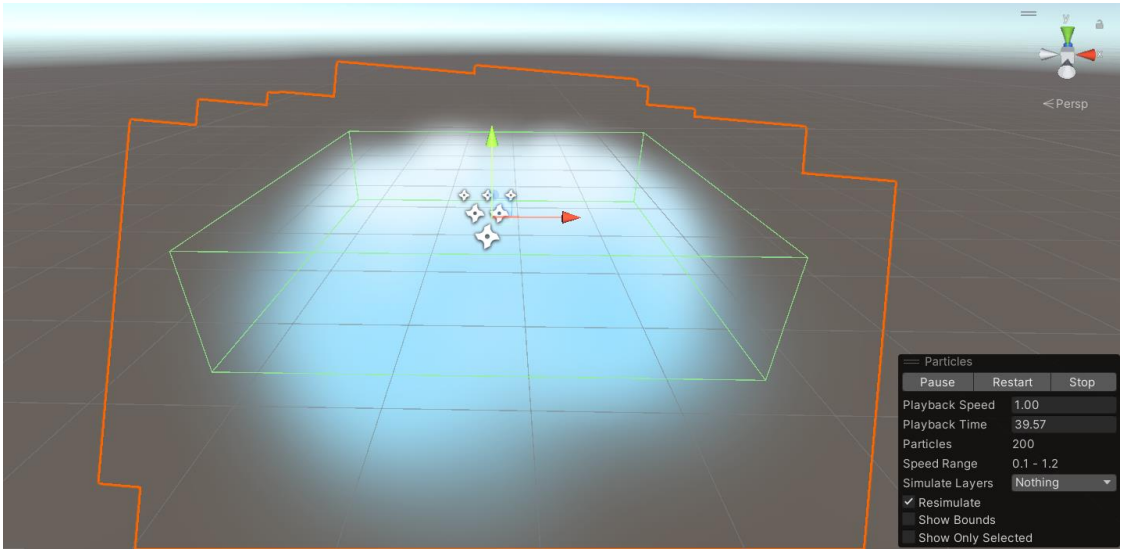
Emitter – це місце появи та напрямок руху частинок.

Modules - впливають на рух, вигляд та поведінку частинок. Також за допомогою цієї функції можно змінити швидкість, розмір, колір та форму.

Colliders - використовуються для взаємодії частинок з іншими об'єктами в грі. Щоб частинки відскакували від поверхонь або взаємодіяли з персонажами чи іншими об'єктами

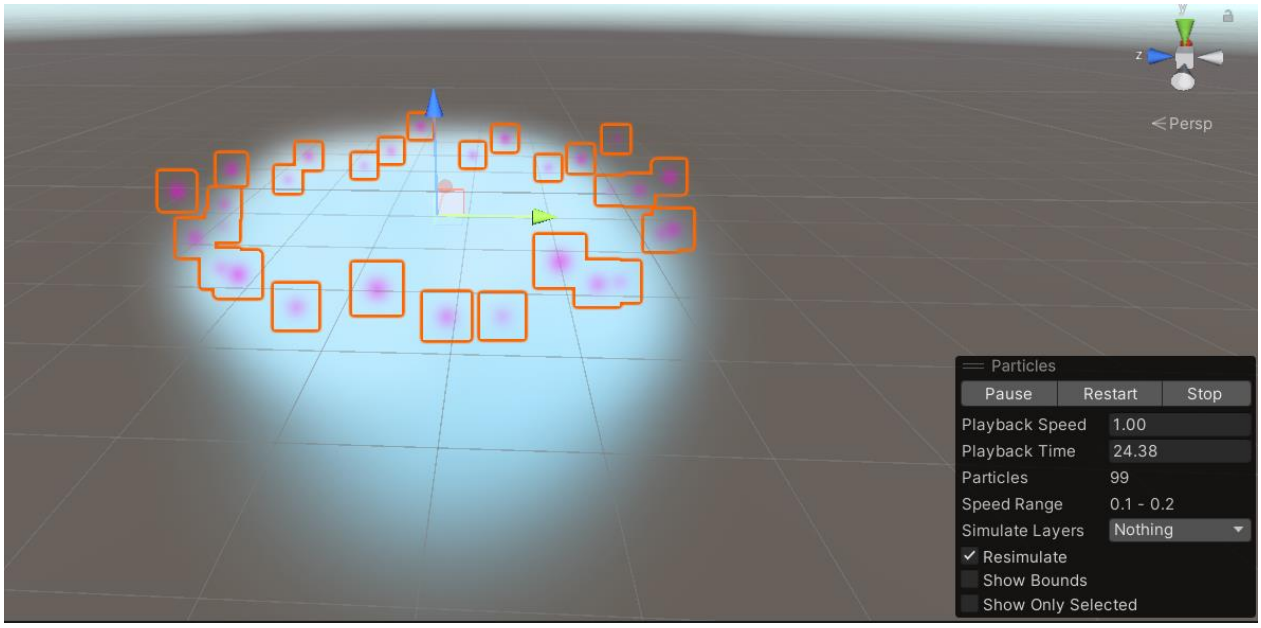
Textures - можно використовувати зображення або текстури для надання частинкам вигляду та стилізації. Наприклад використовувати текстуру пламені для створення ефекту вогню.

За допомогою цих функцій я зробив ось таку хмаринку. Основним моїм завданням, оскільки ця хмаринка слугує платформою на яку потрібно пригнути, було зробити так щоб на неї приємно було стрибати. Тому її дизайн нагадує пухнасту подушку, на яку якщо призимлитися відлітаються частинки хмаринки.



6.3 Платфора у виді хмаринки

Є ще одна проблема, яку може вирішити дизайн, це орієнтація у просторі. Через те що гра зроблена у 3D, та камера не статична, складно буде прорахувати куди та з якою силою треба стрибнути, щоб потрапити на хмарку. Щоб виправити це, я знову використовую систему часток. За допомогою неї я роблю візуальні границі, які допомагають гравцеві орієнтуват наскільки далеко лівітує хмаринка від гравця та дає зрозуміти де закінчуються границі коллайдера хмаринки.



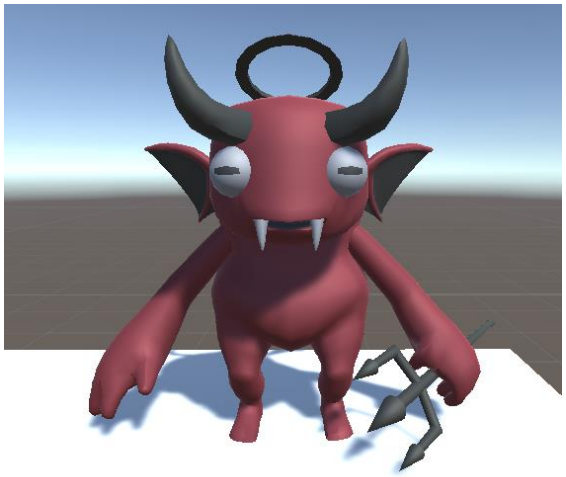
6.4 Вигляд кордонів облака у виді фіолетових частинок

Модель головного персонажу та його анімація

По сюжту гри, головний герой це чертеня яке є жителем аду але потрапило до раю. Отже він повинен мати основні атрибути демона, але одночас виглядати милим створінням, щоб відповідати тиматиці ігрового світу. Головні атрибути демона, які виділять його від інших сворінь, це ріжки, красний колір шкіри, маленькі крильця, та тризубець, яким він катує грішних людей.

7.1 Модель Diablo з гри "Diablo"

Хоча з мене поганий художник, але у юніті є інструмент Unity Asset Store, який допоможе швидко знайти і зручно скачати та імпортувати модель, яка мені потрібна. Після кількох хвилин пошуку, я знайшов цілий пак персонажів, з яких є підходяща для мене модель демона. Які має всі особливості, які мені потрібні.

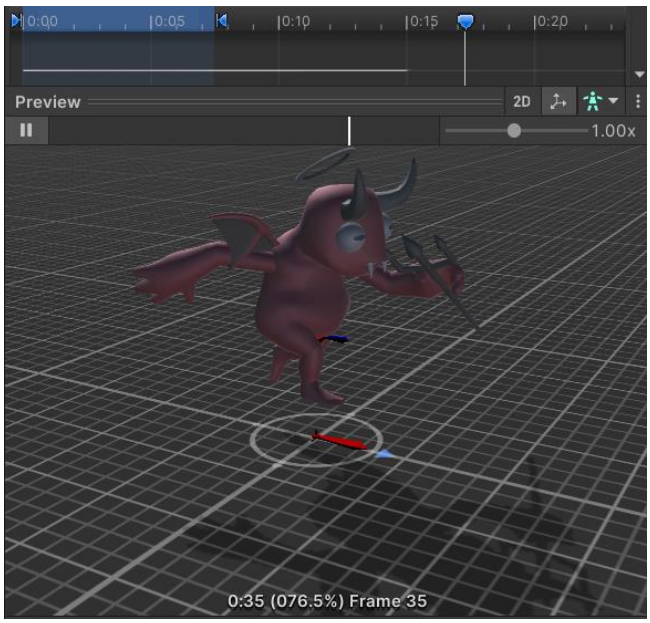


7.2 Модель Чертеня з гри "EscapeClouds"

Анімація персонажів у Unity - це процес надання руху ігровим персонажам. Unity має потужні інструменти для створення, редагування та

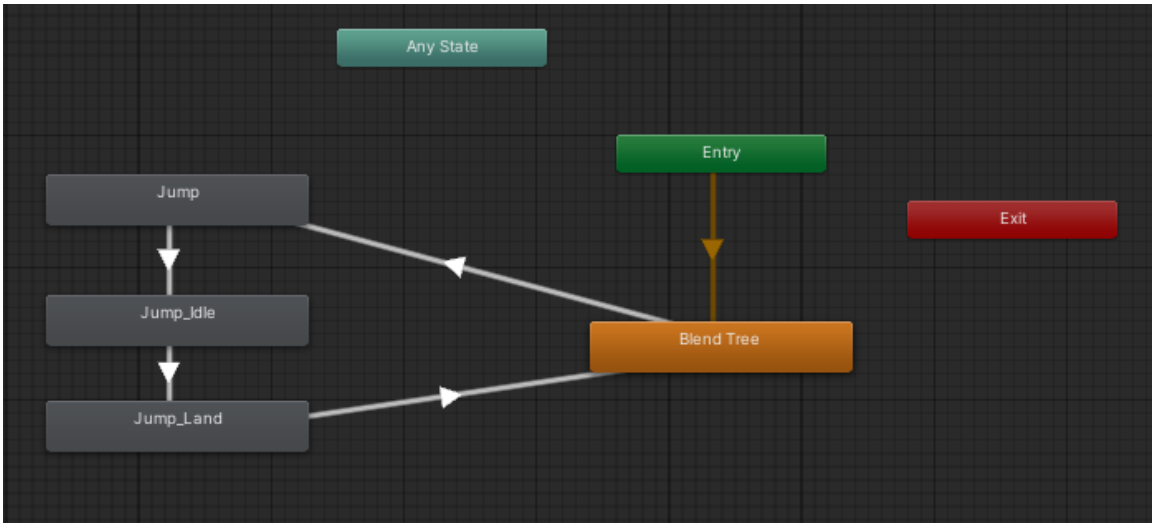
управління анімаціями. Основним елементом анімації персонажа є Animator Controller. Він дозволяє вам керувати різними анімаціями залежно від стану персонажа та взаємодії з оточенням. Контролер анімацій включає набір станів, переходів та анімаційних кліпів.

Щоб створити анімацію персонажа, нам необхідно анімаційний кліп. Це записаний рух персонажа в певному часовому діапазоні. Unity підтримує різні методи створення анімаційних кліпів: візуальний редактор, імпорт анімаційних файлів (наприклад, із програми Blender або Maya) або застосування скриптів, що керують анімацією.



7.3 Анімація головного героя в аніматорі Unity

Після створення анімаційного кліпу можно додати його до контролера анімацій. У контролері визначаються умови переходу між станами та встановити, які анімації будуть відтворюватися при певних подіях або діях гравця. Для керування анімацією персонажа в реальному часі використовують скрипти C# або систему Mecanim, яка надає розширений набір функцій для керування анімаціями.



7.4 Контроль анімації за допомогою Mecanim

Контроль анімації стрибка поділяється на три стани,

Jump – це початок стрибка, коли демон відштовхується від землі, ця анімація застосовується коли буде натиснута кнопа стрибка.

Jump_Idle – це стан стрибка, коли демон не торкається землі і опиняється в повітрі, ця анімація застосовується, коли коллайдер персонажа не перетинається з коллайдером платформи.

Jump_Land – це стан стрибка, коли демон приземляється на поверхню платформи, анімація застосовується при дотику коллайдера персонажа з коллайдером платформи.

Unity надає потужні можливості для створення реалістичної та живої анімації персонажів, яка зручно налаштується та керується за допомогою графічних інструментів, як той що я застосовував в своєму проектв, або через скрипт.

Дизайн UI елементів

UI це інтерфейс користувача, він допомогає ігроку взаємодіяти з грою, активно впливати на подачу інформації та виконання дій через візуальний компонент, в основному на екрані. Це допомогає з навігацією до геймплею в меню та паузі або ж в самій грі.

В меню зазвичай використовується певний шаблон, це декілька кнопок за допомогою яких можно орієнтуватися. Основними кнопками є "Старт" - для початку геймплею, "Налаштування" - де можна змінювати звукові ефекти, графіку гри, а також налаштовувати гарячі клавіші для персоналізації управління та "Вихід" - для виходу з гри.

В самій же грі з інтерфейсом користувача трішечки складніше. Там є безліч елементів які я поділив на дві частини, це інтерактивні та інформативні. Інтерактивний інтерфейс зосереджується на взаємодії з користувачем і може включати такі елементи, як спеціальні здібності персонажа або інструменти для взаємодії з оточенням. Інформативний інтерфейс надає гравцю важливу інформацію про гру, таку як стан здоров'я персонажа, кількість ресурсів, мапу рівня, завдання або цілі гри. Ці елементи можуть бути розташовані на екрані для зручності гравця та допомагати йому приймати рішення та аналізувати ситуацію у грі.

Важливо створити привабливий дизайн, який відображає стиль та тематику гри. Використання чудових анімацій і привабливих картинок для UI елементів допомагає гравцям легше зануритись в атмосферу гри. Оскільки мій проект в жанрі 3д платформер, а тут дуже важливо забезпечити мінімальну кількість UI елементів на екрані, щоб нічого не заважало гравцеві при проходженні рівнів. Однак, ви можете перенести основну інформацію, яка зазвичай передається гравцеві через UI, безпосередньо в елементи самої гри.

Наприклад можно відобразити показник здоровя персонажа на моделі самого персонажа, як цез робили в грі Dead Space

8.1 геймплей Dead Space

Вся необхідна інформація відображається на спині персонажа. Здоров'я відображається через трубку на хребті персонажа, розділене на сектори, є зрозумілим та лаконічним способом відображення стану здоров'я. Зменшення рідини в трубці відповідає зменшенню здоров'я персонажа, що дозволяє гравцю легко оцінювати свій поточний стан. Зміна кольору трубки при низькому рівні здоров'я є ефективним способом привернути увагу гравця до необхідності приділити більше уваги. Відображення сили енергії персонажа внизу правого плеча. І останнє це кількість патронів в зброї, вона відображається на самій зброї.

В своєму проекту я вирішив долучитися до цього ж правила, та перенести основну інформацію, яка зазвичай передається гравцеві через UI, безпосередньо в елементи самої гри. Наприклад очки які даються по мірі

проходження рівнів, яку можно використувавати як валюту, я вирішив відобразити на основній платформі, де після смерті зявляється герой, прям біля магазину, щоб відразу було видно скільки ресурсів залишилося

8.2 Табличка з інформацією кількості золота

За золото можно купити у жителя раю нові рівні для проходження сюжету або покращити економіку для того щоб золото швидше накаплювалося.

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

public class Economica : MonoBehaviour
{

    private int income = 1;
    private int gold = 0;
```

```
private Text goldText;

void Start()
{
    goldText = gameObject.GetComponent<Text>();
    StartCoroutine("Income");

}

void Update()
{

   goldText.text = getGold().ToString();

}



public void setIncome(int income)
{
    this.income = income;
}

public int getIncome()
{
    return income;
}

public void plusIncome(int buff_icome)
{
    income += buff_icome;
}

public void setGold(int gold)
{
    this.gold = gold;
}

public int getGold()
```

```
    {
        return gold;
    }

    private void plusGold()
    {
        gold += income;
    }




    IEnumerator Income()
    {

        while (true)
        {
            yield return new WaitForSeconds(1f);
            plusGold();
        }

    }

}
```

Цей код контролює накопичення рахунку економики гри. Золото нараховується через кожну секунду або її можно зібрати на рандомній платформі на рівні.

2. UI елементи гри

UI елементи це кнопки та інтерактивні елементи які гравець може натискати або взаємодіяти з ними для виконання певних дій. Наприклад, це

можуть бути кнопки для переміщення персонажа, взаємодії з предметами, стрільби, призначення спеціальних навичок або активації інтерфейсних функцій

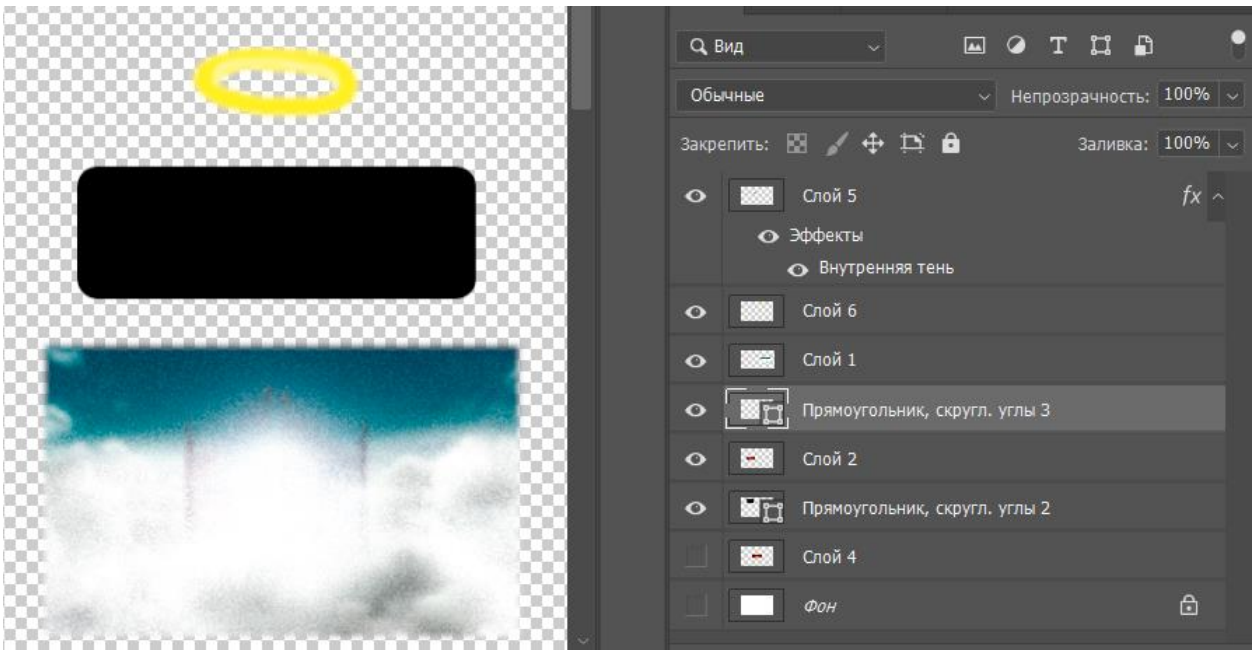
Кнопка «Вихід з гри» була розроблена мною з елементами характеризующим демона. Вона має великі красні рога, а основа її зроблена в красно-темних з малюнком схожий на лаву в пещері. Цей дизайн створює враження загадковості та небезпеки, віддалено асоціюючись з адом. Натискання на цю кнопку може викликати відчуття вихідних воріт, переносячи гравця від гри до реального світу.

9.1 Підготовка елементів для кнопки виходу

За допомогою інструменту фотошоп я створив прямокутник з закругленими кутами. Після цього додаю картинку на слой з чорним прямокутником та за допомогою функцій фотошопу создаю обтравочну маску на картинку та домалював для нього рога.

9.2 Готова кнопка виходу з гри

Те саме я роблю з кнопкою «Грати».



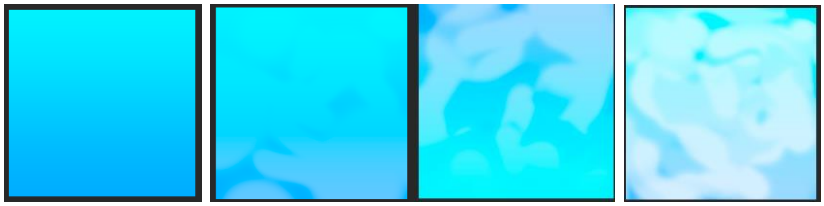
9.3 Підготовка елементів для кнопки “Play”

За допомогою асетів, які я створив, була розроблена кнопка, яка є протилежністю кнопці "Вийти з гри" і відображає стиль гри. Мій підхід

полягав у створенні кнопки, яка передавала позитивні асоціації гравцеві під час початку гри. Замість темних та загадкових елементів, я використав яскраві кольори та елементи, що символізують радість та привітність.

9.4 Готова кнопка "Play"

Наступним елементом є кнопка вибору рівня у жителя раю. Її я також створив в фотошопі, це звичайний квадрат з градієнтом синіго на який було наложено три слоя з білою краскою зі зменшиною прозорістю



9.5 Покроковий малюнок хмарин для кнопки рівнів

Наступним кроком додаю текст з номером рівня, оскільки в грі десять рівнів, я створив такі кнопоки з різнимим цифрами підстать кожного рівня.

9.6 Готові кнопки вибора рівнів

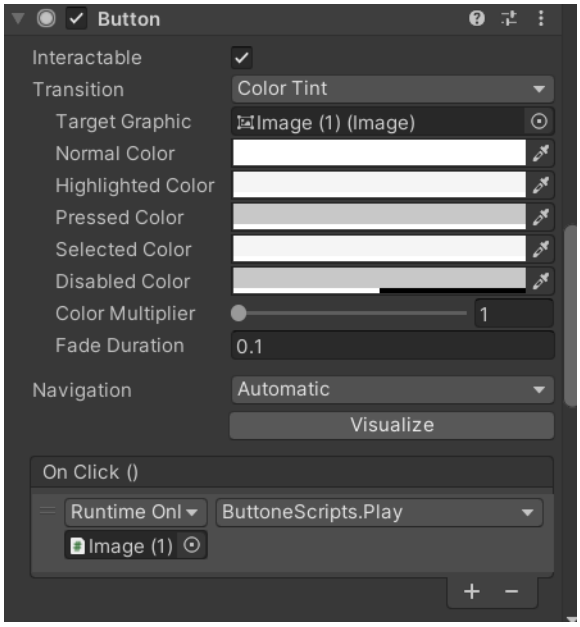
Стартове меню гри

Найголовніше забезпечити гравцеві легку навігацію та зручний доступ до різних функцій. Найкращий варіант для платформера може бути вертикальне меню з кнопками, які представляють основні функції, такі як “Грати” або "Вийти”. Додатково можна включити інформацію про рівні гри або досягнення гравця. Важливо забезпечити зручну навігацію між різними розділами меню та зробити його інтуїтивно зрозумілим для гравців платформера.

Розроблене головне меню в тематичному стилі з кнопками Аду та Раю в оточенні небес і головним героєм - демоном, що стоїть на платформі у вигляді хмарки, миттєво передасть гравцю атмосферу та концепцію гри. Меню надає гравцю візуальну інформацію про фантастичну атмосферу гри, демонструє її основні елементи та зацікавлює гравця з першого погляду. Це може створити сильне перше враження та забезпечити гравцям, та показує які елементи гри можна очікувати.

10.1 Меню гри "Escape Clouds"

В головному меню кожній кнопці присвоєний скрипт, що виконує відповідні функції. Наприклад, кнопка “Play” має вбудований компонент “Button” з функцією On Click(), яка автоматично викликається при натисканні на об’єкт, на якому знаходиться цей компонент.

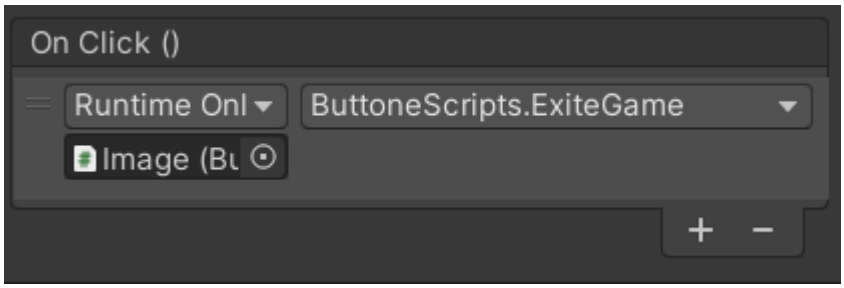


10.2 Компонент “Button” з функцією On Click()

При спрацьовувані компонент визиває функціонал скрипта ButtoneScripts.Play, який за допомогою бібліотеки SceneManager починає загрузку ігрової сцени.

```
Public void Play()
{
  SceneManager.LoadScene(0);
}
```

Так само, для кнопки "Вихід з гри" використовується функція "On Click()", яка викликає клас "ButtonScripts", але з використанням іншої функції під назвою "ExitGame". При виклику цієї функції гра буде закрита, і гравець вийде з програми.

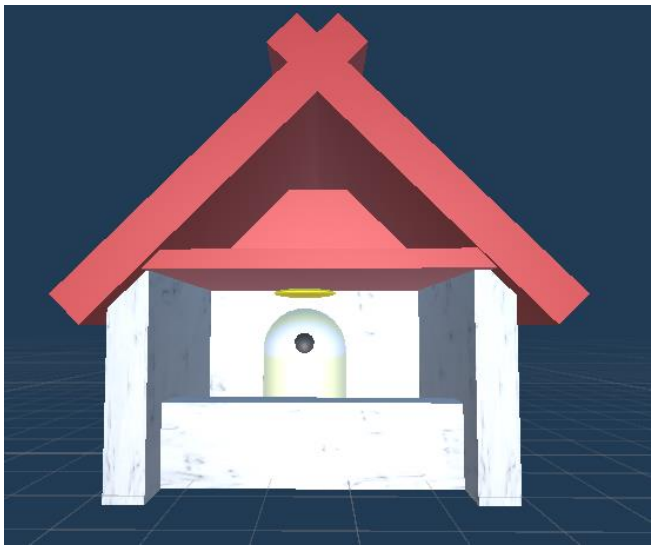


10.3 Функція On Click() з скриптом "ButtoneScreipts"

```
public void ExiteGame()
  {
    Application.Quit();
  }
```

Інтерфейсом взаємодії з NPC

Цей ігровий інтерфейс викликається при взаємодії з NPC, тобото неігровий персонаж, який по сюжету допомагає гравцю з подоланням рівней та втечею з раю.



11.1 Дім жителя раю (NPC)

Щоб взаємодіяти з ним, потрібно підійти на дистанції до двух метрів та натиснути на нього.

Реалізація скрипту взаємодії з неігровим персонажем

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

public class OpenInterface : MonoBehaviour
{
    Camera camera;

    private bool enter = false;
    private float maxDistance = 29f;



    private void Start()
```

```
{
    camera = GetComponent<Camera>();
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}


private void Update()
{

    Vector3 point = new Vector3(camera.pixelWidth / 2, camera.pixelHeight / 2, 0);
    Ray ray = camera.ScreenPointToRay(point);

    if (Input.GetMouseButtonDown(0) || getEnter())
    {
        talkToNPC(ray);
    }
}

public void setEnter(bool enter)
{
    this.enter = enter;
}

public bool getEnter()
{
    return enter;
}

void OnGUI()
{
    int size = 12;
    float posX = camera.pixelWidth / 2 - size / 4;
    float posY = camera.pixelHeight / 2 - size / 2;
    GUI.Label(new Rect(posX, posY, size, size), "*");
}

public void onInterface(bool booler, GameObject canvas)
```

```
{
    canvas.SetActive(booler);
    Cursor.visible = booler;

    if (booler)
    {
        Cursor.lockState = CursorLockMode.None;

    }
    else
    {
        Cursor.lockState = CursorLockMode.Locked;

    }
}


void talkToNPC(Ray ray)
{

    RaycastHit hit;

    if (Physics.Raycast(ray, out hit))
    {
        GameObject target = hit.transform.gameObject;

        var distance = target.transform.position - gameObject.transform.position;

        if (distance.sqrMagnitude < maxDistance)
        {
            if (target != null && target.tag == "FrendlyNPC")
            {

                onInterface(true, target.GetComponent<HeroEnter>().canvas);

            }
        }
```

```
    }
  }
}
```

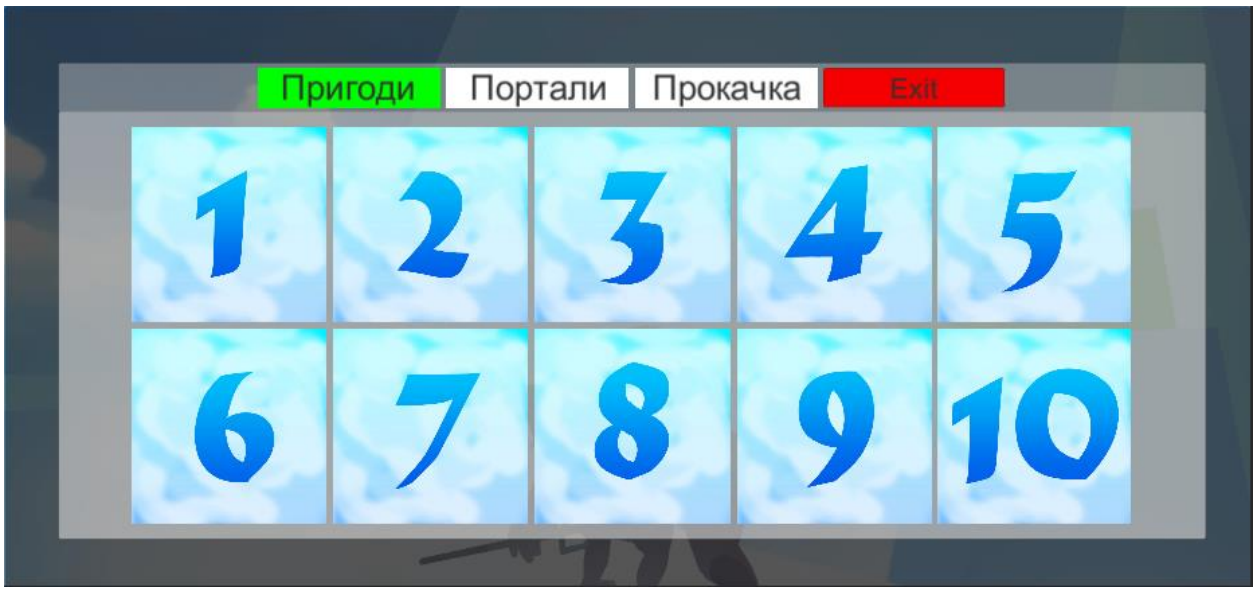
За допомогою цього скрипта відкривається інтерфейс під час взаємодії з дружніми NPC.

Інтерфейс розроблений в мінімалістичному стилі з навігацією, схожою на браузер зі закладками у верхній частині екрана. Основний контент розташований на основній центральній панелі. У закладках зверху є кнопка "Пригоди". При кліку на цю кнопку на основній панелі з'являться рівні, які будуть відкриватися по мірі проходження.

Портали це не сюжетні рівні при проходження яких головний герой стає сильнішим та отримує нові здібності, такі як ривок або заморозка хмаринки, що полегшують проходження сюжетних рівнів.

Перед остання кнопка “Прокачка” за допомогою золото можно прокачати основні характеристикаи героя, такі як дальність стрибку чи зменшення його сили гравитації.

І остання кнопка “Вихід”, якщо клікнути її інтерфейс вимикається до звичайного режима.



11.2 Ігровий інтерфей вибору рівней

Реалізація коду інтерфейса NPC

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

public class TabGroup : MonoBehaviour
{
   public TabButton selectButton;

   public List<TabButton> tabs;
   public List<GameObject> objectsToSwap;


   public void Subscribe(TabButton button)
   {

      if (tabs == null)
      {
         tabs = new List<TabButton>();
      }
```

```
    tabs.Add(button);
}

public void OnTabEnter(TabButton button)
{

    ResetTabs();

    if (selectButton == null || button != selectButton)
    {
        button.backgrounde.color = Color.blue;
    }

}

public void OnTabExite(TabButton button)
{
    ResetTabs();
}

public void OnTabSelected(TabButton button)
{
    selectButton = button;
    ResetTabs();
    button.backgrounde.color = Color.green;

    int index = button.transform.GetSiblingIndex();

    for (int i = 0; i < objectsToSwap.Count; i++)
    {
        if (i == index)
        {
            objectsToSwap[i].SetActive(true);
        }
        else
        {
            objectsToSwap[i].SetActive(false);
```

```
        }

    }


}

public void ResetTabs()
{
    foreach (TabButton button in tabs)
    {
        if (selectButton != null && button == selectButton)
        {
            continue;
        }
        button.backgrounde.color = Color.white;
    }
}
}
```

Цей скрипт в Unity відповідає за управління групою вкладок в інтерфейсі NPC. Він дозволяє перемикати вкладки і показувати відповідний контент в основній панелі на основі вибраної вкладки.

3. Розробка системи фізики

В юніті є вбудована система фізики Rigidbody. Цей компонент додають до грального об'єкта, щоб надати йому фізичні властивості, такі як маса, сила тяжіння і зіткнення. Rigidbody автоматично керує рухом об'єкта з урахуванням зіткнень з іншими об'єктами у сцені. Для реалістичного моделювання зіткнень між об'єктами використовується коллайдер. Коллайдер визначає форму

об'єкта і обробляє зіткнення з іншими коллайдерами. Unity надає різні типи коллайдерів, такі як box collider, sphere collider, capsule collider і mesh collider. У системі фізики Unity також є можливість встановлення різних обмежень і умов для руху об'єктів. Наприклад обмежити об'єкт у просторі або задати його швидкість і обертання.

Натомість в своїй грі я буду використовувати компонент Character Controller. Його відмінність від Rigidbody це те що він надає основну фізичну поведінку персонажа, включаючи рух по землі, стрибки та реакцію на колізії з об'єктами. Він працює на основі просторової фіксації, що дозволяє персонажу "плавно" рухатись без впливу сили гравітації. Цей підхід дозволяє контролювати рух гравця на більш високому рівні абстракції, ніж просте зміщення позиції. Крім того, це дозволяє отримати більш точний контроль над поведінкою персонажа, так як можна встановити різні параметри фізики, такі як гравітація і колізії. Що є клюевим для ігор жанру платформер.

Розробка алгоритмів фізики платформ

Для разноманітності геймплею, було створенно декілька різних платформ-хмаринок. Кожна з них має свою особливу механіку, яка заважає, або навпаки, допомагає гравцеві пройти рівень.

JustPlatform:

В назві все сказано, це звичайна хмаринка, з компонентами фізики Rigidbody та Capsule Collider

PrefabMove:

Особливість цієї хмарки, це те що вона переміщається з точки А до точки Б та назад.

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

public class ManagerPlatform : MonoBehaviour
{

    public float rotSpeed = 20f;

    public Transform startPoint;
    public Transform endPoint;
    public float traveltime;
    public bool rotation_or_move;

    Vector3 currentPos;

    Rigidbody rb;
    CharacterController cc;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }


    private void FixedUpdate()
    {
        if (!rotation_or_move)
        {
            movePlatform();
        }
```

```
        else if (rotation_or_move)
        {
            rotatePlatform();
        }



    }



    private void movePlatform()
    {
        currentPos = Vector3.Lerp(startPoint.position, endPoint.position, Mathf.Cos(Time.time / traveltime +
Mathf.PI * 2) * -.5f + .5f);
        rb.MovePosition(currentPos);
    }

    private void rotatePlatform()
    {
        Quaternion deltaRotation = Quaternion.Euler(new Vector3(0, rotSpeed, 0) * Time.deltaTime) ;
        rb.MoveRotation(rb.rotation * deltaRotation);
    }




    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {

            cc = other.GetComponent<CharacterController>();
        }
    }

    private void OnTriggerStay(Collider other)
    {
```

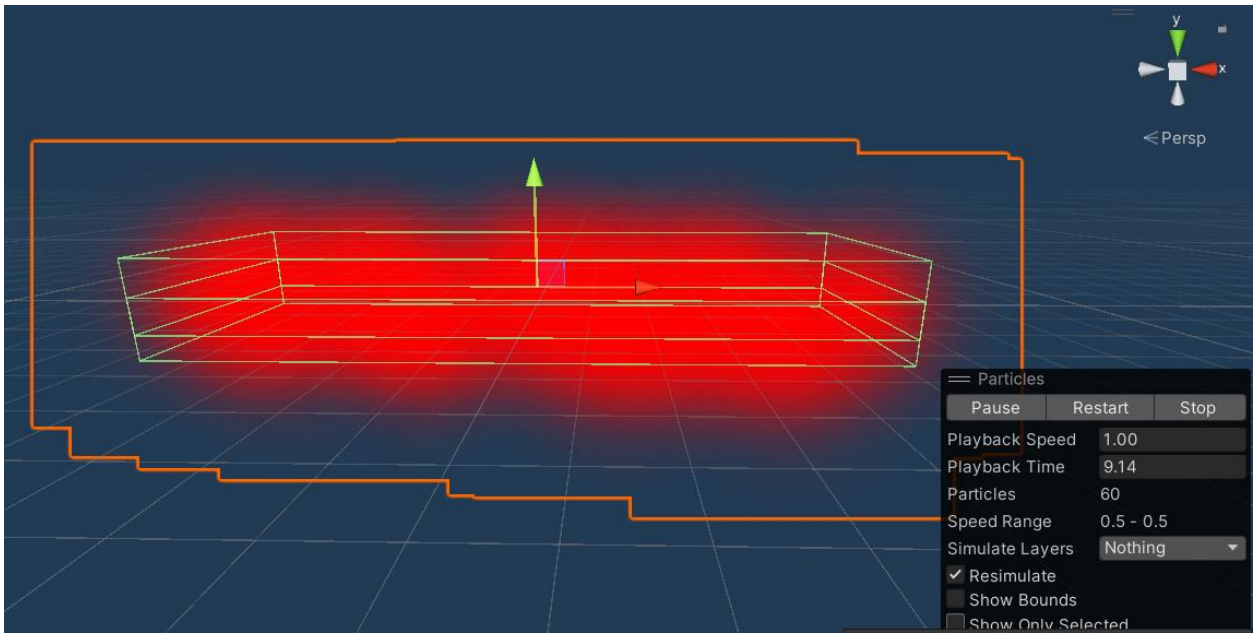
```
        if (other.tag == "Player")
        {
            if (!rotation_or_move)
            {
                cc.Move(rb.velocity * Time.deltaTime / 2);
            }
            else
            {
                var step = rotSpeed * Time.deltaTime;

                cc.Move(Vector3.MoveTowards(other.transform.position, transform.position, step));



            }

        }
    }
}
```

Функція "movePlatform" виконує плавний рух платформи між початковою ("startPoint") та кінцевою ("endPoint") позиціями. Вона використовує функцію Lerp для інтерполяції між цими двома точками на основі часу. Результатом є плавний рух платформи вздовж лінії між початковою та кінцевою точками.

У функції "OnTriggerStay" перевіряється, чи перебуває головний герой на платформі. Залежно від умови "rotation_or_move", яка визначає, чи рухається платформа чи обертається, виконується відповідне діяння. Якщо платформа рухається, герой рухається разом з платформою шляхом зміщення за допомогою "CharacterController.Move". Якщо платформа обертається, герой наближається до центру обертання платформи за допомогою "Vector3.MoveTowards", щоб зберегти відносну позицію на платформі. Це дозволяє головному герою взаємодіяти з рухомою або обертаючоюся платформою в грі, забезпечуючи плавний рух героя разом з платформою і збереження його позиції на платформі.

SlowPlatform:

При попаданні на цю хмаринку, головний герой сповільнюється.



12.1 Дизайн ігрової платформе "SlowPlatform"

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

public class SlowPlatform : MonoBehaviour
{

    private float slowSpeed = 2.5f;



    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
```

```
            other.GetComponent<PlayerMove>().setSpeed(other.GetComponent<PlayerMove>().getSpeed() /
slowSpeed);
        }

    }

    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            other.GetComponent<PlayerMove>().setSpeed(other.GetComponent<PlayerMove>().getSpeed()
* slowSpeed);
        }
    }

}
```

Щоб зменшити швидкість гравця, використовується доступ до компонента "PlayerMove" (або аналогічного компонента) на об'єкті гравця. Застосовується метод "setSpeed", який змінює швидкість гравця, ділену на значення "slowSpeed". Це зменшує швидкість гравця на задане число. Коли гравець покидає область тригера платформи ("OnTriggerExit"), знову перевіряється тег об'єкту. Якщо цей об'єкт має тег "Player", то застосовується зворотне сповільнення швидкості гравця. Знову використовують метод "setSpeed", але тепер швидкість гравця множиться на значення "slowSpeed". Це відновлює оригінальну швидкість гравця після того, як він покинув платформу.

Реалізація фізичної моделі головного героя

Контроль та управління персонажа це дуже складний процес розробки, адже по суті програмуєш міст, між світом гри та справжнім гравцем із справжнього світу. Моделюєш його аватар в свою гру, і його потрібно зробити максимально зручним, щоб гравець міг легко взаємодіяти з грою.

По перше потрібно зробити клас який виконує керування параметрами гравця. Клас PlayerManager виконує роль збереження значень параметрів гравця та надання можливості їх змінювати з інших скриптів.

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

public class PlayerManager : MonoBehaviour
{
    [SerializeField]
    private float jumpSpeed = 15.0f;
    [SerializeField]
    private float gravity = -10;


    public void SetGravity(float gravity)
    {
        this.gravity = gravity;
    }

    public float GetGravity()
    {
        return gravity;
    }


    public void SetJumpSpeed(float jumpSpeed)
    {
        this.jumpSpeed = jumpSpeed;
    }

    public float GetJumpSpeed()
    {
        return jumpSpeed;
    }
```

```
}
```

Цей клас використовує метод, для отримання та встановлення значень приватних змінних класу, тобто використовую геттери і сетери. Вони надають доступ до цих значень ззовні класу і дозволяють контролювати спосіб доступу до даних. Що і використовує наступний клас.

```
using UnityEngine;
using System.Collections;
using System.Runtime.CompilerServices;

[RequireComponent(typeof(CharacterController))]
public class PlayerMove : MonoBehaviour
{
    [SerializeField] private Transform target;

    public float rotSpeed = 15.0f;
    public float moveSpeed = 15.0f;
    public float terminalVelocity = -10.0f;
    public float minFall = -1.5f;

    private float _vertSpeed;
    private float _vertSpeedDemo;


    [SerializeField] public bool stopMove = true;

    float jumpStartInterval = 0;
    public float jumpEndInterval = 1;

    int jumpCount = 0;
    public int maxJump = 1;

    bool doublejump;
```

```
private CharacterController charController;
private PlayerManager playerManager;
private ControllerColliderHit _contact;

public Animator animator;

float jumppower = 2.2f;


void Start()
{

    playerManager = GetComponent<PlayerManager>();
    charController = GetComponent<CharacterController>();
    _vertSpeed = minFall;

}

void Update()
{

    Vector3 movement = Vector3.zero;

    float horInput = Input.GetAxis("Horizontal");
    float vertInput = Input.GetAxis("Vertical");


    if (horInput != 0 || vertInput != 0)
    {
        movement.x = horInput * moveSpeed;
        movement.z = vertInput * moveSpeed;

        animator.SetFloat("Speed", Vector3.ClampMagnitude(movement, 1).magnitude);


        movement = Vector3.ClampMagnitude(movement, moveSpeed);
```

```
        Quaternion tmp = target.rotation;

        target.eulerAngles = new Vector3(0, target.eulerAngles.y, 0);
        movement = target.TransformDirection(movement);

        target.rotation = tmp;
        Quaternion direction = Quaternion.LookRotation(movement);

        transform.rotation = Quaternion.Lerp(transform.rotation, direction, rotSpeed * Time.deltaTime);



    }

    animator.SetBool("isJump", !IsGround());
    movement.y = Jump();
    movement *= Time.deltaTime;

    if (stopMove)
    {
        charController.Move(movement);
    }

    animator.SetFloat("JumpSpeed", _vertSpeed);

}

public bool IsGround()
{
    bool hitGround = false;
    RaycastHit hit;

    if (_vertSpeed < 0 && Physics.Raycast(transform.position, Vector3.down, out hit))
    {
        float check = (charController.height + charController.radius) / jumppower;
        if (hit.transform.tag == "platform")
        {
```

```
            hitGround = hit.distance <= check;

        }

    }


    return hitGround;
}


public float Jump()
{

    if (IsGround())
    {
        jumpStartInterval = 0;
        jumpCount = 0;
        doublejump = false;

        if (Input.GetButton("Jump"))
        {

            _vertSpeed = playerManager.GetJumpSpeed();
            _vertSpeedDemo = _vertSpeed;
        }
        else
        {

            _vertSpeed = minFall;
        }
    }
    else
    {

        if (Input.GetButton("Jump") && jumpStartInterval < jumpEndInterval)
        {
```

```
    if (_vertSpeedDemo <= _vertSpeed)
    {
        jumpStartInterval = jumpStartInterval + 1f * Time.deltaTime;
        _vertSpeed = playerManager.GetJumpSpeed();
    }
    else
    {
        _vertSpeed += playerManager.GetGravity() * 5f * Time.deltaTime;

        if (_vertSpeed < terminalVelocity)
        {
            _vertSpeed = terminalVelocity;
        }
    }

}
else
{
    _vertSpeed += playerManager.GetGravity() * 5 * Time.deltaTime;

    if (_vertSpeed < terminalVelocity)
    {
        _vertSpeed = terminalVelocity;
    }

}

if (Input.GetButtonUp("Jump"))
{
    doublejump= true;
    jumpStartInterval = jumpEndInterval;

}

if (jumpCount < maxJump && doublejump)
{

    if (Input.GetButton("Jump"))
```

```
            {
                jumpCount++;
                _vertSpeed = 13f;
            }


        }



    }

    return _vertSpeed;
}




public void setSpeed(float speed)
{
    moveSpeed = speed;
}

public float getSpeed()
{
    return moveSpeed;
}

public void setJumpPower(float jumppower)
{
    this.jumppower = jumppower;
}

public float getJumpPower()
{
    return jumppower;
}
```

```
}
```

PlayerMove це скрипт для керування рухом гравця за допомогою клавіш на клавіатурі. Він використовує CharacterController, компонент для передачі руху гравця.

Щоб передати рух персонажа, викликається метод charController.Move(movement). charController - це посилання на компонент CharacterController, а movement - вектор руху, який вже був обчислений.

Метод Move() CharacterController виконує кілька дій. По перше виконує врахування колізії інших об'єктами на сцені та робить перевірки для забезпечення правильного переміщення, враховуючи перешкоди. По друге застосовує гравітацію на персонажа, тобто додає силу тяжіння. Значення якого бере у класа PlayerManager. І останнє, обчислює вектор руху тим самим зміщує позицію гравця. Третій етап можливо складніше зрозуміти, тому про нього детальніше.

Після того як скрипт отримує данні про те що гравець натиснув кнопку руху, в коді це реалізовуєця за допомогою функції Input.GetAxis("Horizontal") рух по горизонталі, та рух по вертикалі Input.GetAxis("Vertical"). Після цього скрипт створює вектор руху movement, після отримання значень вводу множе її на швидкість руху moveSpeed для визначення швидкості руху гравця у конкретному напрямку.

Взаємодія головного героя з фізичними об'єктами

Взаємодія головного героя з фізичними об'єктами в Unity може бути реалізована за допомогою різних механік, таких як колізії та фізика. Колізії використовуються для виявлення зіткнень між героєм та іншими об'єктами у світі гри. Після виявлення зіткнення можна виконувати різні дії, такі як зміна стану героя, відтворення анімації, нанесення шкоди та інше.

Взаємодія героя з об'єктами є дуже важливою механікою гри, наприклад взяти нагороду яка находиться на платформі.

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

public class TakeValue : MonoBehaviour
{

   CreateHouse createHouse;
   public GameObject lesopilka;

   private void Start()
   {
      createHouse = lesopilka.GetComponent<CreateHouse>();
   }

   private void OnTriggerEnter(Collider other)
   {

      createHouse.setIsCreateHouse(true);
      Destroy(gameObject);

   }
}
```

Зіткнення регуструє функція OnTriggerEnter, який викликається, коли цей об'єкт зіткнувся з іншим об'єктом, у якого є колайдер. Після цього цей об'єкт, на якому знаходиться скрипт TakeValue, знищується за допомогою Destroy(gameObject), що видаляє сам себе.

Також колайдер головного героя, може взаємодіяти з пастками. Наприклад звичайний шар, який літає біля платформ, при дотику відштовхне героя за кордон хмаринки.

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

public class ExplosionCube : MonoBehaviour
{

   public float explosionRadius;
   public float explosionPower;


   bool isExplosion;

   private void Update()
   {
      if (isExplosion)
      {
         Explode();
      }
   }

   private void OnTriggerEnter(Collider other)
   {
      if (other.tag == "Player")
      {
         StartCoroutine("startExplosion");
      }
   }
   private void Explode()
   {

      Collider[] colliders = Physics.OverlapSphere(transform.position, explosionRadius);
```

```
        for (int i = 0; i < colliders.Length; i++)
        {
            ForceCharacter script = colliders[i].transform.GetComponent<ForceCharacter>();

            if (script)
            {
                var dir = colliders[i].transform.position - transform.position;
                var force = Mathf.Clamp(explosionPower / 3, 0, 15);
                script.AddImpact(dir, force);

            }

        }

    }

    IEnumerator startExplosion()
    {
        isExplosion = true;
        yield return new WaitForSeconds(1.5f);
        isExplosion = false;
    }

}
```

У методі OnTriggerEnter відбувається перевірка, чи об'єкт, з яким стикається цей об'єкт, має тег "Player". Якщо так, запускається корутин startExplosion. Для кожного колайдера перевіряється наявність скрипту ForceCharacter. Якщо такий скрипт є, обчислюється напрямок вибуху та сила впливу, а потім викликається метод AddImpact скрипту ForceCharacter, передаючи йому напрямок та силу.

4. Розробка концепції геймплею

Головна ідея гри полягає в самому геймплеї - взаємодії гравця з грою. Щоб створити захоплюючий досвід, необхідно зробити основну механіку гри, характерну для її жанру, цікавою і захоплюючою. У випадку платформера, це основною механікою є стрибки, які займають центральне місце в геймплеї та використовуються протягом усієї гри. Стрибки в платформері відіграють ключову роль, оскільки гравець використовує їх для подолання перешкод, долання висоти та досягнення нових областей. Вони є основним інструментом гравця для переміщення і дії у грі. Тому велика увага приділяється розробці та налаштуванню механіки стрибків з метою зробити їх захоплюючими та задовольняючими.

Механіка гри та геймплейні елементи

Механік в грі безліч, це і просто стрибки по рівням гри, а і більш складні, такі як головоломки та особливі здібності головного персонажу. Поміж іншими елементами геймплею, платформери можуть також включати бойові механіки, збирання предметів, розв'язування головоломок та інші взаємодії з оточенням. Але основа залишається незмінною - стрибки, які створюють динамічний та захоплюючий геймплей для гравця. Стрибки повинна бути цікавою та захоплюючою, щоб забезпечити задоволення від кожного натискання пробілу по клавіатурі. Для досягнення цього можна дотримуватися деяких правил "гарного платформера"

Відчуття повного контролю над головним героєм стрибків повинна бути точною та передбачуваною.

Реалізація скрипту ForceCharacter

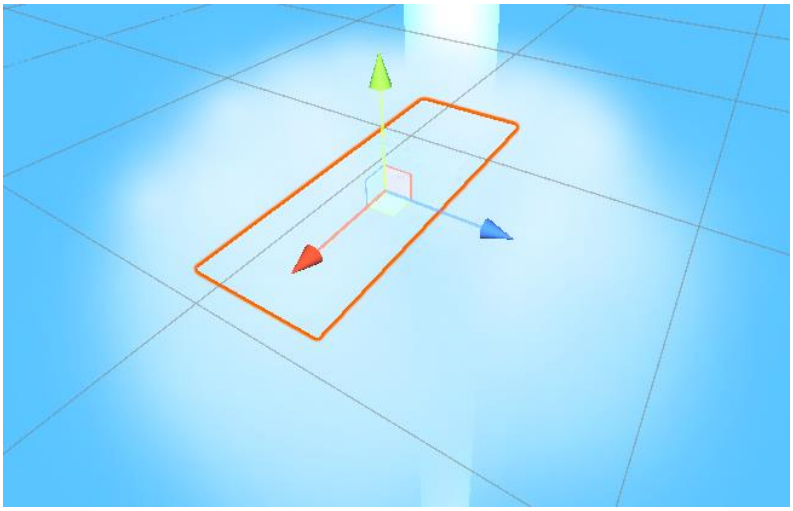
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ForceCharacter : MonoBehaviour
{
   float mass = 3.0F;
   Vector3 impact = Vector3.zero;
   private CharacterController character;

   void Start()
   {
      character = GetComponent<CharacterController>();



   }



   void Update()
   {

      if (impact.magnitude > 0.2F) character.Move(impact * Time.deltaTime);
      impact = Vector3.Lerp(impact, Vector3.zero, 5 * Time.deltaTime);
   }

   public void AddImpact(Vector3 dir, float force)
   {
      dir.Normalize();
      if (dir.y < 0) dir.y = -dir.y;
      impact += dir.normalized * force / mass;
   }
```

}

Цей скрипт дозволяє "штовхати" персонажа у визначеному напрямку з певною силою, що приводить до його руху у стрибку та додає динамічність і можливості для симуляції реалістичних реакцій персонажа на зовнішні впливи.

Різноманітність рівнів, для збереження інтересу гравця протягом гри, важливо мати різноманітність рівнів з різними викликами та складностями. Нові перешкоди, вороги або головоломки на кожному рівні.

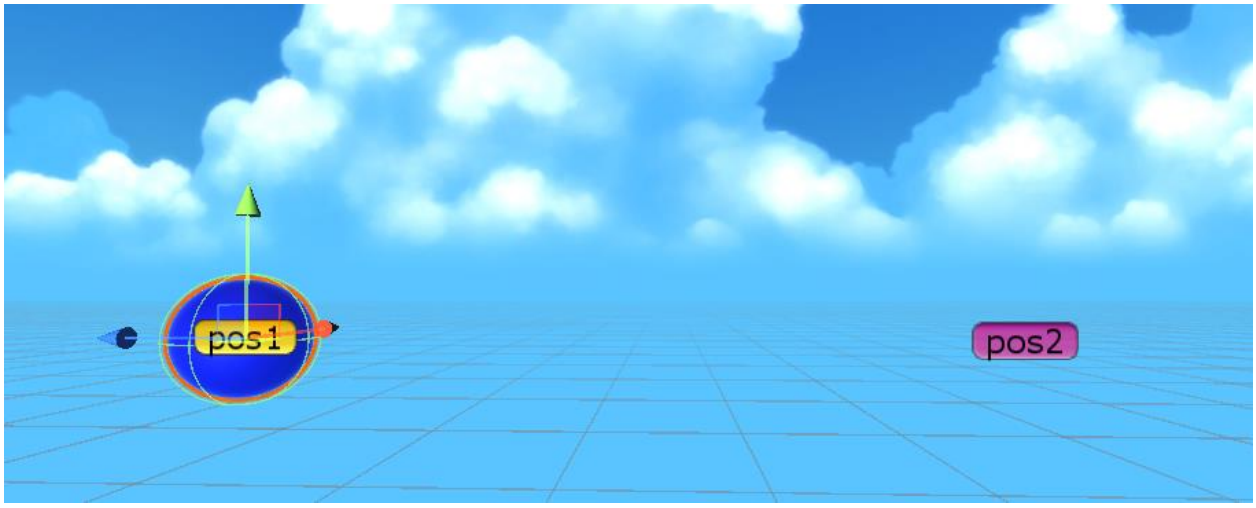
Одна з основних пасток це скритна міні платформа на яку якщо встає гравець, відбувається анімація та за допомогою скрипта ExplosionCube вона відштовхує ігрока за межі платформи.



13.1 Скритна пастка

Ігроку потрібно бути завжди насторожі, коли він стрибає на нову платформу, щоб не потрапити в пастку і не вилітити за межі платформи. Це стимулює гравця бути уважним при кожному стрибку та аналізувати оточуючі платформи перед тим, як на них вступати.

Наступна ловушка в грі працює за схожою схемою, але має свої особливості. Вона поєднує скрипт ManagerPlatform з об'єкта PrefabMove та скрипт ExplosionCube. Ця ловушка представляє собою невеликий шар, який рухається від позиції Pos1 до позиції Pos2 зі заданою швидкістю. Особливість цієї ловушки полягає в тому, що якщо гравець перебуває на шляху цього рухомого шару, то за допомогою скрипта ExplosionCube він буде відштовхнутий на задану силу.



13.2 Друга пастка

Ще одна механіка гри під назвою “Спел бук”. Це предмет у виды книжки, який можно купити у жителя раю, в ній зберігаються здібності головного персонажу по мирі того, як добре він буде досліджувати рівні гри та виконувати завдання.

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

public class DoubleJump : MonoBehaviour
```

```
{

    PlayerMove PlayerMove;
    GameObject player;
    public int count = 1;
    int countjump = 0;

    public GameObject batut;
    public GameObject Particle;

    private CharacterController charController;

    private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        PlayerMove = player.GetComponent<PlayerMove>();
        charController = player.GetComponent<CharacterController>();
    }

    private void Update()
    {

        if (!PlayerMove.IsGround())
        {

            if (Input.GetKeyDown(KeyCode.E))
            {



                if (countjump > 0)
                {
                    Instantiate(batut, player.transform.position - new Vector3(0, -1.5f), Quaternion.identity);
                    countjump -= 1;
                }


            }
        }else if (PlayerMove.IsGround())
        {
```

```
            countjump = count;

        }
    }

}
```

Здібність DoubleJump не просто ще один запасний стрибок. Він працює по іншій логиці, він якби штовхає головного героя вперед, що дає більше варіацій проходження рівня. Наприклад дістатися на нові етапи рівня, або знайти швидшу варіацію його проходження.

Налаштування балансу складності рівнів

Для налаштування балансу складності рівнів зручніше за все використовувати метрику, щоб оцінити рівень складності та зрозуміти, наскільки гравці можуть успішно пройти кожен рівень. Метрика допомагає розробникам зробити налаштування та коригування рівнів. Метрика “процент проходження рівнів” вона вимірює відсоток гравців, які успішно завершили кожен рівень гри. Чим більший процент, тим менша ймовірність, що рівень є надто складним. Хоча гра повинна давати челенж для гравця, на сьогоднишній день коли є стільки розважальних проектів, потрібно привертати максимальну увагу гравця на гру, тому в якщо для більшості гравців рівень занадто складний, потрібно зменшити його складність на оптимальний рівень.

Неможливо недооцінити важливість тестування гри для розробників, особливо коли відсутня спеціальна команда тестувальників. У такій ситуації, я, як розробник, самостійно проводжу внутрішнє тестування, але розумію, що це не є найідеальнішим рішенням. Тим не менше, для досягнення кращих результатів, я просив кількох своїх друзів протестувати основні рівні та механіки гри та надати мені свій фідбек.

Щоб краще зрозуміти пропорції ігрового світу, та налаштувати баланс гри, потрібно використовувати метричні блоки які допомагають зрозуміти наскільки високо та далеко стрибає основний герой для подальшого проектування рівня гри. Використання метричних блоків допомагає визначити оптимальну складність рівнів гри. За допомогою цієї системи можно експериментувати з різними комбінаціями метричних блоків, включаючи їх розташування, висоту та відстань між ними, щоб забезпечити цікавий та викликаючий геймплей. Відповідно до реакцій та фідбеку від гравців, розробники можуть вносити корективи в баланс складності рівнів, забезпечуючи насолоду та виклик для гравців.

14.1 Метричні блоки при розробці гри "Uncharted"

14.2 Метричні блоки мого проекту

Система камери

Основною метою системи камери є забезпечення належного положення та орієнтації камери для забезпечення оптимального огляду ігрового світу. Камера повинна відстежувати рухи гравця, щоб завжди знаходитися у відповідному положенні та адекватно реагувати на небезпеки або виконувати складні рухи.

Реалізація коду Камери

```
using UnityEngine;
using System.Collections;

public class RotationCamera : MonoBehaviour
{
   [SerializeField] private Transform target;

   public float rotSpeed = 1.5f;

   public LayerMask maska;

   private float max_distnce;
```

```
private float _rotY;
private float _rotX;

private Vector3 _offset;
private Vector3 positioner
{
   get
   {
      return transform.position;
   }

   set
   {
      transform.position = value;
   }
}


void Start()
{
   _rotY = transform.eulerAngles.y;
   _rotX = transform.eulerAngles.x;
   _offset = target.position - transform.position;
   max_distnce = Vector3.Distance(positioner, target.position) + 0.1f;
}

void LateUpdate()
{


   _rotX += Input.GetAxis("Mouse Y") * rotSpeed * 2;
   _rotY += Input.GetAxis("Mouse X") * rotSpeed * 2;



   Quaternion rotation = Quaternion.Euler(_rotX, _rotY, 0);
   transform.position = target.position - (rotation * _offset);
   transform.LookAt(target);

   distanceCamera();

}
```

```
  void distanceCamera()
  {
     float distance = Vector3.Distance(positioner, target.position);

     RaycastHit hit;

     if (Physics.Raycast(target.position, transform.position - target.position, out hit, max_distnce, maska))
     {
        positioner = hit.point;
     }
     else if (distance < max_distnce && !Physics.Raycast(positioner, -transform.forward, 0.1f, maska))
     {
        positioner -= transform.forward * 0.05f;
     }

  }

}
```

Цей код відповідає за рух камери відносно цілі у 3D середовищі. Основна функція цього скрипта - забезпечити можливість обертання камери навколо цілі за допомогою миші. Камера може автоматично адаптуватися до змінної висоти або відстані між гравцем та об'єктами навколо.

Метод distanceCamera() використовується для керування відстанню між камерою та ціллю. Він вимірює відстань між ними і перевіряє, чи перешкоджає щось цій відстані. Якщо перешкоду знаходиться на шляху, камера буде розташована на ближчій точці зіткнення. Якщо відстань менша за максимально допустиму і немає перешкоди, камера наближується до цілі на невелику відстань.

5. Висновки третього розділу

У розділі про розробку концепції геймплею були визначені основна мета гри, механіка гри та геймплейні елементи. Фізика забезпечує реалістичну поведінку об'єктів у грі. Алгоритми фізики платформ та фізична модель головного героя дозволяють гравцю контролювати рух персонажа та взаємодіяти з оточуючими об'єктами. Також дизайн UI елементів забезпечити гравцю зручність та легкість взаємодії з грою, а інтерфейс взаємодії з NPC забезпечує можливість взаємодії з неігровим персонажам. Сам дизайн був виконаний в стилі гри, який забезпечив занурення у світ гри ще з самого початку. Стартове меню гри було розроблено з дотриманням в стилі гри, щоб воно задавало правильний настрій перед початком гри.

Посилання

1. Збірник корисної інформації з дизайну рівнів [Електронний ресурс] – Режим доступу до ресурсу: https://docs.google.com/document/d/1fAlf2MwEFTwePwzbP3try1H0aYa9kpVBHPBkyIq-caY/preview?pru=AAABcuV6jKM*_MWUhVg70PB_UoSLJt2wpA

2. Як роблять хороший UI в іграх (а як поганий). Доступно: https://www.youtube.com/watch?v=k7NrZnJ5DtY&t=955s&pp=ygUueHl6INC00LjQt9Cw0LnQvSDRgNCw0LfRgNCw0LHQvtGC0LrQsCDQuNCz0YDRiw%3D%3D

3. Гра Super Mario Bros [Електронний ресурс] – Режим доступу до ресурсу: https://mario.nintendo.com

4. Гра Crash Bandicood [Електронний ресурс] – Режим доступу до ресурсу: https://www.crashbandicoot.com

5. Гра Psychonauts 2 [Електронний ресурс] – Режим доступу до ресурсу: https://www.doublefine.com/games/psychonauts-2

6. Гра Hades [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/1145360/Hades

7. Гра Dead Space [Електронний ресурс] – Режим доступу до ресурсу: https://www.ea.com/ru-ru/games/dead-space

8. Гра Uncharted [Електронний ресурс] – Режим доступу до ресурсу: https://www.playstation.com/en-us/uncharted

9. Як влаштована висока складність в іграх і навіщо потрібний хардкор [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=WAv4Wy9Gjvg&t=420s&pp=ygUueHl6INC00LjQt9Cw0LnQvSDRgNCw0LfRgNCw0LHQvtGC0LrQsCDQuNCz0YDRiw%3D%3D

10. Як створити рівень у грі. Умови, структура, деталізація, план [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=TuRiXIYFz2w&t=394s&ab_channel=DTF

11. Левелдізайн: "Метод Нінтендо". Дизайн рівнів у Super Mario Bros [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=Hb3Aw_TPqH0&pp=ygU70YDQsNC30YDQsNCx0L7RgtC60LAg0LjQs9GAINGO0L3QuNGC0Lgg0L_Qu9Cw0YLRhNC-0YDQvNC10YA%3D

12. Unity Asset Store [Електронний ресурс] – Режим доступу до ресурсу: https://assetstore.unity.com.

13. Manual Unity [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ru/530/Manual/UnityManual.html

14. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5 / Joseph Hocking., 2015. – 352 с.

15. Правила хорошого платформера [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=DmNMi7PG_yQ&t=23s&ab_channel=GDevAcademy

16. #5 - Контрольований стрибок у Unity 2D [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=mMcxl6w__do&t=65s&ab_channel=NoobGameDev

17. #6 - Мультистрибок для персонажа у Unity 2D [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=EaktzIEsSCk&ab_channel=NoobGameDev