МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Розробка гри на основі фреймворку Unity

Текстова частина до курсової роботи за спеціальністю “Інженерія програмного забезпечення”

Керівник курсової роботи

Старший викладач Борозенний С.О.

_____

“____” _____ 2022 р.

Виконав студент

Ставровський М.А.

“____” _____ 2022 р.

Київ 2022

Зміст

Тема: Розробка гри на основі фреймворку Unity

Календарний план виконання роботи:

№ п/п	Назва етапу	Термін виконання етапу	Примітка
1	Отримання теми курсової роботи	Листопад 2021р.	
2	Дослідження фреймворку та планування проекту	Лютий 2022р.	
3	Розробка застосунку	Квітень 2022р.	
4	Написання текстової частини	Квітень 2022р.	
5	Коригування роботи згідно зауважень керівника	Травень 2022р.	
6	Створення презентації	Травень 2022р.	

Студент Ставровський М.А.

Керівник Борозенний С.О.

“_____” _____

Анотація

Ця курсова робота присвячена розробці відео-гри на основі фреймворку Unity. Проведено аналіз предметної області та розглянуто існуючі рішення та технічні засоби, інструменти мобільної розробки.

Вступ

Ігрова індустрія стрімко розвивається в сучасному світі. Те що починалося з гри в pong переросло мультимільярдний ринок насичений як AAA іграми розроблених сотнями людей, так і унікальними інді проектами. Проте рушії усієї індустрії лишаються незмінними - інноваційні креативні ідеї та їхня технічна імплементація, яка потребує постійного рішення різноманітних задач.

Нині ігри створюються за допомогою комплексних інструментів, які мають назву ігрові рушії (Game Engine). Вони задають центральну програмну частину гри, та спрощують процес розробки ігор за допомогою уніфікації та систематизації її внутрішньої структури. [1]

В межах цієї курсової роботи розглянутий ігровий рушій Unity. Він є одним з найпопулярніших інструментів розробки ігор, має обширний функціонал та зручний в користуванні.

Мета курсової роботи - дослідження процесу створення відео-гри за допомогою фреймворку Unity та розробка власної відео-гри під назвою “Raildown”.

У першому розділі описані теоретичні відомості про розробку ігор та геймдизайн. У другому розділі розглянута концепція гри та запропоновані попередні рішення. У третьому розділі проведено аналіз середовища розробки Unity. У четвертому розділі описаний процес створення власної гри. Фінальний розділ демонструє результати тестування та описує можливі покращення.

Розділ 1: Теоретичні відомості

Цей розділ присвячений важливим теоретичним відомостям геймдизайну в контексті даної курсової роботи.

За візуальною репрезентацією ігри зазвичай поділяються на двовимірні та тривимірні. Це зокрема визначає тип об’єктів якими гра оперує. Тривимірні ігри створюються на базі об’єктів які рухаються у тривимірному просторі. Такі об’єкти найчастіше задаються 3D моделями які складаються з полігонів. Двовимірні ігри ж маючи представлення на площині можуть оперувати як 3D моделями які проектуються на 2D площину, так і 2D об’єктами під назвою *спрайт*. У межах цієї курсової роботи розглянута розробка двовимірної гри, яка оперує спрайтами.

Спрайт - це двовимірне зображення, яке є частиною ігрової сцени. Спрайти можуть бути статичними або анімованими. Статичний спрайт зазвичай задається як зображення із одиничного файлу.[2]

(Рис. 1.1) Статичний спрайт[2]

Анімований спрайт теж задаєься одиничним файлом, так званим *аркушем спрайтів*, який містить на одному зображенні набір кадрів спрайту, які потім програмно комбінуються в анімацію. Кадри розташовані поруч у вертикальному або горизонтальному порядку.

(Рис. 1.2) Анімований спрайт[2]

Одиничні спрайти найчастіше використовують для конкретних унікальних об’єктів, наприклад, для спрайтів гравця. Проте існує інший спосіб визначення спрайтів - за допомогою *тайлсетів*.

Тайлсет - це файл який містить великий набір спрайтів, розташованих на квадратній сітці. Тайлсет розбивається на масив спрайтів, які потім можуть бути використані будь-якими об’єктами. Тайлсети найчастіше використовують для створення навколишнього середовища. [4]

(Рис. 1.3)Приклад тайлсету[4]

Для використання тайлів гра повинна бути побудована на основі *тайлової системи*. Гра на базі тайлів - це гра, ігрова частина якої складається із зазвичай квадратних об’єктів (тайлів) розташованих на сітці. [5]

(Рис. 1.4)Приклад гри на базі тайлів[6]

Ключовим аспектом будь-якої гри є *геймплей*. *Геймплей* - це опис взаємодії гравця та гри, який включає в себе опис правил, опис цілей та способи їх досягти, опис керування грою, опис відображення гри та загальний бажаний досвід гравця. [7]

Одним з аспектів геймплею є перспектива гри. Від перспективи залежить положення камери і відповідно те, як гравець бачить гру. Існують такі можливі перспективи:[8]

- *Від першого обличчя* - вигляд гри, при якому камера прикріплена до гравця так, щоб світ було видно ніби від обличчя гравця. Такий вигляд використовується найчастіше для ігор жанру шутер.
- *Від третього обличчя* - вигляд гри, при якому камера розташована за спиною гравця на певній відстані, щоб було видно як світ довкола так і самого гравця. Такий вигляд використовується для ігор жанру слешер, 3D RPG тощо.

- *Вид збоку* - вигляд гри, при якому камера розташована збоку, а світ проектується на площину. Такий вигляд є найбільш притаманним для ігор жанру платформер, файтинг тощо.
- *Вид згори* - вигляд при якому камера знаходиться над полем гри, даючи гравцеві більше інформації про навколишнє ігрове середовище. Такий вигляд зазвичай використовується для ігор жанру RPG, стратегій тощо.

Вид від першого та третього обличчя більш притаманні тривимірним іграм, в той час коли вид збоку та згори найчастіше використовується для двовимірних ігор. В контексті цієї курсової роботи розглянутий тільки вид згори.

В залежності від геймплею існує велика кількість жанрів ігор. У цій курсовій роботі актуальними наступні жанри: головоломки та пригодницькі ігри.

Ігри головоломки - це жанр, який зосереджений на розв'язанні певних задач із заданими правилами. Ігри головоломки зазвичай складаються з окремих невеликих рівнів, де гравцеві потрібно вирішувати певні задачі на логіку, стратегію, розпізнавання образів тощо.

Пригодницькі ігри - це жанр, який характеризується дослідженням та розв'язанням головоломок в рамках наративу гри. [9]

Наратив також є складовою геймплею, яка відповідає за подачу гравцеві сюжету, правил, сенсу гри. Наратив може бути використаним як для задання мотивації гравцеві так і для надання інструкцій як правильно взаємодіяти з грою. Виділяють наступні типи наративу:[10]

- *Вбудований наратив* - це наперед визначений прямий наратив, на який гравець ніяким чином не може вплинути. Такий наратив може задавати статичний сюжет, за яким гравець повинен слідувати. Також такий наратив використовується для надання обов'язкових чітких інструкцій.
- *Виникаючий наратив* - це прямий наратив, який так само як і вбудований має певні чіткі правила, проте також містить елементи на які може впливати гравець. Це досягається у вигляді надання гравцеві рішень які змінюють

наратив, наприклад через діалоги з персонажами гри, вибір дії в певній ключовій сюжетній частині, пряме питання від самої гри тощо.

- *Евокативний наратив* - це непрямий наратив, який в більшій мірі розрахований на домислення самого гравця. Такий наратив не надає чітких інструкцій або сюжетної експозиції, лишаючи гравцям лише певні образні елементи, які можуть бути сприйняті по різному, відповідно створюючи суб'єктивний наратив для кожного гравця.
- *Розіграний наратив* - це наратив, який залежить від внутрішньо ігрового розвитку гравця, радше ніж від його конкретних рішень, наприклад в залежності від рівня, кількості певних ресурсів тощо. Такий наратив може бути як прямим так і не прямим.

Зазвичай ігри змішують різні наративи у різних пропорціях, створюючи більш динамічні наративи.

Останній аспект геймплею, який я розглядатиму - лінійність прогресу. Прогрес в іграх може бути лінійним та нелінійним. Лінійні ігри прогресують лінійно, наприклад є чітка лінійна прогресія рівнів, або гра слідує за чітким вбудованим наративом, який веде до одного рішення. Лінійні ігри зазвичай мають одне рішення, один кінець.

Нелінійні ігри - це ж ігри які прогресують нелінійно. Нелінійність може виражатися у наступних аспектах:[11]

- *Сюжет* - така нелінійність створюється за допомогою створення нелінійного сюжетного наративу, наприклад використовуючи виникаючий наратив, надаючи гравцеві можливість робити рішення, які впливають на прогресію наративу.
- *Різні рішення* - нелінійність яка досягається за допомогою надання різних можливих рішень однакових проблем, що дозволяє гравцям отримувати нелінійний та різноманітний досвід тих самих частин гри.

- *Порядок рівнів* - порядок рівнів може бути не лінійним, даючи гравцеві більше свободи у виборі напряму прогресування. Така нелінійність запобігає стагнації досвіду гравця. Наприклад, якщо гравець не має успіху в одному напряму, він може спробувати прогресувати в іншому.
- *Вибірковість* - це нелінійність, яка досягається наданням гравцеві вибору які аспекти гри він хоче або не хоче мати. Прикладами такої нелінійності слугують вибір складності, необов'язкові рівні та можливість налаштування різних аспектів гри вручну.

Базуючись на всіх вищезгаданих відомостях, я виділю ще один актуальний для цієї курсової роботи жанр: *метроідванія*. *Метроідванія* - це піджанр пригодницьких відео-ігор, який у своїй основі має нелінійність рівнів та прогресію засновану на знаходженні та використанні певних покращень та інструментів. Свою назву піджанр отримав від ігор Metroid (1986) та Castlevania (1986), від яких розробники ігор унаслідували характеристики та методики дизайну рівнів та прогресії. [12]

Розділ 2: Концепція гри

Я вирішив назвати гру “**Raildown**”. Гра є двовимірною та має вигляд згори. Концепція гри наступна: коліями без упину рухається неконтрольована вагонетка. Гравець не може взаємодіяти з самою вагонеткою, але може взаємодіяти з навколишнім середовищем. Якщо вагонетка сходить з колій або відбувається зіткнення з відповідним об’єктом гравець програє. Ціль гри - знайти вихід та не дати вагонетці розбитися.

Гравець взаємодіє з навколишнім середовищем за допомогою мишки, перетягуючи коробки, які при зіткненні змінюють напрямок вагонетки, та перемикаючи важелі, які змінюють напрямок колій.

Усі рівні в Raildown будуть побудовані на основі *тайлової системи.* Таким чином рівні будуть прості у створенні. Я обрав двовимірну ретро стилістику для гри, тож тайлсет, який буде використаний має бути відповідним.

Одна з ключових характеристик геймплею Raildown - розв’язування головоломок на швидкість. Це здійснюється за допомогою обмеження поля зору гравця до невеликої ділянки навколо вагонетки. Таким чином, потрібно шукати рішення проблем не наперед, а залежно від руху вагонетки. Також, вагонетка може рухатися з різною швидкістю залежно від колій по яких вона їде, створюючи більше динаміки.

Загалом, рейки якими може рухатися вагонетка мають такі типи:

- *Звичайні* - вертикальні або горизонтальні рейки які дозволяють вагонетці рухатися далі в заданому напрямку. Вони мають три типи за швидкостями:
    - Швидкі
    - Звичайні
    - Повільні
- *Поворотні* - рейки, які слугують як поворот, та змінюють напрям руху вагонетки.

- *Змінні* - рейки, які мають 2 стани, які змінюються при перемиканні важеля поруч. Будь-який з двох станів може бути як повортоним так і звичайним.
- *Пружинні* - рейки, які при зіткненні змінюють напрям руху вагонетки на протилежний.
- *Зламані* - рейки, при зіткненні з якими вагонетка сходить з колій, та гравець програє.

Змінні рейки змінюються важелями. При перемиканні важеля мишкою, усі змінні рейки поруч з важелем в радіусі одного тайлу змінюють свій стан на альтернативний. Існує два типи важелів:

- *Звичайні* - звичайні важелі, які мають два стани між якими гравець перемикається мишкою.
- *Часові* - важелі, при перемиканні яких альтернативний стан заморожується на заданий час, після спливання якого важіль повертається до оригінального стану.

Коробки рухаються гравцем за допомогою перетягування мишки. Коробки не можуть знаходитися на тайлі, де є будь-які перешкоди. Перешкодами вважаються елементи середовища такі як важелі, каміння, інші коробки тощо.

Одна з особливостей Raildown також полягає в нелінійності прогресі. Нелінійність в цьому випадку досягається за допомогою порядку рівнів та різних рішень. Гра матиме велику кількість рівнів з різноманітними розгалуженнями та можливістю обрати шлях яким рухатися далі. Таким чином, будь-який рівень може мати більше ніж один вхід та вихід. Також з кожного рівня можливо повернутися назад тим самим входом. Змінювати рівні можна лише внутрішньо ігровими методами, не маючи меню вибору рівнів, що дає прогресуванню більше ваги для гравця.

Крім нелінійного прогресування рівнів гра міститиме елементи метроідванії у вигляді потреби знаходження певних інструментів аби подолати певні перешкоди, що створює потребу у дослідженні рівнів.

Наратив гри в своїй більшості є евокативним. Будь-які прямі сюжетні або інструкційні елементи є опціональними та знаходяться у табличках на рівнях. Гравець може відкрити діалогове вікно таблички натиснувши мишкою на неї. Також наратив може подаватися у вигляді деяких декоративних елементів рівнів.

Підсумовуючи, Raildown - гра жанру пригодницька головоломка з елементами метроідванії. Найголовніші технічні аспекти розробки гри - розробити ігрову систему, яка відповідає вказаній у концепції та розробити систему нелінійності переходів рівнів. Точний список вимог та їх розробка описані у четвертому розділі. У наступному розіділі розглянуте середовище розробки Unity.

Розділ 3: Фреймворк Unity

Unity - ігровий рушій для створення двовимірних та тривимірних крос-платформних ігор, вперше випущений у 2005 році. [13] Даний рушій є одним із найбільш популярних інструментів для розробок ігор. Зокрема він користується популярністю для створення малобюджетних (інді) та мобільних ігор за рахунок простоти інтерфейсу, великої кількості готових рішень та низького порогу входження для початку розробки. Рушій має такі готові рішення як готовий фізичний рушій, система аудіо, інструменти для роботи з двовимірними та тривимірними об’єктами, способи взаємодії гравця з додатком та інші.

Unity надає всі необхідні інструменти для розробки Raildown. У цьому розділі описані структура проектів Unity, графічний інтерфейс та найважливіші інструменти, які у подальшому були використані у розробці додатку.

3.1 Структура проекту

Основою будь-якого проекту на Unity є *сцени.* Сцена - це елемент ігрової структури який може містити в собі основне наповнення гри. Сцена може містити в собі як увесь ігровий контент, так і його частину. Прості ігри можуть бути зроблені повністю в межах однієї сцени, проте деякі потребують окремих сцен на різні рівні із власними об’єктами, середовищами, декораціями, інтерфейсом тощо. [14]

Кожна сцена складається з певного набору об’єктів (GameObject), які певним чином розташовані в ігровому просторі. Об’єкт також може містити під-об’єкти. Будь-який об’єкт всередині сцени має набір властивостей, які називаються компонентами. *Компонент* - це окрема структурна одиниця, яка собою гуртує певні змінювані характеристики, які задають поведінку батьківського об’єкту. Unity має велику кількість готових компонентів, таких як компоненти фізичних та світлових властивостей об’єкту, компоненти анімації, аудіо тощо. Кожен об’єкт має невід'ємний стандартний компонент - Transform, який визначає положення об’єкту в ігровому просторі (*Рис 3.1*).

(Рис. 3.1) Компонент Transform

Якщо готові компоненти не можуть описати потрібні властивості об’єкта розробник також може створювати власні компоненти у вигляді *скриптів*. Скрипт - це окремий файл, що містить певний код, який може бути виконаний. Unity написаний на мові програмування C, проте скриптингова система якою оперує рушій базована на C#. Таким чином, розробник може написати власну поведінку об’єкту, яка потім може бути додана як компонент.

Аналогічно до компонентів, Unity також має шаблони об’єктів, які містять певний набір наперед налаштованих компонентів. Існують наступні категорії об’єктів:

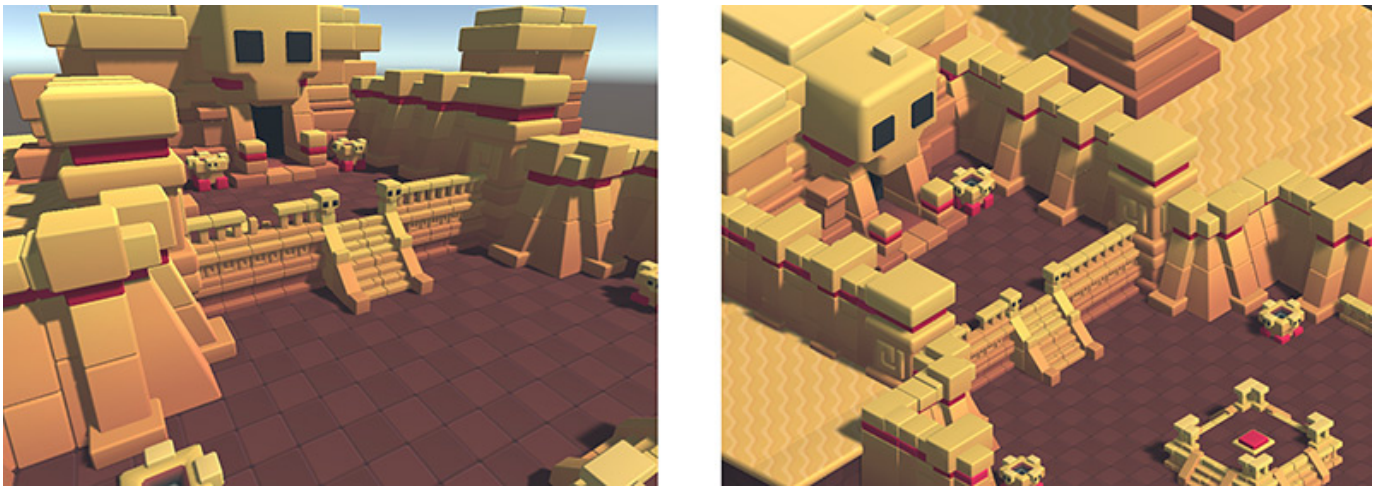
- *2D об’єкти* - об’єкти для використання у двовимірному контексті
- *3D об’єкти* - об’єкти для використання у тривимірному контексті
- *Ефекти* - об’єкти для створення систем візуальних ефектів
- *Освітлення* - об’єкти що слугують джерелом освітлення
- *Аудіо* - об’єкти джерела звуку або звукового ефекту
- *Відео* - об’єкти відігравачі відео файлів
- *UI* - об’єкти користувацького інтерфейсу

Також існує об’єкт поза вище названими категоріями - камера. Це ключовий об’єкт будь-якої сцени. Камера визначає ділянку ігрового простору чи площини яка буде відображатися гравцеві. Без камери рушій нічого не відображатиме, навіть якщо на сцені присутні інші об’єкти. Камера також визначає чи відображати гру двовимірною чи тривимірною.

Компонент камери містить тип проекції. Для коректного відображення тривимірного простору камера матиме перспективну проекцію, що означає що

камера враховує тривимірний простір та перспективу об’єктиву камери, параметри якого також налаштовуються, наприклад глибина поля зору.

Unity за замовчуванням оперує в тривимірному просторі. Для розробки двовимірної гри камера має мати ортографічний (ортогональний) тип проекції. Цей тип проекції проектує усі об’єкти на площину без урахування перспективи. Таким чином не потрібно змінювати систему координат усього двигуна для двовимірних ігор. Також це дозволяє створювати двовимірні ігри з використання тривимірних об’єктів. Візуальне порівняння перспективної та ортографічної камери зображено на *Рис 3.2*.



(Рис. 3.2) Порівняння перспективної та ортографічної камери[15]

Об’єкти кожної сцени зберігаються локально кожній сцені. При потребі використання однакових об’єктів на різних сценах, Unity має систему збірних об’єктів (Prefab System). Ця система дає можливість створювати, налаштовувати та зберігати ігровий об’єкт, разом з усіма його компонентами та підпорядкованими об’єктами, як об’єкт повторного використання (Prefab)[16]. Надалі, я називатиму такі об’єкти *шаблонними*. Шаблонні об’єкти зберігаються у файловій системі, і можуть бути використані на будь-якій сцені, створивши екземпляр шаблону. Їх можна редагувати як локально в контексті одного екземпляру, так і глобально, змінюючи характеристики для кожного екземпляру. Також, допускається створення шаблонних

об’єктів всередині шаблонних об’єктів, що дає можливість створювати комплексні шаблони об’єктів.

Об’єкти мають також такі властивості для ідентифікації та групування:

- *Назва* - назва об’єкту
- *Тег* - ярлик, який можна використовувати для групування об’єктів одного типу в одну категорію
- *Шар* - властивість, яка дозволяє групувати об’єкти за шарами

Теги і шари створюються та зберігаються окремо. Вони дозволяють групувати об’єкти схожого типу та схожого положення відносно камери для використання у скриптах.

3.2 Основні Об’єкти та Компоненти

У цьому підрозділі описані основні готові об’єкти та компоненти використані в контексті даної курсової роботи.

Для розробки двовимірної гри розглянуті наступні компоненти:

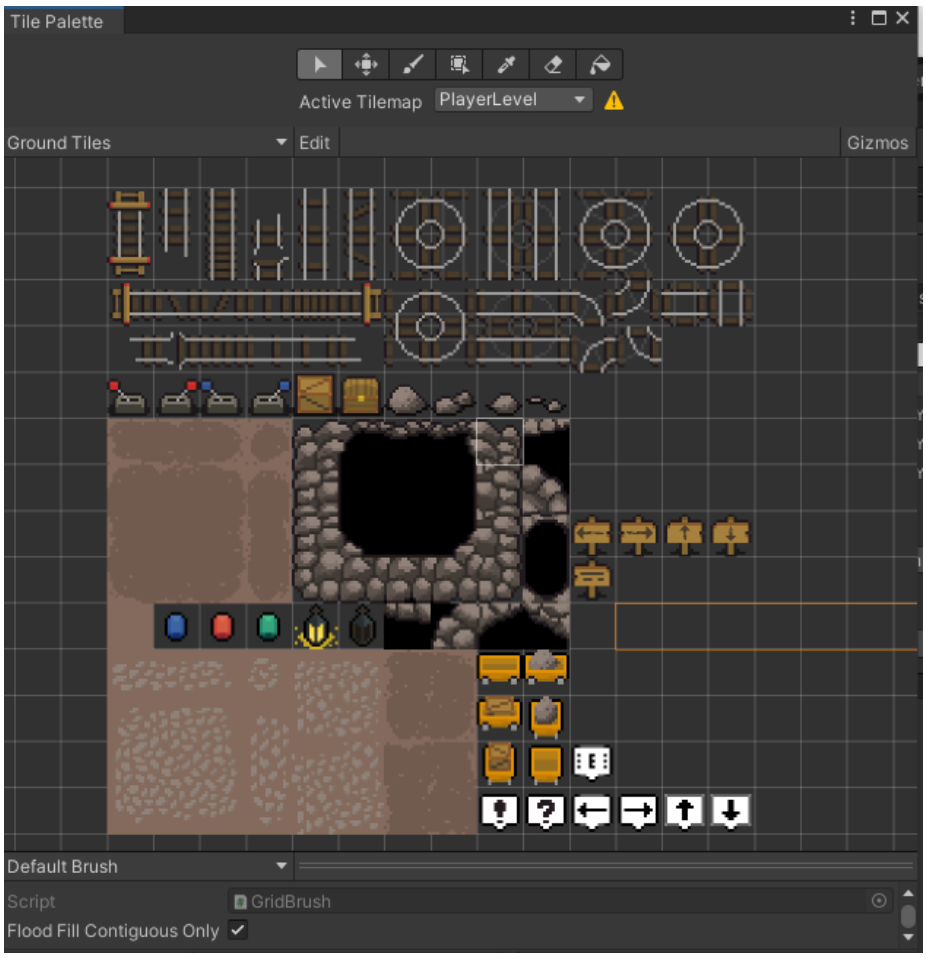
- *Sprite Renderer* - компонент який задає об’єкту спрайт для відображення. Всередині цього компоненту можна задатии сам спрайт, матеріал для рендерингу, шар рендерингу та порядок в ньому. Шар рендерингу визначає ієрархію об’єктів відносно камери. Таким чином, об’єкти що знаходяться на вищому шарі, будуть завжди відображатися поверх об’єктів нижчого шару. Також об’єкт всередині одного шару відображається поверх інших цього ж шару, якщо у нього вищий порядок в шарі.
- *Box Collider 2D* - компонент, який задає об’єкту межі зіткнення в двовимірному просторі. Всередині компоненту можна налаштувати форму та розмір меж. Також компонент має два стани - колайдер та трігер. У стані колайдера, при зіткненні об’єкт буде реагувати відповідно до заданих фізичних властивостей об’єкту. У стані трігеру, при зіткненні об’єкт не реагує фізично, а лише подає сигнал про зіткнення, що дозволяє використовувати його в коді.

- *Rigidbody 2D* - компонент, що задає фізичні властивості об’єкту у двовимірному просторі, які симулюються фізичним рушієм Unity. Компонент дозволяє налаштувати тип симуляції, вагу об’єкту, згладжування симуляції руху, вплив сили гравітації тощо. Існує три типи симуляції [17]:
    - Динамічний - симуляція руху фізичним рушієм здійснюється константно, на об’єкт діють усі зовнішні сили зокрема і сили гравітації.
    - Кінетичний - симуляція руху відбувається лише при конкретних діях гравця, відповідно на об’єкт не діють постійні сили як гравітація.
    - Статичний - симуляція руху фізичним рушієм не здійснюється, та при будь-якому зіткненні з іншим об’єктом статичний об’єкт вважається як нерухомий, тобто той, що має нескінченно велику вагу.

      Статичний та кінетичний типи симуляції потребують значно менше ресурсів рушія для симуляції.

Крім компонентів також були розглянуті наступні об’єкти:

- *Tilemap* - це комплексний об’єкт, який дозволяє імплементувати тайлову систему в Unity. Цей об’єкт створюється та складається з об’єкту Grid та підпорядкованих об’єктів Tilemap. Grid - об’єкт який задає тайлову сітку, відповідно якій розставляються тайли. Кожен Tilemap об’єкт - задає один шар тайлів. Tilemap також містить тайлсет з яким тайлмапа працює. Unity надає візуальний інструмент під назвою Tile Palette для використання тайлсету та малювання ним на тайлмапі (Рис 3.3). Цей інструмент надає можливість обрати ділянку тайлсету, малювання, видалення та заливки. Об’єкт Tilemap може містити в собі компонент *Tilemap Collider*, який задає область зіткнення для всієї тайлмапи. Принцип дії аналогічний до Box Collider 2D.



(Рис. 3.3)Tile Palette

- *UI об'єкти* - це категорія об'єктів, які дозволяють створювати користувацький інтерфейс, зокрема меню. Для коректного відображення усі UI об'єкти повинні бути підпорядковані об'єкту типу *Canvas*. Canvas задає абстрактний простір, який використовується для рендерингу елементів інтерфейсу всередині. [18] Panel - це елемент UI, який задає візуальну площину, або панель, на якій можуть бути розташовані інші елементи. Інші часто використовувані об'єкти користувацького інтерфейсу налічують:
    - Text - текстова ділянка
    - Button - кнопка, яка містить метод OnClick якому можна присвоїти потрібні дії, які повинні виконуватись при натисненні на кнопку.

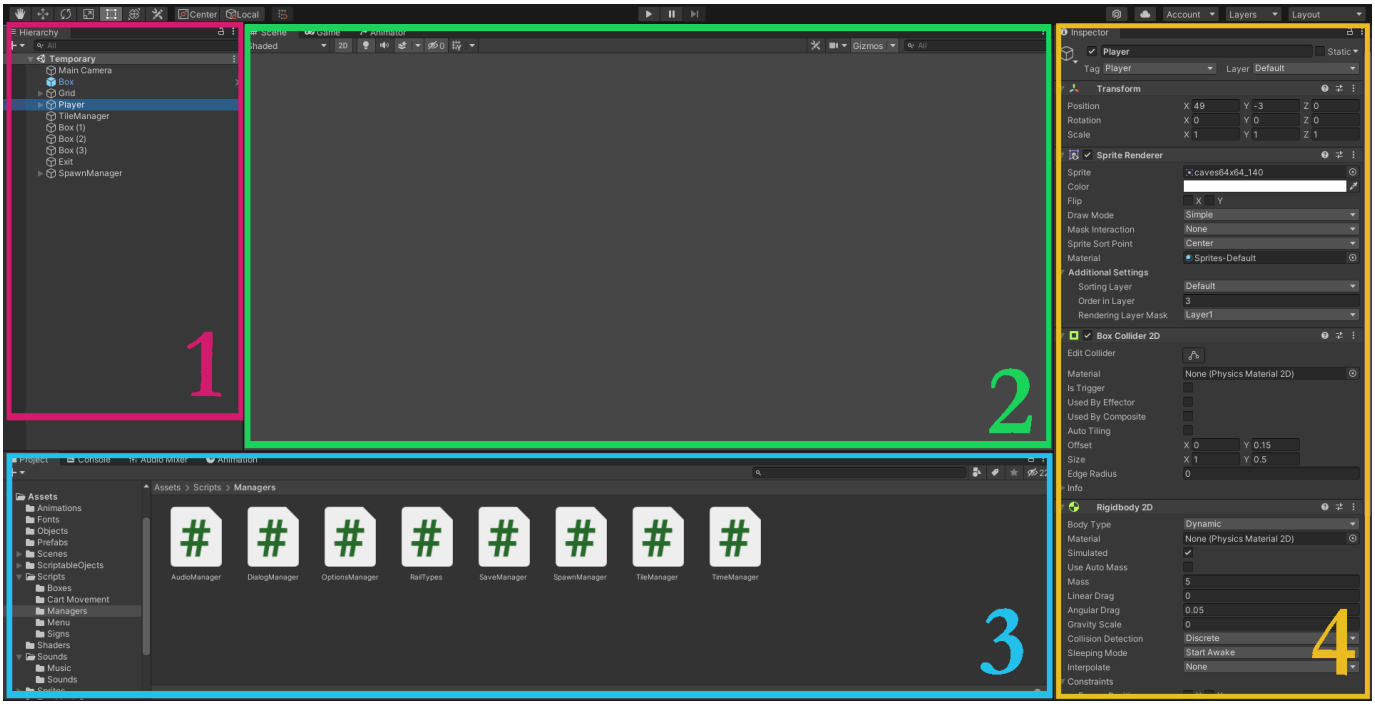
- Toggle - аналогічно до кнопки, але при кожному натисканні Toggle візуально змінює свій стан на протилежний.
- Slider - повзунок, який зберігає певне значення, яке при зміні положення повзунка змінюється відповідно.
- Dropdown - спадне меню, яке дозволяє обрати значення з масиву можливих значень.

Вищезгадані елементи, використовують в собі об’єкт Text для відображення тексту. Для рендерингу тексту Unity має дві систему спрощену та TextMeshPro. TextMeshPro дозволяє рендерити текст власними нестандартними шрифтами та надає більше контролю над ними. Через це, для використання TextMeshPro кожен об’єкт Text потрібно змінювати відповідно.

- *Audio Mixer* - це глобальний об’єкт який задає аудіо систему гри. Об’єкти *Audio Source* - є джерелом аудіо, з можливістю налаштувати їх індивідуальну гучність, повторюваність, стерео тощо. Audio Mixer же надає можливість створювати кілька різних аудіо каналів, на які можуть відправлятися різні аудіо джерела, та які можна індивідуально налаштовувати змінюючи гучність, стерео та додаючи ефекти. Це надає можливість розділяти та групувати аудіо джерела та змінювати характеристики звуку для всієї групи одразу.

3.3 Графічний Інтерфейс

Одна з особливостей Unity - це обширний графічний інтерфейс, що значно спрощує розробку. На Рис 3.4 зображено основні 4 секції стандартного розташування графічного інтерфейсу Unity. В курсовій роботі я розглядаю стандартне розташування, адже саме з ним мав справу. Користувацький інтерфейс рушія гнучкий, тому будь-яку з вкладок керування можна пересунути на іншу позицію.

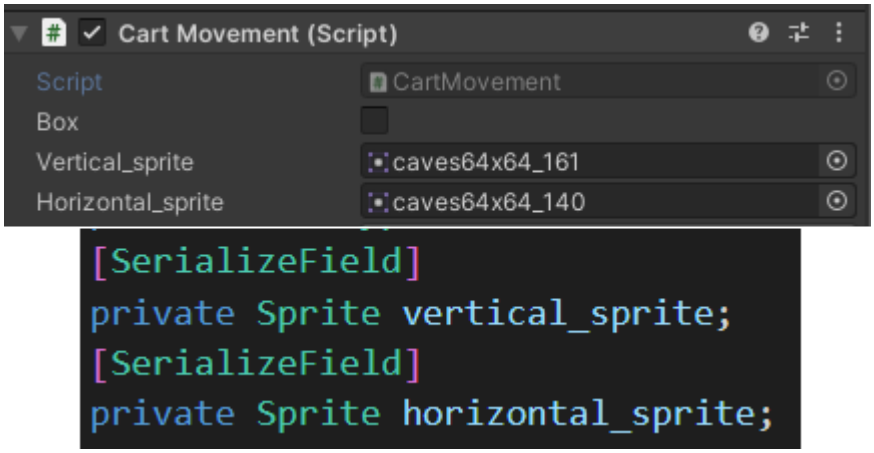


(Рис. 3.4) Графічний Інтерфейс Unity: 1. Панель ієрархії сцени; 2. Панель попереднього перегляду 3. Панель ієрархії файлів 4. Панель властивостей обраного об'єкту

Зліва - панель ієрархії сцени відображає усі об'єкти які знаходяться на поточній сцені. На цій же панелі відбувається додавання об'єктів на сцену. Центральна панель - панель попереднього перегляду дозволяє побачити сцену як у вільному режимі розробника так і в ігровому режимі для попереднього перегляду як сцена буде виглядати та поводити себе в реальному часі. В нижній частині знаходиться панель ієрархії файлів, де відображаються усі локальні файли проекту та панель консолі. Справа - панель властивостей обраного об'єкту, де відображаються усі компоненти обраного об'єкту. За допомогою цього інтерфейсу здійснюється більша частина взаємодії користувача з рушієм.

На панелі властивостей об'єкту, кожен компонент виглядає як візуальний блок із параметрами відповідними до полів виконуваного скрипту компоненту. Додати компонент можна знизу панелі натиснувши на кнопу "Add Component". Деякі компоненти можна повністю вимикати без їх видалення з властивостей об'єкту. При

додаванні скрипту як компоненту, всередині скрипту можна зазначити деякі поля як SerializeField, що зробить їх візуально видимими в користувацькому інтерфейсі рушія, як це зображено на Рис 3.5. Це дозволяє графічно змінювати параметри скриптів всередині Unity без взаємодії з кодом.



(Рис. 3.5) Параметри скрипту всередині графічного інтерфейсу

Панель попереднього перегляду сцени дозволяє візуально змінювати позицію та форму об’єктів за допомогою інструментів як інструмент руху, повороту, масштабування у лівому верхньому куті. На панелі попереднього перегляду гри відображається те, на що сфокусована камера сцени. Якщо камери немає цей перегляд буде пустим. По центру згори є кнопки для запуску та паузи симуляції гри, що дозволяє запустити сцену та побачити її дію в реальному часу.

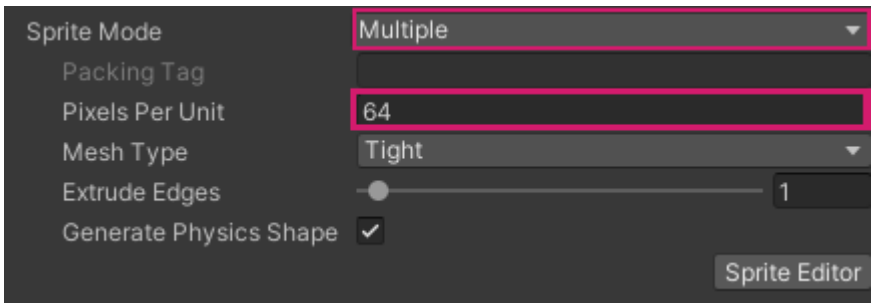
Розділ 4: Розробка додатку

В цьому розділі буде описаний процес розробки гри Raildown на основі фреймворку Unity. Наступний функціонал має бути реалізованим:

- Тайлова система для побудови рівнів
- Рух вагонетки по рейках різних типів
- Важелі, які при перемиканні змінюють стан рейок навколо
- Важелі, які при перемиканні змінюють стан рейок навколо на 3 секунди, після чого повертаються в попереднє положення
- Коробки, які можна рухати мишкою, та які змінюють напрям вагонетки при зіткненні
- Камера, яка буде слідувати за вагонеткою
- Головне меню, налаштування
- Пауза та меню паузи
- Вікно програшу
- Збереження прогресу та стану гри
- Система аудіо
- Таблички та діалогові вікна, які з’являються після натискання
- Система освітлення

4.1 Тайлова система

Враховуючи, що вся концепція гри побудована на тайлах, тайлова система - фундаментальна частина розробки. Першим моїм кроком був вибір тайлсету для розробки. Замість створення власного я використав готовий тайлсет Caves and Rails [19]. Для того щоби використовувати спрайт тайлсету він повинен бути розбитий на квадрати однакового розміру. Для цього у властивостях спрайту я обрав режим спрайту Multiple, який розбиває спрайт на сітку вказаного розміру. Оригінальний спрайт був збільшений до розміру 896x896 пікселів, тому всередині Unity я розбиваю його на квадрати 64x64 пікселів (Рис 4.1).

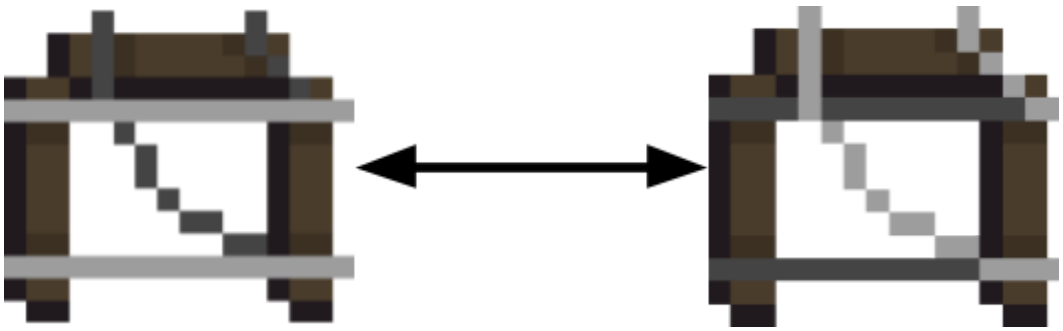


(Рис. 4.1) Розбиття спрайту тайлсету на квадрати 64x64

Наступним кроком було розбити цей спрайт на окремі об’єкти тайлів, які можна використовувати на тайловій сітці. Для цього, за допомогою інструменту Tile Palette я створив новий об’єкт - Ground Tiles, який є тайловою палетою. Перетягнувши в нього спрайт тайлсету, були згенеровані об’єкти типу TileBase з кожного квадрату з тайлсету, розміщені в папці Tiles. На базі цієї тайлової палети, було створено Tilemap, на базі якого відбуватиметься уся наступна розробка. Для тайлмапи я створив 4 шари у вигляді Tilemap об’єктів:

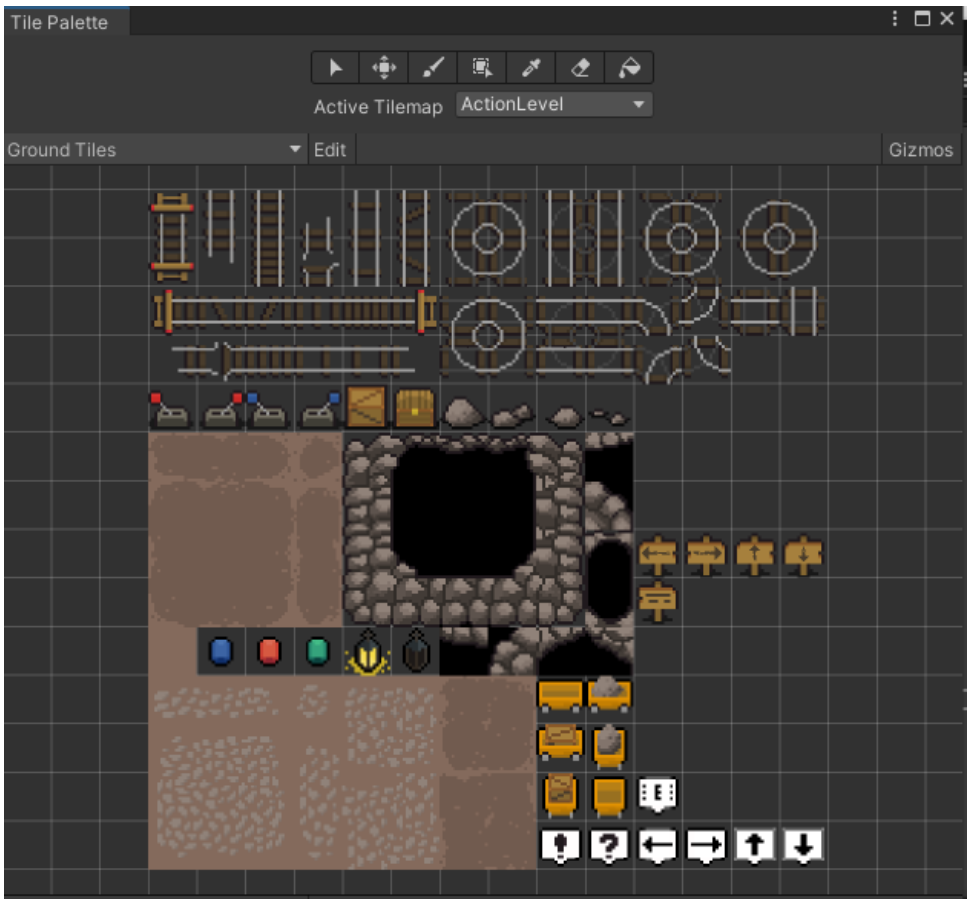
- *Action Level* - шар для об’єктів з якими може інтерактувати гравець.
- *Rail Level* - шар рейок.
- *Terrain Level* - шар декоративних об’єктів на кшталт стін, каміння тощо.
- *Ground Level* - шар фонової землі.

Наступним кроком було додати можливість деяких тайлів змінюватись на інший тайл. Це потрібно для того, щоб поворотні рейки могли змінювати свій стан при перемиканні важеля поруч (Рис 4.2). Для цього було створено скриптований клас *ChangeTile* який наслідує TileBase, та містить одне поле TileBase alt - альтернативний тайл, на який буде замінятися поточний при зміні. На базі цього класу було створено всі скриптовані тайли які містять важелі та рейки, які мають альтернативний стан у відповідних підпапках Levers та Turns папки Special.



(Рис. 4.2) Приклад зміни стану тайлу

Оскільки ці об’єкти наслідують TileBase, вони можуть бути додані до тайлової палети та використовуватись на тайловій сітці. Таким чином була сформована остаточна тайлова палета включаючи усі змінні тайли (Рис 4.3).



(Рис. 4.3) Готова тайлова палета

На цьому етапі тайли не мають жодних властивостей, за якими їх можна було б розрізняти. Для вирішення цієї проблеми був створений скриптований клас TileData

та статичний клас RailTypes. RailTypes - клас, містить в собі два enum списки - directionEnum і typeEnum, для напрямку та типу тайлу. Типи включають:

- normal - звичайні рейки.
- turn - поворотні рейки.
- broken - зламані рейки.
- bounce - пружинні рейки.
- lever - важіль.
- timed_lever - часовий важіль.

Напрямки рейок включають в себе:

- horizontal та vertical - напрями звичайних прямих рейок.
- left, right, up, down - напрями для пружинних рейок, які задають з якого боку відбувається зміна напряму.
- right_up, right_down, left_up, left_down - напрями для поворотних рейок, які задають два пов’язані напрями.

TileData - це клас, який задає категорію об’єктів одного типу, поєднуючи в собі їхні властивості. Клас містить в собі наступні поля:

- TileBase[] tiles - тайли які належать даній категорії.
- float speed - швидкість руху на цьому тайлі, у випадку якщо це тайл рейки.
- RailTypes.directionEnum direction - напрям рейок, якщо це тайл рейки.
- RailTypes.typeEnum type - тип тайлу

На базі цього класу були створені відповідні екземпляри кожного типу тайлів рейок та важелів. Для того, щоб взаємодіяти з цією тайловою системою я створив скриптований клас TileManager. Цей клас містить в собі наступні поля:

- Tilemap railMap - посилання на тайлмапу з рейками.
- Tilemap actionMap - посилання на тайлмапу з важелями.
- List<TileData> TileDatas - список TileData використаних у системі, який задається зовнішньо в інтерфейсі Unity.

- Dictionary<TileBase, TileData> tileDictionary - словник, в якому ключ - це відсилка та тайл, а значення - TileData якій цей тайл належить.

Основна задача TileManager обробляти натискання лівої кнопки мишки гравцем. TileManager наслідує класу MonoBehavior, що прив'язує його до ігрового часу. MonoBehavior об'єкти мають методи Awake() та Update() - метод що викликається при створенні та який викликається кожен тік часу гри відповідно. Їх можна перевизначити для додавання власної поведінки класу. У випадку TileManager у методі Awake() відбувається заповнення tileDictionary з указаних в tileDatas тайлових категорій. Після цього в Update() відбувається перевірка на натискання лівої кнопки мишки за допомогою методу Input.GetMouseButton(0). При натисканні спершу дістаються координати екрану де відбувся клік: Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition). Ці координати конвертуються в координати на тайловій сітці: Vector3Int gridPos = railMap.WorldToCell(mousePos). З координат тайлової сітки можливо дістати тайл на даній позиції з потрібної тайлмапи. Оскільки гравець має взаємодіяти тільки з важелями, тайл береться з actionMap: TileBase clickedTile = actionMap.GetTile(gridPos). Оскільки TileBase не містить усіх потрібних характеристик для дії важеля, відбувається спроба піднести цей тайл до класу ChangeTile. Якщо ця дія успішна, це означає, що отриманий тайл - змінник. Далі перевіряється тип ChangeTile на lever та timed_lever. Якщо тип тайлу lever - викликається метод pullLever. Цей метод приймаючи позицію кліку і тайл важеля змінює стан важеля на альтернативний та викликає метод changeRails, який змінює усі рейки навколо. Цей метод виконує наступне: перевіряється кожен тайл в радіусі однієї клітинки від позиції важеля, та якщо цей тайл є змінною рейкою, то її тайл змінюється на альтернативний.

Також TileManager містить поле bool loss, яке використовується для визначення умови програшу гри. Метод IsLoss приймає координати тайлу, та якщо там немає рейок - loss стає true. Це пізніше буде використане для дій програшу.

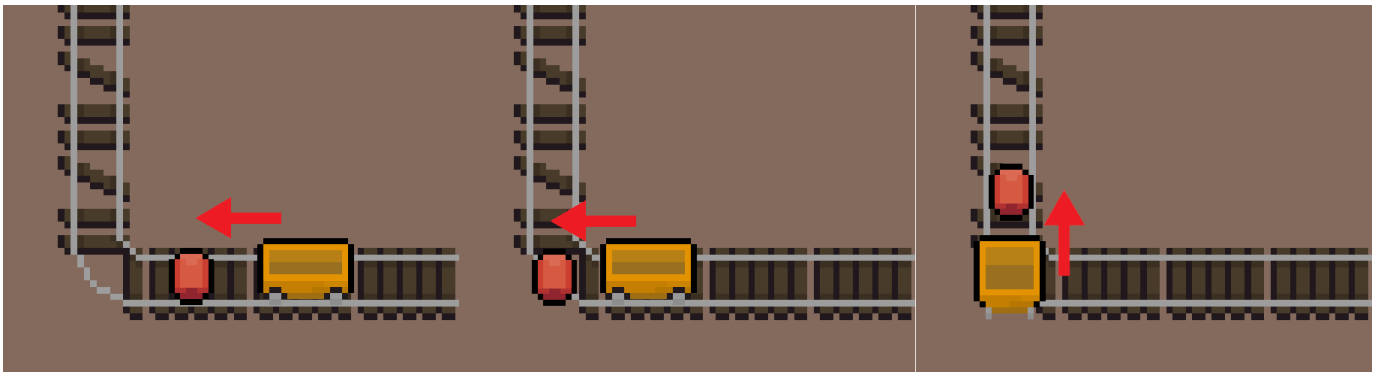
4.2 Рух вагонетки

Unity надає готові рішення для руху гравця, проте Raildown потребує власного рішення, оскільки рух вагонетки не залежить від прямих дій гравця, та засноване на розробленій тайловій системі. Об’єкт вагонетки для руху повинен існувати поза тайловою системою. Це означає, що вагонетка оперує світовими координатами Unity а не координатами тайлової сітки. При русі вагонетки, потрібно перевіряти на яких рейках вона знаходиться для того, щоб задати їй правильну поведінку руху. Були розглянуті кілька різних рішень.

Перше рішення - рух вагонетки потайлово, тобто так щоб вагонетка рухалась за один рух на тайл вперед, перевірялись рейки на яких вона знаходилась та задавала напрям руху наступного кроку. Таке рішення легко імплементувати, проте тоді рух вагонетки буде не плавним.

Друге рішення - плавний рух вагонетки, рухаючись в потрібному напрямку з певною швидкістю, тобто пересуваючись на певну відстань кожного тіку гри. Перевірка рейок має відбуватись лише кожного тайлу, відповідно можна конвертувати світові координати вагонетки до координат тайлової сітки та змінювати напрям вагонетки лише тоді, коли вагонетка заходить у координати наступного тайлу. Таке рішення створює іншу проблему - для правильного руху, вагонетка повинна змінювати свій напрям по центру тайлу, щоб продовжувати рухатись вирівняно по тайлах рейок. Це створює потребу у додаткових розрахунках та ускладнює імплементацію такого рішення.

Я зупинився на третьому рішенні, яке поєднує два попередні рішення. У цьому рішенні об’єкт вагонетки складається з самої вагонетки та невидимого об’єкту Move Point. Вагонетка плавно рухається у напрямку Move Point, а Move Point, в свою чергу, при зіткненні з вагонеткою рухається потайлово далі та при русі перевіряє які рейки знаходяться на новій позиції. Таким чином, невидима рухома точка коректно рухається рейками та вчасно задає зміну напряму, в той час як вагонетка плавно рухається до цієї точки (Рис 4.4)



(Рис. 4.4) Демонстрація концепції руху вагонетки

Для імплементації цього рішення було створено об’єкт Player який має підоб’єкт MovePoint. Об’єкт був збережений як шаблон для використання на багатьох сценах. Player містить скрипт CartMovement, який задає всю поведінку руху вагонетки. CartMovement має такі поля:

- bool turning, bouncing, box - булеани для позначення чи вагонетка зараз повертає, відпружинює чи торкається коробки.
- Sprite horizontal_sprite, vertical_sprite - горизонтальний та вертикальний спрайти вагонетки.
- Transform movePoint - посилання на Move Point
- TileManager tileManager - посилання на Tile Manager
- moveEnum cart_direction - поточний напрямок вагонетки із enum списку moveEnum, який містить напрямки left, right, up, down.

CartMovement так само як і TileManager наслідує MonoBehavior. MonoBehavior крім Awake() та Update() також містить метод Start(), який працює аналогічно до Awake, але виконується лише коли скрипт увімкнений в інтерфейсі Unity. Увесь алгоритм руху імплементований в методі Update(). Алгоритм Update наступний: спершу дістається тайл рейок на позиції Move Point. Якщо рейки звичайні, то вагонетка рухається до точки руху. Коли відстань до точки руху дорівнює 0 Move Point рухається в залежності від поточного напряму вагонетки. Крім цього на цьому

етапі відбувається перевірка на те, чи потрібно змінювати спрайт вагонетки на протилежний, якщо відбувається поворот.

Якщо ж рейки поворотні, то перевіряється чи цей поворот можливий при поточному напрямку вагонетки методом canTurn(). Якщо так - то у змінну last_turn зберігається поворот, і поки під MovePoint не інших рейок, вагонетка рухається до рухомої точки, після чого рухома точка рухається відповідно зміненого напрямку. Якщо ж поворот не можливий - то ініціюється програш.

У випадку пружинних рейок викликається метод bounce, в якому вагонетка рухається до рухомої точки, після чого Move Point змінює напрям на протилежний. Для кожного типу рейок, стани інших дій обнуляються, тобто якщо вагонетка в даний момент знаходиться в стані bouncing, вона не може одночасно бути в стані turning, тож в методі bounce turning стає false. Також в кожній ітерації циклу Update відбувається перевірка на те, чи тайл на якому знаходиться MovePoint коректний, зокрема методом IsLoss зі скрипту TileManager, для того щоб у випадку не можливості руху далі ініціювався програш.

Під час розробки цього алгоритму виникали деякі проблеми, зокрема неправильна обробка поворотів, зміна спрайту до того, як вагонетка доїде до точки повороту, дизфунктивна зміна положення Move Point тощо. Усі ці проблеми були вирішені.

4.3 Коробки

Коробки - це ще один ключовий аспект геймплею Raildown. Відповідно концепції, гравець має можливість рухати коробки перетягуючи їх мишкою. Мною було запропоновано два варіанти імплементації цієї системи.

Перше рішення - пересування коробки лише після відпускання мишки. Таке рішення означає, що гравець натискає лівою кнопкою мишки на коробку, рухає курсор на іншу позицію, і лише після того як гравець відпускає кнопку мишки коробка змінює свою позицію. Поки гравець рухає курсор, він бачить напівпрозору

проекцію коробки, яка не є функціональною, для того щоб візуально відобразити що гравець перетягує коробку на іншу позицію. В такому підході, коробки повинні ставати лише на тайлову сітку, відповідно після відпускання коробки, позиція коробки повинна бути переведена в координати тайлової сітки. Крім цього, коробка має стати лише туди, де є на це місце, а отже при спробі поставити коробку на каміння, стіну або іншу декорацію, коробка має лишитися на своєму попередньому місці. Таке рішення потребує додаткових обрахувань та може викликати незручності геймплею

Друге рішення ж використовує фізичний рушій Unity - коли гравець натискає та перетягує коробку, коробка з повільнішою за швидкість руху курсора рухається зі сталою швидкістю до нього. При цьому відбувається фізична симуляція коробки, тобто при зіткненні з іншими твердими об’єктами коробка не може рухатись далі в цьому ж напрямку. У такому випадку положення коробки постійно змінюється при русі курсора та дає гравцеві більше інтеракцій з навколишнім середовищем.

Для імплементації цього рішення, було створено об’єкт Box, який також був збережений як шаблон. Для фізичної симуляції, коробка містить в собі компоненти Box Collider 2D та Rigidbody 2D. Тип симуляції для цього об’єкту - динамічний, та гравітація дорівнює 0. Скрипт який описує поведінку коробки - DragAndDrop. Цей скрипт використовує стандартний методи MonoBehavior - OnMouseDrag, який виконується, коли після затисканні мишею на об’єкті курсор рухається. Всередині цього методу описано плавну зміну положення коробки в напрямку курсора за допомогою метода Vector3.MoveTowards.

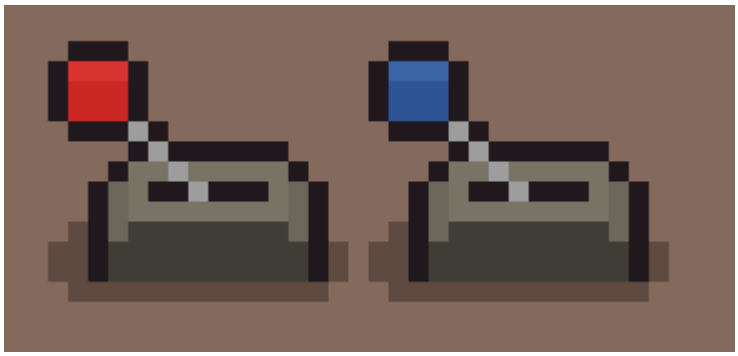
Для опрацювання зіткнення коробки та інших об’єктів я додав компонент Tilemap Collider 2D та Rigidbody 2D до тайлмап декорацій та важелів та додав Box Collider 2D та Rigidbody 2D до об’єкту вагонетки. Для відстеження зіткнення в MonoBehavior існує метод OnCollisionEnter2D. Цей клас був використаний в скрипті CartMovement, та при зіткненні з об’єктом з тегом Box, виконується метод checkBox,

який змінює напрям вагонетки, ініціює рух в новому напрямку та знищує об’єкт коробки.

Недолік такої постійної перевірки на зіткнення полягає у тому, що при такій перевірці вагонетка буде змінювати свій напрям навіть якщо доторкнеться своєю задньою частиною до коробки, що не є коректним. Тому для того, щоб зіткнення опрацьовувалось лише спереду було додано компонент BoxCollider2D до точки руху, яка завжди знаходиться попереду вагонетки. Також оскільки в цьому випадку ніякої фізичної симуляції не має бути, BoxCollider2D має стан isTrigger. Цей трігер тоді використовується у перевірці зіткнення у CartMovement, що означає що зміна напряму можлива лише тоді коли як Move Point так і сама вагонетка торкаються коробки.

4.4 Часові важелі

Як згадувалося до цього, часові важелі - це важелі, які через визначену кількість секунд перемикаються назад. В грі вони візуально відрізняються від звичайних кольором: червоні - звичайні, сині - часові (Рис 4.5).



(Рис. 4.5) Зовнішній вигляд різних видів важелів

Для системи часових важелів потрібно, щоб асинхронно рахувався час від активації важелю. Для цього було створено статичний клас TimeManager. Він приймає в себе об’єкт Action<Vector3Int, ChangeTile> action, два аргументи Vector3Int arg1 та ChangeTile arg2 та кількість секунд очікування float timer. Action - це клас

Unity, який дозволяє абстрактно передати метод. Оскільки цей клас є статичним, але є потреба у використанні внутрішньо-ігрового тіку часу, було створено підклас MonoBehaviorHook, який наслідує MonoBehavior. Таким чином, кожного разу, коли часовий важіль перемикається, статичний TimeManager створює MonoBehavior об’єкт, який всередині використовує метод Update() визначений у TimeManager. Update містить у собі перевірку чи час не дійшов до 0, та відповідно його декремент на час ігрового тіку Time.deltaTime. Коли таймер доходить до 0, об’єкт виконує заданий Action та самознищується.

Всередині TileManager, якщо тайл на який клікає гравець - часовий важіль, то важіль змінює позицію та запускається таймер TimeManager із методом pullLever як Action. Тобто коли час створеного таймеру досягне нуля, то виконається друге перемикання важеля, що поверне його до початкового стану.

4.4 Таблички

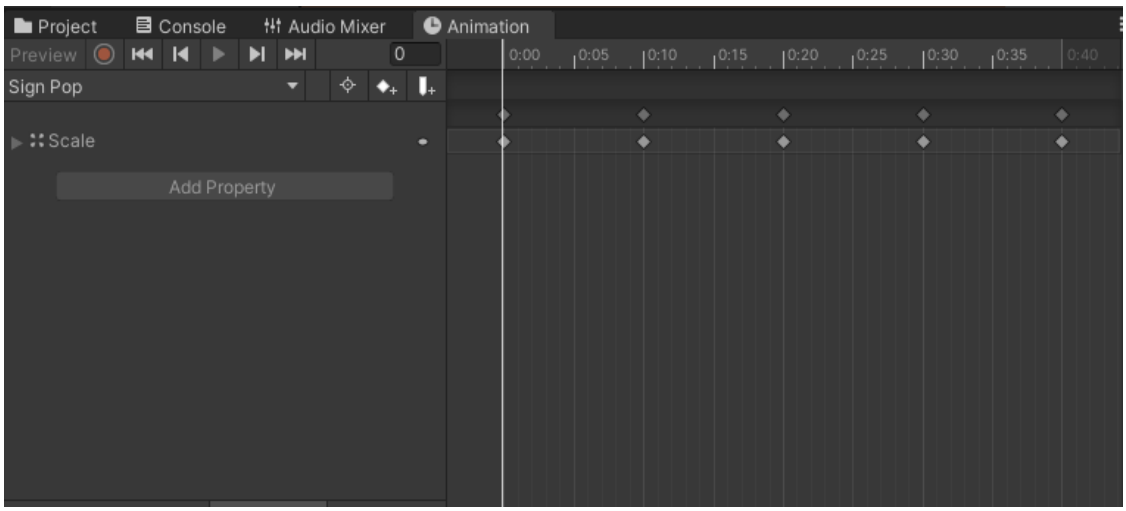
Таблички - це об’єкти, через які гра зможе комунікувати інформацію гравцеві. Це є основним елементом наративу гри. Концепція системи табличок наступна: об’єкт таблички знаходиться поза тайловою сіткою; при натисканні лівою кнопкою мишки на табличку, гра призупиняється та з'являється діалогове вікно із текстом заданим в табличці.

Для імплементації цієї концепції потрібно дві речі: канва користувацького інтерфейсу, яка відображатиме гравцеві текст таблички та система зупинки часу гри.

Для створення канви, я створив об’єкт UI Canvas, всередині якого знаходиться панель з текстом та кнопка Continue. Усією канвою керує скрипт Dialog Manager. За замовчуванням, канва вимкнена, для того щоб вона не відображалась гравцеві коли це не потрібно, а коли натискається табличка, канва вмикається та зупиняється час. Dialog Manager містить чергу Queue<string> text, яка буде відображатись на канві. Вона заповнюється при активації таблички в методі ShowDialogue, беручи дані з індивідуальної таблички. При натисненні на кнопку Continue виконується метод

DisplayNextSentance, де з черги відбирається одне значення допоки черга не буде пуста. Коли черга пуста, кнопка Continue виконує метод EndDialogue, який деактивує канву та відновлює час.

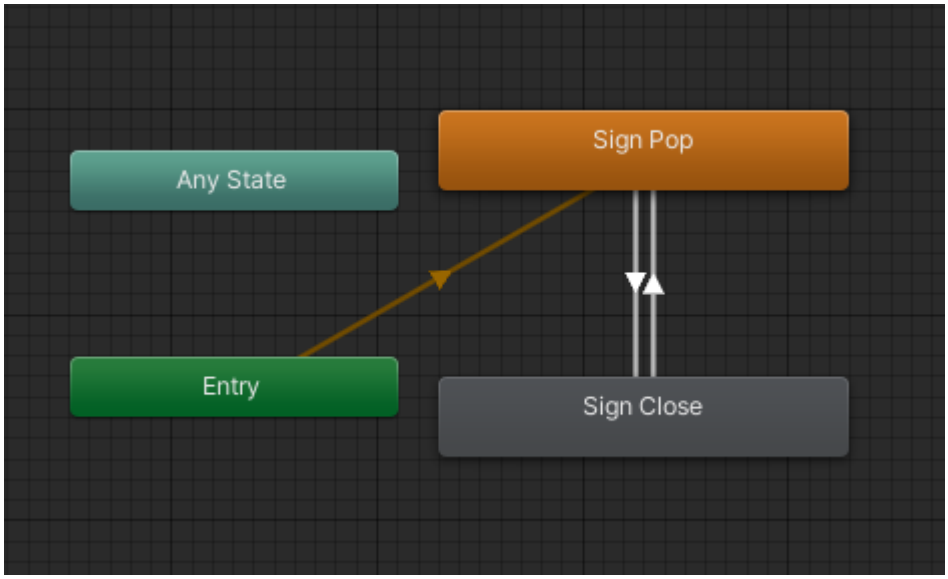
Крім цього до канви було додано візуальну анімацію активації та деактивації. Для цього Dialog Manager також містить компонент Animator, який на вхід приймає об’єкт Animation Controller. Animation Controller задає групу анімацій Animation Clip, якими потім можна керувати. Я створив такий об’єкт під назвою Panel. Animation Clip - це анімація об’єкту, до якого прив’язаний об’єкт анімації. Така анімація створюється в графічному інструменті Unity під назвою Animation. Він з себе представляє часову лінію, на якій можна обрати параметри об’єкту, які потребують анімації та на якій можна задати так звані якірні точки, між якими відбувається анімація (Рис 4.6).



(Рис. 4.6) Зовнішній вигляд різних видів важелів

Таким чином, якщо задано дві точки з різними значеннями певного параметру, то різниця між ними ділиться на кількість кадрів між цими точками, після чого при відіграванні анімації кожного кадру значення параметру змінюється на пораховану різницю за кадр. Відповідно до цього було створено дві анімації, які змінюють розміри панелі - Sign Pop та Sign Close для активації та деактивації таблички.

Потім ці анімації були додані до Animation Controller Panel створеного раніше. За допомогою графічного інструменту Unity Animator можна налаштувати контроллер анімації, задаючи які стани переходять в одне одного та за яких умов. Налаштування відбувається візуально у вигляді сполучення нод різних станів анімацій, як це зображено на Рис 4.7.



(Рис. 4.7) Вигляд інструменту Animator

Наступним кроком створення системи табличок є система паузи гри. Для цього створений статичний скрипт PauseScript, який зберігає стан паузи, та має два методи: Resume() та Pause() для продовження гри та паузи відповідно. Пауза відбувається за допомогою зміни внутрішньо ігрового коефіцієнту плину часу Time.timeScale. Коли timeScale має значення 0 плин часу зупинений, коли ж 1 - плин часу звичайний. Значення більше одиниці призведе до пришвидшення плину часу, а менше одиниці - до його уповільнення.

Останнім аспектом системи табличок є самі таблички. Таблички - об’єкти з Box Collider 2D, через які не можуть проходити коробки. Основний компонент який задає їхню поведінку - Sign Trigger. Цей скрипт містить в собі мною створений скриптований об’єкт Dialogue. Dialogue - це простий тип, який містить масив string, обмежений у кількості елементів до 5. Сам Sign Trigger при натисканні мишкою по

табличці знаходить об’єкт та передає Dialog Manager у методі StartDialogue текст який потрібно відобразити. Це відбувається лише у випадку, якщо гра уже не знаходиться на паузі.

4.5 Камера

Оскільки Raildown є двовимірною, камера кожної сцени налаштована в ортографічному режимі. Для гри критично важливим є рух камери за вагонеткою. Для цього я створив скриптовий компонент Camera Follow. Даний скрипт має наступні поля:

- Transform cart - об’єкт вагонетки, за якою камера слідує.
- float smoothFactor - коефіцієнт згладжування руху камери.
- Vector3 minV, maxV - мінімальні та максимальні координати, в межах яких камера може рухатися.

Camera Follow як MonoBehavior працює наступним чином: кожного тіку гри камера дістає координати вагонетки, після чого формується вектор обмежених рамок руху камери за допомогою методу Mathf.Clamp. Він потрібен для того, щоб камера не могла зайти за межі рівня, коли вагонетка знаходиться коло його меж. Межі рівня задаються вручну для кожної сцени окремо. Далі вираховується позиція, на якій має бути камера в наступний тік, при плавному русі до вагонетки. Це здійснюється за допомогою методу Vector3.Lerp(transform.position, bounds, smoothFactor * Time.fixedDeltaTime). Ці координати присвоюються положенню камери.

4.6 Система спавнерів

На даному етапі головний геймплей імплементований у межах однієї сцени. Цю сцену я зберіг як шаблонну сцену, на базі якої можна створювати інші рівні. Тепер була потрібна система переходу між сценами. Unity має окремий модуль SceneManager, який дозволяє керувати сценами. Для того, щоб працювати з множиною сцен їх потрібно додати в чергу побудови Unity. Коли додаток буде

експортуватися ці сцени будуть вшиті в готову гру, в той час як сцени поза цією чергою будуть проігноровані.

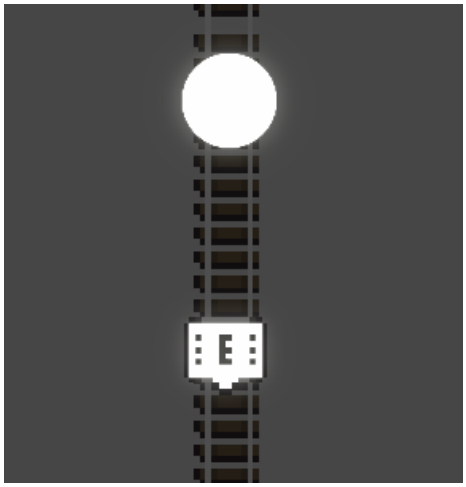
Концептуально система переходу між сценами наступна. Існує два типи об’єктів - спавнер та вихід. Вихід при зіткненні з вагонеткою ініціює процес переходу на наступну задану сцену. Спавнер - об’єкт який задає місце, де має з’являтися вагонетка при завантаженні. Враховуючи, що Raildown розрахована на нелінійну прогресію рівнів кожен рівень може мати більше ніж один вихід та спавнер. Це означатиме, що вихід повинен мати можливість делегувати наступній сцені який спавнер використовувати, та вихідний спавнер з попередньої сцени повинен зберігатися задля можливості повертатися назад.

Для імплементації цієї системи було створено відповідні класи Exit та Spawner на основі скриптів ExitScript та SpawnerScript. Exit містить в собі назву наступної сцени string nextScene, індекс спавнера поточної сцени який має зберігатися для повернення назад int spawnIndex, параметр який відповідає за делегування bool narrateNext та відповідно індекс спавнера на наступній сцені який має використовуватись int nextSpawn. При зіткненні з вагонеткою, відбувається збереження стану поточної сцени та якщо делегується спавнер наступної сцени - наперед перезаписується стан наступної сцени, після чого SceneManager методом LoadScene завантажує наступну сцену. Збереження стану сцени описано у наступному підрозділі курсової роботі.

Оскільки виходи ніяк не пов’язані з одне одним та можуть всі працювати одночасно вони не потребують централізованого класу для керування ними. Спавнери ж виконують активну роль при завантаженні сцени. Через це крім об’єкту Spawner створено також клас SpawnManager, який зберігає в собі список спавнів List<Spawner> spawners, задає індекс int spawnerIndex та запускає лише поточний спавн при запуску сцени. Також при запуску сцени відбувається збереження спавну. Сам спавнер містить в собі посилання на об’єкт вагонетки та напрям в якому вона

має ініціюватися. При запуску вагонетка змінює своє положення на положення спавнеру та починає рух в заданому спавнером напрямку.

Виходи та спавнери для зручності розробки не є невидимими, тому для того щоб вони не були видимими під час гри, вони знаходяться поза межами, в яких діє камера. Тому усі виходи та входи починаються за межами візуально доступного рівня. Спавнери мають вигляд білого кола, а виходи - таблички з літерою E (Рис 4.8).



(Рис. 4.8) Візуальний вигляд спавнерів та виходів

4.7 Система аудіо

Важлива частина будь-якої гри - це аудіо оформлення. Один із варіантів імплементації аудіо в Unity - це створення окремих Audio Source об’єктів які будуть прив’язані до певних об’єктів або дій. Проте, таке рішення дає ускладнення для керування аудіо всередині гри. Тому я вирішив створити централізований статичний об’єкт AudioManager, який буде відповідати за всі звуки гри, та матиме розділення звуків на дві категорії: музика та звукові ефекти. Таким чином можна буде легко налаштовувати одразу усю категорію.

Перш ніж налаштовувати AudioManger потрібно було створити окрему структуру, яка міститиме аудіо дані та їх налаштування для кожного окремого об’єкту. Так я створив скриптований клас AudioAsset, який містить в собі наступні поля:

- string name - назва аудіоданих.
- AudioClip clip - аудіо файл, який має відігруватися.
- float volume, pitch, spatialBlend - параметри звукових даних гучність, висота та відсоток урахування відстані від слухача аудіо (камери) до об’єкту.
- bool loop - параметр, який задає чи аудіо має повторюватись.
- string type - тип аудіо, можливі значення music та fx.
- AudioSource source - об’єкт джерело звуку.

Крім AudioAsset для коректної регуляції потрібно було створити глобальний об’єкт Unity - Audio Mixer. Це інструмент рушія, який дозволяє розділити аудіо джерела на умовні окремі канали, які потім додаються в так званий мастер (головний) канал, звук з якої в результаті чує гравець. Audio Mixer має окрему вкладку в Unity для візуального регулювання. Там я створив Audio Mixer та два канали: Music та FX для музики та звукових ефектів відповідно (Рис 4.9). Я також додав фільтр нижніх частот як ефект на кожен з них. Цей фільтр буде використовуватись пізніше.

(Рис. 4.9) Вигляд створеного Audio Mixer

Для того, щоб доступатися до внутрішніх параметрів Audio Mixer потрібно відкрити потрібні параметри за допомогою інструменту Unity Exposed Parameters. Таким чином я відкрив параметри гучності обох каналів та частоти зрізу кожного з накладених фільтрів як зображено на Рис 4.10.

(Рис. 4.10) Відкриті параметри Audio Mixer

Всередині AudioManager зберігаються наступні поля:

- SoundAsset[] sounds - масив SoundAsset, які можуть бути використані.
- string area_theme - назва SoundAsset, який буде використаний як фонова музика рівня.
- bool playing - стан програвання AudioManager.
- AudioMixer mixer - відсилка до вище описаного Audio Mixer.
- static AudioManager instance - оскільки AudioManager наслідує MonoBehavior але потребує статичного перебування на сцені, всередині зберігається статичний об’єкт який міститиме даний скрипт як компонент.

При створенні AudioManager ініціює усі AudioAsset об’єкти додані до масиву, задаючи себе як джерело аудіо. Також тут відбувається завантаження збережених опцій та відповідне налаштування Audio Mixer, описане пізніше в курсовій роботі. Наостанок починає відігратись музика поточної сцени. Для керування під час ігрового процесу скрипт має наступні методи:

- void Play, Stop - методи які приймають на вхід назву аудіоданих, які потрібно відтворити або зупинити.

- void ChangeMusic - метод зміни музики поточної сцени на музику з заданою на вхід назвою аудіо даних.
- void StopGame, Loss - методи, які вимикають усі відповідні звуки у випадку зупинки гри (переходу до меню) та у випадку поразки.

До масиву sounds були додані усі аудіодані включаючи звуки та музику потрібні для гри. Використана музика авторська, написана мною.

Після створення AudioManager в інші попередньо створені скрипти CartMovement та TileManager було додано відповідний код, який керує AudioManager та запускає музику або звуки. Для цього було додано посилання на AudioManager як поле, яке при ініціалізації об’єктів знаходиться на сцені за допомогою GameObject.FindGameObjectsWithTag("AudioManager"). Потім були додані відповідні методи відтворення звуків, зокрема відтворення звуків важелів при їх перемиканні, відтворення звуку руху вагонетки, коли вона рухається, звук розбиття коробки та звук відпружинення від пружинних рейок. Також для задавання музики кожної сцени я створив скрипт MusicLoader який було додано до камери. Даний скрипт завантажує AudioManager та задає його area_theme відповідно до вказаного значення в полі.

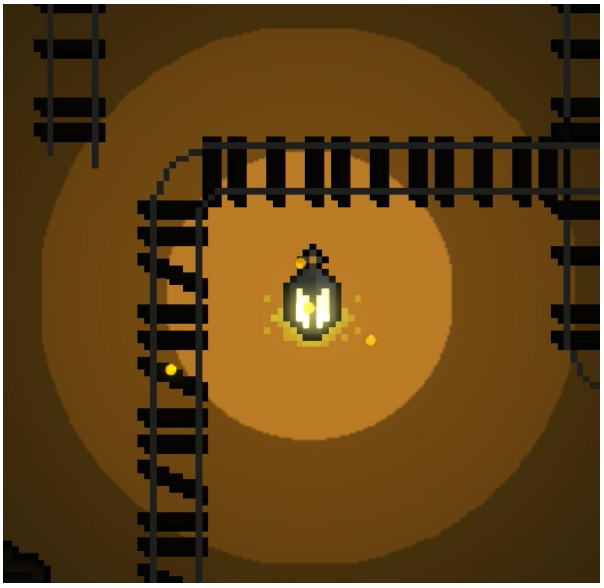
4.8 Система освітлення

Останньою частиною візуального оформлення геймплею є система освітлення. Unity має потужні інструменти для створення освітлення. Для того, аби додати систему освітлення, до проекту потрібно додати додатковий модуль Universal Render Pipeline, який заміняє стандартну систему рендерингу об’єктів Unity. При переході на цей рушій рендерингу потрібно задати кожному об’єкту підходящий матеріал. Матеріали задають те, як рендеринговий рушій має відображати об’єкт. Всередині Universal Render Pipeline є два стандартні типи матеріалів - Sprite-Lit та Sprite-Unlit, для об’єктів на які має впливати освітлення середовища, та для об’єктів які є константно освітленими. Усім об’єктам крім вагонетки було присвоєно Sprite-Lit метеріал.

Освітлення створюється за допомогою об’єктів джерел освітлення Light 2D. Джерело світла має наступні типи:

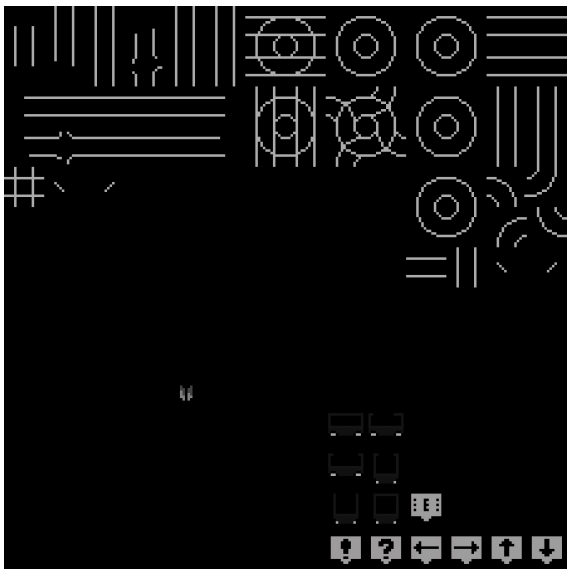
- *Параметричне* - світло форми полігону із заданою кількістю граней
- *Вільної форми* - світло довільної налаштованої форми
- *Спрайтове* - світло форми заданого спрайту
- *Точкове* - радіальне світло з однієї точки
- *Глобальне* - ілюмінація всієї сцени

Кожна сцена містить об’єкт глобальної ілюмінації, зі своєю заданою інтенсивністю. До об’єкту вагонетки також було додано точкове джерело світла для того, щоб середовище навколо вагонетки завжди залишалося освітленим. Деякі сцени містять окремо створені декоративні об’єкти освітлення, такі як світло згори або ліхтарі. Ці об’єкти крім освітлення містять також систему частинок, які створюються та рухаються траєкторією текстури шуму. (Рис 4.11)



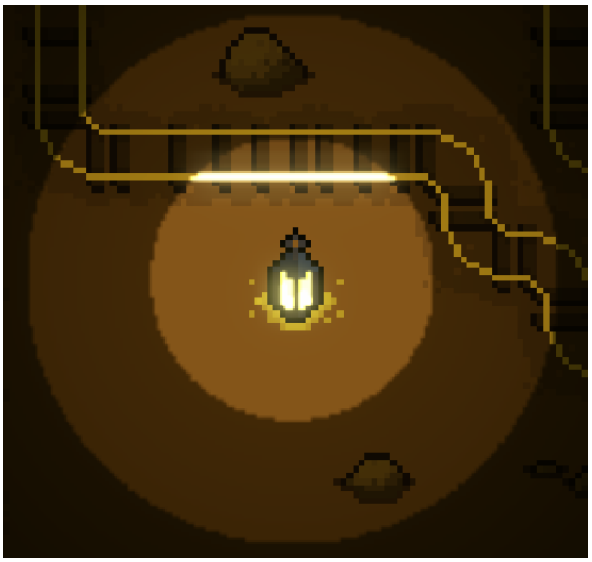
(Рис. 4.11) Ліхтар з освітленням та частинками навколо

Інший елемент освітлення, який я додав, це блиск. Для цього я створив власний матеріал Sprite Glow. Даний матеріал призначений для спрайту тайлсету гри. Спершу я сформував маску блиску тайлмапи, де білим позначено які ділянки будуть блистіти (Рис 4.12).



(Рис. 4.12) Приклад симуляції тіней в реальному часі

Потім за допомогою графічного інструменту Unity для роботи з шейдерами, я створив матеріал, який перемножує дані з маски блиску на заданий колір, після чого ці дані додаються до даних оригінального спрайту та подаються на вихід (Рис 4.13).



(Рис. 4.13) Приклад блиску

Крім світла для довершеності системи освітлення потрібні тіні. Для того, щоб об’єкт міг давати тінь, потрібно додати до нього стандартний компонент Unity під назвою Shadow Caster 2D. Цей компонент симулює падіння тіней в реальному часі,

що надає об’єктам об’єму. Я присвоїв цей компонент коробкам, важелям, вагонетці та табличкам. Приклад падіння тіней зображено на Рис 4.14.

(Рис. 4.14) Приклад симуляції тіней в реальному часі

4.9 Меню

Крім геймплею лишається створити меню, за допомогою якого гравець зможе навігувати грою. Розроблені були два меню: головне меню на початковому екрані, меню налаштувань меню паузи, яке з’являється тоді коли гравець ставить гру на паузу та меню програшу.

Головне меню було створено в окремій сцені Menu Scene. Меню розташоване на UI Canvas, та містить панель Main Menu з такими кнопками (Рис 4.15):

- *New Game* - запускає нову гру, переходячи на перший рівень
- *Continue* - завантажує збережений рівень
- *Options* - переходить до підменю опцій
- *Quit* - остаточно виходить з гри

Меню керується скриптом MenuScript, методи якого використовуються на кнопках для виконання своєї функції. Увесь текст, присутній в меню та інших частинах гри імплементований за рахунок текстових об’єктів TextMeshPro з

використанням шрифту LunchDS. Для того, щоб використовувати шрифт в середині TextMeshPro було створено окремий об’єкт TMP_Font Asset із заданим відповідним файлом шрифту.

(Рис. 4.15) Головне меню

Кнопка Options вимикає головне меню, та вмикає меню опцій. Меню опцій містить налаштування розширення екрану у вигляді випадаючого списку, можливість увімкнути або вимкнути повноекранний режим, налаштування гучності музики та звукових ефектів та кнопку повернення до головного меню. Меню опцій керується скриптом OptionsScript. При запуску, в методі Awake ініціюються усі елементи меню значеннями, які завантажуються зі збережених. Також відбувається прив’язка слайдерів гучності до реальної гучності зазначеної в Audio Mixer. До випадаючого списку розширень додаються усі можливі значення, які дозволяються поточним монітором за допомогою Screen.resolutions. Зміна повноекранного режиму відбувається через параметр Screen.fullscreen, а розширення через Screen.resolution. Зміна гучності каналів AudioMixer відбувається через раніше відкриті параметри та метод mixer.SetFloat(“назва параметру”, [значення]). Усі параметри зберігаються

лише при переході назад до головного меню. Для графічного урізноманітнення інтерфейс було доповнено спрайтами з ігрового тайлсету та слайдерам гучності надані власні кольори (Рис 4.16).

(Рис. 4.16) Меню опцій

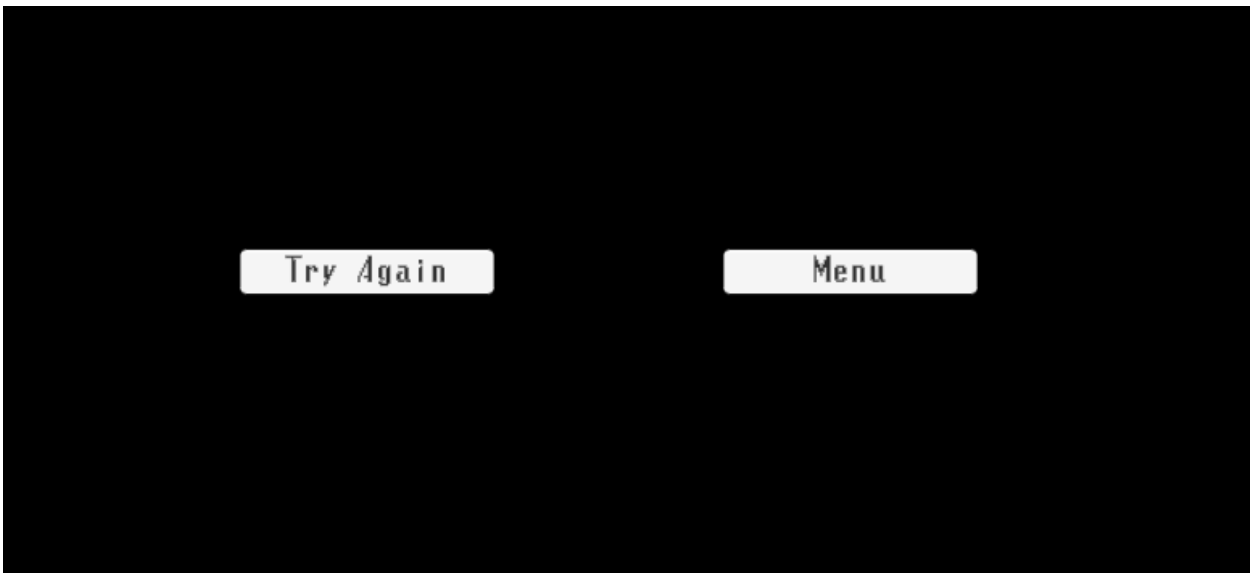
Меню паузи повинне з’являтися під час ігрового процесу при певних діях гравця. Для цього було створено ще один UI Canvas, на якому знаходиться Pause Panel. Ця панель працює згідно скрипту Pause Menu Script. Існує конвенція використання клавіші клавіатури ESC як кнопки для ігрової паузи, тому в скрипті в циклі Update перевіряється натискання цієї кнопки. Коли гравець натискає ESC панель активується. При активації меню також зупиняється час за допомогою PauseScript описаного раніше, а також на каналі музики Audio Mixer збільшується частота зрізу фільтру нижніх частот. Таким чином, коли вмикається меню паузи, гра зупиняється, а музика стає приглушеною. Меню паузи містить наступні кнопки, функціонал яких прописаний всередині скрипта (Рис 4.17):

- *Resume* - продовжує гру, продовжує плин часу та повертає звук до норми.
- *Restart* - завантажує поточну сцену заново.
- *Options* - відкриває меню опцій, аналогічне до того, яке було описано вище.

- *Menu* - завантажує сцену головного меню.
- *Quit* - остаточно виходить з гри.

(Рис. 4.17) Меню паузи

Останнім меню є меню поразки. Це меню має виникати після того, як вагонетка гравця розбивається. Це меню знаходиться на тому ж UI Canvas що і меню паузи. Меню має непрозору чорну панель як фон, та дві кнопки: Try Again та Menu для перезапуску рівня та повернення до головного меню відповідно (Рис 4.18). Меню керується скриптом LossScript, який також використовується всередині CartMovement в методі Loss де зупиняється плин часу, вимикаються усі звуки Audio Manager та активізується меню поразки.



(Рис. 4.18) Меню поразки

Усі кнопки меню також використовують AudioManager для того щоб відтворювати звук натискання кнопок.

4.10 Збереження прогресу та налаштувань

Завершальним функціональним сегментом розробки Raildown є збереження прогресу та налаштувань. Концепція збереження наступна: певний статичний об’єкт містить в собі як поля дані; при запуску гри, ці дані зчитуються та записуються в цей об’єкт; в процесі гри, об’єкт змінюється, та по завершенню гри або після певних внутрішньо ігрових дій дані з об’єкту серіалізуються та записуються у файл.

Для зберігання прогресу було створено скриптовий клас GameData. Він містить наступні поля:

- int current_level - індекс останнього збереженого рівня. Саме цей індекс буде використовуватись при завантаженні за кнопкою Continue в головному меню.
- int N_levels - кількість рівнів.
- Dictionary<int,int> level_spawn - словник, де ключ - це індекс рівня, а значення - індекс збереженого спавнера для цього рівня.

GameData також містить відповідні методи для роботи з цими полями. Далі я створив статичний клас який використовується для опрацювання GameData під назвою SaveManager. Основні три методи цього класу включають: Save, Load та New. В методі Save поточна GameData оновлюється поточними значеннями та за допомогою об’єкту типу BinaryFormatter та файлового потоку серіалізується та записується у файл savedata.rld. Rld - це неіснуюче текстове розширення лише для файлів Raildown. Метод Load працює навпаки, спершу файловим потоком зчитується вміст savedata.rld та потім десеріалізується для того щоб перезаписати поточний GameData. Метод New лише ж ініціює нову GameData. Метод Load використовується всередині MenuScript для завантаження останнього рівня на кнопці Continue, а New на кнопці New Game для завантаження першого рівня і відповідно прибирання будь-якої попередньо збереженої інформації. Метод Save використовується при переході з одного рівня на інший всередині ExitScript.

Збереження налаштувань гри відбувається за аналогічним принципом. Як об’єкт даних було створено клас OptionsData, який містить усі ті поля, що присутні в меню налаштувань: width та height - ширина та висота розширення екрану, стан повноекранного режиму, гучність музики та звуків. Аналогічно до SaveManager для OptionsData було створено статичний скриптовий клас OptionsManager, який має методи Save, Load та Reset які працюють так само як і Save, Load та New SaveManager. Ці методи потім використовуються в OptionsScript для початкового завантаження параметрів з файлу та їх зберігання після виходу з меню.

4.11 Оптимізація інтерфейсу

На даному етапі, ігрове меню лишається частиною ігрової сцени, через що виникає потреба повторно провантажувати одні й ті самі об’єкти. Також це створює проблему того, що якщо музика двох різних рівнів не відрізняється, то вона починатиеться заново при переході, через те що завантажується новий Audio Manager. Для того, щоб усунути цю проблему потрібно створити окрему сцену

користувацького інтерфейсу, яка буде лишатися завантаженою в пам’ять, в той час коли ігрові сцени будуть змінюватись.

Я створив сцену під назвою UI Scene, яка містить канву з меню паузи, меню поразки та Audio Manager. Ця сцена не містить в собі камеру, тому що вона буде адитивно завантажуватись до іншої сцени. Unity дозволяє мати декілька сцен завантаженими одночасно, що дозволяє виносити деякі елементи в окремі сцени без потреби у дублікуванні їх на кожній сцені. Для завантаження UI Scene до поточної я створив скрипт UILoader, який при запуску з використанням Scene Manager асинхронно завантажує UI Scene в адитивному режимі (Рис 4.19). Цей скрипт я додав як компонент до кожної камери ігрової сцени.

```
void Start()
{
  if (SceneManager.GetSceneByName("UI Scene").isLoaded == false)
  {
    AsyncOperation operation = SceneManager.LoadSceneAsync("UI Scene", LoadSceneMode.Additive);
    operation.allowSceneActivation = true;
    isLoaded = true;
  }
}
```

(Рис. 4.18) Асинхронне завантаження UI Scene

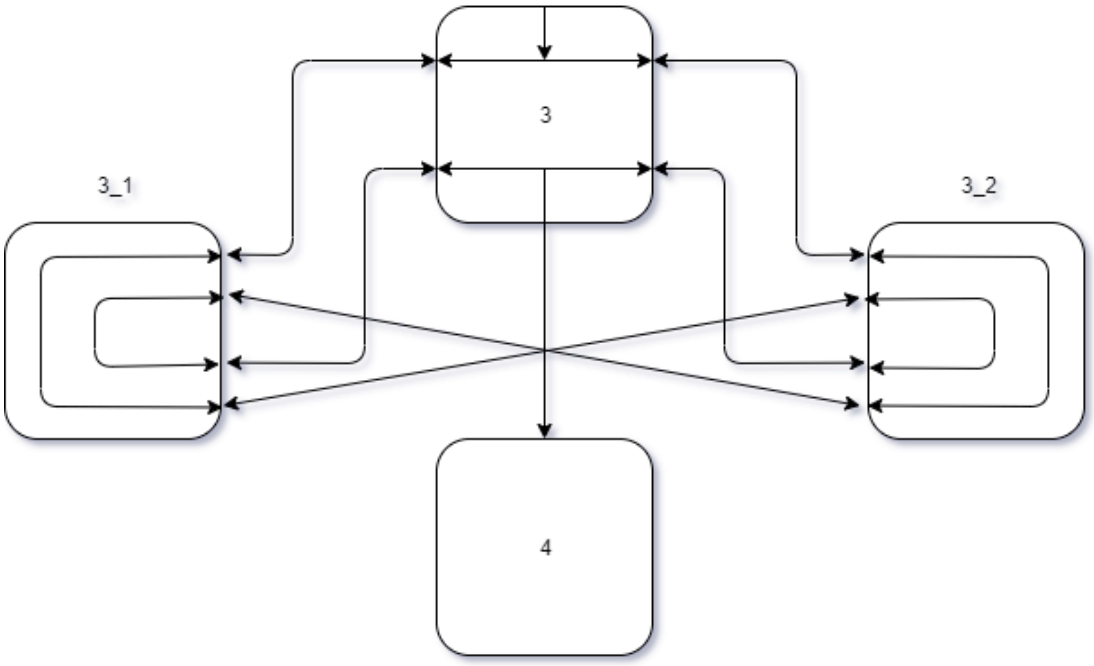
Крім цього для усіх елементів даної сцени потрібно додати властивість незнищенності при вантаженні наступної сцени. За замовчуванням, при завантаженні іншої сцени, усі поточні сцени та об’єкти відвантажуються. Проте для GameObject можливо виконати метод DontDestroyOnLoad, який відміняє цю властивість, та залишає об’єкт завантаженим. Саме це я і додав до AudioManager, PauseScript, PauseMenu.

Попри оптимізацію завантаження такий спосіб додавання об’єктів створює проблеми з доступом до об’єкту однієї сцени з іншої при асинхронному завантаженні. Наприклад, CartMovement не зможе зразу напряму доступитися до Audio Manager який тепер знаходиться в іншій сцені, яка адитивно додається до поточної. Через це, при першій спробі доступитися до нього, CartMovement отримає

у відповідь об’єкт null. Для того щоб це не відбувалося в CartMovement, TileManager та інших класах, які потребують взаємодії з елементами поза сценою, було створено асинхронні методи для пошуку цих елементів. Здійснюється це за допомогою методу StartCoroutine, який приймає метод типу IEnumerator та запускає цей метод асинхронно. Таким чином, потрібний елемент буде шукатися до того моменту, поки він не буде знайдений, і тільки після цього він зможе експлуатуватися.

4.12 Сцеии

На основі розробленого функціоналу було створено 9 сцен, дві з яких - це головне меню та UI Scene. Всі інші 7 сцен - це ігрові рівні. Рівні розташовані в порядку зростання складності. Перші два слугують як навчання для гравця. Для підтвердження концепції нелінійного прогресування рівня з розробленими системами було створено рівні 3, 3_1 та 3_2. Рівень 3 має два різні виходи в різних напрямках, один переходить на рівень 3_1 інший на 3_2. Підтвердженням роботи нелінійності буде можливість повернутись до рівня 3 іншим шляхом від того, який був останнім. Таким чином я створив приклад використання цієї системи для створення нелінійного циклу, де різними шляхами можливо прийти до того самого місця. На Рис 4.19 схематично зображено взаємозв’язок між цими сценами. Рівень 4 - фінальний, та містить додатковий UI Canvas з подякою та моїм підписом.



(Рис. 4.19) Схематичне зображення взаємопов'язаності рівнів 3, 3_1 та 3_2

4.13 Можливі покращення

В результати розробки було реалізовано увесь запланований функціонал додатку. Проте, гра має великий потенціал для поліпшення. Нижче я навів кілька значних потенційних покращень:

- Більше рівнів - нелінійна прогресія розкривається краще при великій кількості рівнів. Також велика кількість рівнів може урізноманітнити геймплей. Крім більшої кількості рівнів значним покращенням було б введення ігрових інструментів або об'єктів, які впливали б на можливі рішення. Це додало б грі більше елементів метроідванії.
- Редактор рівнів та зберігання рівнів у вигляді XML файлів - наразі рівні зберігаються як наперед визначені об'єкти кожної сцени. Винесення рівнів у файли, які завантажуються на одну сцену значно оптимізувало б процес гри. Крім цього з такою системою було б можливо створити редактор рівнів, за допомогою якого гравці могли б створювати та ширити власні рівні.

- Більше геймплейних елементів - більша кількість ігрових механік урізноманітнила б геймплей Raildown. Потенційно можна було б додати більше елементів залежних від часу, на основі Time Manger, або об’єктів які можна пересувати, та які будуть виконувати інакшу функцію від коробок.

Також я надав гру для тестування трьом іншим людям. Як результат їхнього тестування, вони визначили наступні можливі покращення у роботі уже наявного функціоналу:

- Гра не є інтуітивно зрозумілою, тому потенційним покращенням може бути додавання елементів які дають підказки гравцеві, демонструють як взаємодіяти з грою та вказують що робити в тих чи інших ситуаціях.
- Коробки можуть бути досить повільними, через що гравці були незадоволені. Збільшення швидкості, з якою коробки рухаються до курсору, було б значним покращенням цієї механіки.

Крім цього, тестування продемонструвало, що гравці не мали жодних інших проблем з налаштуванням та взаємодією з грою. Усі гравці змогли успішно пройти гру від початку до кінця. Швидкодія гри є стабільною, та всі механіки працюють як було задумано.

Висновки

Впродовж курсової роботи, я дослідив методики та принципи розробки відеоігор. Я детальніше дізнався про процес створення та концепції якими оперують двовимірні відеоігри. Також я дослідив класифікації відео ігор за жанрами, візуальним виглядом та наративом.

Технічно, я познайомився з ігровими рушіями та отримав досвід та навички роботи з фреймворком розробки ігор Unity. Я на практиці переконався у потужності рушія, та можу зазначити, що Unity містить обширний набір інструментів для розробки ігор такі як фізичний рушій, система контролю сцен, система освітлення, роботи з аудіо та багато інших.

Також я отримав досвід планування власної відеогри на основі всіх проведених досліджень та створив концепцію гри Raildown. В результаті курсової роботи, я розробив гру на рушію Unity, використовуючи увесь потенціал рушія. Так я спромігся виконати увесь запланований функціонал та провести тестування гри. Гра містить повністю функціонуючий геймплей, оригінальне візуальне та аудіо оформлення та різноманітні меню та налаштування для навігації.

Список літератури

1. Ігровий Рушій[Електронний ресурс], 29.04.2022 https://en.wikipedia.org/wiki/Game_engine
2. Definition: Sprite[Електронний ресурс], 2015 https://www.educative.io/edpresso/definition-sprite
3. Tileset Definition[Електронний ресурс], 10.08.2019 https://solarus-games.org/doc/latest/quest_tileset_data_file.html
4. New Amazing Tilesets[Електронний ресурс], 24.05.2011 https://vxresource.wordpress.com/2011/05/24/new-amazing-tilesets/
5. Tile-based Video Game [Електронний ресурс], 11.12.2021 https://en.wikipedia.org/wiki/Tile-based_video_game
6. Stardew Valley [Електронний ресурс], 2022 https://www.stardewvalley.net/
7. Gameplay[Електронний ресурс], 21.12.2016 https://en.wikipedia.org/wiki/Equalization_(audio)
8. How Camera Placement Affects Gameplay in Video Games [Електронний ресурс], Markos Naftis, Georgios Tsatiris, Kostas Karpouzis, 08.09.2021 https://arxiv.org/pdf/2109.03750.pdf
9. Electronic Adventure Games[Електронний ресурс], William L. Hosch, 01.07.2018 https://www.britannica.com/topic/electronic-adventure-game
10. 4 Types of Narrative In Video Games[Електронний ресурс, Elle McFadzean, 22.10.2019 https://ellemcfadzean.com/4-types-of-narrative-in-games/
11. Game Design - Theory and Practice: The Elements of Gameplay [Електронний ресурс], Richard Rouse III, 27.07.2001 https://www.gamedeveloper.com/design/game-design---theory-and-practice-the-elements-of-gameplay

12. Metroidvania [Електронний ресурс], 19.04.2022
https://en.wikipedia.org/wiki/Metroidvania
13. Unity(game engine)[Електронний ресурс], 07.05.2022
https://en.wikipedia.org/wiki/Unity_(game_engine)
14. Scenes[Електронний ресурс], 07.05.2022
https://docs.unity3d.com/Manual/CreatingScenes.html
15. Cameras Overview[Електронний ресурс], 2020
https://docs.unity3d.com/2020.2/Documentation/Manual/CamerasOverview.html
16. Prefabs[Електронний ресурс], 07.31.2018
https://docs.unity3d.com/Manual/Prefabs.html
17. Rigidbody 2D[Електронний ресурс], 2021
https://docs.unity3d.com/Manual/class-Rigidbody2D.html
18. Canvas [Електронний ресурс], 2018
https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html
19. Caves and Rails Tileset[Електронний ресурс], 08.04.2020
https://heyitswidmo.itch.io/caves-rails-tileset