МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: **Розробка ігрового додатку на платформі UNITY**

Виконав студент групи **КН-20**
спеціальності 122 Комп'ютерні науки
Плетос Володимир Григорович

Керівник **ст. викладач**
Штефан Наталія Зінов'ївна

Консультант **техн. наук, професор**
Казакова Надія Феліксівна

Рецензент **к.т.н., доцент**
Сергієнко Андрій Володимирович

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

ООП – Об’єктно-орієнтоване програмування

Ассети – ігровий ресурс, цифровий об'єкт, який переважно складається з однотипних даних

Спрайти – двомірне зображення, що застосовується в комп'ютерній графіці

Платформери – жанр комп'ютерних ігор, у яких основу ігрового процесу становлять стрибки по платформах, лазіння сходами, збирання предметів, необхідні перемоги над ворогами чи завершення рівня

Cryengine –потужний движок 3D-ігор

C# – мова програмування

GameMaker Studio 2 – ігровий рущій

Unity – кроссплатформне середовище розробки комп'ютерних ігор

Unreal Engine – ігровий рущій

ВСТУП

Інді-гра – це не жанр, клас чи навіть тип гри. Її створюють незалежною командою або навіть одним розробником. Це гра без великої фінансової підтримки, але зі свободою та сміливістю поставити ідею гри вище прибутку. Зазвичай інді-студії складаються лише з кількох людей. Найчастіше їх кількість не перевищує 10 експертів. Щоб досягти успіху, інді-студії мають шукати інноваційні ідеї та нові способи створення ігор.

Деякі з найбільших успіхів на ринку 2D-відеоігор досягли незалежні розробники ігор. Через нижчий рівень складності розробники ігор витрачають менше часу та грошей на створення 2D-ігор. Розробники можуть більше зосередитися на експериментуванні з різними сюжетними лініями та художніми стилями, а стежити за оновленнями легше. Для ефективнішого виробничого процесу незалежні розробники залучають 2D концепт-арт персонажів та інші виробничі процеси.

У прихільників 2D ігор є кілька обґрунтованих причин, чому вони люблять грати у відеоігри, створені у двох вимірах. Наприклад, вони не перевантажуються контролем персонажа, тому що елементи керування прості. Двовимірні ігри схожі на «торкнись і грай» із простими інструкціями гри. Крім того, ігровий інтерфейс досить простий, що дозволяє гравцям зрозуміти, як грати, забезпечуючи безпроблемну роботу користувача. Будучи менш складними, розмір 2D ігор менший, що полегшує їх завантаження та ідеально підходить для мобільних пристроїв.

Метою дипломного проекту – розробка гри мовою C# , використовуючи сучасні засоби розробки. Це дозволить закріпити навички в ООП та створити власний проект.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Огляд предметної області

Двомірні ігри можуть відрізнятися художнім стилем, жанром, перспективним виглядом і платформою. Жанр відеоігор – це категорія ігор, яка має схожі ігрові характеристики. Жанри відеоігор майже однакові як для 2D, так і для 3D ігор. Це рольові ігри, платформери, бойовики, файтинги, симулятори, головоломки тощо. Але ключовою відмінністю між двовимірними іграми є вибір перспективи. Ось найпоширеніші типи 2D-ігор на основі їх перспективи:

1. Вид збоку. Це один із найпопулярніших видів 2D-ігор. Вигляд збоку є класичним вибором для платформерів, коли персонажі рухаються переважно зліва направо, вгору та вниз.
2. Вигляд зверху вниз. Гра виглядає так, ніби камера знаходиться над головою. Ігрове поле або видно з висоти пташиного польоту, або має трохи нахилену камеру.
3. Ізометрична перспектива. Ігрове поле показано під певним кутом камери. Це ідеальний варіант, щоб створити ілюзію тривимірного простору, показуючи три сторони об’єкта.
4. Один екран. Кожен рівень розташований у новій кімнаті, яка заповнює весь простір екрана. Коли рівень пройдено, ви переходите до наступної кімнати [1].

1.2 Вибір засобів розробки

Ігровий рущій – це основа, на якій створюється гра: пишуться правила, вибудовується інтерфейс і опрацьовується фізика гри, звук, анімація та багато іншого. Двигун збирає всі компоненти гри воєдино, як паззл, щоб у результаті різних елементів вийшов один працюючий продукт.

Деякі ігрові двигуни вимагають знань програмування, а деякі – ні. Другі працюють за принципом візуального програмування, і сам процес розробки гри в робочому середовищі таких двигунів нагадує складання конструктора, де блоки - це частини коду [2].

Двигун відеоігор дозволяє додавати:

- фізику;
- візуалізацію;
- створення сценаріїв;
- виявлення зіткнень;
- штучний інтелекті багато іншого без необхідності їх програмування.

Game Engines забезпечують розробників ігор інструментами для скорочення часу розробки. Ігрові рущії – це багаторазові компоненти, які розробники використовують для створення основи гри. Це дає їм більше часу, щоб зосередитися на унікальних елементах, таких як моделі персонажів, текстури, взаємодія об’єктів тощо [3].

Якби кожен створював свої ігри з нуля без допомоги чудових механізмів розробки ігор, ігри займали б більше часу, і їх було б складніше створювати. З огляду на це, все ще є багато великих компаній і навіть незалежних команд, які створюють власний механізм.

Для вибору засобу розробки було оглянуто декілька топових рущіїв для створення ігор.

1.2.1 Unreal Engine

Це одним із найпопулярніших і широко використовуваних ігрових движків є Unreal Engine, який належить Epic Games. Оригінальна версія була випущена ще в 1998 році, і через, 40 років вона продовжує використовуватися в деяких з найбільших ігор.

Движок Unreal (рис. 1) ефективніше справляється з кількома складними завданнями завдяки широкому використанню в сфері розробки ігор. Також спільнота постійно вдосконалює движок Unreal.

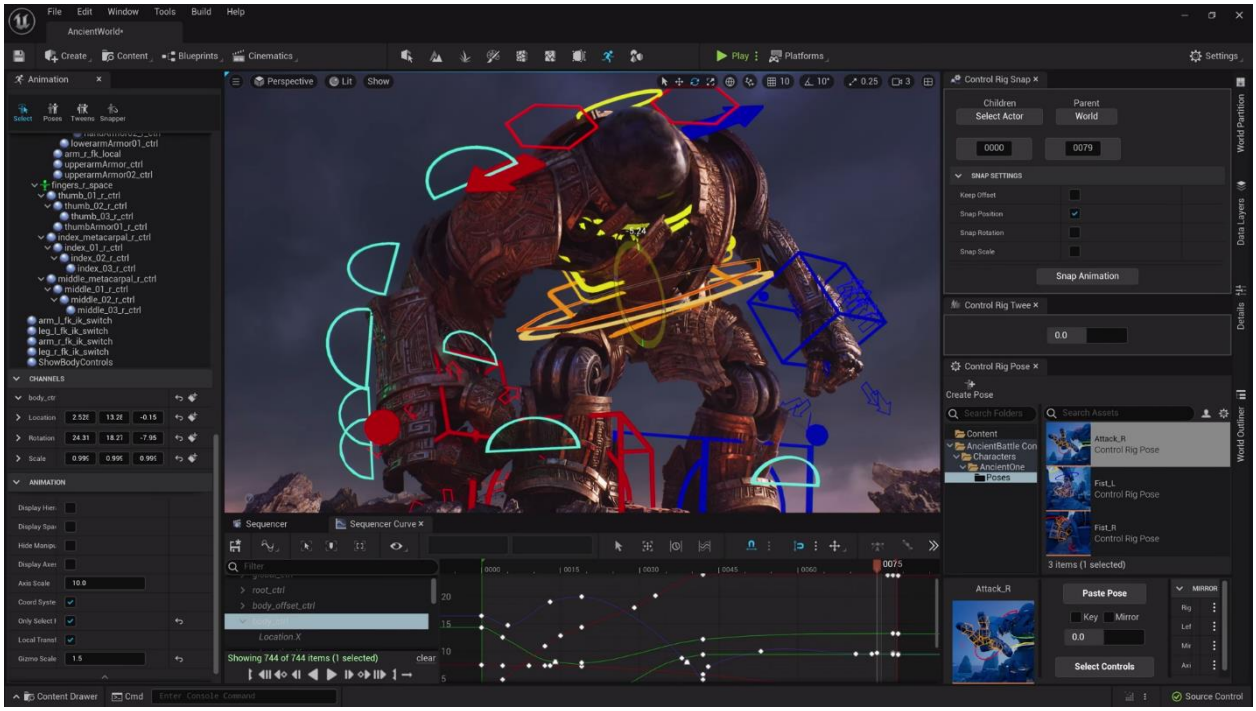


Рисунок 1 – Робоче вікно рушія Unreal

Функція візуального планування движка Unreal дозволяє навіть непрограмістам створювати ігри, і це універсальна потужна система, яка піднімає VR на нові висоти [4].

Переваги Unreal Engine:

- прихильникам графіки сподобається;
- двигун, який працює краще за інші;
- найкращий вибір vr;
- непрограмісти можуть використовувати візуальні креслення;
- необмежені активи доступні на великому ринку.

Недоліки Unreal Engine:

- прості або сольні проекти не рекомендуються;

– потрібні комп’ютери з високопродуктивною графікоюігровий процес кращий у 3D, ніж у 2D.

1.2.2 GameMaker Studio 2

Популярний ігровий движок GameMaker Studio, створений у 2017 році, є найновішою версією продукту, який існує з 1999 року під багатьма назвами та ітераціями.

Ігри, розроблені за допомогою ігрового движка GameMaker (рис. 2), підтримуються на багатьох платформах, включаючи Nintendo Switch. Окрім візуального редактора, GameMaker Language можна використовувати для програмування налаштованої поведінки, яка виходить за рамки, доступні за допомогою візуального програмування.

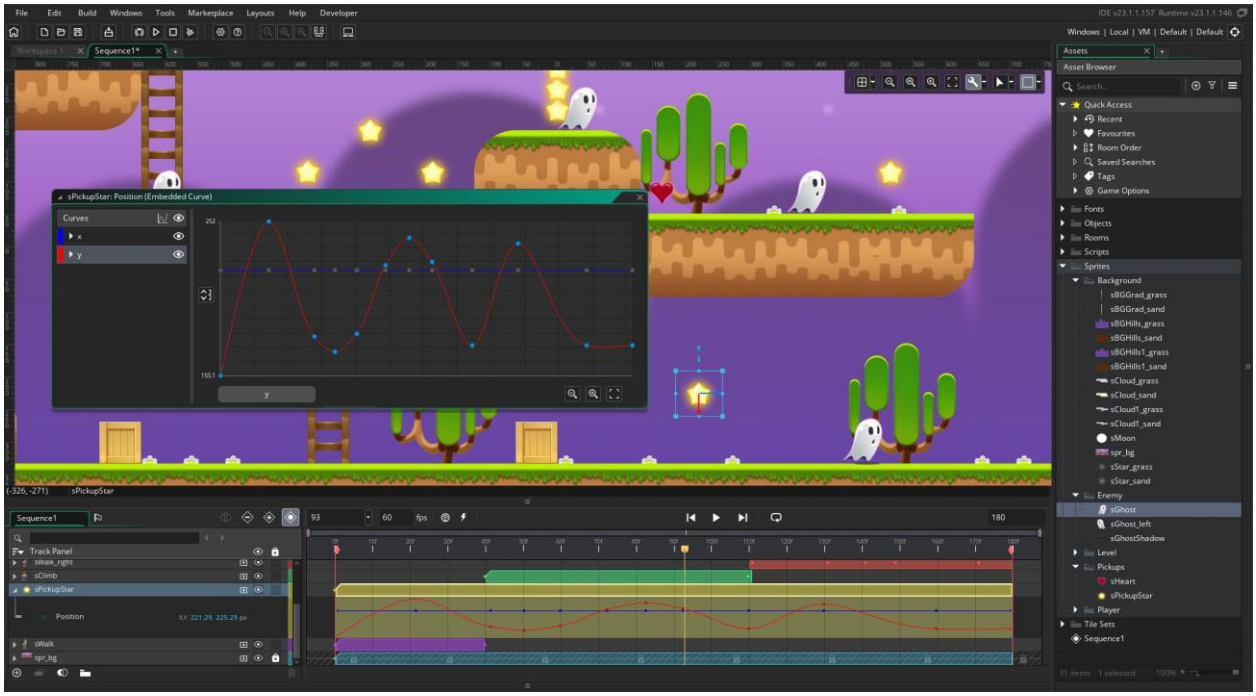


Рисунок 2 – Робоче вікно рушія GameMaker Studio

Як простий у використанні ігровий механізм, кожен може легко розробляти ігри з ним. Як власний ігровий движок і фреймворк GameMaker

Studio 2 не ідеальний для тих, хто шукає бюджетне рішення. Крім того, він також спрямований на інші двигуни, такі як Phasers. Розроблений ігровий движок дійсно має деякі обмеження щодо можливостей 3D, але він не може зрівнятися з візуальним редактором ігрових движків Unity, Unreal [5].

Переваги GameMaker Studio 2:

- підтримуються численні платформи;
- програмувати легко за допомогою перетягування;
- простий у використанні для початківців.

Кілька недоліків:

- призначений переважно для 2d ігор;
- отримується за певну ціну.

1.2.3 CryEngine

Cryengine – це потужний движок 3D-ігор, який забезпечує найсучаснішу графіку для консолей і ПК. Розробники, які хочуть створювати фотореалістичні ігри чи ігри наступного покоління на платформі, як Steam, знайдуть CryEngine (рис. 3) привабливим завдяки підтримці віртуальної реальності та вдосконаленим візуальним ефектам.

Рисунок 3 – Робоче вікно рушія CryEngine

Загальна мета цього ігрового движка – надавати якісний контент з дуже реалістичними та деталізованими персонажами. Завдяки ігровому движку CryEngine розробка ігор доступніша, ніж Unity або Unreal 4.

Коли справа доходить до розробки ігор високого рівня, це ігровий движок, який ви не повинні пропустити. Редагування рівнів можна ефективно виконувати навіть напівпочатківцям за допомогою редактора рівнів CryEngine.

Коли розробник зрозуміє движок, він зможе розробити повну робочу гру. На відміну від більшості ігрових движків, CryEngine менш бажаний для початківців через те, що його важко використовувати та вивчати. Аби зрозуміти, як працює ігровий движок, повним новачкам, ймовірно, варто почати з іншого місця [6].

1.2.4 Unity

Unity – кроссплатформне середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies. Unity дозволяє створювати додатки, що працюють на більш ніж 25 різних платформах, включаючи персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші.

Завдяки постійним оновленням і щороку у Unity (рис. 4) додаються нові важливі функції, такі як Unity Reflect, движок користується неймовірним рівнем підтримки. Оскільки багато компаній і розробників створюють зручні SDK для машини, це не тільки популярний вибір для 2D і 3D ігор, але також віртуальної реальності та доповненої реальності [6].

Переваги ігрового движка Unity:

- для початківців із доходом менше 100 тисяч доларів це безкоштовно;
- підходить для 2d і 3d ігор;
- підтримка розробки мобільних ігор;
- доступні SDK для Vr і Ar;

– безкоштовні ресурси в Asset Store.

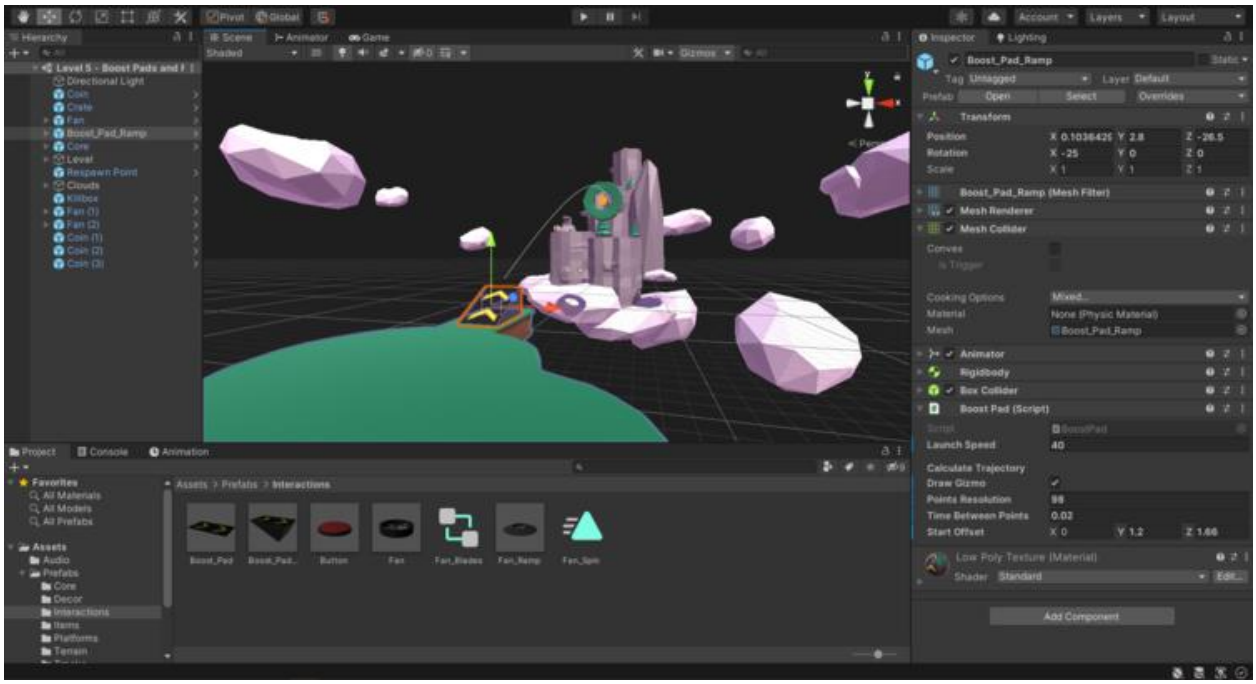


Рисунок 4 – Робоче вікно рушія CryEngine

Скрипти в Unity розробляються на мові C#, яка має багаті можливості і прост у вивченні. Оскільки C# є керованою мовою. Керовані мови набагато менше схильні до помилок, ніж некеровані мови, і, отже, загалом у 5–10 разів продуктивніші.

1.3 Огляд аналогів, створених на базі Unity

Для дипломного проекту було обрано в якості рущія Unity, так як він ідеально підходить до новачків у game development та використовує мову програмування C#. Крім того, в Unity є Tutorial Microgame, який дозволяє за короткий час ознайомитись з палітрою інструментів Unity, та отримати навички в програмуванні основної механіки, яка присутня в комп’ютерних іграх.

Для аналізу можливостей та ідей для майбутнього проекту було проведено аналітичний огляд існуючих ігор на базі обраного рущія.

1.3.1 Hearthstone

Hearthstone (рис. 5) – колекційна карткова онлайн-гра за мотивами всесвіту Warcraft, розроблена компанією «Blizzard Entertainment» і розповсюджується за моделлю free-to-play.

Рисунок 5 – Скрін гри «Hearthstone»

Є багато чудових карткових відеоігор. Багато популярних карткових ігор на основі фентезі, таких як «Magic: The Gathering», випустили адаптації відеоігор, які показали себе досить добре. Blizzard, маючи власну велику фентезі-франшизу, вирішила стрибнути в неї з «Hearthstone».

У гру можна грати безкоштовно, у ній представлені персонажі та теми із серії Warcraft. Два гравці протистоять один одному, використовуючи колоди з

30 карт і героя з унікальною здатністю. Мета полягає в тому, щоб знищити героя супротивника, а перемога створює потік нагород.

Гра мала великий успіх для Blizzard і була перенесена на кілька платформ, доводячи гнучкість Unity [7].

1.3.2 Genshin Impact

«Genshin Impact» (рис. 6) – це рольова гра-екшн, випущена в 2020 році китайським розробником «MiHoYo». Гра має середовище відкритого світу в стилі аніме з елементарною магічною бойовою системою, заснованою на дії, і перемиканням персонажів. «Genshin Impact» стала самим вдалим прикладом серед гри на рух Unity, і розробники дуже грамотно використовують його можливості.

Рисунок 6 – Скрін гри «Hearthstone»

1.3.3 Hollow Knight

Саме спільна ностальгія за класичною грою підштовхнула трьох початківців австралійських розробників ігор до створення власної студії

«Team Cherry» у 2012 році. Арі Гібсон, Вільям Пеллен і Девід Казі з Аделаїди об’єдналися навколо «Zelda II: The Adventure of Link» і додали власний популярний титул до королівства Метроідванія. Як і багато інших успішних студій, ця мала початок зі скромного досвіду [8].

«Hollow Knight» (рис. 7) – це складна та хитра пригода, заснована на комахах. Як милий маленький жучок, гравець протистоїть смертоносним створінням і хитромудрим пасткам, розгадуючи таємниці грибних відходів, кістяних храмів і зруйнованих «готичних» підземних міст. Купуючи карти по дорозі, ви орудуєте цвяхом лицаря, зброєю, схожою на меч, щоб утримувати ворогів і босів.

Рисунок 7 – Скрін гри «Hollow Knight»

У грі наголошується на вправності та дослідженні величезного, взаємопов’язаного підземного королівства, відомого як Хеллоунест, як сказав Пеллен Red Bull Games: «Ключовим елементом, який ми хотіли відтворити, було відчуття грандіозної пригоди в дивній країні, повній монстрів і кумедні персонажі та таємні куточки. Нам дуже подобається відчуття прокладати власний шлях на незвіданій території та стикатися з невідомими небезпеками».

1.3.4 Androrium

Ця гра є представником відразу декількох жанрів:

- 3D (реалістичний 3D-простір / 6-осьове керування кораблем);
- «advanced spaceship simulator» (будівництво корабля/ багато кнопок);
- виживання в космосі (корабельний повітряний бій / статистичні дані для 6 гравців);
- дослідження (станції / сектори / уламки / планети / кораблі).

Для гравця відкривається широкий перелік елементів керування: паливні елементи, електролізер, трансформатор, термоядерний реактор, нова будівельна система, ефекти RCS тощо (рис. 8).

Рисунок 8 – Скрін гри «Androrium»

В грі можна створювати свій корабель і керувати ним, захищатись від безпілотних хвиль, грабувати уламки, досліджувати планети та підкорювати галактику.

Цілі:

- досягти центру галактики;
- захист від дронів;

- оновлення свого корабля;
- грабування дронів та станцій;
- керування своїм кораблем;
- залишитися живим.

За результатами аналітичного огляду було прийнято рішення створити гру у стилі «Advanced Spaceship Simulator» з 2D, в якості рущія обрано Unity та мову програмування C#.

2 ПРОЕКТУВАННЯ

2.1 Концепт-документ проекту

Концепт-документ (дизайн-документ) є першим ігровим документом, що дозволяє отримати загальне уявлення про гру, фіксує її основні особливості.

Вступ: безмежні космічні простори та нове покоління шатлів, які керуються штучним інтелектом – все це невід’ємні частини ігрового світу. На борту корабля знаходиться один астронавт. Під час метеоритного дощу виникають неполадки у системі керування. Відправити сигнал «SOS» до бази неможливо поки не налагодити систему зв’язку.

Тема гри: пошук варіантів налагодження системи керування космічним шатлом.

Ідея гри: виконати усі завдання гри на різних поверхах космічного кораблю для відкриття дверей до рятівного борту.

Мета гри: розвинути логічне мислення гравця, покращити його логічні здібності.

Цільова аудіторія: дана гра розрахована на вік від 12+.

Опис елементів гри: головний герой – космонавт, який керував космічним кораблем, може рухати деякі речі, пересуватись по борту корабля, робити стрибки.

Стиль гри – логічна гра, яка примушує гравця застосовувати логічне мислення для розв'язання головоломок та загадок, які є основою для проходження рівнів.

Декорації – це усе те, що відноситься до елементів космічного кораблю: шестерні, комп’ютери, ящики з припасами, кабелі та багато іншого, з чим доводиться взаємодіяти головному герою.

Опис елеметів гри, які впливають на досвід гравця та яким чином – це логічні загадки, які він змушений вирішувати для того, щоб вижити та пройти далі для того, щоб вибратись з шатлу

2.2 Призми гейм-дизайну

Призми – це невеликий набір питань, які розробнику слід задавати собі на різних етапах розробки. Це не схеми або рецепти, а скоріше, інструменти для вивчення дизайну. Їх запровадив Джессі Шелл, професор університету Карнеги- Меллона та засновник власної студії розробки [10].

До призм гейм-дизайну відносяться: призма резонансу, призма виникнення, призми дій та цілей. Далі наведено опис кожної з них.

2.2.1 Призма резонансу

Для того, щоб розробити призму резонансу, треба відповісти на питання:

1. Що робить гру особливою з точки зору розробника?
2. Що саме інші люди знаходять в грі особливим?
3. Якщо будуть відсутні обмеження щодо створення гри , то якою вона вийде?
4. Що дозволяє розробнику інтуїтивно та чітко уявляти гру?

Гру особливою робить сам ігровий процес, де гравець одразу попадає до складної ситуації, яка змушує швидко знаходити рішення проблеми. Також, сам ігровий сеттінг: зараз дуже популярні ігри, які мають космічну тематику та кораблі, які виведено зі строю.

Інші знаходять особливим те, що гра відповідає тенденціям ігор про космос, де гравець сам змушений розуміти через підказки, або через власну логіку, які кроки необхідні для забезпечення виживання, а також штучний інтелект, що вийшов з-під контролю та який намагається вбити головного героя.

Якщо розробника не будуть обмежувати, то гру можна буде розвинути дуже сильно, придумати багато цікавих рівнів за межами космічного шатлу.

Інтуїтивно і чітко уявляти гру м допомагає уявлення, опис гри та досвід гри у інші схожі ігри.

2.2.2 Призма виникнення

Для того, щоб скласти призму виникнення, необхідно відповісти на наступні запитання:

1. Скільки дієслів є у гравця?
2. На скільки об'єктів поширюються дії кожного дієслова?
3. Скільки гравців може досягти мети?
4. Скільки іменників контролює гравець?
5. Яким чином побічні ефекти змінюють рамки?

Отже, для початку необхідно встановити, які дієслова доступні гравцю: пересуватись, взаємодіяти, активувати, підбирати, тягти, підійматися, спускатися, стрибати, вирішувати, проходити.

Дії кожного дієслова поширюються безпосередньо на головного героя гри, а також тих об’єктів, з якими він взаємодіє, тобто від 1 до 3 (перетягнути коробку на кнопку, активувавши тим самим саму кнопку). Досягти поставленої мети може лише один гравець, оскільки дана гра є одиночною грою.

Гравцю підконтрольні наступні іменники: головний герой, рухомі предмети, механізми, головоломки.

Оскільки дана гра є квестом, то вона має певний сценарій проходження, отже побічні ефекти, окрім визначених грою, не матимуть впливу на рамки даної гри.

2.2.3 Призма дій

До цієї призми відносятся наступні питання до розробників:

1. Які базові дії доступні користувачеві у грі?
2. Які стратегічні дії доступні?
3. Які стратегічні дії хотів би побачити розробник? Як він може змінити гру, щоб зробити їх можливими?

4. Які дії гравці хотіли б виконувати у грі, але не можуть? Чи можливо дати їм таку змогу за допомогою базових чи стратегічних дій?

Базові дії, доступні гравцеві – це дії, що відповідають за переміщення – стрибки, ходьба, підіймання/опускання. Також до базових дій можна віднести фізичну взаємодію головного героя з об’єктами на карті – перетягування, підіймання та опускання.

До стратегічних дій можна віднести взаємодію гравця з механізмами, що відкриває інтерфейс проходження логічних завдань, та самі дії гравця безпосередньо в самому інтерфейсі.

Серед стратегічних дій, що бракують, можна виділити механіку взаємодії гравця з дверима, тобто неможливість пройти певний сектор, доки не вирішена головоломка в поточному секторі.

Виходячи з жанру даної гри, співвідношення базових та стратегічних дій є достатнім для гарного досвіду гравця під час проходження гри. Можливо, гравці хотіли б мати інвентар гравця, де б можна було зберігати предмети для проходження головоломок. Натомість, гравцю достатньо стратегічних дій в інтерфейсі певної головоломки для її вирішення, тобто необхідні «предмети» чи механізми знаходяться безпосередньо в рамках самої головоломки.

2.2.4 Призма цілі

Для того, щоб скласти призму цілі, необхідно відповісти на наступні запитання:

1. Які кінцева мета гри?
2. Мета очевидна для гравця?
3. Якщо цілей декілька, гравці зможуть це зрозуміти?
4. Чи є смисловий зв’язок між різними цілями?
5. Чи притаманні цілям конкретність, реальність досягнення та гідна винагорода?
6. Чи достатньо збалансовані короткострокові та довгострокові цілі?

7. Чи можуть гравці вибирати цілі на свій розсуд?

Кінцевою ціллю даної гри є вибратись з космічного корабля, попутно полагодивши зламані механізми. Данна ціль є очевидною для гравця, оскільки всі механіки спеціальних дії пов’язані саме на процесі взаємодії з механізмами у вигляді головоломок.

Оскільки ціль є одна – вибратись, то гравці це розуміють з самого початку. Дану ціль досягнути реально, вирішивши всі головоломки.

Данну ціль можна розбити на підгрупи – короткострокові цілі та одна довгострокова. Короткострокові цілі – це вирішення головоломок задля того, щоб полагодити зламані механізмі. Довгострокова ціль – завершити подорож, вибравшись із корабля.

Оскільки, для досягнення загальної цілі необхідно вирішувати поточні головоломки, то гравець може обирати порядок, в якому він буде проходити надані головоломки.

2.3 Моделювання системи

Щоразу, коли розробник щось планує, він записує ідеї, тому що це робить усю картину зрозумілою з усіма вимогами. Подібним чином, проектуючи та розробляючи програмне забезпечення чи гру, необхідно підготувати план того, що буде реалізовано.

UML розшифровується як «Unified Modeling Language» і може використовуватися для візуалізації різних функцій гри за допомогою діаграм UML. Діаграми UML створені не лише для розробників програмного забезпечення/ігор, але й для бізнес-менеджерів, інженерів, звичайних людей та всіх, хто зацікавлений у розумінні системи, а графічне представлення полегшує її розуміння.

Система (візуалізована та створена за допомогою UML) може бути програмним забезпеченням, грою або будь-яким іншим цивільним чи механічним проектом.

Концептуальну модель можна визначити як модель, яка складається з понять та їхніх зв’язків. Це робиться ще до початкового планування того, як все працюватиме. Це демонструє ранню концепцію та стосунки. Концептуальна модель є першим кроком перед малюванням діаграми UML для будь-якого проекту. Це допомагає отримати ширшу та цілісну картину ідеї проекту.

Відповідно до концепт-документу можна побудувати діаграму варіантів використання для відображення функцій та можливостей гравця у ігровому просторі (рис. 9).

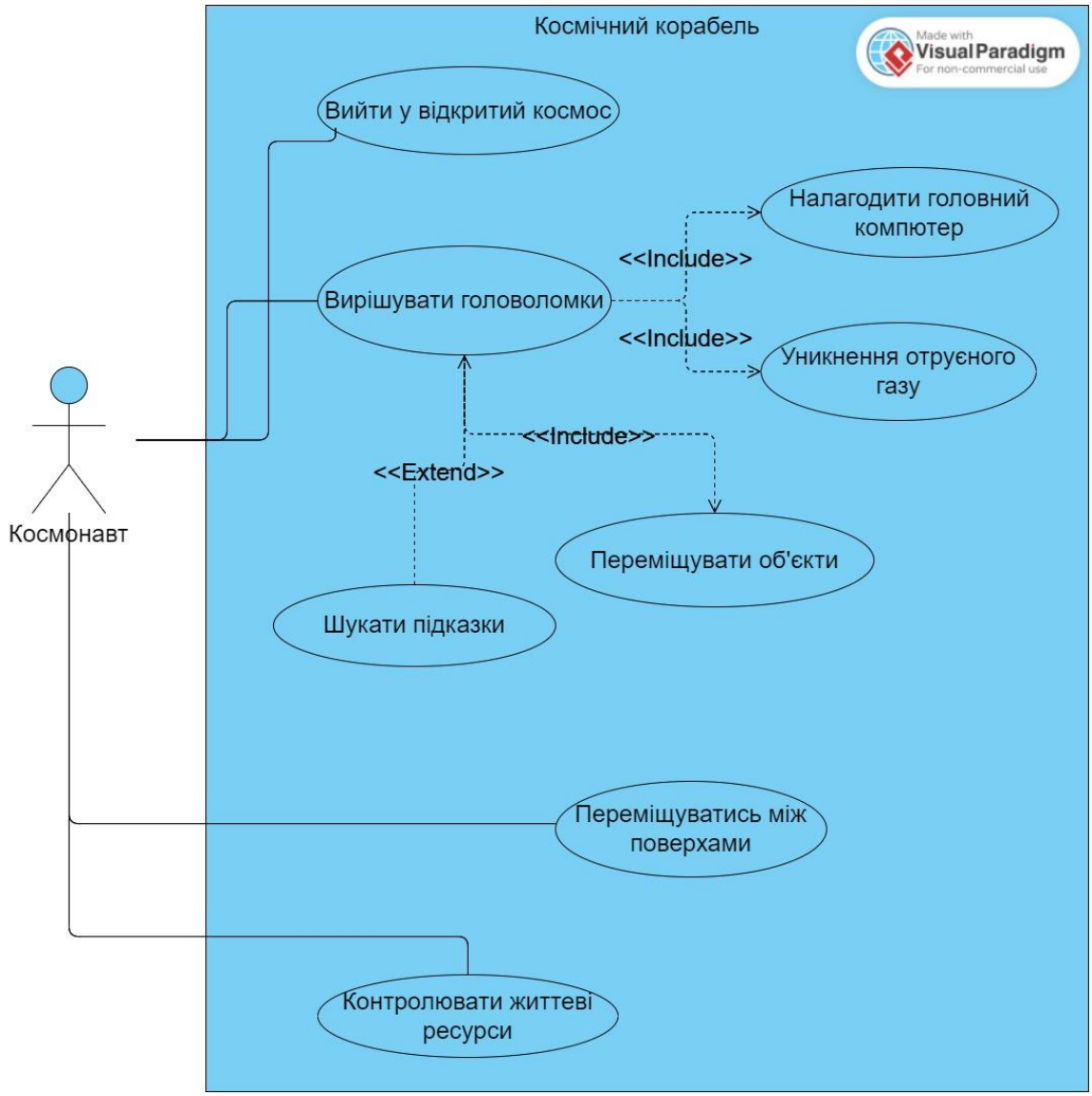


Рисунок 9 – Діаграма Use-Case

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Основні механіки гри

Основними механіками гри є взаємодія головного героя з оточуючим світом, а саме з елементами космічного кораблю перебуває головний герой гри. По-перше, головний герой має змогу пересуватися по кораблю на вже звичні для усіх кнопки «a, w, s, d», також він має змогу взаємодіяти з предметами, які його оточують через клавішу «e», стрибати через кнопку «space» та ставити на паузу або відміняти дію на кнопку «esc».

Головний упор зроблений на головоломки, які з однієї сторони виглядають дуже простими, але якщо погано прораховувати ходи, то можна зробити помилку и знадобиться починати алгоритм пошуку рішення знову. Для вирішення деяких головоломок зроблені спеціальні помітки на кораблі, які, у цілому є підказками, які повинні допомогти з пошуком рішення.

Елементами гри, з якими можна взаємодіяти є:

- кнопка (на підлозі), на неї треба класти ящики, для того, щоб їх активувати;
- кнопка (на стіні), на неї треба просто натиснути, для того, щоб відбулась якась дія (відкриття дверей та інше);
- кнопка (двигун), використовується для того, щоб вирішити головоломку з двигуном за запустити корабель;
- драбина, за допомогою їх можна збиратись догори або злазити донизу для того, щоб пересуватись між рівнями;
- ящик, їх треба брати та будувати собі гору для того, щоб дібратись до деяких місць, або просто покласти на кнопки, які знаходяться на підлозі;
- перемикач, для того, щоб перемикати положення деяких елементів;
- маніпулятор для безпечного пересування ящиків, щоб не витрачати час на пересування у небезпечних зонах;

– пересувний комп’ютер, який можна брати, так користуватись маніпулятором у секціях де є маніпулятори.

3.2 Створення головного персонажа

По-перше, необхідно створити 2D проект в редакторі Unity та налаштувати його оточення для більш зручного користування.

Для дипломного проекту було прийнято рішення використовувати власні спрайти для графічного контенту. Так, головний герой – космонавт був створений за допомогою редактору Photoshop, базуючись на інших представниках даного жанру, він має бути у скафандрі з шоломом (рис. 10).

Рисунок 10 – Спрайт головного героя

Також було додано багато спрайтів для героя, наприклад як він біжить або як він переміщує ящик.

Розглянемо реалізацію основної механіки для головного героя (клас «Player»):

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Player : MonoBehaviour
{
```

```
private float speedX;
private float horizontalSpeed;
private float verticalImpulse;
private Transform transform_Player;
private SpriteRenderer spriteRenderer_Player;
private Rigidbody2D rigidbody2D_Player;
private bool player_InAir;
public bool itemInHands { get; set; }
```

Після оголошення глобальних змінних слід описати метод Start():

```
void Start()
     {
          transform_Player = GetComponent<Transform>();
          spriteRenderer_Player = GetComponent<SpriteRenderer>();
          rigidbody2D_Player = GetComponent<Rigidbody2D>();
          horizontalSpeed = 0.2f;
          verticalImpulse = 12;
          player_InAir = false;
          itemInHands = false;
     }
```

Наступні методи (Update(), FixedUpdate()) потрібні для апдейту та задавання швидкості і координат початкової локації космонавта.

```
void Update()
{
  if (Input.GetKeyUp(KeyCode.Space) && player_InAir == false)
{
  player_InAir = true;
  rigidbody2D_Player.AddForce(new Vector2(0, verticalImpulse),
  ForceMode2D.Impulse);
}
}

void FixedUpdate()
{
  if (Input.GetKey(KeyCode.A))
   {
     if (spriteRenderer_Player.flipX == true)
       {
         spriteRenderer_Player.flipX                              =
         !spriteRenderer_Player.flipX;
       }
     speedX = horizontalSpeed;
     transform_Player.Translate(speedX, 0, 0);
  }
```

```
  else if (Input.GetKey(KeyCode.D))
  {
    if (spriteRenderer_Player.flipX == false)
    {
     spriteRenderer_Player.flipX = !spriteRenderer_Player.flipX;
    }
     speedX = -horizontalSpeed;
     transform_Player.Translate(speedX, 0, 0);
  }
  speedX = 0;
 }

 private void OnCollisionEnter2D(Collision2D collision)
 {
  if (collision.gameObject.layer == 0)
   {
     player_InAir = false;
   }
 }
}
```

Для анімації відображення послідовності рухів космонавту використовувались наступні спрайти (рис. 11).

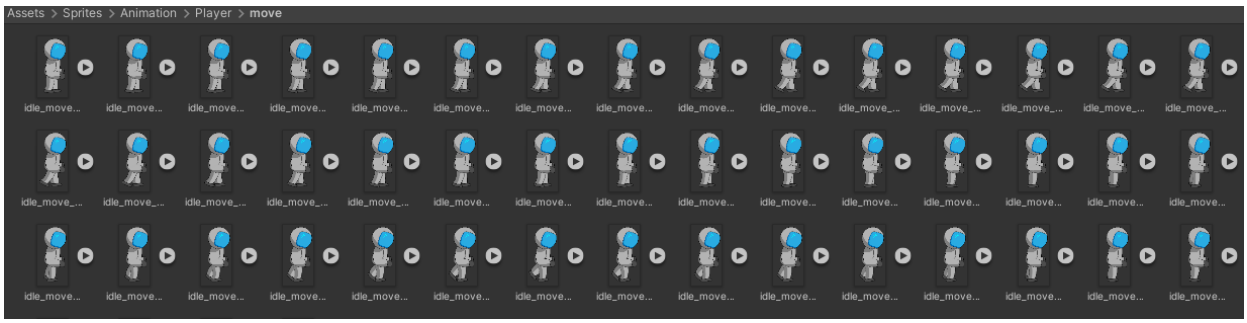


Рисунок 11 – Ассет спрайтів для анімації космонавту

Скрипти для гравця:

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

public class Animation : MonoBehaviour
{
    public Player player;
    public Animator anim;
```

```
        private bool control;
        public GameObject GasTrigger;
        public schafa[] schafa = new schafa[3];

        void Start() {
        anim = GetComponent<Animator>();
            control = false;
        }
        void Update()
        {
           if (MoveObj)
            {
                anim.SetBool("move", true);
            } else
            {
                anim.SetBool("move", false);
            }
            anim.SetBool("jump", false);
            if  (schafa[0].GetComponent<schafa>().ActiveCollider
||       schafa[1].GetComponent<schafa>().ActiveCollider       ||
schafa[2].GetComponent<schafa>().ActiveCollider)
            {
                anim.SetBool("move", true);
            }
            else
            {
                anim.SetBool("move", false);
            }
```

При кожному крокі героя будемо перевіряти наявність «антогоніста» – отруєного газу, при якому космонавт вмирає.

```
      if (GasTrigger.GetComponent<gas>().Timer <= 0)
            {
                control = true;
            }
            if (control)
            {
                DieAnim();
            } else
            {
                anim.SetBool("die", false);
            }
            if (Input.GetKey(KeyCode.A) ||
Input.GetKey(KeyCode.D))
            {
                if (player.itemInHands)
                {
                    anim.SetBool("box_walking", true);
```

```
                    //anim.SetBool("box_iddle", false);
                } else
                {
                    anim.SetBool("walking", true);
                }
        } else
            {
                if (player.itemInHands)
                {
                    anim.SetBool("box_walking", false);
                    anim.SetBool("box_iddle", true);
                } else
                {
                    anim.SetBool("box_iddle", false);
                    anim.SetBool("walking", false);
                }
            }
            if (Input.GetKeyUp(KeyCode.Space))
            {
                anim.SetBool("jump", true);
            }
        }
```

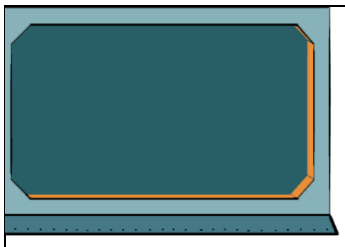
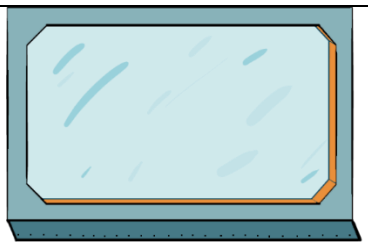
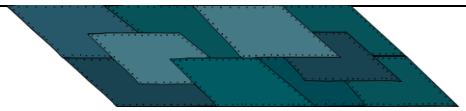
Наступні методи будуть зчитувати направлення переміщення героя через кнопки.

```
    void OnTriggerStay2D (Collider2D other) {
            if (other.gameObject.CompareTag("stairs"))
         {
                if (Input.GetKey(KeyCode.A) ||
Input.GetKey(KeyCode.D))
                {
                    anim.SetBool("walking", true);
                }
                else
                {
                    anim.SetBool("walking", false);
                }
                if (Input.GetKey(KeyCode.W) ||
Input.GetKey(KeyCode.S))
                {
                    anim.speed = 1f;
                    anim.SetBool("ladder", true);
                }

    else if (anim.GetBool("ladder"))
                {
                    anim.speed = 0f;
                }
```

```
            }
private void OnTriggerExit2D(Collider2D other)
        {
            if (other.gameObject.CompareTag("stairs"))
            {
            anim.speed = 1f;
            anim.SetBool("ladder", false);

            if (Input.GetKey(KeyCode.A) ||
Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.W) ||
Input.GetKey(KeyCode.S))
                {
                    anim.SetBool("walking", true);
                }
                else
                {
                    anim.SetBool("walking", false);
                }
            }
        }

      void DieAnim()
        {
            anim.SetBool("die", true);
            control = false;
        }
    }
```

Для візуалізації фону гри використовувались такі ассети (таблиця 1):

Таблиця 1 – Ассети для фону сцен гри

		
Стінка	Вікно	Елемент підлоги

Ассети для візуалізації лазіння по драбині астронавта представлено на рисунку 12.

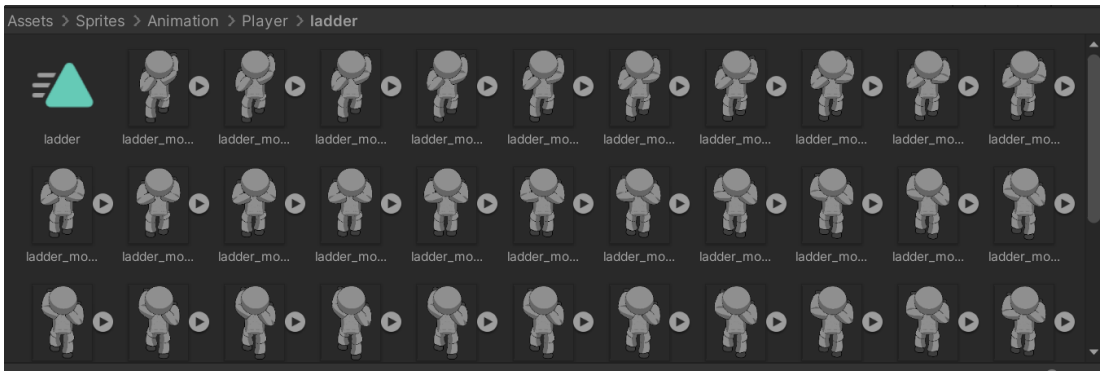


Рисунок 12 – Ассети для відображення руху по дробині

Скрипт для взаємодії гравця з драбиною:

```
using UnityEngine;
using UnityEngine.UI;

public class Ladder : MonoBehaviour
{
[SerializeField] float coolDown;
    [SerializeField] float timer;
    [SerializeField] float speed = 5;
    private Vector2 playerVelocity;
    public Player Player;

    void Update()
    {
        playerVelocity =
Player.GetComponent<Rigidbody2D>().velocity;
    }
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")&&playerVelocity != new
Vector2(0, 0))
        {
            Player.GetComponent<Rigidbody2D>().velocity = new
Vector2(0, 0);
            Player.GetComponent<Rigidbody2D>().gravityScale = 0;
        }
    }
```

За концепт-документом, космонавт може перетягувати ящики для будови споруд та комп’ютер (рис. 13):

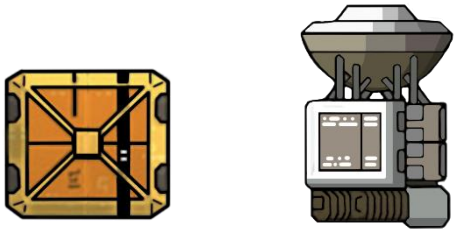


Рисунок 13 – Спрайти ящику та пересувного комп’ютера

Крім цього, за допомогою ящиків можна натискати на кнопки (рис. 14). Для цієї мети було розроблено ассети для гри (рис 15).

Рисунок 14 – Спрайт космонавта з ящиком у руках

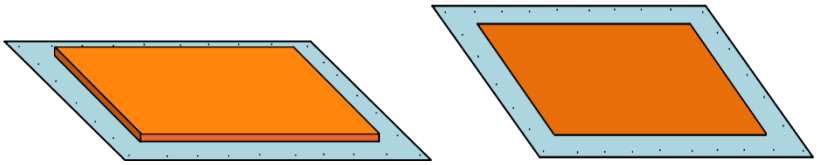


Рисунок 15 – Ассети для кнопки

Дані складові являють собою частину першої головоломки з якими можна взаємодіяти. Було додано головоломки на деяких рівнях корабля, які виглядають як кнопки, на які треба або натискати у певній послідовності, або класти на них ящики для того, щоб викликати виклик ліфта, для проходження на наступний ярус (рівень) космічного корабля.

Було написано скрипти для взаємодії гравця з даними елементами. Приклад розміщення спрайтів на карті:

Рисунок 16 – Скрін сцени гри з натисканням на кнопку ящиком

Скрипт для пересувного комп’ютера:

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

public class schafa:MonoBehaviour
{
    private BoxCollider2D collider2D;
    public bool ActiveCollider;
    Rigidbody2D rigid;
    private void Start()
    {
        rigid = this.GetComponent<Rigidbody2D>();
        collider2D = this.GetComponent<BoxCollider2D>();
        collider2D.enabled = false;
        ActiveCollider = false;
    }

void Update()
    {
        if (ActiveCollider)
        {
            collider2D.enabled = true;
```

```
                rigid.constraints &=
 ~RigidbodyConstraints2D.FreezePositionY;
            }
            else
            {
                collider2D.enabled = false;
                rigid.constraints =
 RigidbodyConstraints2D.FreezePositionY;
            }
        }

     public void OnTriggerStay2D(Collider2D other)
        {
            if (other.CompareTag("Player") &&
 Input.GetKeyUp(KeyCode.E))
            {
                ActiveCollider = !ActiveCollider;
            }
        }

        public void OnTriggerExit2D(Collider2D other)
        {
            if (other.CompareTag("Player"))
            {
                ActiveCollider = false;
                collider2D.enabled = false;
          }}}
```

Скрипт для ящика:

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

public class Box : MonoBehaviour
{
    public Player player;
    private bool box_status;
    private float flip;
    private Vector3 position;
    void Start()
    {
        flip = 1f;
        box_status = false;
    }
    void FixedUpdate()
    {
        if (box_status == true)
        {
            position = player.transform.position;
```

```
his.gameObject.GetComponent<SpriteRenderer>().enabled = false;
this.gameObject.GetComponent<BoxCollider2D>().enabled = false;
this.gameObject.GetComponent<Rigidbody2D>().gravityScale = 0;
                if (Input.GetKey(KeyCode.D))
                {
                  position.x = player.transform.position.x + 3f;
                    flip = 1f;
                }
                else if (Input.GetKey(KeyCode.A))
                {
                  position.x = player.transform.position.x - 3f;
                    flip = -1f;
                }
              position.x = player.transform.position.x+flip*3f;
                this.transform.position =
Vector3.Lerp(this.transform.position, position, 100f*
Time.deltaTime);
            } else
            {
this.gameObject.GetComponent<SpriteRenderer>().enabled = true;
            }
        }

    public void OnTriggerStay2D(Collider2D other)
        {
            if (other.CompareTag("Player") &&
Input.GetKeyDown(KeyCode.E) && box_status == false &&
player.itemInHands == false)
            {
                box_status = true;
                player.itemInHands = true;

            }
            else if (other.CompareTag("Player") &&
Input.GetKeyDown(KeyCode.E) && box_status == true &&
player.itemInHands == true)
            {
                box_status = false;
                player.itemInHands = false;
this.gameObject.GetComponent<BoxCollider2D>().enabled = true;
this.gameObject.GetComponent<Rigidbody2D>().gravityScale = 3;
            }
    }
    }
```

3.2 Реалізація антогоніста гри

Введено в гру антагоніста (отруйна хмара газу) та вбивство головного героя. Для цього було створено відповідні ассети та проставлено на карті розміщення перешкоди (рис. 17).

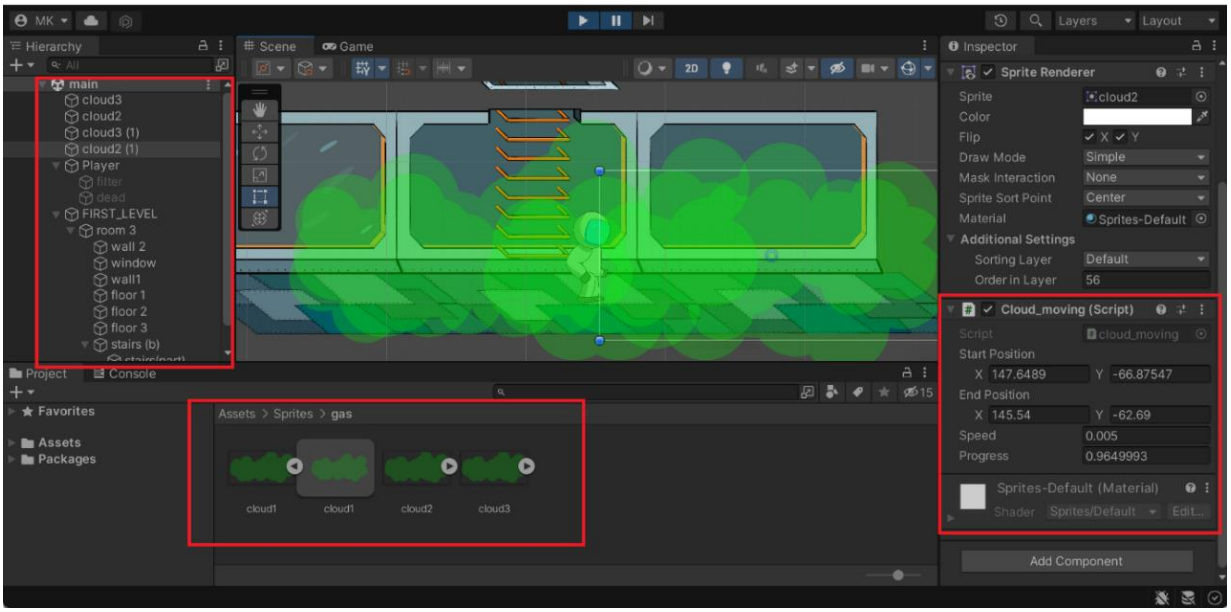


Рисунок 17 – Ассет отруйної хмари

Після чого було додано динаміку до гри. Код для руху газової хмари за допомогою скрипта «cloud_moving»:

```
using UnityEngine;

public class cloud_moving : MonoBehaviour
{
    private SpriteRenderer spriteRenderer_Robot;
    public Vector2 startPosition;
    public Vector2 endPosition;
    public float speed;
    public float progress;

    void Start()
    {
        transform.position = startPosition;
        spriteRenderer_Robot = GetComponent<SpriteRenderer>();}
```

```
    void FixedUpdate()
    {
        transform.position = Vector2.Lerp(endPosition,
startPosition, progress);

        if (progress >= 1)
        {
            speed = -speed;
        }
        else if (progress < 0)
        {
            speed = -speed;
        }
        progress += speed;
    }}
```

Створено новий 2D Object для виділення області дії газу (при потраплянні гравця в цю зону починае працювати таймер). Написано скрипт для вбивства героя, після того як герой входить до зони дії хмари, починається зворотній відлік часу (код таймеру до смерті героя = 60 секунд).

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

public class gas: MonoBehaviour
{
    public Player Player;
    public float Timer;
    private float TimerTemp;
    public Vector2 RestartPossition;
    private bool TimerOn;
    void Start()
    {
        TimerTemp = Timer;
        TimerOn = false;
    }

    void FixedUpdate()
    {
        if (TimerOn)
        {
            Timer -= Time.deltaTime;
        }
        if (Timer <= 0)
        {
            Invoke("Reload", 2f);
        }
```

```
    }
    void OntriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            TimerOn = true;
        }
    }
    void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            TimerOn = true;
        }
    }


    void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            Timer = TimerTemp;
            TimerOn = false;
        }
    }
}
```

3.3 Реалізація рівня «Головоломка з шестернями»

Наступними елементами, які присутні в грі є настінні елементи для активації головоломки та елементи самої задачі (рис. 18):

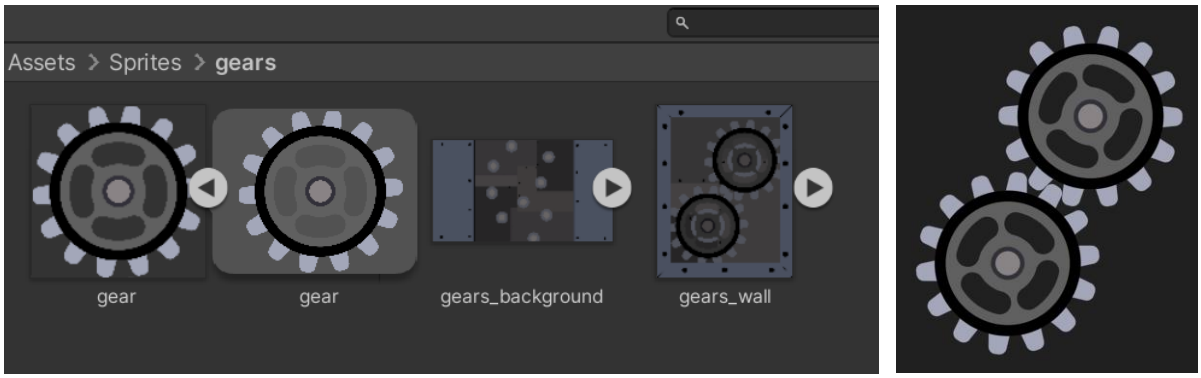


Рисунок 18 – Елементи головоломки

Завдання: перемістити усі деталі, що змінили своє місцеположення в результаті аварії, на правильні місця (рис. 19).

Перешкоди: щоб дістатися до механічного блоку необхідно пройти через отруйний газ на якому встановлено таймер, при тривалому перебування у газовій хмарі гравець вмирає. Для імплементації головоломки: створили таку ієрархію, де “gear” – статичні шестерні, усі чайдли від ActiveGears – динамічні шестерні, що потрібно встановити назад у механізм на tartget-и.

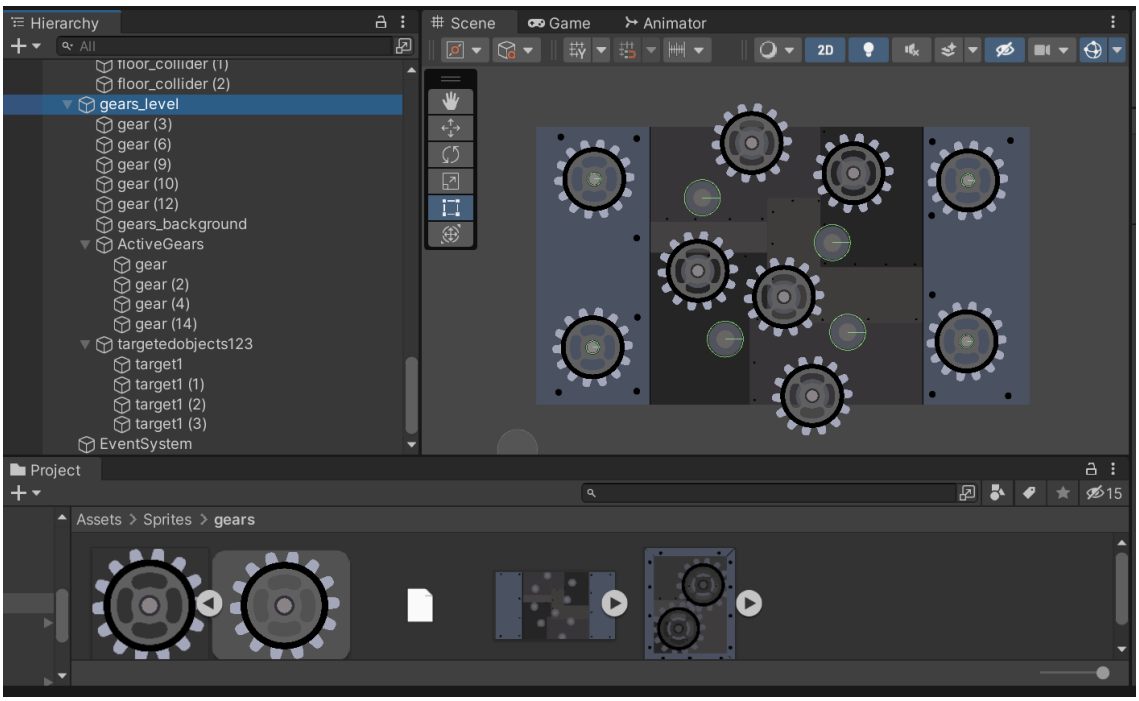


Рисунок 19 – Приклад зібранної головоломки

Об’єкти target є екземплярами класу Trigger, який має поле ObjectInTrigger – об’єкт у позиції. При встановленні елементу на gear_target ObjectInTrigger приймає значення true; Скрипт:

```
using UnityEngine;
public class Trigger : MonoBehaviour
{
    public bool ObjectInTrigger { get; set; }

    void Start()
    {
```

```
            ObjectInTrigger = false;
        }
        void OnTriggerStay2D(Collider2D other)
        {
            if (other.CompareTag("gear_target") &&
Input.GetMouseButtonUp(0))
            {
                other.transform.position =
this.transform.position;
                ObjectInTrigger = true;

other.gameObject.GetComponent<Mouse_MovingObj>().enabled =
false;
            }
        }
        void OnTriggerEnter2D(Collider2D other)
        {
            if (other.CompareTag("Key"))
            {
                ObjectInTrigger = true;
            }
        }
    }
```

Створили настінний об’єкт і додали тригер для відкриття головоломки гравцев. Скрипт:

```
using UnityEngine;
public class gears_level : MonoBehaviour
{
    public GameObject Gears_Level;
    public GameObject Camera;
    public GameObject Junk;
    public Trigger[] gears = new Trigger[4];
    public Player player;
    public gas gas;
    [System.Obsolete]
    void Start()
    {
        Junk.SetActive(false);
    }
    void FixedUpdate()
    {
        if (gas.Timer<=1)
        {
```

Перевіряємо:якщо таймер вийшов закриваемо вікно. У разі якщо всі об’єкти на позиціях – закриваемо вікно.

```
                Invoke("BacktToGame", 0.5f);
            }
                  if (gears[0].ObjectInTrigger &&
gears[1].ObjectInTrigger && gears[2].ObjectInTrigger &&
gears[3].ObjectInTrigger)
            {
                Camera.transform.position =
player.transform.position;
                Junk.SetActive(true);
                Invoke("BacktToGame", 0.5f);
            }
        }
```

Якщо гравець знаходиться поруч з об’єктом і натиснув клавішу «Q» – переносимо вид камери на розроблену схему головоломки та блокумо можливіть користувача управляти камерою:

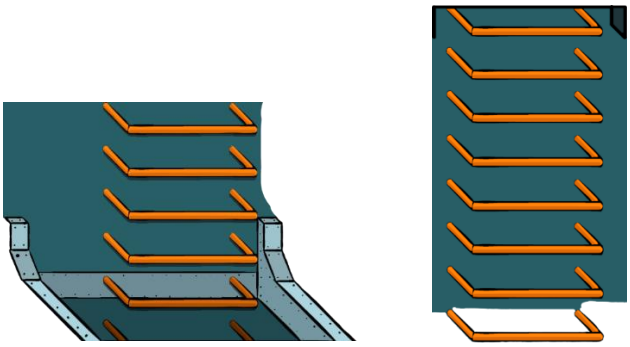
```
        [System.Obsolete]
        void OnTriggerStay2D(Collider2D other)
        {
            if (other.CompareTag("Player") &&
Input.GetKey(KeyCode.Q))
            {
                Gears_Level.SetActive(true);
                Vector3 newCamPos = new Vector3(32.4f, 103.9f, -
57f);
                player.gameObject.GetComponent<Player>().enabled
= false;

Camera.gameObject.GetComponent<camera_moove>().enabled = false;
                Camera.transform.position = newCamPos;
            }
        }
        void BacktToGame ()
        {
            Gears_Level.SetActive(false);
            player.gameObject.GetComponent<Player>().enabled   =
true;

Camera.gameObject.GetComponent<camera_moove>().enabled = true;
            this.gameObject.GetComponent<gears_level>().enabled
= false;
        }
    }
```

Після встановлення усіх елементів на місця головоломка закривається.

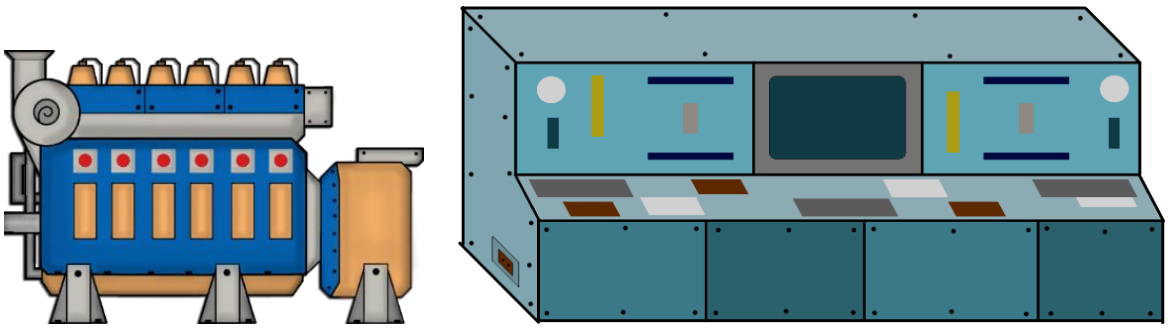
3.4 Реалізація моделі двигуну корабля

Для створення моделі двигуну постала необхідність у створенні ассетів для двигуну – модель самого двигуна, модель блоку, що необхідно під’єднати та інтерфейс самої головоломки. Окрім цього необхідно було створити об’єкти, що знаходитимуться на фоні секції, де знаходиться об’єкт, а також ассети самої драбини, якою лазатиме головний герой.

Ассети, що було створено для мапи корабля представлено на рисунку 20:



а) драбина (верх) б) драбина (низ)



в) двигун корабля г) комп’ютер

Рисунок 20 – Набір ассетів для корабля

За задумкой гри, герой повинен розгадати квести щодо запуску двигуну. Отже, потрібні ассети для відображення двигуна зблизу (рис. 21а) та блок двигуна, якій гравцю потрібно буде зєднувати (рис. 21б).

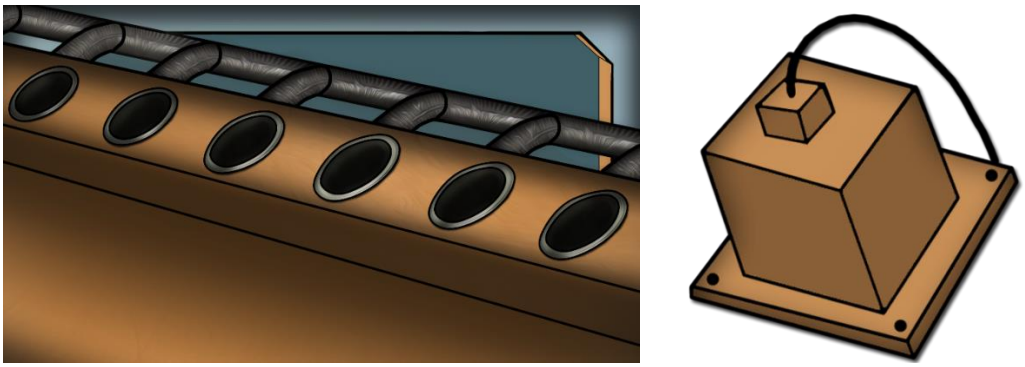


Рисунок 21 – Ассети, пов’язані з двигуном

Головоломка «Двигун»: завданням є під’єднати всі блоки двигуна до відповідних роз’ємів. Для реалізації головоломки було створено наступну ієрархію:

1. Engine_level – окремий інтерфейс, що відкривається на час вирішення головоломки.
2. Engine_2 – це фон, вид двигуна зблизу.
3. Rozetka (1-5) – це блоки, що необхідно під’єднати до двигуна.
4. GameObject (1-4) – це об’єкти, колізія з якими активує тригер «під’єднання» блоку до двигуна.

Після під’єднання всіх блоків двигуна (рис. 22), інтерфейс головоломки закриється, що дасть змогу продовжити проходження гри.

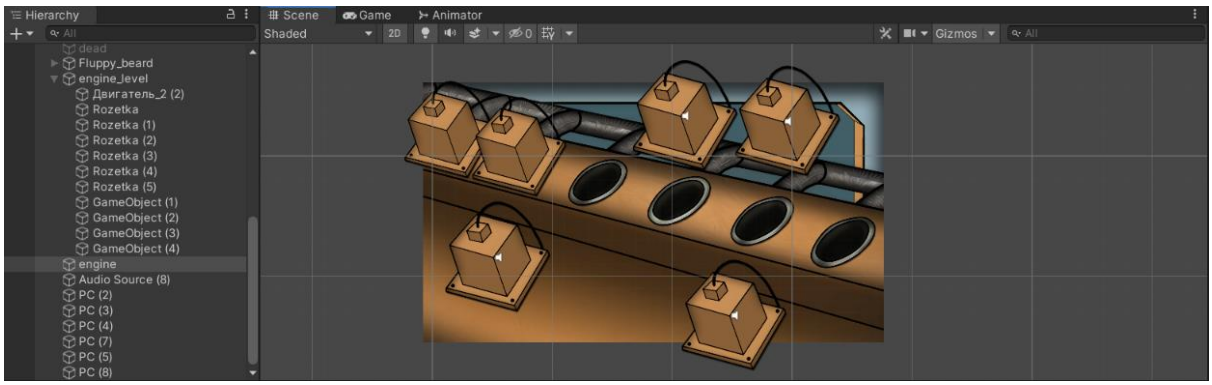


Рисунок 22 – Скрін елементу головоломки з двигуном

Розлгянемо скрипт для переміщення блоку:

```
using UnityEngine;
public class Mouse_MovingObj : MonoBehaviour
{
    private Vector3 offset;
    public Camera curCam;
    void OnMouseDown()
    {
            Vector3 temp_V = Input.mousePosition;
            temp_V.z = 57f;
            offset = this.transform.position -
curCam.ScreenToWorldPoint(temp_V);
    }
    void OnMouseDrag()
    {
        Vector3 newPosition = new Vector3(Input.mousePosition.x,
Input.mousePosition.y, 57f);
        this.transform.position =
(curCam.ScreenToWorldPoint(newPosition) + offset);
    }
}
```

Об’єкти GameObject також є елементи класу «Триггер», вони відповідають за прогрес проходження головоломок після переміщення блоків двигуну на GameObject-об’єкти:

```
using UnityEngine;
public class Trigger : MonoBehaviour
{
    public bool ObjectInTrigger { get; set; }
    void Start()
    {
        ObjectInTrigger = false;
    }
    void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("gear_target") &&
Input.GetMouseButtonUp(0))
        {
            other.transform.position = this.transform.position;
            ObjectInTrigger = true;

ther.gameObject.GetComponent<Mouse_MovingObj>().enabled = false;
        }
    }
```

```
    void OnTriggerEnter2D(Collider2D other)
        {
            if (other.CompareTag("Key"))
            {
                ObjectInTrigger = true;
            }
        }
    }
```

Модель двигуну має скрипт переходу від звичного інтерфейсу до інтерфейсу головоломки і навпаки (після проходження головоломки). Об’єкти gear є екземпляру класу «Trigger», які відповідають за об’єкти GameObject, тобто активація цих об’єктів запускає триггер гри:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class engineScript : MonoBehaviour
{
    public GameObject Engine_Level;
    public GameObject Camera;
    public Trigger[] gears = new Trigger[4];
    public Player player;
    [System.Obsolete]

void Start()
    {
        this.gameObject.GetComponent<AudioSource>().enabled =
false;
    }
    void FixedUpdate()
    {
        if (gears[0].ObjectInTrigger && gears[1].ObjectInTrigger
&& gears[2].ObjectInTrigger && gears[3].ObjectInTrigger)
        {
            Camera.transform.position =
player.transform.position;
            Invoke("BacktToGame", 0.5f);
        }
    }
    [System.Obsolete]


 void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("Player") &&
Input.GetKey(KeyCode.E))
        {
```

```
            Engine_Level.SetActive(true);
            Vector3 newCamPos = new Vector3(-80.74f, 155.6f, -
57f);
            player.gameObject.GetComponent<Player>().enabled =
false;

Camera.gameObject.GetComponent<camera_moove>().enabled = false;
            Camera.transform.position = newCamPos;
        }
    }

void BacktToGame()
    {
        Engine_Level.SetActive(false);
        player.gameObject.GetComponent<Player>().enabled = true;
        Camera.gameObject.GetComponent<camera_moove>().enabled =
true;
        this.gameObject.GetComponent<AudioSource>().enabled =
true;
        this.gameObject.GetComponent<engineScript>().enabled =
false;
    }
}
```

3.5 Огляд ризиків

3.5.1 Ризики пов’язані з ходом гри

По-перше, це дисбаланс гри між протагоністом та антагоністом. Причиною виступає недосвідченість розробника, погане тестування гри може призвести до неможливості завершення певних рівнів гри, або навпаки робить їх занадто легкими.

Для оптимізації такого ризику є тестування усіх рівнів гри при різних умовах сторонніми особами, що не є розробниками цієї гри.

Наступний ризик – створення однорідної(не цікавої) гри. Оскільки гра є невеликою і досить простою, це може причинити брак контенту, відсутність бажання повторного проходження гри, непродуманість і неузгодженість ігрових задумів. Так, наприклад, перетягування деталей є досить простою задачею і не вимагає особливих зусиль від гравця.

З точки зору оптимізації можна створити ідею загальної гри, а не зосереджетися на одній цікавій «фічі», після проходження якої користувач

втратить інтерес до ігрового процесу. Засвічення частини космічного корабля, для того щоб не відкривати гравцю усіх завдань, а пробудити в ньому інтерес до того, що його чекає далі. Проектування лінії сюжету, з можливістю подальшого масштабування, додання відео-вставок перед і пост історій.

Ще один ризик – незрозумілість гравцем наступного кроку. Розроблено три різних рівні з завданнями, які можна проходити у довільному порядку, що може заплутати гравця і ввести в непорозуміння, що виконувати далі.

Для оптимізації такого ризику слід додати підказки, додаткове освітлення на місця карти, що потребують завершення, або створення індикатора процесу виконання всіх завдань.

Останній ризиком з даної категорії можна вважати навмисний дисбаланс гри: оскільки ігри розробляє людина з невеликим досвідом, то більша частина часу і сил приділяється функціоналу гри, а не забезпеченню безпеки, тому може виникнути ризик додавання «чітів» до гри гравцями, через що втрачається інтерес. Через малий досвід розробки ігор на Unity, розробник не зможе оптимізувати цей ризик. Але через простоту гри, цей ризик ігнорується.

3.5.2 Ризики пов’язані з розгортанням гри

По-перше, це неможливість запуску гри на непотужних/середніх машинах.

Невдосконалена архітектура гри, відсутність валідаторів, може призвести до підвисання гри, або використовувати більше ресурсів машини, ніж вона насправді потребує.

Як рішення: екстремальне тестування і навантаження гри може виявити значні недоліки, які можна ліквідувати, але через недостатність часу і досвіду розробника оптимізувати гру до кінця не вдасться.

Наступний ризик – мала цільова аудиторія. Гра розроблена для десктопів, а зараз більшість невеликих ігор створюються для мобільних пристроїв (більша аудиторія). Рішенням може стати спрощення процесу

установки і запуску гри для того щоб не втратити тих користувачів, хто буде стикатись з помилками встановлення гри на десктоп. Або ігнорування ризику, оскільки проект не є комерційним і не ставить за мету отримати фінансовий успіх. Основна ідея проекту навчитися розробляти ігрові застосування та накопичення досвіду.

3.5.3 Ризики, пов’язані з розробником

Якщо планується проект реалізувати як комерційний продукт, то тут постає загроза не завершення роботи відповідно плану. Обсяг роботи недооцінюється в рази, внаслідок чого ентузіазм закінчується раніше часу. Більша частина ідей і дизайну залишається нереалізованою. Доводиться жертвувати ідеями для того, щоб встигнути завершити роботу над основним функціоналом, від чого страждає загальний концепт гри і в результаті отримуємо не той результат на яких розраховували на початку.

Даний проект є некомерційним і розробляється у рамках навчальної програми, і тому основний важливий ресурс це час, а не гроші. Якщо після витрачення половини запланованого часу проект не має іграбельного прототипу, то гра потенційно може бути не завершена.

У такому разі може допомогти планування проекту, раціональний розподіл роботи над функціоналом. Встановлення дедлайнів, моніторинг прогресу за допомогою гнучких методологій розробки.

Неможливість реалізації функціоналу також відноситься до цього типу ризиків, тому що є можливість недостатності досвіду/знань/практики розробника. Як рішення: використання напрацювань, готових рішень, аналогів бажаного функціоналу, якщо це можливо. Якщо ні, спрощення реалізації зі збереженням загальної ідеї.

ВИСНОВКИ

У результаті було розроблено працездатний прототип гри з базовим набором головоломок.

Користувача мотивує грати в гру її зовнішня складова та також цікаві головоломки, які не такі прості, якими вони здаються на перший погляд, а потім посилюються з кожною секцією кораблю.

Користувач буде прагнути знайти рішення головоломки, адже небагато людей зможуть зміритися з тим, що гра їх подолала на звичайній головоломці, це буде породжувати нові спроби, що також сприяє покращенню логічного мислення. Крім цього, гравець буде прагнути побачити, які елементи та головки є в грі – це підкріпить внутрішню мотивацію гравця. У грі є багато елементів, якими можна користуватись, а також елементів, які є декором або скритою підказкою для того, щоб вирішити головоломку.

Фундаментальними правилами гри є те, що гравцю треба взаємодіяти з оточенням, вирішуючи головоломки та проявляючи кмітливість, якщо не слідкувати правилам гри то гравця буде очікувати рестарт, або переробка алгоритму пошуку рішення головоломки, все залежить від певної головоломки.

Звісно, в грі є свої закони та правила, за якими натиснення на ту, чи іншу кнопку має за собою певні наслідки, які будуть вести до різних варіацій рішення, наприклад, невірне вирішення головоломки буде сприяти тому, що головний герой загине або користувач буде думати як треба правильно вирішити головоломку, це додасть грі елементу планування та стратегії, що також покращить аналітичні та стратегічні здібності гравця.

В грі немає різних режимів та вибору складності, це зумовлено тим, що гра досить проста для проходження, але все залежить від аналітичного складу розуму користувача. Виконання правил контролюють механізми кораблю, який вийшов з ладу, через це користувачу треба знаходити правильні рішення та застосування приладів, які мають бути інтуїтивно зрозумілі, також елементи

керування розташовані на най розповсюджені кнопки, що не повинно викликати питання щодо подальшого кроку. Лише від гравця залежить результат, який отримає у кінці гравець.

В грі немає домінуючих стратегій, лише вірне вирішення головоломок, яке дасть змогу пройти далі та врятувати себе та корабель. Також стоїть відмітити на те, що уся гра супроводжується музикою та ігровими звуками при взаємодії з предметами, що додає грі живості, наприклад, у деяких моментах ігри звук змінюється, що буде натикати на думки, що щось не так.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ:

1. ARE 2D GAMES STILL POPULAR? URL: https://retrostylegames.com/blog/are-2d-games-still-popular/ (дата звернення 12.04.2023)
2. Самые популярные движки для создания игр. URL: https://vokigames.com/bazovyj-soft-dlya-sozdaniya-prostoj-mobilnoj-igry/ (дата звернення 12.04.2023)
3. How to Choose the Best Video Game Engine. URL: https://www.gamedesigning.org/career/video-game-engines/ (дата звернення 14.04.2023)
4. Unreal Engine . URL: https://habr.com/ru/hub/unreal_engine/ (дата звернення 14.04.2023)
5. Все, що потрібно для створення гри. URL: https://gamemaker.io (дата звернення 20.04.2023)
6. The Best Gaming Engines You Should Consider for 2023. URL: https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider (дата звернення 20.04.2023)
7. 10 Best Games Developed In Unity, Ranked. URL: https://screenrant.com/best-games-developed-unity-ranked/#hearthstone-2015 (дата звернення 20.04.2023)
8. Hollow Knight . URL: https://unity.com/madewith/hollow-knight (дата звернення 20.04.2023)
9. The Art of Game Design: A Book of Lenses, Third Edition 3rd Edition by Jesse Schell ISBN-13 978-1138632059