КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-20

Крупа З.А.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Крупа Захар Анатолійович

УДК 004.4

Розробка 3D-гри типу Top-Down на ігровому двигуні Unity

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавр

Нормоконтроль	Студент
_____Стисло О.В. (підпис, дата, розшифрування підпису)	_____Крупа З.А. (підпис, дата, розшифрування підпису)
Допускається до захисту Завідувач кафедри	Керівник роботи
_____к.т.н., доц. Ващишак С.П. (підпис, дата, розшифрування підпису)	_____к.т.н., доц. Слабінога М.О (підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Крупі Захару Анатолійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Розробка 3D-гри типу Top-Down на ігровому двигуні Unity

керівник роботи:
Слабінога Мар'ян Остапович, к.т.н., доцент

затверджена наказом вищого навчального закладу від « 12 » березня 2024 року № 19/1

2. Термін подання студентом роботи 05.06.2024

3. Вихідні дані роботи: ігровий двигун Unity, мова програмування C#, програмний пакет Blender, середовище редагування коду Visual Studio

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Огляд предметної області
   2. Розробка структури
   3. Реалізація та тестування системи

5. Дата видачі завдання 14.03.2024

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд предметної області	20.03.2024	Виконано
2.	Огляд інсуючих рішень	28.03.2024	Виконано
3.	Розробка загальної структури проекту	13.04.2024	Виконано
4.	Розробка гри, оформлення локації, тестування	27.04.2024	Виконано
5.	Формування висновків	29.05.2024	Виконано
6.	Оформлення пояснювальної записки	02.06.2024	Виконано
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	15.06.2024	Виконано

Студент _____ Крупа З.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Слабінога М.О.
(підпис) (прізвище та ініціали)

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

28	Вміст каталогу Prefabs	50	Тестування генерації системних блоків та взаємодії з ними
29	Вміст каталогу ScriptableObjects	50	Тестування відображення поміщених компонентів
30	Вміст каталогу Scripts		

АНОТАЦІЯ

В даній кваліфікаційній роботі описується процес розробки гри з видом зверху на двигуні Unity. Робота містить у собі три розділи, які охоплюють різні аспекти проекту.

У вступі пояснюється актуальність обраної теми та вказується мета і завдання роботи.

Перший розділ представляє собою огляд предметної області. Він містить у собі загальну інформацію про ігри, жанр Top-Down, історію розвитку даного жанру та особливості. Розглядаються та аналізуються уже наявні популярні рішення подібного або такого ж самого жанру. Аналізуючи дані рішення, створено власні задачі для розробки гри.

У другому розділі показано розробку структури гри. Було розглянуто засоби та їхні переваги та технології, а саме такі як ігровий двигун Unity, мову програмування C#, програмний пакет Blender. Описується загальна структура системи, компоненти які входять в проект, а також модулі та їх взаємодія.

Третій розділ містить у собі опис реалізації розробленої гри. Показано етапи тестування різних функцій та механік, інтерфейс користувача та інших компонентів.

У висновку усі результати розробки підсумовуються, проводиться оцінка поставлених задач та можливості для вдосконалення проекту.

У списку використаних джерел вказаний перелік літератури, інших ресурсів чи матеріалів, які були застосовані у процесі розробки та написання кваліфікаційної роботи.

КЛЮЧОВІ СЛОВА: 3-D ГРА, ВИД ЗВЕРХУ, UNITY, РОЗРОБКА ІГОР, BLENDER, C#.

ANNOTATION

This qualification paper describes the process of developing a top-down game using the Unity engine. The work includes three chapters covering different aspects of the project.

The introduction explains the relevance of the chosen topic and indicates the purpose and objectives of the work.

The first chapter is an overview of the subject area. It contains general information about games, the Top-Down genre, the history of this genre, and its features. It also analyzes the existing popular solutions of a similar or the same genre. Analyzing these solutions, we create our own tasks for game development.

The second section shows the development of the game structure. The tools and their advantages and technologies were considered, namely, the Unity game engine, the C# programming language, and the Blender software package. The general structure of the system, the components included in the project, as well as the modules and their interaction are described.

The third chapter describes the implementation of the developed game. It shows the stages of testing various functions and mechanics, the user interface and other components.

In the conclusion, all the results of the development are summarized, the tasks set and the possibilities for improving the project are evaluated.

The list of references contains a list of literature, other resources or materials that were used in the process of developing and writing the qualification work.

KEYWORDS: 3-D GAME, TOP VIEW, UNITY, GAME DEVELOPMENT, BLENDER, C#.

ЗМІСТ

ВСТУП

Актуальність теми. Розробка гри про збирання комп'ютерів на Unity несе в собі значну цінність, адже вона не лише розважає, а й навчає. Граючи, користувачі знайомляться з основними комп'ютерними комплектуючими, складанням, що робить її корисною як для новачків, так і для досвідчених користувачів. Цей ігровий процес може стати захоплюючим способом вивчення комп'ютерів, роблячи навчання цікавим та практичним.

Мета і завдання дослідження. Метою роботи є розробка 3D гри з виглядом зверху на базі ігрового двигуна Unity. Мета передбачала виконання наступних завдань:

- аналіз предметної області;
- аналіз аналогічних ігор даного жанру;
- проектування загальної структури системи;
- реалізація гри;
- тестування.

Об’єкт дослідження. Процес комп’ютерної симуляції з використанням ігрових двигунів.

Предмет дослідження. Методи та інструменти створення 3D ігор з виглядом зверху.

Методи дослідження. Для досягнення мети дослідження та вирішення поставлених завдань у роботі застосовувалися наступні методи дослідження:

- аналіз;
- симуляція;
- моделювання;
- програмування;
- тестування.

Практичне значення одержаних результатів. Результат цього дослідження полягає у створенні 3D гри з виглядом зверху, що дало можливість створити інструмент для організації дозвілля користувачів.

Апробація результатів дослідження. Апробація результатів дослідження.(напівжирний) Матеріали кваліфікаційної роботи були представлені на XI Міжнародній науковій конференції «Студентські наукові дискусії поза форматом», яка відбулася 11 квітня 2024 року в Університеті Короля Данила.

Структура. Дана робота складається з трьох основних розділів, перший з яких – це огляд предметної області, другий – розробка структури, третій – реалізація та тестування системи. Загальний обсяг основної частини – 41 сторінок. Список використаних джерел – 20.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд предмету області

TechBuild – це інтерактивна 3D гра з видом зверху, яка навчатиме гравців вибору, комбінуванню та оптимізації компонентів для створення “ідеальних” комп’ютерів, відповідно до зазначених замовлень. У теперішній час ігри присутні в житті майже кожної людини. Їх використовують як для роботи так і для навчання, спілкування та багато чого іншого. Тому завжди буде потреба в людях, які розуміють принципи роботи комп’ютерів, володіють навичками їх збирання та налаштування.

Якщо враховувати початкову та саму головну мету ігор, то без вагань можна сказати, що ігри створюються для розваг, адже ігри дають можливість розслабитися, відволіктися від повсякденних проблем і отримати задоволення.

Навчання, також невід’ємна частина нашого життя, в цьому плані ігри також не відстають, ігри можуть бути потужним інструментом для навчання. Вони можуть використовуватися для викладання різних предметів, таких як математика, історія, наука, або розвиток навичок, таких як логічне мислення, вирішення проблем, координація та стратегічне планування або навіть вивчення коду на певній мові програмування.

Традиційні методи навчання основам складання комп’ютерів, такі як лекції, практичні заняття та читання книг, можуть бути не ефективними для деяких людей. Оскільки ці методи можуть бути складними, нудними для розуміння, або нема можливості практикувати навички в реальному середовищі.

TechBuild може запропонувати інноваційний та захоплюючий спосіб навчання основам складання комп’ютерів. За допомогою ігрового процесу, гра робить навчання цікавим та доступним для людей з різним рівнем досвіду.

Застосування:

– навчання школярів та студентів основам складання комп'ютерів. Гра може бути інтегрована в шкільну програму з інформатики або використовуватися як додатковий ресурс для самостійного навчання;

– підготовки користувачів, які бажають самостійно навчитися збирати комп'ютер. TechBuild може допомогти їм уникнути помилок і зробити процес складання комп'ютера цікавішим;

– підвищення кваліфікації співробітників сервісних центрів та магазинів комп'ютерної техніки. Гра може допомогти їм покращити свої знання та навички збирання та налаштування комп'ютерів.

Ігри можуть бути чудовим способом спілкування з друзями, сім’єю та людьми з усього світу. Вони можуть дати людям можливість познайомитися з новими людьми, знайти спільні інтереси та співпрацювати для досягнення спільної мети.

Ігри також можуть виступати у ролі терапії для людей з різними проблемами, такими як тривога, депресія, аутизм, PTSD. Ігри можуть допомагати розслабитися, емоційно регулюватися, розвивати соціальні навички та покращити когнітивні функції.

Для творчих людей ігри можуть виступати у ролі мистецтва. Вони можуть використовуватися для розповіді історій, вираження емоцій, дослідження складних тем і створення красивих візуальних і звукових пейзажів.

Для дослідників також знайдеться місце, адже ігри можуть використовуватись для дослідження різних тем, таких як поведінка людей, штучний інтелект, економіка та соціальні проблеми. Вони можуть допомогти дослідникам зрозуміти складні системи, протестувати теорії та збирати дані.

Якщо розглядати ігри як бізнес, то і тут є чудові можливості для просування продуктів і послуг, навчання співробітників, підвищення продуктивності та покращення обслуговування клієнтів.

Потреби користувачів у сучасних іграх:

Захоплюючий та динамічний ігровий процес: Гравці хочуть, щоб ігри були захоплюючими та динамічними, щоб їм було не нудно.

Важливість хорошого ігрового процесу:

- цікавий сюжет: сюжет має бути захоплюючим, щоб гравцям було цікаво дізнатися, що буде далі. Якщо ігровий процес буде недостатньо захоплюючим, гравці швидко втратять інтерес;
- складні завдання: завдання повинні бути досить складними, щоб кинути виклик гравцям, але складними в міру, так як вони можуть здатись користувачам неможливими, що викличе у них негативні емоції.

Як приклад можемо виділити таку гру як The Legend of Zelda: Breath of the Wild:

The Legend of Zelda: Breath of the Wild – гра пропонує гравцям один з найбільших відкритих світів, який можна досліджувати будь-яким способом.

Високоякісна графіка та візуальні ефекти:

Гравці очікують від гри приємну для очей графіку. Розробники повинні вивчати та використовувати сучасні технології та інструменти для створення наступних пунктів:

- деталізоване середовище: навіть з обмеженої перспективи, світи гри мають бути візуально цікавими для гравців та сповненими деталей;
- реалістичність персонажів: персонажі повинні мати привабливий візуальний вигляд а також чітко окреслені риси обличчя, одяг та анімацію;
- вражаючі візуальні ефекти: візуальні ефекти мають доповнювати ігровий процес, але головне щоб вони не відволікали гравців від основної дії.

Як приклад можна розглянути гру Path of Exile: Гра виконана в темних та похмурих тонах, що надають їй готичної атмосфери. Персонажі та монстри візуально цікаві та різноманітні, а візуальні ефекти використовуються для підкреслення бойових дій.

Доступність:

Ігри повинні бути максимально доступними для всіх гравців, незалежно від їхніх фізичних або когнітивних здібностей. Наприклад:

– зрозумілий ігровий процес: під цим пунктом мається на увазі чітке пояснення правил гри, надання підказок, а також можливість налаштувати складність гри;

– різні варіанти керування: гравці повинні мати можливість використовувати не тільки клавіатуру та мишу, а ще й різні контролери, джойстики, сенсорні екрани;

– допоміжні технології: можна також використовувати мовні команди, зчитувачі екрана та інші інструменти, які б допомагали людям з інвалідністю грати в ігри.

Розробники повинні пам’ятати, що доступність – це проблема яку не можна ігнорувати. Роблячи свої ігри доступними для всіх гравців, розробники мають можливість розширити аудиторію та зробити світ гри більш інклюзивним.

1.2 Огляд існуючих рішень

3D ігри з видом зверху (Top-Down 3D Games) стали доволі популярним жанром, що пропонує захоплюючий ігровий процес, який поєднує в собі багато рішень. Такі ігри часто характеризуються ізометричною перспективою, де камера розташована над ігровим світом, що дає можливість гравцям отримати чітке уявлення про навколишнє середовище та своїх персонажів.

Метою даного дослідження є огляд існуючих Top-Down 3D ігор, розглядання їх механік, особливостей та функціоналу, що надали їм такої популярності. Аналізуючи та вивчаючи даний жанр, можна отримати певні висновки про розвиток, слабкі та сильні сторони даних проектів.

У цій роботі будуть розглянуті існуючі рішення жанру Top-Down 3D ігор, від самого його заснування, до сучасних проектів. В випадку звернемо увагу на функціонал, механіки, візуальний стиль та інші ключові аспекти. Список відкриває гра Ultima Underworld: The Stygian Abyss (1992) (рис. 1.1).

Рисунок 1.1 – Геймплей гри Ultima Underworld: The Stygian Abyss (1992)

"Ultima Underworld: The Stygian Abyss" (1992) стала визначною ланкою у розвитку RPG та top-down ігор. Хоча гра використовувала перспективу від першої особи, але її іновації значно вплинули на дизайн та механіку top-down та RPG ігор загалом.

Гра запропонувала повноцінний тривимірний світ з текстурами та динамічним освітленням, що дало можливість їй виступити зразком для інших ігор, включаючи ті, що використовують вид зверху. Впровадження реалістичної фізики та взаємодії з об’єктами додало глибини ігровому процесу, що стало натхненням для top-down жанру, де об’єкти та персонажі та персонажі взаємодіють більш реалістично.

"Ultima Underworld"[20] ввела нелінійний геймплей, дозволяючи гравцям вільно досліджувати світ та виконувати завдання у власному темпі. Цей підхід вплинув на топ-даун RPG, де свобода дослідження та вибір гравця стали важливими елементами. Високий рівень інтерактивності з оточенням, включаючи збирання предметів, вирішення головоломок та використання різних об'єктів, став стандартом для ігор з видом зверху, де взаємодія з оточенням є ключовою складовою геймплею.

"Ultima Underworld" стала зразком для багатьох майбутніх RPG, включаючи топ-даун ігри, які наслідували та вдосконалювали її інновації. Вплив гри відчутний у таких серіях, як "Baldur's Gate", "Fallout" та "Divinity: Original Sin". Інновації та механіки, представлені в "Ultima Underworld", були вдосконалені та розширені в наступних топ-даун RPG, що сприяло розвитку жанру та створенню більш глибоких та цікавих ігор.

Таким чином, "Ultima Underworld: The Stygian Abyss" (1992) зробила значний внесок у розвиток топ-даун жанру, запропонувавши нові технологічні досягнення, складний геймплей та високий рівень інтерактивності. Її вплив відчутний у багатьох наступних іграх, які наслідують та вдосконалюють її інновації. "Ultima Underworld" встановила нові стандарти, які залишаються актуальними і до сьогодні, продовжуючи формувати майбутнє жанру. Список продовжить гра Legend of Zelda (1986)(рис. 1.2).

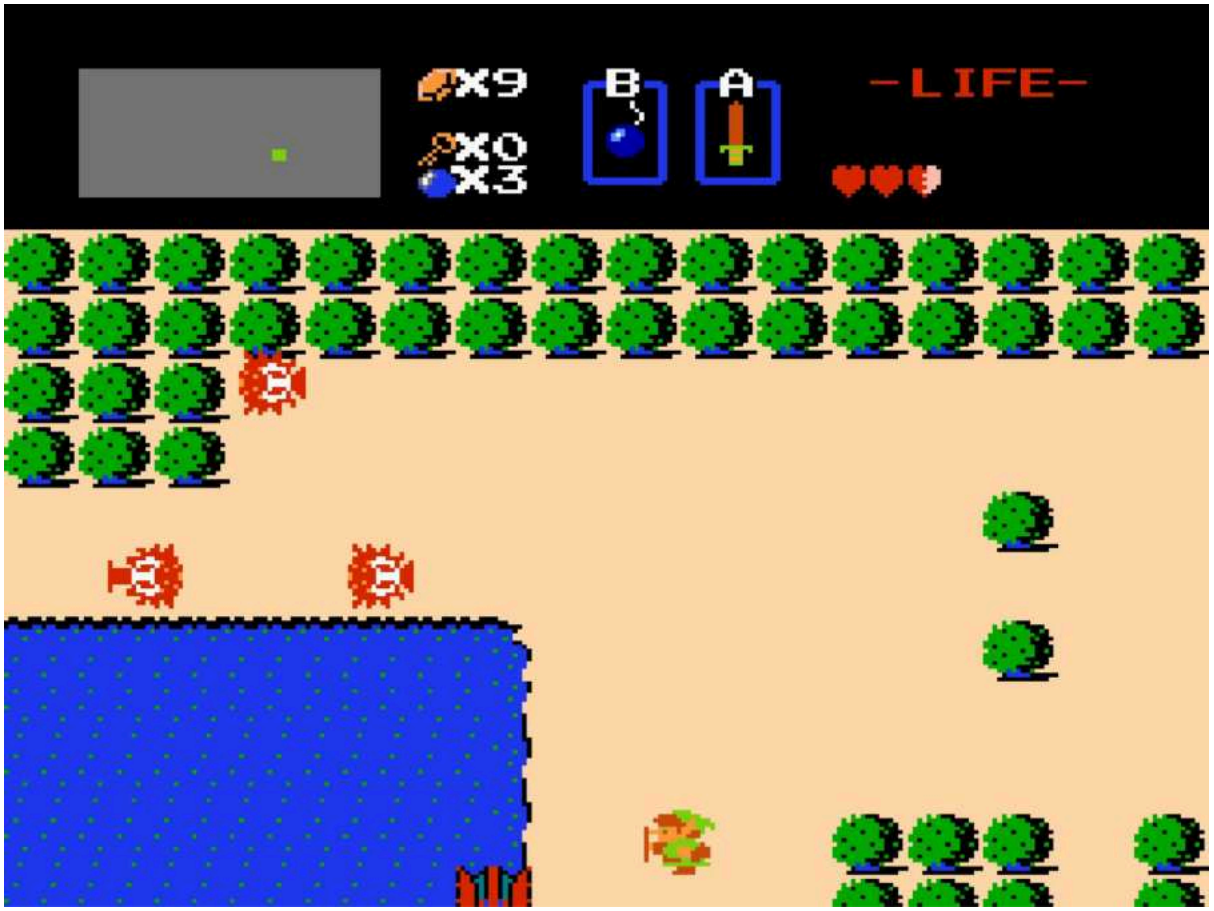


Рисунок 1.2 – Геймплей гри Legend of Zelda (1986)

"Legend of Zelda" (1986) стала революційною грою, яка внесла значний вклад у формування та розвиток Top-Down жанру в відеоіграх. Гра подала гравцям великий, відкритий світ для дослідження, що було новаторським для свого часу. Гравці могли вільно пересуватися по різних локаціях і виконувати завдання в будь-якому порядку, що створювало новий рівень свободи в ігровому процесі. Нелінійний геймплей заохочував дослідження і експерименти, дозволяючи кожному проходженню бути унікальним[19].

Дизайн рівнів в "Legend of Zelda" також був новаторським. Різноманітні локації з унікальними характеристиками та викликами робили гру більш багатогранною та захопливою. Складні та багаторівневі підземелля з безліччю пасток, ворогів і головоломок вимагали від гравців ретельного планування і стратегічного мислення.

Успіх "Legend of Zelda" встановив нові стандарти для Top-Down ігор, які були прийняті багатьма майбутніми іграми. Її вплив можна побачити в таких серіях, як "Secret of Mana", "Alundra", "Terranigma" та багатьох інших. Успіх гри надихнув багатьох розробників створювати ігри з відкритим світом, складними головоломками та інтерактивними елементами.

Таким чином, "Legend of Zelda" (1986) зробила значний внесок у розвиток Top-Down жанру, запропонувавши нові підходи до дизайну ігрового світу, геймплею та механік. Вона встановила нові стандарти, які залишаються актуальними і до сьогодні, і вплинула на багато наступних ігор у цьому жанрі. Чудовим прикладом стане гра Diablo III(рис. 1.3).

"Diablo III" (2012) внесла значний вклад у розвиток Top-Down жанру, запропонувавши низку інновацій та вдосконалень. Гра відзначилася своєю поліпшеною графікою, використанням сучасних графічних технологій та фізичного рушія Havok, що додало реалістичності та інтерактивності ігровому оточенню. Покращена механіка геймплею включала різноманітні класи персонажів з унікальними навичками, систему рун для модифікації навичок, що збільшило можливості для налаштування персонажів та стратегічного підходу до бою [9].

Рисунок 1.3 – Геймплей гри Diablo lll(2012)

Гра отримала інтуїтивний користувацький інтерфейс, який спрощував керування персонажем, взаємодію з предметами та навігацію в ігровому світі, роблячи гру доступнішою для нових гравців та зручнішою для ветеранів серії. Впровадження автоматизованого збирання золота та луту зменшило рутину, дозволяючи гравцям зосередитися на бою та дослідженні. "Diablo III" також запропонувала зручний багатокористувацький режим, що дозволяв гравцям легко з'єднуватися з друзями та спільно проходити гру, підвищуючи соціальну складову та додаючи елемент спільного досвіду. Хоча аукціонний дім був спочатку суперечливим, він додав економічний аспект до гри.

Успіх "Diablo III" встановив нові стандарти для графіки, геймплею та користувацького інтерфейсу, ставши зразком для багатьох майбутніх ігор у жанрі action RPG та hack and slash. Впровадження онлайн-елементів, таких як постійне підключення до інтернету та оновлення контенту, стало стандартом для багатьох сучасних ігор, які прагнуть підтримувати активну базу гравців та надавати новий контент. Таким чином, "Diablo III" зробила значний внесок у розвиток Top-Down жанру, запропонувавши нові технологічні досягнення та інновації, що продовжують формувати майбутнє жанру. А завершить список гра Stardew Valley (рис 1.4).

Рисунок 1.4 – Геймплей гри Stardew Valley

"Stardew Valley" (2016) зробила свій внесок у розвиток топ-даун жанру, продемонструвавши, як успішно можна поєднувати елементи RPG, симулятора фермерства та соціального симулятора. Гра об'єднала механіки RPG з елементами симулятора фермерства, надаючи гравцям можливість керувати фермою, взаємодіяти з мешканцями села, виконувати квести та досліджувати підземелля. Такий підхід показав, що різні жанрові елементи можуть гармонійно співіснувати у грі з видом зверху.

Гра пропонує гравцям відкритий світ, де вони мають свободу дій. Вони можуть займатися різними видами діяльності – від фермерства та риболовлі до соціальних взаємодій та дослідження підземель. Цей аспект свободи та відкритого світу став натхненням для багатьох топ-даун ігор, де гравцям надається можливість обирати свій власний шлях та розвивати персонажа на свій розсуд [18].

"Stardew Valley" акцентувала увагу на соціальних взаємодіях, дозволяючи гравцям будувати стосунки з NPC, що впливає на ігровий процес і розвиток сюжету. Гравці можуть одружуватися, заводити друзів та взаємодіяти з мешканцями Пелікан Таун. Це підштовхнуло інших розробників до включення

глибоких соціальних елементів у свої топ-даун ігри, де взаємодія з NPC стала важливою частиною геймплею.

Використання піксельної графіки у "Stardew Valley" стало прикладом того, як класичний візуальний стиль може бути привабливим і успішним у сучасній грі. Цей стиль надихнув багато інді-розробників на створення ігор з ретро-візуальною естетикою, де піксельна графіка використовується для створення привабливих і атмосферних світів.

Таким чином, "Stardew Valley" (2016) зробила значний внесок у заснування та розвиток топ-даун жанру. Її інноваційний підхід до інтеграції жанрів, створення глибокого та насиченого світу, акцент на соціальних взаємодіях, привабливий візуальний стиль, гнучкість у налаштуванні та управлінні ресурсами, а також постійний розвиток та підтримка гри встановили нові стандарти для топ-даун ігор. Завдяки всьому цьому "Stardew Valley" стала орієнтиром для багатьох майбутніх проектів у цьому жанрі, впливаючи на розробників і гравців та демонструючи, як створити глибокий, насичений та цікавий ігровий досвід.

1.3 Постановка задачі

Виходячи з проведеного аналізу, було прийнято рішення розробити 3D-гру з видом зверху, з використанням ігрового двигуна Unity. Гра буде присвяченою віртуальній збірці комп’ютера. В процесі проектування гри, буде розроблено та реалізовано наступні компоненти:

- ігровий світ: головною локацією було вирішено зробити затишну майстерню, яка може зустрітись у кожному куточку світу. Вона не відрізняється розкішшю, але в ній панує атмосфера творчості та майстерності;
- ігровий процес: гравець розпочинає гру з отримання замовлення на збирання комп’ютера, замовлення містить необхідні комплектуючі. У майстерні розміщені певні тумби з посортованими компонентами, відповідно до

замовлення гравець обирає потрібний йому компонент. Усі поставленні компоненти у системний блок відображаються над самим корпусом;

− інтерфейс користувача: інтерфейс користувача реалізовано доволі просто та зрозуміло, у лівому верхньому куті екрана розміщений список замовлень, а у на тумбах позначки відповідних комплектуючих;

− ігровий аудіосупровід: містить у собі мелодію, щоб ігровий процес був цікавішим.

Висновки до розділу 1

Розділи свідчать про важливість інтерактивних ігор у навчанні та розвагах, зокрема в контексті віртуальної збірки комп'ютерів. Аналіз ключових ігор у жанрі Top-Down 3D підкреслює значний внесок таких проектів, як "Ultima Underworld", "Legend of Zelda", "Diablo III" та "Stardew Valley", у розвиток жанру і встановлення нових стандартів якості та інновацій. Ці ігри надихнули подальший розвиток галузі, збагативши її інтерактивністю, складністю геймплею та візуальною привабливістю.

На основі цього аналізу було прийнято рішення розробити 3D-гру з видом зверху, яка буде допомагати користувачам вивчати основи складання комп'ютерів. Це відповідає потребам сучасних гравців у цікавому та педагогічно цінному віртуальному досвіді, а також сприяє подальшому розвитку жанру Top-Down 3D.

РОЗДІЛ 2. РОЗРОБКА СТРУКТУРИ

2.1. Вибір засобів для реалізації

Цей розділ описує засоби реалізації, які використовувалися для створення гри. Мета цього розділу - прозоро продемонструвати технологічний стек проекту та надати розуміння інструментів, які були необхідні для втілення ігрової концепції в життя.

В основі проекту лежить потужний ігровий двигун Unity. Він слугує фундаментом, на якому будується весь віртуальний світ. Unity надає все необхідне для створення захоплюючих 2D та 3D ігор, пропонуючи широкий спектр інструментів для роботи з графікою, анімацією, звуком, фізикою та ігровою логікою.

Чому Unity:

– універсальність: Unity дозволяє створювати ігри для ПК, мобільних пристроїв, консолей, віртуальної та доповненої реальності. Це робить його чудовим вибором для розробників, які прагнуть охопити широку аудиторію;

– простота використання: Unity має інтуїтивно зрозумілий інтерфейс та візуальну систему програмування, що робить його доступним навіть для початківців. Це значно знижує бар'єр для входу та дозволяє зосередитися на створенні контенту;

– продуктивність: Unity оптимізований для високої продуктивності, що гарантує плавний ігровий процес навіть на слабких пристроях. Це робить його чудовим вибором для розробників, які прагнуть створити доступні та динамічні ігри;

– спільнота: Unity має велику та активну спільноту розробників, яка завжди готова допомогти та поділитися своїми знаннями. Це робить його цінним ресурсом для новачків та досвідчених розробників.

Можливості Unity[14]:

– 2D та 3D розробка: Unity підтримує як 2D, так і 3D розробку, що робить його універсальним інструментом для створення різних типів ігор;

– створення персонажів та анімація: Unity надає потужні інструменти для створення реалістичних персонажів та анімації. Це дозволяє розробникам створювати живі та динамічні віртуальні світи;

– фізика та реалістичність: Unity має реалістичну фізичну систему, яка дозволяє створювати динамічні та захоплюючі ігри;

– штучний інтелект: Unity підтримує інтеграцію з системами штучного інтелекту, що дозволяє розробникам створювати більш складних та розумних не ігрових персонажів;

– мережеві можливості: Unity підтримує мережеві можливості, що дозволяє створювати мультиплеєрні ігри;

– розширюваність: Unity має великий магазин ассетів та плагінів, що дозволяє розширювати його функціональність та створювати унікальні ігри.

Unity в дії: Unity використовується для створення безлічі популярних ігор, таких як Pokémon GO, Cuphead, Among Us та Monument Valley. Ці ігри демонструють різноманітність та потужність Unity, а також його здатність створювати захоплюючі та інноваційні ігрові світи.

Unity - це потужний та універсальний ігровий двигун, який надає розробникам безліч інструментів та можливостей для створення захоплюючих 2D та 3D ігор. Завдяки своїй простоті використання, продуктивності та великій спільноті, Unity є чудовим вибором як для початківців, так і для досвідчених розробників.

Важливо зазначити, що unity використовує мову програмування C#, яка також використовується при створенні ігрової логіки та скриптів.

C# - це сучасна та багатофункціональна мова програмування, яка ідеально підходить для розробки ігор. Її елегантний синтаксис, потужні можливості та висока продуктивність роблять її чудовим вибором для створення захоплюючого геймплею [11].

Переваги C# для розробки ігор:

– простота використання: C# має чіткий та зрозумілий синтаксис, що робить його доступним навіть для початківців. Це дозволяє розробникам швидко освоїти мову та зосередитися на створенні ігрової логіки;

– гнучкість: C# підтримує парадигми об'єктно-орієнтованого програмування, що робить його ідеальним для розробки складних ієрархічних структур в іграх;

– продуктивність: C# компілюється в машинний код, що забезпечує високу продуктивність та швидкість виконання програм. Це робить його чудовим вибором для розробки динамічних ігор з високими вимогами до продуктивності;

– підтримка Unity: C# - це мова програмування, що використовується за замовчуванням в Unity. Це робить його ідеальним вибором для розробників, які працюють з цим популярним ігровим двигуном;

– спільнота: C# має велику та активну спільноту розробників, яка завжди готова допомогти та поділитися своїми знаннями. Це робить його цінним ресурсом для новачків та досвідчених розробників.

C# в дії: C# використовується для створення безлічі популярних ігор, таких як World of Warcraft, League of Legends, Unity, Minecraft та Factorio. Ці ігри демонструють різноманітність та потужність C#, а також його здатність створювати складні та захоплюючі ігрові механіки.

C# - це потужна та елегантна мова програмування, яка є незамінним інструментом для розробки ігор. Завдяки своїй простоті використання, гнучкості, продуктивності та тісній інтеграції з Unity, C# дозволяє розробникам оживляти свої віртуальні світи та створювати захоплюючі ігрові враження.

C# оживляє віртуальний світ логікою та динамікою, але саме Blender наповнює його візуальною красою та реалістичністю. Blender - це 3D-пакет, який слугує нам вірним помічником у створенні захоплюючих 3D-моделей, анімацій та декорацій[10].

Переваги Blender для розробки ігор:

– безкоштовність: Blender - це безкоштовний програмний пакет, що робить його доступним для розробників з будь-яким бюджетом;

– гнучкість: Blender пропонує широкий спектр інструментів для моделювання, скульптингу, текстурування, анімації та динамічної симуляції, що дозволяє реалізувати візуальні ефекти будь-якої складності;

– продуктивність: Blender постійно оновлюється та оптимізується, що забезпечує його високу продуктивність та швидкість роботи;

– спільнота: Blender має велику та активну спільноту користувачів, яка завжди готова допомогти та поділитися своїми знаннями. Це робить його цінним ресурсом для новачків та досвідчених художників;

– інтеграція: Blender легко інтегрується з Unity, що робить процес створення ігрових ассетів максимально зручним.

Blender в дії: Blender використовується для створення 3D-моделей та анімацій для безлічі популярних ігор, таких як Doom Eternal, Half-Life: Alyx, Assassin's Creed Valhalla та The Witcher 3: Wild Hunt. Ці ігри демонструють різноманітність та потужність Blender, а також його здатність створювати візуально вражаючі та реалістичні віртуальні світи[3].

Blender - це потужний та універсальний 3D-пакет, який є незамінним інструментом для розробки ігор. Завдяки своїй безкоштовності, гнучкості, продуктивності та тісній інтеграції з Unity, Blender дозволяє художникам та розробникам створювати візуально вражаючі та реалістичні віртуальні світи.

2.2. Загальна структура системи

Загальна структура складається з наступних пунктів: ігровий світ, ігровий процес, інтерфейс користувача, звук, інші компоненти.

Ігровий світ:

– майстерня: це віртуальна майстерня, де персонаж буде збирати комп'ютери. Вона обладнана різними робочими столами для комплектуючих,

інструментами та іншими елементами, які створюють атмосферу реальної майстерні;

– зона замовлень: це місце, де персонаж буде отримувати нові замовлення на збірку комп'ютерів. Тут можна буде побачити інформацію про замовлення, включаючи список необхідних комплектуючих.

Ігровий процес: а) система замовлень: ця система буде генерувати замовлення на збірку комп'ютерів з різними характеристиками; б) система вибору комплектуючих: гравець буде мати доступ до комплектуючих які стоять у нього в майстерні по окремим тумбам; в) система збірки: гравець буде використовувати клавіатуру, щоб віртуально збирати комп'ютер. Це включатиме встановлення процесора, оперативної пам'яті, відеокарти.

Інтерфейс користувача: індикатор замовлення – цей індикатор буде показувати інформацію про поточне замовлення, включаючи список необхідних комплектуючих, бюджет клієнта та час, що залишився до закінчення замовлення.

Звук – музика: гра буде містити фонову музику, яка буде створювати особливу атмосферу.

2.3. Структура проекту

Вміст кореневої директорії (рис. 2.1).

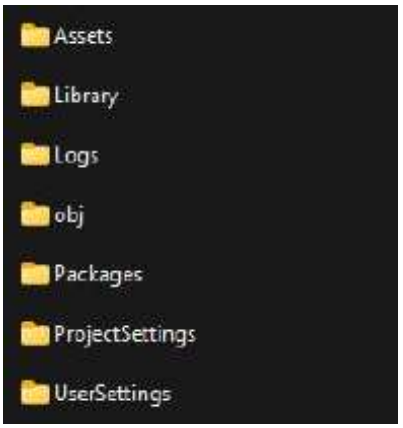


Рисунок 2.1 – Коренева директорія проєкту

В кореневій директорії розміщені важливі каталоги проєкту[14], а саме:

– assets – являють собою ресурси, які використовуються для розробки ігор та інтерактивних додатків. Вони можуть включати в себе зображення, звуки, моделі, тексти, анімації, скрипти, шейдери та інші файли, які використовують для створення візуального змісту програми. Assets можна загрузити як зовнішнім шляхом так і створити у самому середовищі Unity;

– library –даний каталог являється одним з найголовніших каталогів, які створюються автоматично самим середовищем. Каталог містить тимчасові дані та файли, які генеруються під час компіляції, імпортування ресурсів, а також під час виконання проєкту;

– logs – даний каталог являє собою місце, де зберігаються лог-файли, які пов’язані з роботою самого середовища. Логи редактора містять інформацію про події, помилки та іншу діагностичну інформацію, що стосується роботи редактора, такі як завантаження, запуск проєкту, використання скриптів тощо.

– obj – каталог у якому зазвичай зберігаються тимчасові файли, пов’язані з компіляцією та імпортом ресурсів, а також побудовою сцен та іншими процесами роботи з проектом;

– packages – даний каталог зберігає в собі дані та файли, які пов’язані з управлінням пакетами, які в свою чергу застосовуються для організації та керування залежностями проєкту. Є декілька видів, інформаційні – це файли конфігурації та метадані, кешовані – це завантажені версії пакетів які кешуються для прискорення процесу роботи з ними, локальні – власні пакети які використовуються для легкого доступу та управління проєктом, залежності та версії – зберігаються файли, які вказують на залежності між різними пакетами та їх версіями;

– projectSettings – даний каталог містить у собі конфігураційні файли та налаштування, які використовуються для налаштування параметрів проєкту та середовища розробки Unity;

– userSettings – користувацький каталог, який містить у собі користувацькі файли та конфігурації.

Інші файли слугують для коректної роботи проєкту. Тепер розглянемо кожен каталог більш детально.

Вміст каталогу Assets (рис. 2.2)

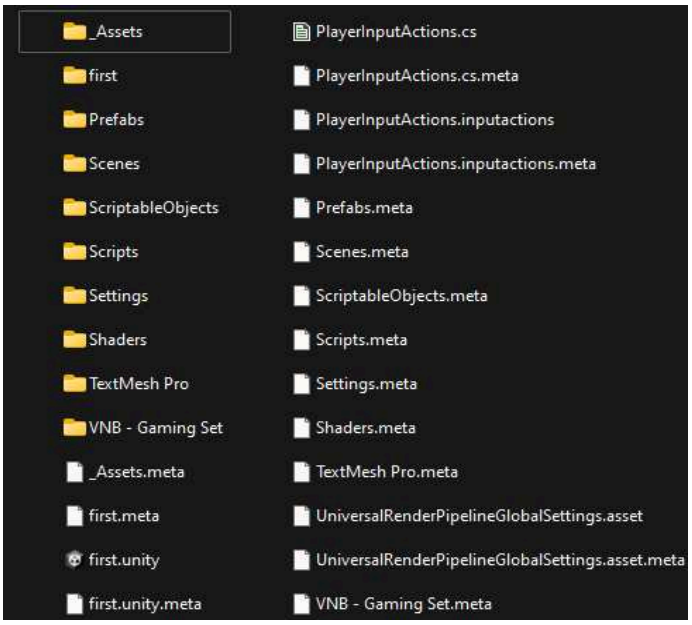


Рисунок 2.2 – Вміст каталогу Assets

_Assets являє собою завантажений набір ассетів (рис. 2.3), які використовуються для створення візуальної частини гри, а також використовуються об'єкти з якими взаємодія головний герой [17].

Рисунок 2.3 – Вміст завантаженого пака _Assets

Загалом готовий набір складається за наступних каталогів:

– animations – каталог у якому містяться послідовність рухів для моделей та налаштування для керування анімаціями;

– materials – каталог у якому знаходяться зображення, які наносяться на моделі для надання їм реалістичного вигляду, а також налаштування для матеріалів;

– prefabVisuals – каталог який зазвичай містить ресурси, які необхідні для швидкого та ефективного створення об’єктів у грі;

– sounds – каталог містить у собі звукові ефекти для гри;

– textures – один з найголовінших каталогів в готових паках, оскільки містить текстури, які використовуються для надання візуального вигляду різним об’єктам у грі.

Prefabs(рис. 2.4) являє собою набір особливих об'єктів які дозволяють зберігати, відтворювати а також редагувати моделі чи групи об'єктів. Вони являються основним способом створення складних і повторюваних об'єктів у грі, що дозволяє економити час і зусилля при розробці.

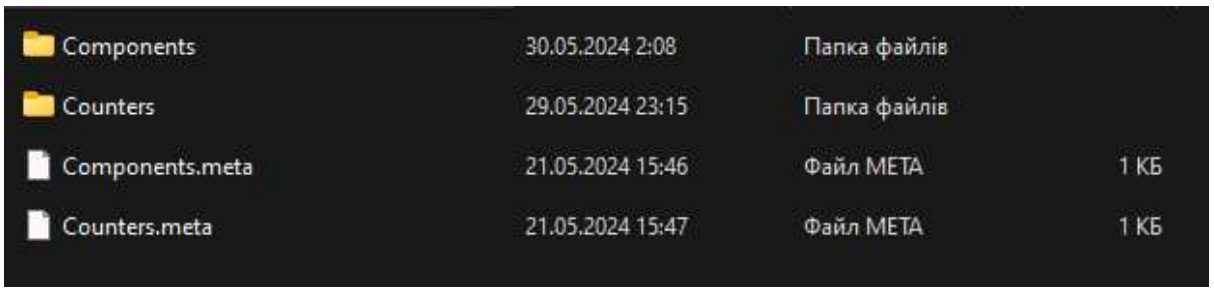


Рисунок 2.4 – Вміст каталогу Prefabs

У даному каталозі наявні два підкаталога:

– components - цей каталог зазвичай містить у собі окремі частини або модулі, які можна використовувати для створення складніших об’єктів або функцій у грі;

– counters – даному випадку містить об’єкти які неодноразово використовуються у грі.

Scenes містить основні блоки, які представляють собою окремі рівні, меню,екранні інтерфейси або будь-які інші частини проекту які вимагають різних налаштувань і елементів.

ScriptableObjects (рис. 2.5) каталог, який містить у собі класи, які являються потужним інструментом для зберігання і управління даними. Вони дають можливість створювати об'єкти які не прив'язані до сцени чи конкретного ігрового об'єкта, що дає можливість зберігати дані незалежно від ігрових об'єктів.

Рисунок 2.5 – Вміст каталогу ScriptableObjects

Підкаталоги ComponentSO та RecipeSO являє собою набір компонентів, які використовуються для компонування певних аспектів проєкту.

Scripts (рис. 2.6) є основним каталогом для зберігання усіх скриптів проєкту. Це дозволяє легко і організовано знаходити скрипти, що використовуються для управління логікою гри, взаємодії між об'єктами, обробки подій та інших завдань.

Виділимо кілька найосновніших файлів файли:

– player.cs – відповідає за логіку головного героя та його взаємодію з іншими об’єктами;

– components.cs – відповідає за логіку компонентів;

– deliveryManager.cs - відповідає за генерацію замовлень.

Settings даний каталог використовується для зберігання різноманітних налаштувань проєкту, таких як конфігурація гри, налаштування графіки,звуку та інших параметрів які являються легкодоступними та які мають можливість

редагуватися. Це допомагає централізувати управління налаштуваннями, що робить їх легшими для підтримки і зміни.

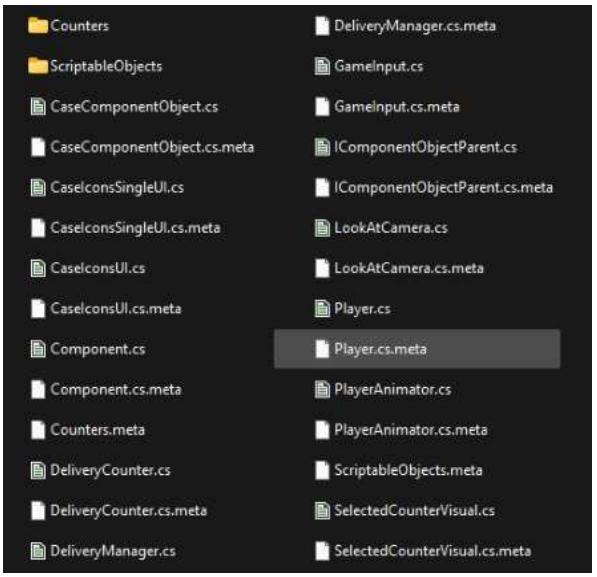


Рисунок 2.6 – Вміст каталогу Scripts

Shaders каталог для зберігання всіх шейдерів проєкту. Шейдери використовуються для створення візуальних ефектів, також для визначення того, як будуть виглядати об'єкти при відображенні на екрані, і для управління складними графічними процесами. Правильна організація даного каталогу забезпечить порядок та ефективне управління графічними ресурсами проєкту.

TextMesh Pro каталог який містить в собі файли, що пов'язані з шрифтами та налаштуваннями тексту, які використовуються в проекті.

VNB-Gaming Set - каталог в якому містяться завантаженні з зовнішнього ресурсу асети, деякі елементи використовуються в самому проєкті.

Переважно усі файли з розширенням .meta створюються автоматично середовищем при додаванні або зміні ресурсів та об'єктів у проєкті. Вони використовуються для зберігання додаткової інформації, а саме налаштування компонентів, теги, шари та посилання на інші ресурси. Це дозволяє Unity зберігати інформацію, яка не входить безпосередньо до файлу об'єкта, але безпосередньо важлива для правильної роботи проєкту.

Висновки до розділу 2

Вибір засобів для реалізації гри був здійснений з урахуванням їхньої потужності, простоти використання та підтримки широкого спектру можливостей. Unity, як основний ігровий двигун, був обраний за його універсальність, продуктивність та підтримку різних платформ, а мова програмування C# використовується для реалізації ігрової логіки та скриптів, завдяки своїй простоті та інтеграції з Unity. Blender використовується для створення візуальних елементів гри завдяки своїй безкоштовності, гнучкості та великому набору інструментів для моделювання та анімації.

Загальна структура проекту надає чітке уявлення про організацію гри, забезпечуючи логічність та доступність ресурсів для розробників, а також ефективну роботу з ними під час розробки та підтримки проекту.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація системи

У даному розділі представлено реалізацію гри, розробленої в рамках кваліфікаційної роботи. Основною метою розробки було створення ефективного та захоплюючого ігрового програмного забезпечення. Для досягнення цієї мети були використані сучасні методи та підходи до розробки ігор.

У цьому розділі буде детально розглянуто, як була розроблена ігрова логіка персонажа, включаючи його рухи та взаємодію з іншими ігровими елементами. Показані фрагменти коду ілюструють ключові програмні рішення та методи, використані для забезпечення захоплюючої поведінки персонажа.

Розглянемо код файлу Player:

```
    public static Player Instance
      {
          get; private set;
      }
      public event EventHandler OnPickedSomething;
       public event EventHandler<OnSelectedCounterChangedEventArgs>
OnSelectedCounterChanged;
      public class OnSelectedCounterChangedEventArgs : EventArgs
      {
          public BaseCounter selectedCounter;
      }
      [SerializeField] private float moveSpeed = 7f;
      [SerializeField] private GameInput gameInput;
      [SerializeField] private LayerMask counterLayerMask;
      [SerializeField] private Transform componentObjectHoldPoint;
```

Дана частина код оголошує клас Player з реалізацією шаблону Singleton, дозволяючи лише один екземпляр цього класу в грі. Він також має дві події:

одна для моменту, коли гравець підбирає предмет (OnPickedSomething), а друга для моменту, коли змінюється вибраний лічильник (OnSelectedCounterChanged). Крім того, він визначає кілька полів, які можуть бути налаштовані через інспектор Unity, таких як швидкість руху гравця, ввід гравця, маска шару лічильників та точка тримання об'єктів [13].

```
        private bool isWalking;
        private Vector3 lastInteractDir;
        private BaseCounter selectedCounter;
        private Component component;
        private void Awake()
        {
            if (Instance != null)
            {
                Debug.LogError("There is more than one Player
instance");
            }
            Instance = this;
        }
        private void Start()
        {
            gameInput.OnInteractAction += GameInput_OnInteractAction;
        }
        private void GameInput_OnInteractAction(object sender,
System.EventArgs e)
        {
            if (selectedCounter != null)
            {
                selectedCounter.Interact(this);
            }
        }
```

Цей код дозволяє гравцю взаємодіяти з об'єктами (лічильниками) у грі за допомогою вводу, оброблюючи відповідні дії через події. Поля зберігають інформацію про стан гравця, останній напрямок взаємодії, вибраний лічильник і компонент. Метод Awake реалізує Singleton, гарантуючи, що є тільки один

екземпляр Player. Метод Start підписується на подію взаємодії з вводу гравця. Метод GameInput_OnInteractAction обробляє подію взаємодії, викликаючи метод Interact на вибраному лічильнику, якщо він існує.

```
        private void Update()
        {
            HandleMovement();
            HandleInteractions();
        }
        public bool IsWalking()
        {
            return isWalking;
        }

        private void HandleInteractions()
        {
            Vector2 inputVector =
gameInput.GetMovementVectorNormalized();

            Vector3 moveDir = new Vector3(inputVector.x, 0f,
inputVector.y);
            if (moveDir != Vector3.zero)
            {
                lastInteractDir = moveDir;
            }
```

Код викликає методи HandleMovement і HandleInteractions, кожен кадр для обробки руху і взаємодії персонажа з оточенням. Метод IsWalking повертає стан ходьби персонажа через змінну isWalking. У методі HandleInteractions відбувається перевірка напрямку руху на основі введення користувача і оновлюється змінна lastInteractDir, якщо вектор руху не є нульовим.

```
            float interactDistance = 2f;
            if (Physics.Raycast(transform.position, lastInteractDir,
out RaycastHit raycastHit, interactDistance, counterLayerMask))
            {
```

```
                if (raycastHit.transform.TryGetComponent(out
BaseCounter baseCounter))
                {
                    if (baseCounter != selectedCounter)
                    {
                        SetSelectedCounter(baseCounter);

                    }
```

Цей фрагмент коду призначений для визначення об'єктів, які знаходяться в полі зору (або в межах досяжності) і мають компонент BaseCounter. Якщо такий об'єкт знайдений і він відрізняється від поточного вибраного об'єкта (selectedCounter), він стає новим вибраним об'єктом.

```
                } else
                {
                    SetSelectedCounter(null);
                }
            }else
            {
                SetSelectedCounter(null);
            }
        }
        private void HandleMovement()
        {
            Vector2 inputVector =
gameInput.GetMovementVectorNormalized();

            Vector3 moveDir = new Vector3(inputVector.x, 0f,
inputVector.y);
```

Ця частина коду обробляє взаємодії персонажа з об'єктами на певній відстані (за допомогою індикатора вибору предмету, який підсвічує певний предмет) і рух персонажа на основі користувацького введення. Якщо промінь влучає в об'єкт, він перевіряє наявність компонента BaseCounter і встановлює цей об'єкт як вибраний.

```
            float moveDistance = moveSpeed * Time.deltaTime;
            float playerRadius = .7f;
            float playerHeight = 2f;
            bool canMove = !Physics.CapsuleCast(transform.position,
transform.position + Vector3.up * playerHeight, playerRadius, moveDir,
moveDistance);
```

Ця частина коду обчислює відстань, яку персонаж повинен пройти за цей кадр, визначає розміри капсули, яка представляє об'єм персонажа, і перевіряє, чи є перешкоди на шляху руху цієї капсули. Якщо на шляху немає перешкод, персонаж може рухатися в заданому напрямку.

```
            if (!canMove)
            {
                //Якщо не вдається переміститись за moveDir
                //Робимо рух лише по осі X
                Vector3 moveDirX = new Vector3(moveDir.x, 0,
0).normalized;
                canMove = !Physics.CapsuleCast(transform.position,
transform.position + Vector3.up * playerHeight, playerRadius, moveDirX,
moveDistance);
                if (canMove)
                {
                    //Рух тільки по осі X
                    moveDir = moveDirX;
                }
                else
                {
                    //Якщо рух по осі X неможливий
                    //Тоді робимо рух  по осі Z
                    Vector3 moveDirZ = new Vector3(0, 0,
moveDir.z).normalized;
                    canMove = !Physics.CapsuleCast(transform.position,
transform.position + Vector3.up * playerHeight, playerRadius, moveDirZ,
moveDistance);
```

Цей код перевіряє можливість руху персонажа у випадку, якщо початковий напрямок заблокований. Спочатку пробує рухатися тільки по осі X, а якщо це неможливо, то по осі Z. Якщо рух по одній з осей можливий, то відповідний напрямок стає новим напрямком руху персонажа[8].

```
                    if (canMove)
                    {
                        //Рух тільки по осі Z
                        moveDir = moveDirZ;
                    }
                    else
                    {
                        //Рух не можливий ні по якій осі
            if (canMove)
            {
                transform.position += moveDir * moveDistance;
            }
            isWalking = moveDir != Vector3.zero;

            float rotateSpeed = 15f;
            transform.forward = Vector3.Slerp(transform.forward,
moveDir, Time.deltaTime * rotateSpeed);

        }
```

Ця частина коду завершує обробку руху персонажа, перевіряючи можливість руху по осі Z, якщо рух по осі X неможливий, і оновлює позицію персонажа, якщо рух можливий. Крім того, оновлюється змінна isWalking, що вказує на те, чи рухається персонаж, і напрямок обертання персонажа за допомогою плавної інтерполяції.

```
        private void SetSelectedCounter(BaseCounter selectedCounter)
        {
            this.selectedCounter = selectedCounter;
            OnSelectedCounterChanged?.Invoke(this, new
OnSelectedCounterChangedEventArgs
```

```
        {
            selectedCounter = selectedCounter
        });
    }
    public Transform GetComponentFollowTransofrm()
    {
        return componentObjectHoldPoint;
    }
```

SetSelectedCounter(BaseCounter selectedCounter): Оновлює вибраний об'єкт selectedCounter і викликає подію OnSelectedCounterChanged для повідомлення про зміну. GetComponentFollowTransofrm(): Повертає Transform об'єкта componentObjectHoldPoint, що використовується як точка для утримання або слідування інших об'єктів.

```
    public void SetComponent(Component component)
    {
        this.component = component;
        if (component != null)
        {
            OnPickedSomething?.Invoke(this, EventArgs.Empty);
        }
    }
    public Component GetComponent()
    {
        return component;
    }
    public void ClearComponent()
    {
        component = null;
    }
    public bool HasComponent()
    {
        return component != null;
    }
}
```

Цей код включає методи для обробки вибраного об'єкта (лічильника) і взаємодії з компонентами.

Код файлу PlayerAnimator:

```
public class PlayerAnimator : MonoBehaviour
{
    private const string IS_WALKING = "IsWalking";
    [SerializeField] private Player player;
    private Animator animator;
    private void Awake()
    {
        animator = GetComponent<Animator>();
    }
    private void Update()
    {
       animator.SetBool(IS_WALKING, player.IsWalking());
    }
}
```

Даний код відповідає за анімацію гравця. Він синхронізує стан анімації з поточним станом гравця та його рухом. Таким чином клас ‘PlayerAnimator’ забезпечує автоматичне оновлення анімаційного стану гравця на осонові його поточних дій.

Код файлу GameInput:

```
public class GameInput : MonoBehaviour
{
    public event EventHandler OnInteractAction;
    private PlayerInputActions playerInputActions;
    private void Awake()
    {
        playerInputActions = new PlayerInputActions();
        playerInputActions.Player.Enable();
        playerInputActions.Player.Interact.performed +=
Interact_performed;
    }
```

Дана частина коду показує, як налаштувати обробку вводу за допомогою Input System. Він створює новий екземпляр класу PlayerInputActions, активує його і підписує на подію, яка викликається при взаємодії гравця.

```
        private void
Interact_performed(UnityEngine.InputSystem.InputAction.CallbackContext
obj)
        {
                OnInteractAction?.Invoke(this, EventArgs.Empty);
        }
        public Vector2 GetMovementVectorNormalized() {
            Vector2 inputVector =
playerInputActions.Player.Move.ReadValue<Vector2>();
            inputVector = inputVector.normalized;
            return inputVector;
        }
    }
```

Даний код забезпечує обробку взаємодії та руху гравця з іншими компонентами гри, використовуючи події(івенти)[10].

Код файлу BaseCounter:

```
    public class BaseCounter : MonoBehaviour, IComponentObjectParent
    {
        [SerializeField] private Transform counterTopPoint;
        private Component component;
        public virtual void Interact(Player player)
        {
            Debug.LogError("BaseCounter.Interact();");
        }
        public Transform GetComponentFollowTransofrm()
        {
            return counterTopPoint;
        }
```

Цей код визначає клас BaseCounter, який наслідує від MonoBehaviour і реалізує інтерфейс IComponentObjectParent.

```
    public void SetComponent(Component component)
    {
        this.component = component;
    }
    public Component GetComponent()
    {
        return component;
    }
    public void ClearComponent()
    {
        component = null;
    }
    public bool HasComponent()
    {
        return component != null;
```

Клас, слугує для створення інтерактивних об’єктів у грі, що забезпечує базову функціональність для роботи з компонентами та взаємодії з гравцем.

Код файлу ClearCounter:

```
public class ClearCounter : BaseCounter
{
    [SerializeField] private ComponentSO componentSO;
    public override void Interact(Player player)
    {
        if (!HasComponent())
        {
            // Компонента нема
            if (player.HasComponent())
            {
                //Гравець щось несе в руках

player.GetComponent().SetComponentObjectParent(this);
            }
```

В даній частині йде оголошення класу ClearCounter, який наслідує від BaseCounter. Це означає, що ClearCounter успадковує всі властивості та методи BaseCounter, а також може їх перевизначати і додавати нові. Атрибут [SerializeField] дозволяє відображати приватну змінну componentSO в інспекторі Unity.

```
                else
                {
                    //Гравець немає нічого в руках
                }
            } else
            {
                //Компонент є
                if (player.HasComponent())
                {
                    //Гравець щось несе в руках
                    if(player.GetComponent().TryGetCase(out
CaseComponentObject caseComponentObject))
                    {
                        //Гравець тримає системник
                        if
(caseComponentObject.TryAddComponent(GetComponent().GetComponentSO())){
                            GetComponent().DestroySelf();
                        }
```

Розглянемо продовження методу Interact класу ClearCounter. Цей метод перевіряє стан компонентів, які гравець несе або які вже знаходяться на тумбі, і відповідним чином обробляє взаємодію.

```
                    } else
                    {
                        if (GetComponent().TryGetCase(out
caseComponentObject))
                        {
```

```
                            if
(caseComponentObject.TryAddComponent(player.GetComponent().GetComponent
SO()))
                            {
                                player.GetComponent().DestroySelf();
                            }
                        }
                    }

                }
                else
                {
                    //Гравець немає нічого в руках
                    GetComponent().SetComponentObjectParent(player);
                }
            }
        }

    }
```

Даний код додає конкретну логіку для роботи з компонентами, забезпечуючи при цьому інтерактивність і гнучкість взаємодії між гравцем і об’єктами гри.

Код файлу DeliveryManager:

```
public class DeliveryManager : MonoBehaviour
{
    public static DeliveryManager Instace { get; private set; }
    [SerializeField] private RecipeListSO recipeListSO;

    private List<RecipeSO> waitingRecipeSOList;
    private float spawnRecipeTimer;
    private float spawnRecipeTimerMax = 4f;
    private int waitingRecipesMax = 4;
    private void Awake()
    {
        Instace = this;
```

```
            waitingRecipeSOList = new List<RecipeSO>();
        }
```

Код реалізовує управління доставкою замовлень у гру. Він зберігає список очікуючих замовлень, встановлює таймер для появи нових замовлень та визначає максимальну кількість очікуючих замовлень.

```
        private void Update()
        {
            spawnRecipeTimer -= Time.deltaTime;
            if (spawnRecipeTimer <= 0f)
            {
                spawnRecipeTimer = spawnRecipeTimerMax;

                if (waitingRecipeSOList.Count < waitingRecipesMax)
                {
                                        RecipeSO waitingRecipeSO =
recipeListSO.recipeSOList[Random.Range(0,
recipeListSO.recipeSOList.Count)];
                    Debug.Log(waitingRecipeSO.recipeName);
                    waitingRecipeSOList.Add(waitingRecipeSO);
                }
            }
        }
```

Даний фрагмент кожен кадр відслідковує таймер для появи нового замовлення. Якщо таймер досягає нуля, він обирає випадкове замовлення зі списку доступних замовлень і додає його до списку очікуючих замовлень, якщо кількість замовлень у списку не перевищує максимально допустиму кількість[7].

```
                public void DeliverRecipe(CaseComponentObject
caseComponentObject)
        {
            for (int i = 0; i < waitingRecipeSOList.Count; i++)
            {
```

```
                RecipeSO waitingRecipeSO = waitingRecipeSOList[i];

                    if (waitingRecipeSO.componentSOList.Count ==
caseComponentObject.GetComponentSOList().Count)
                {
                    bool caseContentMatchesRecipe = true;
                        foreach (ComponentSO recipeComponentSO in
waitingRecipeSO.componentSOList)
                    {
                        bool componentFound = false;
```

Цей код починає процес перевірки того, чи відповідають компоненти у caseComponentObject замовленню. Далі він перебирає всі компоненти у замовленні та шукає відповідні компоненти у caseComponentObject.

```
                        foreach (ComponentSO caseComponentSO in
caseComponentObject.GetComponentSOList())
                        {
                            if (caseComponentSO == recipeComponentSO)
                            {
                                componentFound = true;
                                break;
                            }
                        }
                        if (!componentFound)
                        {
                            caseContentMatchesRecipe = false;
                        }
                    }
```

Код перевіряє, чи компоненти в об'єкті caseComponentObject відповідають певному замовленню, представленому змінною recipeComponentSO.

```
                    if (caseContentMatchesRecipe)
                    {
                        Debug.Log("Гравець відпарвив правильну
збірку");
```

```
                    waitingRecipeSOList.RemoveAt(i);
                    return;
                }
            }
        }
        Debug.Log("Гравець не відправив правильну збірку");
    }
}
```

Даний фрагмент закінчує перевірку відповідності компонентів у caseComponentObject замовленню. Якщо жодна збірка не відповідає замовленню, то виводиться повідомлення, що гравець не відправив правильну збірку.

3.2 Тестування роботи системи

Тестування переміщення персонажа (рис. 3.1), в консолі виводиться швидкість завантаження файлу та дає фідбек чи працює система переміщення персонажа. Логіка полягає в тому, що коли користувач натискає одну з клавіш “W, A, S, D” або стрілки на клавіатурі, то персонаж рухається у відповідну сторону.

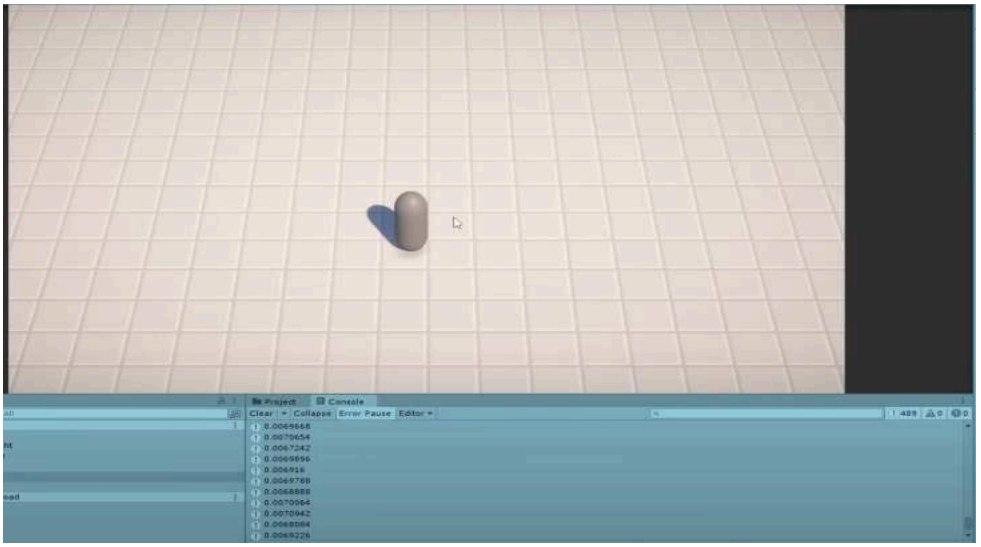


Рисунок 3.1 – Тестування логіки переміщення

Тестування колізії перед її реалізацією (рис. 3.2), персонаж проходить крізь об’єкти. При створенні інших об’єктів персонаж проходить крізь стіни, оскільки логіка зіткнень з об’єктами ще не була реалізована це є нормою.

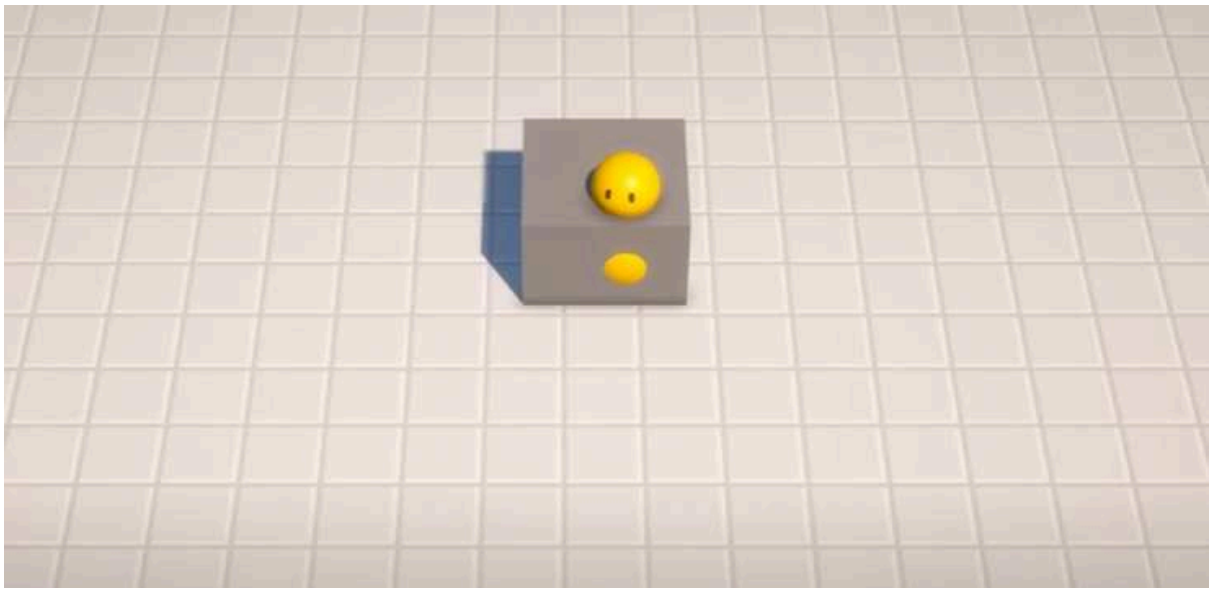


Рисунок 3.2 – Тестування взаємодії з об'єктами до реалізації колізії

Тестування колізії після її реалізації(рис. 3.3), в результаті виконання коду, персонаж замість того щоб проходити - “вдаряється” в інший об’єкт.

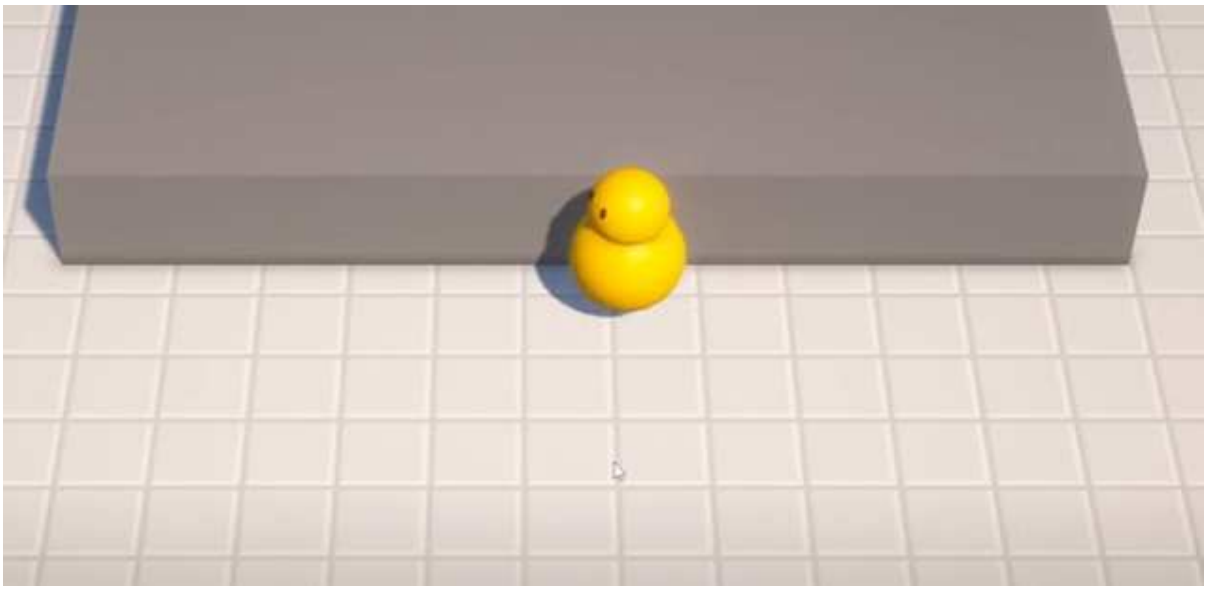


Рисунок 3.3 – Тестування взаємодії з об'єктами після реалізації колізії

Тестування реалізації вибору об’єкта (рис. 3.4), щоб гравцям було зрозуміліше з яким об’єктом персонаж буде взаємодіяти, було вирішено підсвічувати об’єкти з яким ось ось відбудеться взаємодія.

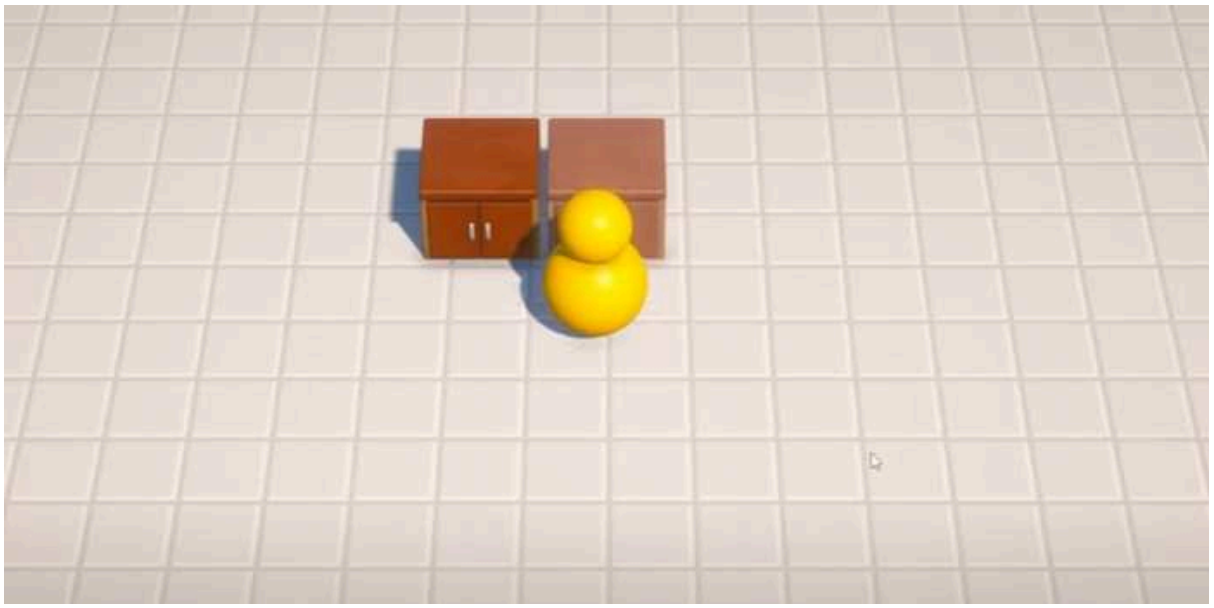


Рисунок 3.4 – Тестування вибору об'єкта

Тестування взаємодії з об’єктами (рис. 3.5), персонаж може брати об’єкти з тумб та ставити їх назад.

Рисунок 3.5 – Тестування взаємодії з об'єктами

Тестування взаємодії персонажа зі сховищем комплектуючих (рис. 3.7), є важливим, оскільки це одна з головних механік.

Рисунок 3.7 – Тестування взаємодії персонажа зі сховищем комплектуючих

Для кожної комплектуючої є своя окрема тумба, звідки персонаж буде її брати для своїх замовлень, тут важливо щоб компоненти не повторювались.

У грі реалізована функція видалення предмету (рис. 3.8), якщо гравець випадково взяв не той компонент що потрібно, або просто хоче викинути лишній компонент, ця функція була реалізована у вигляді смітника, що додає грі трішки гумору.

Рисунок 3.8 – Тестування видалення компонента з робочої області

У відповідному місці генеруються системні блоки (рис. 3.9), які гравець використовуватиме в подальшому для замовлення, системні блоки генеруються у максимальній кількості 4шт, а також гравець одразу може взаємодіяти з ними.

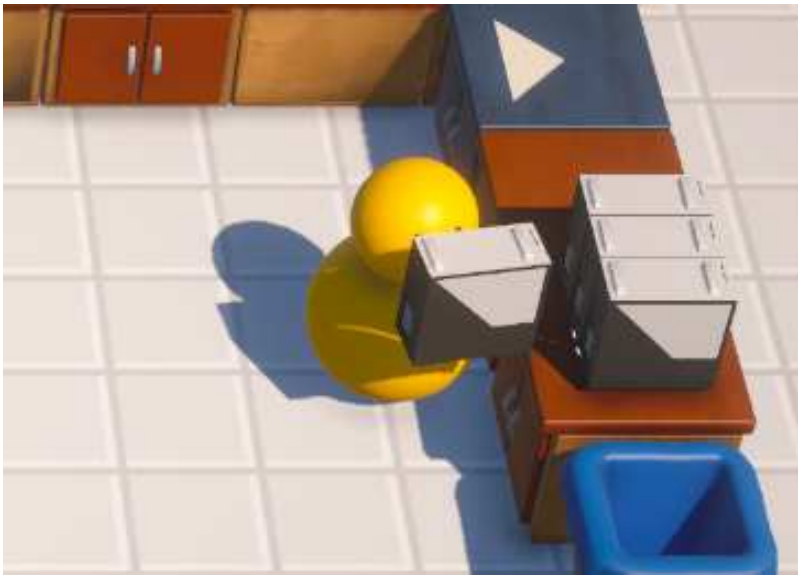


Рисунок 3.9 – Тестування генерації системних блоків та взаємодії з ними

Візуальне відображення компонентів (рис. 3.10), які гравець помістив у системний блок.

Рисунок 3.10 – Тестування відображення поміщених компонентів

Висновок до розділу 3

У даному розділі було ретельно розглянуто реалізацію гри, зосереджуючись на ключових аспектах, таких як ігрова логіка персонажа, взаємодія з об'єктами та обробка введення гравця. Завдяки використанню сучасних методів розробки ігор та глибокому аналізу коду, вдалося створити ефективне та цікаве програмне забезпечення.

Проведені тести виявили різні аспекти функціонування гри, такі як правильність переміщення персонажа, коректність взаємодії з об'єктами та належність реалізації основних механік гри.

Результати тестування підтвердили працездатність та ефективність системи гри, а також виявили можливі проблеми та недоліки, які було успішно виправлено. Гнучка структура коду та використання оптимальних алгоритмів дозволили досягти високої якості та стабільності геймплею.

ВИСНОВКИ

В процесі виконання роботи було спроектовано 3D гру з виглядом зверху на базі ігрового двигуна Unity.

В першому розділі було здійснено аналіз предметної області, розглянуто аналоги та виконано постановку задачі.

В другому розділі було описано стек технологій, що включає ігровий двигун Unity, мову програмування C#, 3D-пакет Blender, а також інші ресурси, такі як ассети, звукові ефекти.

Варто зазначити, що Unity слугує фундаментом для віртуального світу, надаючи широкий спектр інструментів для роботи з графікою, анімацією, звуком, фізикою та ігровою логікою. C# використовується для створення ігрової логіки та скриптів, забезпечуючи елегантний синтаксис, потужні можливості та високу продуктивність. Blender застосовується для створення 3D-моделей, анімацій та декорацій, роблячи віртуальний світ візуально вражаючим та реалістичним.

Загалом, обраний стек технологій дозволив створити захоплюючу та візуально привабливу гру, яка відповідає всім поставленим вимогам.

В третьому розділі описано реалізацію гри. Було представлено код ключових компонентів гри, а також продемонстровано результати тестування.

Основні моменти реалізації:

— логіка персонажа: персонаж може керуватися за допомогою клавіатури, переміщатися, взаємодіяти з об'єктами, підбирати та розміщувати компоненти комп'ютера;

— робота з компонентами: гравець може підбирати компоненти з тумб, розміщувати їх у системних блоках, а також видаляти їх з робочої області;

— система замовлень: гра генерує замовлення, які гравець повинен виконати, правильно підібравши та розмістивши компоненти в системному блоці;

— візуальне оформлення: гра має 3D-моделі, текстури та інтерфейс, що забезпечує зручне та приємне для користувача ігрове середовище;

— анімація: персонаж та ігрові об'єкти мають анімацію, що робить гру більш динамічною та реалістичною;

— звук: у грі присутні звукові ефекти, які доповнюють ігровий процес.

Тестування:

Проведено тестування ключових компонентів гри, таких як логіка переміщення персонажа, взаємодія з об'єктами, робота з компонентами, система замовлень, візуальне оформлення, анімація та звук. Тестування показало, що всі компоненти гри функціонують правильно.

Загалом, проєкт є успішним і може бути використаний як основа для створення більш складних та багатофункціональних ігор.

Практична цінність розробки полягає, в тому що гра може бути не лише захоплюючою розвагою, але й корисним інструментом для навчання. Гравці стикаються з викликом зібрати ПК виконуючи замовлення клієнтів, що дарує їм відчуття досягнення та задоволення. Водночас, гра знайомить з різними комп'ютерними комплектуючими, що викликатиме інтерес ознайомитись з їхніми функціями та сумісністю, а також дає можливість зрозуміти без яких компонентів ПК не зможе обійтись.

Таким чином, “TechBuild” може стати не лише джерелом позитивних емоцій, але й цінним ресурсом для людей, які хочуть дізнатися більше про комп'ютери та навчитися їх збирати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Blandford, N. Blender 3D: Complete Guide to Creating 3D Models, Sculptures, Animations, and Effects. Нью-Йорк, США: Focal Press, 2023. 720 с.

2. Roosendaal T. The Blender Master Class: A Professional Guide to 3D Creation. Нью-Йорк, США: Focal Press, 2022. 512 с.

3. Wageningen J. van. Blender for Dummies. Індіанаполіс, США: Wiley Publishing, 2022. 384 с.

4. Blender Guru. URL: https://www.youtube.com/@blenderguru (дата звернення: 20.03.2024)

5. Blender Tutorial. URL: https://www.youtube.com/watch?v=nIoXOplUvAw (дата звернення: 28.03.2024)

6. Davidson R. Unity Game Development with C#. Packt Publishing, 2023. 624 с.

7. Lakandella, R. C# for Game Development. No Starch Press, 2016. 464с.

8. Jensen T. K. Pro C# Game Programming. Apress, 2013.736 с.

9. GECID. URL: https://ua.gecid.com/games/retsenziya_na_igru_diablo_iii/ (дата звернення: 28.03.2024)

10. Jason M. Learning C# for Unity Game Development. Packt Publishing, 2019. 624 с.

11. Microsoft C# Documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 10.04.2024)

12. Preece, Andrew. The Blender Bible: A guide to 3D modeling and animation. Focal Press, 2011. 768 с.

13. Antonoff A. Unity 2023 Cookbook: Recipes for Creating Professional Cross-Platform Games. Бірмінгем, Велика Британія: Packt Publishing, 2023. 512 с.

14. Unity документація. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 17.04.2024)

15. Richter J.L. Learning C# by Developing Games with Unity 5. Бірмінгем, Велика Британія: Packt Publishing, 2016. 722 с.

16. McCormack R. Unity game development with C#: Learn to program and build 2D and 3D games. Річка Верхнє Сідло, Нью-Джерсі: Pearson Education, 2021. 672 с.

17. Unity Learn. URL: https://learn.unity.com/ (дата звернення: 23.04.2024)

18. Stardew Valley. URL: https://www.stardewvalley.net/ (дата звернення: 28.04.2024)

19. Zelda Wiki. URL: https://zelda.fandom.com/wiki/The_Legend_of_Zelda#Gameplay (дата звернення: 14.05.2024)

20. Ultima Underworld. URL: https://www.gog.com/en/game/ultima_underworld_1_2 (дата звернення: 15.05.2024)

метадані

Заголовок

Розробка 3D-гри типу Top-Down на ігровому двигуні Unity

Автор

Крупа Захар Науковий керівник / Експерт

підрозділ

King Danylo University

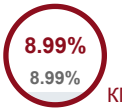
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

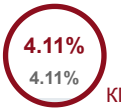
Заміна букв	ß	3
Інтервали	A→	0
Мікропробіли	·	3
Білі знаки	ß	15
Парафрази (SmartMarks)	a	35

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

8146
Кількість слів

65598
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	http://repository.ukd.edu.ua/bitstream/handle/123456789/395/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0%20%D0%A1%D1%82%D0%B5%D0%BF%D0%B0%D0%BD%D1%8E%D0%BA.pdf?sequence=1	91	1.12 %
2	http://repository.ukd.edu.ua/bitstream/handle/123456789/394/%D0%A0%D1%83%D0%B4%D0%B8%D0%B9%20%D0%90%D0%BD%D0%B4%D1%80%D1%96%D0%B9%20%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%D0%BD%D0%B0.pdf?sequence=1	39	0.48 %
3	https://pastebin.com/94zYSwGS	38	0.47 %