Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

СТВОРЕННЯ ГРИ НА UNITY/C# В ЖАНРІ ПАЗЛ-ПЛАТФОРМЕР

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2023 р.

Виконав студент _____
____Близнюк І.О._____
(прізвище та ініціали)
“____” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Близнюк Іван Олегович факультету інформатики 3-го курсу

ТЕМА Створення гри на Unity/C# в жанрі пазл-платформер

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Жанр пазл-платформер: історія та місце в індустрії
2 Використання технології збереження повторів для симуляції переміщень у часі
3 Реалізація
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2023 р. Борозенний О.С. _____ (підпис)
Завдання отримав _____ (підпис)

Тема: Створення гри на Unity/C# в жанрі пазл-платформер

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	06.09.2022	
2.	Аналіз матеріалів за темою	30.12.2023	
3.	Розробка та програмування алгоритму	08.01.2023	
4.	Написання текстової частини до курсової роботи	01.05.2023	
5.	Коригування виконаної роботи	08.05.2023	
6.	Створення слайдів для доповіді та написання доповіді.	10.05.2023	
7.	Остаточне оформлення роботи та слайдів	11.05.2023	
8.	Захист курсової роботи	25.05.2023	

Близнюк І. О. _____

Борозенний О. С. _____

“_____” _____

Анотація

Робота присвячена дослідженню можливостей рушія Unity з використанням мови C# на прикладі розробки гри в жанрі “пазл-платформер”. Гра містить елементи платформінгу та головоломок і має наступні функціональні можливості: переміщення персонажа за допомогою клавіатури, пересування та стрибки по платформах і сходах, переміщення у часі та взаємодія з об’єктами оточення.

Вступ

На сьогоднішній день відеоігри є чи не найпопулярнішим хобі в світі. Одним з найстаріших жанрів ігор є “платформер”, який нещодавно знову набув популярності, зокрема у поєднанні з іншими жанрами, як, наприклад, “пазл” або головоломка. Тому за мету роботу була поставлена розробка саме такого програмного продукту з додатковими механіками, такими як переміщення у часі, що реалізовано за допомогою технології збереження повторів.

Основна частина роботи складається з 3 розділів.

Перший розділ присвячено історії жанрів “пазл” та “платформер” та їх поєднанню

Другий розділ присвячено використанню технології збереження повторів для реалізації механіки переміщення у часі

Третій розділ присвячено реалізації програмного продукту інструментами рушія Unity та мови C#

Основна частина

Розділ 1: Жанр пазл-платформер: історія та місце в індустрії

1.1 Що таке “платформер”

Платформер або Платформна гра – це жанр відеоігор (що є піджанром жанру екшн), ігровий процес в якому складається з переміщення персонажа по різноманітним платформам (звідси й пішла назва) та через перешкоди. В якості перешкод можуть виступати як нерівності ландшафту, так і різноманітні вороги або інші джерела загрози.

Рис. 1 Гра Donkey Kong

Основоположником жанру вважається гра Donkey Kong[1] випущена у 1981 компанією Nintendo для аркадних автоматів. Вона першою дала можливість гравцям перестрибувати через перешкоди та прогалини. Гравець брав на себе роль відомого персонажа “Маріо”, який тоді ще називався просто “Jump Man” або чоловік-стребунець. На кожному рівні герою потрібно підніматися по драбинах на будівельному майданчику щоб врятувати принцесу з лап гігантської мавпи, що жбурляє в нього бочки. Гра стала дуже популярною

та породила серії ігор “Donkey Kong” та “Super Mario Bros”. Найбільшим недоліком гри було статичне положення камери.

Рис. 2 Гра Jungle King

Невдовзі, в середині 1982, цей недолік було прибрано у грі “Jungle King”[2], яка поєднувала платформинг та “Parallax-scrolling” або паралаксну прокрутку, що дало змогу ігровій камері рухатися разом з персонажем в горизонтальному напрямку. Такі ігри ще називають “сайд-скроллер” (від англ. Side – бік та scrolling - прокрутка).

Наступним логічним кроком став перехід жанру від 2х до 3х вимірної графіки, який почав відбуватись ще у середині 1980х, але через недостатній розвиток технологій вони в основному симулювали 3д перспективу.

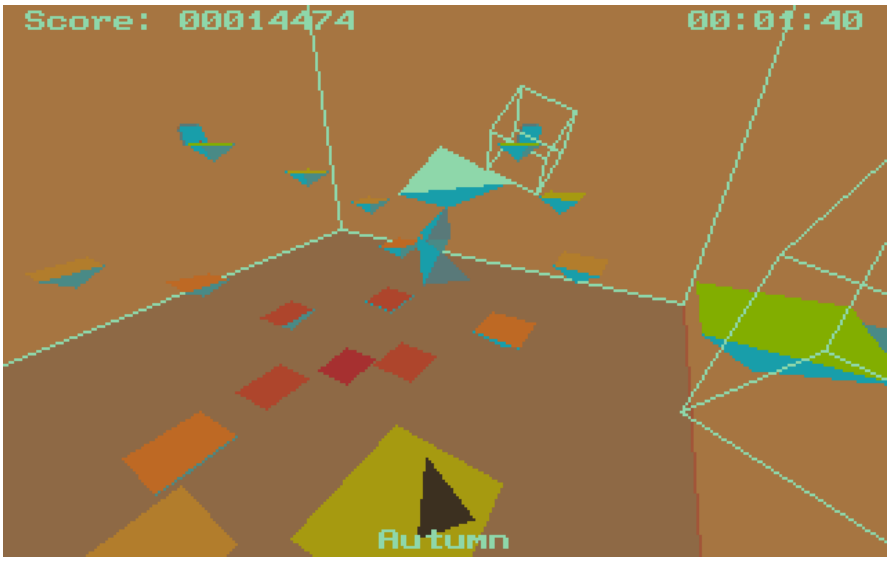

Рис. 3 Гра Alpha Waves

Першим платформером, що використовує справжній 3д простір вважають гру “Alpha Waves”[3],що вийшла у 1990 році де гравцю треба було стрибати по лабіринту з простих геометричних фігур.

З появою 3д проектів та збільшенням доступності обладнання, що їх підтримувало (консолі нових поколінь, більш потужні ПК), 2д ігри почали втрачати популярність.

Але нещодавно жанр знову почав набирати популярність в 2000х та 2010х зокрема через появу інді-ігор (ігор, що створюються невеликими командами без фінансової та технічної підтримки великого видавця).

1.2 Що таке “пазл”

Відеоігри-головоломки (або пазли) складають широкий жанр відеоігор, у яких основна увага приділяється розв’язуванню головоломок. Типи головоломок можуть перевіряти навички вирішення задач, зокрема логіку, розпізнавання образів, вирішення послідовності, просторове розпізнавання та завершення слів. Багато ігор-головоломок включають також елемент реального

часу та вимагають швидкого мислення, наприклад Tetris (1984) або Lemmings (1991).

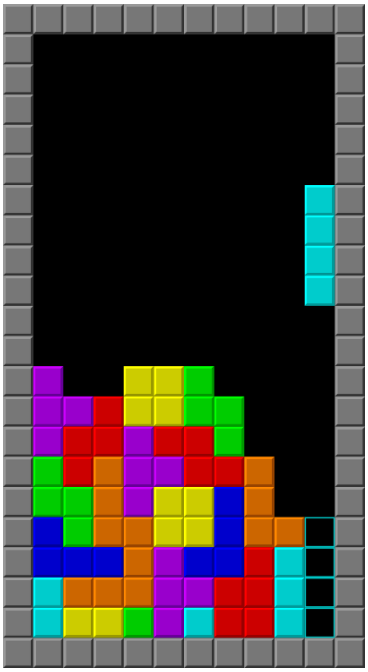

Рис. 4 Гра Tetris

Відеоігри-головоломки завдячують своїм походженням головоломкам і загадкам, що існували протягом всієї історії людства. Математична стратегічна гра Nim та інші традиційні ігри, що вимагають логічне мислення, такі як Шибеник і Бики та корови, були популярними цілями для комп’ютерної реалізації.

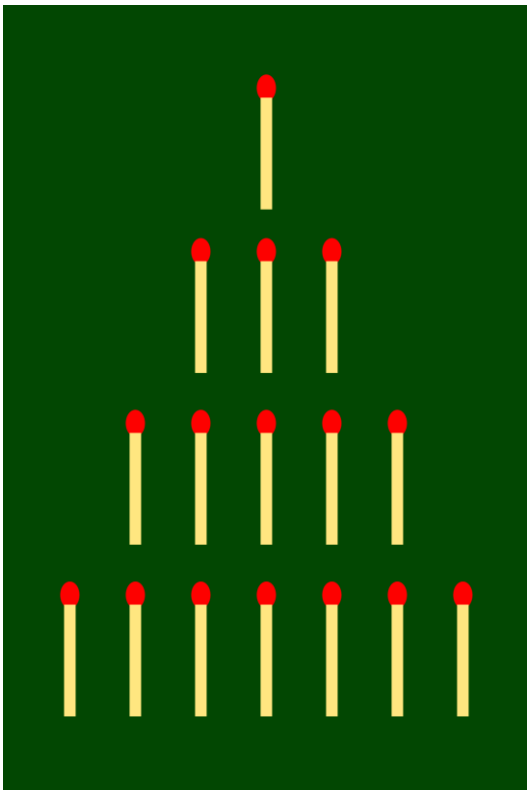

Рис. 5 Гра Nim

Серед сучасних ігор-головоломок виділяють наступні піджанри:

- Фізичні ігри – ігри, де гравець повинен використовувати ігрову фізику щоб проходити рівні.
- Кодингові ігри – ігри, що вимагають елементи програмування.
- Дослідницькі ігри – ігри, де гравець повинен експериментувати з різними механізмами на кожному рівні і вимагається скоріше індуктивний підхід, ніж логічний.
- Ігри з прихованим об’єктом – ігри, де гравець повинен знайти на екрані приховані об’єкти з певного списку.
- Ігри зі співставленням плиток – ігри, де гравець маніпулює плитками, щоб змусити їх зникнути відповідно до правил гри (наприклад, вишукувати 3 однакові в ряд)

1.3 Поєднання жанрів платформер та пазл

Пазл-платформери характеризуються використанням ігрової структури платформерів (присутність платформ, по яких гравець може стрибати та іншими способами переміщуватись), перешкоди якої в основному представляють собою різноманітні головоломки[4].

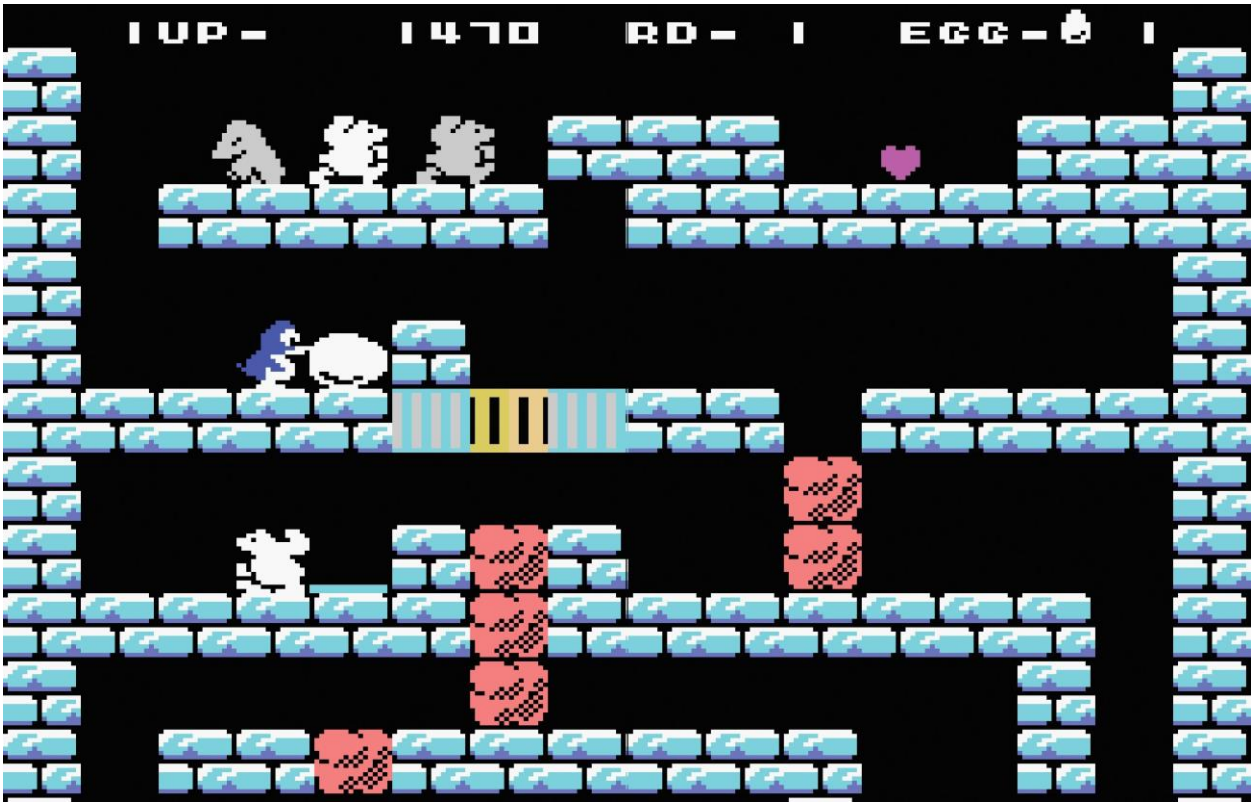

Рис. 6 Гра Doki Doki Penguin Land

Першим представником вважається “Doki Doki Penguin Land”[5] розроблена компанією Sega. В ній гравці могли бігати та стрибати в типовому платформінговому стилі, але також можна було ламати блоки, з яких складався ігровий світ. Основною задачею гравців було донести яйце до низу рівня, при цьому не розбивши його.

Багато представників цього піджанру виходили на портативні консолі завдяки зазвичай простій графіці, що вимагає небагато ресурсів.

Як і весь жанр платформер, пазл-платформери набули нової хвилі популярності на початку 2000х з виходом таких проектів як Portal, Portal 2 або The Entropy Centre[6].

Розділ 2: Використання технології збереження повторів для симуляції переміщень у часі

2.1 Повтори в іграх

2.1.1 Загальні відомості про повтори

Повтор гри, також відомий як демо, є формою користувацького контенту. У більшості випадків повтор гри – це запис битви чи перегонів між суперниками у відеогрі, який потім можуть переглядати інші гравці. Деякі ігри, такі як TrackMania, Doom[7] і N, використовують файли повторів, що є дуже ефективними за розміром, які записують вхідні дані, щоб мати можливість відтворити ігрову ситуацію без втрат.

Повтори найчастіше зустрічаються в стратегіях у реальному часі, таких як StarCraft, Command & Conquer, World in Conflict, Company of Heroes і Age of Empires, а також у деяких шутерах від першої особи, таких як Counter-Strike[8]. Існують веб-сайти, що дозволяють користувачам завантажувати повтори певних ігор, щоб інші гравці могли завантажувати та переглядати їх задля розваги чи вдосконалення власних навичок.

Існує декілька найпоширеніших способів створення повторів, які можна розділити на дві категорії: повтори у рушії та повтори з використанням сторонніх ресурсів[9].

2.1.2 Повтори з використанням сторонніх ресурсів

До повторів з використанням сторонніх ресурсів можна віднести застосунки для запису екрану, такі як Fraps, Bandicam, OBS або ShadowPlay. Вони дозволяють записувати все, що відбувається на екрані та конвертувати це у відеофайли, також є можливість записувати ігрове аудіо та навіть звук з мікрофона користувача.

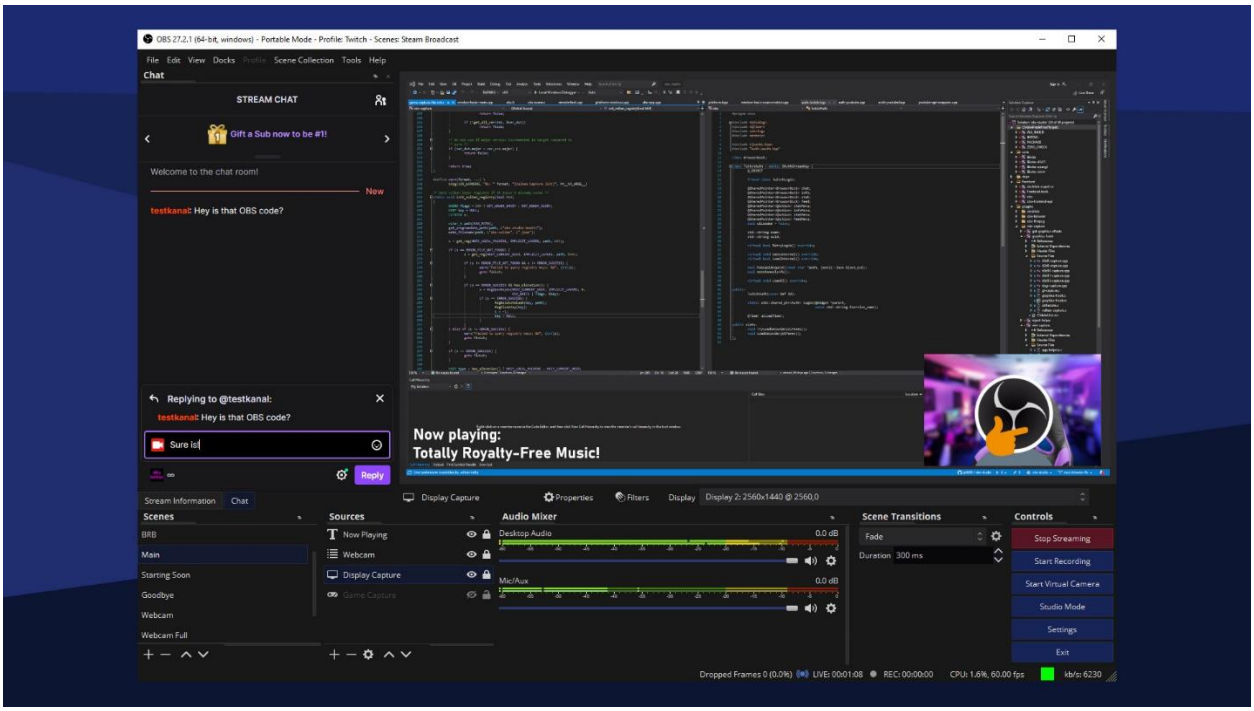

Рис. 7 Інтерфейс програми OBS Studio

Основною перевагою такого підходу є універсальність, адже використовуючи один і той самий застосунок можна записувати ігровий процес будь-якої гри. Також для перегляду записаних відеороликів не потрібно додаткових інструментів, адже вони зберігаються у стандартних відео форматах (найчастіше – MP4, MOV або AVI) і їх навіть можна завантажити на сервіси відеохостингу (такі як youtube).

Але такий підхід містить декілька вагомих недоліків, серед яких:

- Великий розмір вихідних файлів, адже 1 хвилина відео в роздільній здатності full hd займає в середньому 205 мегабайтів пам’яті[10]
- Велике навантаження на систему, адже окрім відображення зображення на екрані, воно повинно паралельно зберігатись на диск і процес має відбуватись у реальному часі, чого може бути складно досягнути, бо сучасні відеоігри зазвичай по максимуму використовують ресурси системи.

- Відсутність можливості розглянути ігрову ситуацію загалом. Оскільки відеозапис містить лише інформацію про зображення на екрані гравця, немає можливості відновити стан об’єктів, що не потрапляють в його поле зору, або оглянути те, що відбувається з іншого ракурсу (а не від лиця гравця, що робив запис).

2.1.3 Повтори у рушії

Натомість, до реалізації повторів у рушії полягає в використанні ігрового рушія, на якому працює гра для повторного відображення ігрової ситуації. Такий підхід дозволяє розробникам використовувати власні пропрієтарні формати для збереження даних, що дозволяє значно зменшити розмір файлів повтору (наприклад, у оригінальному Doom 1 хвилина повтору займала 8.4 кілобайти[11]), але при такому підході переглядати повтори можна лише у рушії гри (звідси і походить його назва). Також оскільки інформація відображається прямо в рушії, частота оновлення має бути постійною, адже якщо під час запису повтору кількість кадрів на секунду одна, а під час відображення – інша, повтор може виглядати сповільненим або прискореним, або взагалі відображатись некоректно.

2.1.4 Порівняння способів реалізації повторів у рушії

До реалізації повторів у рушії є два основні підходи[12]:

1. Збереження стану.

   При цьому підході зберігається стан кожного ігрового об’єкта (положення в просторі, поворот, кадр анімації, тощо) кожну одиницю часу. Такий підхід породжує більші за розміром файли, але дозволяє переглядати повтор починаючи з будь-якого моменту часу. Для цього лише потрібно завантажити стан всіх об’єктів з файлу повтору та

змінювати його кожну одиницю часу відповідно до збереженої інформації.

При використанні даного метода, кожний кадр маємо зберігати позицію гравця, номер та кадр його поточної анімації (поворот гравця зафіксований, тому його не треба зберігати) та позицію та поворот інтерактивних об’єктів (наприклад, ящиків, з якими треба взаємодіяти).

В Unity, для збереження позиції використовується структура Vector3, яка являє собою 3 числа з плаваючою точкою (тип float), але оскільки гра двовимірна, можна зберігати лише дві координати (для цього можна використати Vector2), що вимагає 2*4=8 байтів пам’яті на кожний кадр (оскільки тип float займає 4 байти в C#), плюс по одному байту на збереження номеру анімації та кадру (теоретично, можна було б об’єднати ці два значення в один байт, але тоді всі анімації мали б бути коротші за 16 кадрів та самих анімацій мало б бути не більше за 16). Тобто маємо 10 байтів на кадр існування гравця.

Для інтерактивних об’єктів треба зберігати позицію, тобто 8 байтів інформації та поворот. Поворот в Unity описується структурою Quaternion (кватерніон), який складається з 4 чисел типу float, але оскілки об’єкти обертаються лише навколо однієї осі, яка напрямлена перпендикулярно до площини, в якій проходить гра (в Unity – вісь Z), можна зберігати лише один кут (тобто, одне число з плаваючою точкою), тому всього витрачаємо 12 байтів місця на кадр.

2. Збереження введень.

При цьому підході зберігається початковий стан ігрових об’єктів та кожну одиницю часу зберігаються введення гравця (натиснуті кнопки та рухи миші). Таким чином, під час повтору, рушій емулює поведінку

гравця, повторюючи записані введення, що приводить до точного відтворення ігрової ситуації. Але для правильної роботи, ігрова логіка має бути повністю детермінованою, тобто однакові дії гравця повинні призводити до одного й того самого результату. Але ця вимога не виключає використання елементу випадковості у грі, адже можна використати генератор псевдовипадкових чисел та зберігати в файлі повтору насіння (seed), що використовується для задання випадкової послідовності. При такому підході, файли повтору мають найменший обсяг, бо треба зберігати лише початковий стан ігрових об’єктів та дії користувача, які доволі легко піддаються стисненню.

Оскільки стан кожної клавіші у певний момент часу є бінарним (вона або натиснута, або ні), дії гравця можна описати набором таких бінарних значень. Всього дій, які записуються для повтору 5: рух вліво, вправо та вниз, стрибки, та взаємодія (відповідає за підбирання або відпускання об’єктів, таких як коробки). Ще гравець може переміщуватись у часі, та видаляти існуючі копії, але ці дії недоступні самим копіям, тому не зберігаються.

Оскільки маємо декілька бінарних значень, для їх збереження можна використати пакування бітів (bit packing[13]) звести до присвоєння відповідним бітам в байті значень 1 якщо кнопка натиснута та 0, якщо ні. Так як дій всього 5, дії гравця на кожному кадрі можна описати 1 байтом інформації (теоретично, можна і менше, але використання саме 1 байта в 1 кадр дає простір для додавання нових дій у майбутньому та сильно полегшує процес стиснення та розтиснення інформації). Таким чином, враховуючи, що рушій Unity за замовченням використовую частоту 50 оновлень на секунду для

прорахунку фізики, кожна копія займає 50 байтів оперативної пам’яті на секунду свого існування.

На відміну від першого способу, інформація про інтерактивні об’єкти та анімації не зберігається зовсім, адже може бути відтворена відповідно до збережених введень гравця.

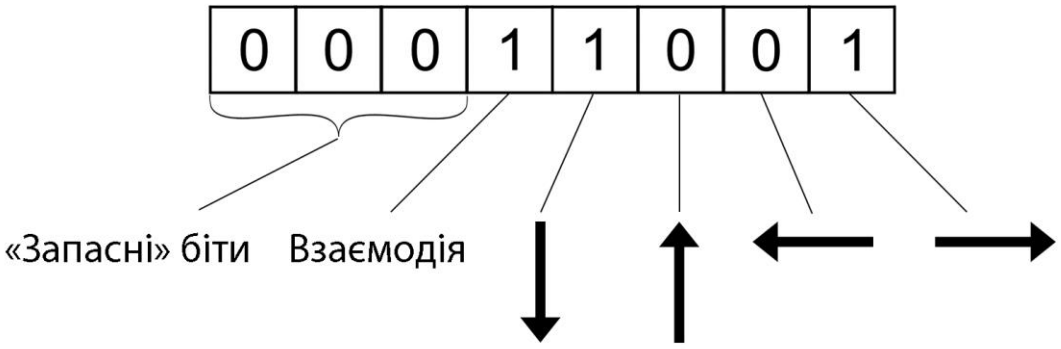

Рис. 8 Схема пакування бітів, що використовується у грі

Таким чином, при використанні збереження введень, маємо виграш по пам’яті більш ніж у 10 разів (точне число залежить від кількості інтерактивних об’єктів, адже при збереженні введені, інформація про них зовсім не зберігається і тому не займає місця).

2.2 Симуляція переміщень у часі

Однією з основних ігрових механік моєї гри є можливість головної героїні переміщуватись назад у часі до моменту входу у кімнату, що створює 2 її копії: ту, що увійшла до кімнати першого разу, та повторює дії гравця до

моменту переміщення у часі й та, яка вже перемістилася в часі і якою керує гравець. Якщо повторно переміщуватись у часі, копій може бути й більше двох.

Оскільки копії повністю відтворюють дії гравця, для реалізації даної механіки було використано технологію збереження повторів у рушії, зокрема збереження введень. Такий підхід є оптимальним у даному випадку, адже інформація про копії (фактично, повтори) зберігається в оперативній пам'яті, тому має займати мінімальну кількість місця, у грі не використовується генерація випадкових чисел, запис та відображення запису відбувається на одній машині (тому не виникає проблем з, наприклад, операціями з числами з плаваючою точкою, які можуть давати різні результати для однакових вхідних даних на різних операційних системах), також копії повторюють дії гравця від початку до кінця і не потребують перемотування. Таким чином, всі дії гравця (в даному випадку, всі натиснуті клавіші управління, адже у грі не передбачається використання миші) записуються до моменту переміщення в часі, після цього створюється копія головної героїні, яка вже не керується з клавіатури, а використовує збереженні введення, щоб в точності відтворити дії гравця, а справжня головна героїня (якою керує гравець) переміщується у точку входу у кімнату.

Для зручності гравця були реалізовані наступні особливості повторів:

- Якщо у копії закінчуються дії, що вона повинна робити ця копія просто завмирає на місці, хоча мала б зникати, адже мала б повторити переміщення гравця в часі. Така поведінка була реалізована для збільшення динамічності гри, бо дуже часто копії використовуються для того, щоб активувати натискні пластини своїм тілом аби відкрити гравцю шлях до виходу і якщо б вони зникали, гравцю б доводилось завмирати на декілька секунд перед створенням копії, аби вихід був відкритий якомога довше.

- Копії візуально відрізняються від головної героїні, вони пофарбовані в інший колір та напівпрозорі, аби гравцю було легше вирізняти, де знаходиться та версія головної героїні, якою він керує.
- Кількість копій обмежена та може відрізнятись у кожній кімнаті, аби складніше було проходити загадки «в лоб», використовуючи більшу кількість копій, ніж задуману дизайнером рівнів.

Розділ 3: Реалізація

3.1 Огляд основних принципів Unity

Unity – сучасний ігровий рушій, що активно використовується як великими студіями, так і незалежними розробниками, адже містить в собі цілу низку корисних та потужних інструментів для ігрової розробки.

Сам рушій написаний мовою C++[14], але для розробки на ньому використовується в основному C#, бо він є значно простішим у використанні та підтримує збирача сміття.

Рис. 9 Зовнішній вигляд Unity Editor

Розробка на Unity ґрунтується на трьох основних складових[15]:

- Game Object – контейнер для компонентів (Components). Будь який об’єкт (головний герой, куб, джерело світла і т.д.) на сцені є екземпляром цього класу. Кожний GameObject може мати дочірні об’єкти (тобто інші об’єкти, які є його складовими) та батьківський об’єкт (для якого сам є дочірнім).
- Компоненти (Components) – складові, що визначають поведінку ігрового об’єкта (Game Object). Їх можна вільно додавати та видаляти щоб змінити цю поведінку (Таким чином, щоб додати об’єкту можливість рухатись під дією сили тяжіння можна додати до нього компонент RigidBody, який відповідає за поведінку твердих тіл, а щоб заставити його світитись можна додати компонент Light).
- Змінні – компоненти зазвичай містять набір властивостей(особливий механізм мови C#, що поєднує в собі геттер та сеттер для певного

приватного поля[16]) або змінних, який можна редагувати прямо в графічному інтерфейсі рушія (В попередньому прикладі такими змінними можуть виступати маса об’єкта або яскравість світла, що він випромінює).

Незважаючи на те, що в Unity є багато готових компонентів під різні задачі, зазвичай лише їх недостатньо для реалізації геймплейних особливостей конкретної гри. Для вирішення цієї проблеми у нагоді стають скрипти.

Скрипти – C# класи, що наслідуються від вбудованого MonoBehaviour[17], які в подальшому можна додавати до ігрових об’єктів аби вплинути на їх поведінку за допомогою коду на C# (так само, як і звичайні компоненти, тому ці назви часто взаємозаміняють). У скриптах можуть як визначати власні методи, які будуть викликатись іншими скриптами, так і використовувати стандартні методи MonoBehaviour, які викликаються у певних ситуаціях. Наприклад, метод Start() викликається у момент створення об’єкту, а метод OnCollisionEnter(Collision collision) – коли об’єкт вступає у колізію, тобто фізично зіштовхується, з іншим об’єктом.

Оскільки багато ігрових об’єктів зазвичай є типовими та часто використовуються повторно в Unity можна створювати префаби (Prefabs[18]), тобто заздалегідь налаштовані та збережені ігрові об’єкти (включаючи список компнентів, їх змінних та дочірніх об’єктів), які потім можна як використовувати в інших сценах, так і створювати їх копії прямо в процесі гри.

Також, якщо потрібно зберегти просто набір даних, а не цілий ігровий об’єкт, можна використати скриптовий об’єкт (ScriptableObject[19]), який є контейнером для збереження даних, та з яким можна зручно працювати у графічному інтерфейсі рушія. Їх можна використовувати для зменшення об’єму оперативної пам’яті, що використовує гра. Якщо маємо наприклад, головну героїню та її копії, які мають незмінні числові характеристики (швидкість

переміщення, швидкість стрибка, тощо) замість того, щоб копіювати їх для кожної копії або використовувати статику, можна створити скриптовий об’єкт з їх характеристиками та дати головній героїні та копіям посилання на нього (що можна зробити прямо у інтерфейсі Unity).

3.2 Опис програмного продукту

3.2.1 Основні механіки

- Переміщення – головна героїня має можливість переміщуватись по горизонталі, підніматись та спускатись сходами, стрибати вверх та вниз крізь відповідні платформи.
- Взаємодія з навколишніми об’єктами – героїня може ставати на натискні пластини аби активувати різноманітні двері, ліфти або силові поля. Також можна підбирати та пересувати деякі коробки аби використати при розв’язуванні головоломок.
- Переміщення у часі – головна героїня може переміщуватись у часі до моменту входу в кімнату (тобто, початку головоломки), що створює її копію, яка в точності відтворює всі її дії з останньої «спроби» (тобто, до моменту переносу).
- Система кімнат – кожна головоломка представляє собою кімнату, в якій необхідно виконати певні дії, аби дістатись до виходу, після досягнення якого гравець переноситься у наступну кімнату.

3.2.2 Ієрархія сцени

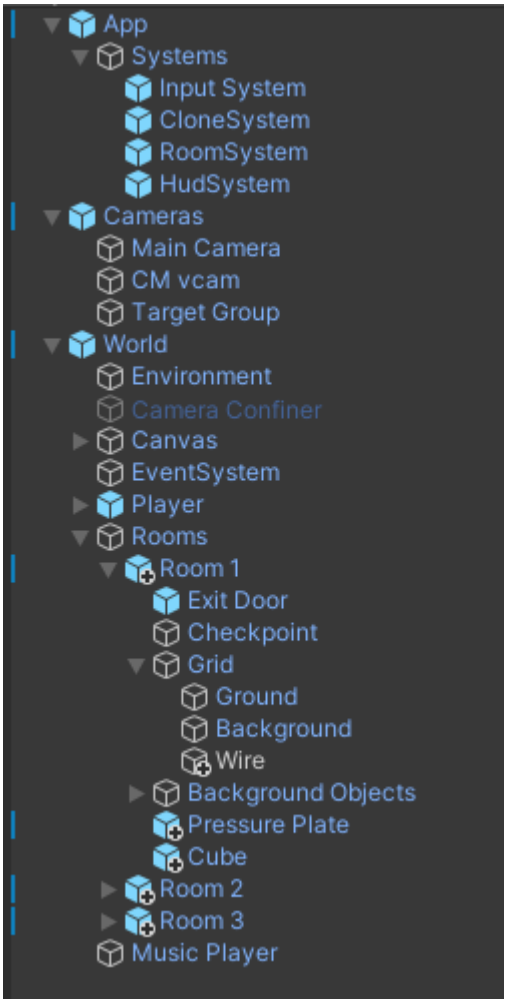

Рис. 10 Ієрархія сцени

Всі сцени в грі побудовані типовим чином, аби розробка нових рівнів займала мінімальну кількість часу.

В корені сцени знаходяться об’єкти App, World та Cameras.

- App – об’єкт, що містить посилання на об’єкти основних систем та виконує реалізацію патерну Dependency Injection[20] (далі - DI).
- Cameras – пустий об’єкт, що містить у собі камери та все, що потрібно для їх роботи.

- World – об’єкт, що містить у собі все, що фізично знаходиться у ігровому світі та містить компонент Objects Container, який зберігає посилання на важливі об’єкти сцени (гравця, кімнати тощо).

В свою чергу, об’єкт World містить в собі такі основні дочірні об’єкти:

- Canvas – графічний компонент, який використовується для відображення користувацького інтерфейсу[21] (UI).
- Player – об’єкт гравця, який містить у собі його візуальне представлення, відповідні компоненти та коллайдери для взаємодії з навколишнім середовищем.
- Rooms – містить кімнати, кожна з яких представляє собою головоломку із входом, виходом та інтерактивними об’єктами.
- Music Player – об’єкт, який відповідає за програвання музики.

3.2.3 Опис основних систем

Як вже було описано раніше, в проекті використовується патерн DI, який спрощує встановлення зв’язків між об’єктами. За його реалізацію відповідає компонент App, який ініціалізує всі системи та дає їм всі потрібні посилання одна на одну та на потрібні об’єкти світу, що зберігаються всередині Objects Container.

Список основних систем:

- Input System - система, що відповідає за обробку натискань клавіш гравцем (далі - інпути) та передає їх контроллеру гравця та системі копій (Clone System).
- Clone System – система, що відповідальна за роботу з копіями головної героїні. Отримує інпути гравця та зберігає їх у масив байтів, використовуючи кожний біт для задання того, чи натиснута відповідна клавіша на кожному кадрі. Коли приходить команда про створення копії,

всі вже інсуючі копії повертаються до початкового стану, створюється новий, який починає використовувати сформований масив інпутів, після чого створюється новий масив.

- Room System – система кімнат, яка зберігає в собі масив кімнат, що присутні на рівні та відповідає за перенесення головної героїні між ними.
- Hud System – система, що відповідає за відображення елементів інтерфейсу на екрані, таких як кількість доступних копій або затемнення екрану при переході між кімнатами.

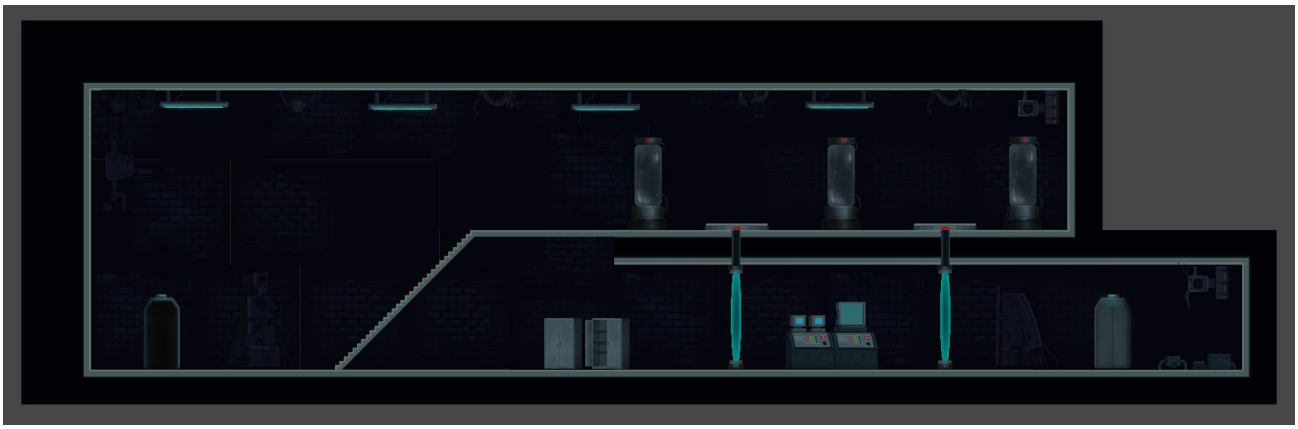

Рис. 11 Приклад кімнати

Рис. 12 Головна героїня з двома копіями

Також, варто виокремити схему роботи інтерактивних об’єктів. Вони поділяються на дві категорії:

- Активатори – об'єкти, що можуть активувати інші об'єкти (наприклад, натискні пластини). Містять список об’єктів з якими зв’язані (тобто, одна натискна пластина може відкривати двері до виходу, активувати ліфт та вимикати силове поле).
- Активовані об’єкти – об’єкти, стан яких змінюється, якщо увімкнути зв’язаний з ним активатор (наприклад, силові поля, ліфти або двері для виходу з кімнати). Можуть бути поєднані з декількома активаторами, тоді перейде в активний стан лише якщо всі активатори будуть увімкнуті.

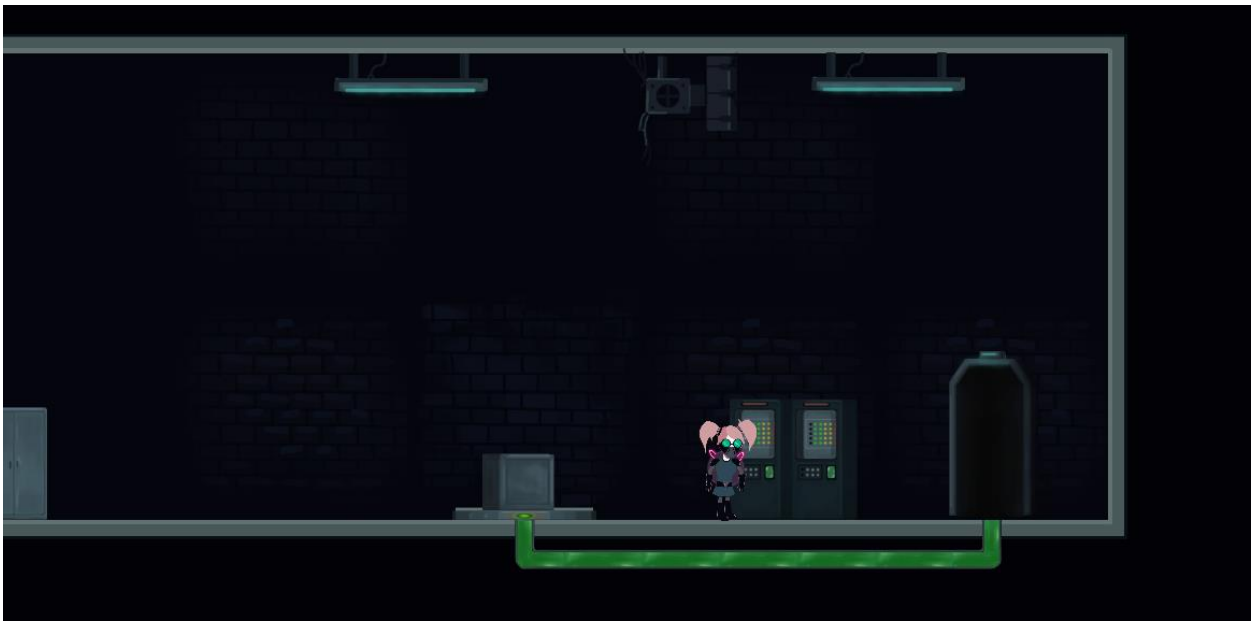

Рис. 13 Приклад використання інтерактивних об'єктів

Висновок

У процесі розробки гри було досліджено види повторів в відеоіграх, особливості жанру пазл-платформер та ігровий рушій Unity як інструмент для реалізації.

Unity надає сучасний та гнучкий інструментарій для розробки ігор будь-яких жанрів, що значно спрощує роботу розробників. Але оптимізація та використання ефективних алгоритмів все-ще є дуже важливою складовою ігрової розробки. Тому у даній роботі було досліджено використання технології створення повторів, яка стала у нагоді для реалізації механіки переміщення у часі.

Як результат, було створено повноцінну гру в жанрі «пазл-платформер», у якій реалізовано як стандартні для жанру механіки, такі як рух та стрибки по платформах, рух крізь платформи, взаємодія з інтерактивними об’єктами, анімації, звуки, та збереження стану гри, так і нестандартні рішення, що вирізняють продукт серед вже існуючих. Такою механікою є можливість переміщуватись у часі, що є результатом нестандартного використання технології збереження повторів.

Список літератури

[1] https://archive.org/details/Electronic_Games_Volume_01_Number_11_1983-01_Reese_Communications_US/page/n53/mode/2up?view=theater

[2] https://retrovolve.com/jungle-hunt-was-a-terrible-waste-of-quarters/

[3] https://grenouillebouillie.wordpress.com/2013/01/18/is-it-worth-disputing-the-title-of-first-3d-game-on-a-pc-to-john-carmack/

[4] https://web.archive.org/web/20190109062923/http://robsretroreviews.blogspot.com/2016/07/17-sub-genres-of-platformer-games-list.html

[5] http://www.hardcoregaming101.net/doki-doki-penguin-land/

[6] https://gamerant.com/puzzle-games-impossible-complete/#portal-2

[7] http://web.archive.org/web/20071211231114/http://demospecs.planetquake.gamespy.com/lmp/lmp.html

[8] https://afkgaming.com/csgo/guide/7939-csgo-replay-controls-for-demo-mode-what-are-they-how-do-they-work-detailed-guide

[9]https://www.gamedeveloper.com/programming/implementing-a-replay-system-in-unity-and-how-i-d-do-it-differently-next-time

[10] https://web.library.uq.edu.au/library-services/it/learnuq-blackboard-help/learnuq-assessment/eportfolio/video-content-recommendations

[11]https://doom.fandom.com/wiki/Demo#Technical_information

[12]https://www.gamedeveloper.com/design/things-that-can-muddle-your-replay-feature#close-modal

[13] https://kinematicsoup.com/news/2016/9/6/data-compression-bit-packing-101

[14] https://docs.unity3d.com/Manual/unity-architecture.html

[15] https://unity.com/how-to/programming-unity

[16] https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties

[17] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[18] https://docs.unity3d.com/Manual/Prefabs.html

[19] https://docs.unity3d.com/Manual/class-ScriptableObject.html

[20] http://w3sdesign.com/?gr=u01&ugr=proble#gf

[21] https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html