Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Лисенка Андрія Олеговича*
(ПІБ)

академічної групи *122-20-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка мобільної гри з використанням рушія Unity*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Ширін А.Л.*			
розділів:				
спеціальний	*доц. Ширін А.Л.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-3* (група) *Лисенка Андрія Олеговича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка мобільної гри з використанням рушія Unity*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *доц. Ширін А.Л.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Лисенко А.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *24.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 65 с., 30 рис., 3 дод., 20джерел.

Об’єкт розробки: мобільний ігровий додаток з використанням рушія Unity.

Мета кваліфікаційної роботи: розробка мобільної гри з використанням рушія Unity.

Вступ включає аналіз поточного стану проблеми, уточнення постановки завдання, мету кваліфікаційної роботи та галузь її застосування. Це важливий етап, оскільки він визначає контекст і обґрунтовує актуальність теми.

У першому розділі проводиться дослідження предметної області, аналізуються існуючі рішення. Проводиться аналіз існуючих додатків. Визначається призначення розробки та його актуальність. Також в цьому розділі формулюється постановка завдання.

У другому розділі проектується та розробляється програма. Описується алгоритм та структура функціонування додатку, визначаються вхідні та вихідні дані, а також характеристики параметрів технічних засобів. Розділ також містить інформацію про роботу та завантаження програми.

В економічному розділі розраховується трудомісткість розробленого програмного продукту, вартість роботи по його створенню та час на його розробку. Цей розділ допомагає оцінити ефективність проекту.

Результатом проведеної роботи є ігровий додаток створений за допомогою рушія Unity.

Список ключових слів: ГРА, КОМПОНЕНТИ, ДИЗАЙН, UNITY, ІГРОВИЙ ДОДАТОК, РЕАКТОР, ІНКРЕМЕНТАЛЬНА ГРА.

ABSTRACT

Explanatory note: 65 p., 30 pictures, 3 add., 20 sources.

Object of development: a mobile game application using the Unity engine.

The purpose of the qualification work: Development of a mobile game using the Unity engine.

The introduction includes an analysis of the current state of the problem, clarification of the statement of the task, the purpose of the qualification work and the field of its application. This is an important stage because it sets the context and justifies the relevance of the topic.

In the first section, a study of the subject area is carried out, existing solutions are analyzed. Analysis of existing applications is carried out. The purpose of development and its relevance is determined. This section also formulates the statement of the task.

In the second section, the program is designed and developed. The algorithm and structure of the application are described, the input and output data are defined, as well as the characteristics of the parameters of the technical means. The section also contains information about the operation and download of the program.

In the economic section, the labor intensity of the developed software product, the cost of work on its creation and the time for its development are calculated. This section helps to evaluate the effectiveness of the project.

The result of the work is a game application created using the Unity engine.

Keywords list: GAME, COMPONENTS, DESIGN, UNITY, GAME APP, REACTOR, INCREMENTAL GAME.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CPU – Central Processing Unit;

GPU – Graphics Processing Unit;

ОС – Операційна система;

JSON – JavaScript Object Notation;

URL - Uniform Resource Locator;

ПК - Персональний Комп’ютер;

ООП - Об'єктно-Орієнтоване Програмування;

GIT - Global Information Tracker

ЗМІСТ

ВСТУП

Ігрова індустрія мобільних ігор є однією з галузей розваг сучасного світу, що швидко розвивається. Вона об'єднує людей різного віку за допомогою легкості використання та доступності.

Галузь відеоігор охоплює широкий спектр діяльності, включаючи створення, візуальне оформлення, написання коду, художню роботу, звукове оформлення та просування продукту. Всі ці елементи взаємодіють, щоб створити захопливий та емоційно насичений геймплей для гравців.

Метою даної кваліфікаційної роботи є розробка мобільної гри з використанням рушія Unity.

Реалізація мобільної гри «Energy Clicker» буде допомагати розвивати логічне мислення, розвивати посидючість та знижувати рівень стресу.

Розроблений застосунок повинен дозволяти виконувати наступні функції для гравця:

- купувати або видаляти частини реактора;
- покращувати частини реактора;
- змінювати загальну гучність;
- змінювати гучність музики;
- змінювати гучність звуків;
- змінювати швидкість перебігу часу;
- керувати ресурсами реактору такими як тепло, енергія та гроші.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Розробка ігор - це захопливий процес, що об'єднує творчість і технічні навички, який призводить до створення захопливих світів та унікальних ігрових переживань. Кожна відеогра, чи то захопливий екшн, чи то захоплива рольова гра, чи то захопливий пазл, починається з ідеї, перетворюється на детально опрацьований проєкт і, нарешті, стає реальністю для мільйонів гравців по всьому світу. На цьому захопливому шляху розробки ігор втілюються в життя найсміливіші та найкреативніші концепції. [1]

Дизайн.

Етап “Дизайн” у розробці відеогри є одним із ключових, адже саме на цьому етапі визначаються основні візуальні та інтерактивні аспекти гри. Ось основні компоненти етапу “Дизайн”:

1. Визначення концепції гри. На цьому етапі розробники формують загальну ідею гри, включаючи сеттинг, історію, персонажів та основні механіки.

2. Розробка геймплею. Визначаються правила гри, задачі, які повинен виконувати гравець, та механіки взаємодії з ігровим світом.

3. Створення арт-концепцій. Художники розробляють візуальний стиль гри, створюють концепт-арти персонажів, локацій, предметів тощо.

4. Дизайн інтерфейсу користувача (UI). Розробка інтерфейсу, який буде зручним та інтуїтивно зрозумілим для гравців.

5. Аудіодизайн. Вибір та створення музичного супроводу, звукових ефектів, голосів персонажів.

6. Прототипування. Створення прототипу гри для тестування основних ідей та механік.

7. Тестування та ітерації. Проведення тестів та внесення змін на основі отриманого зворотного зв’язку.

Програмування.

Етап “Програмування” у розробці відеогри є одним з найважливіших, адже саме тут відбувається перетворення ідеї в реальний продукт. Ось ключові аспекти цього етапу:

1. Написання коду. Програмісти пишуть код, який буде основою ігрової механіки, файлової архітектури проєкту, інтерфейсу користувача та інших систем, необхідних для функціонування гри.

2. Розробка ігрової механіки. Команда працює над створенням і налагодженням ігрових правил, руху персонажів, взаємодії з ігровим світом та інших елементів, які визначають геймплей.

3. Інтеграція графіки та аудіо. Програмісти співпрацюють з художниками та звукорежисерами для інтеграції візуальних та аудіо елементів у гру.

4. Оптимізація та налагодження. Пошук та виправлення помилок у коді, оптимізація продуктивності та забезпечення стабільності гри на різних платформах.

5. Тестування. Ретельне тестування всіх аспектів гри для виявлення та усунення помилок, багів та інших проблем, які можуть вплинути на досвід користувача.

6. Підготовка до випуску. Після завершення розробки та тестування, гра готується до випуску, включаючи фінальні штрихи та підготовку маркетингових матеріалів.

Анімація.

Етап “Анімація” у розробці відеогри - це процес, де оживають персонажі та елементи ігрового світу. Ось ключові аспекти цього етапу:

1. Пред-продакшн. На цьому етапі збирається та обробляється інформація про проект. Розробляється концепція та сценарій, створюються персонажі та їхні образи.

2. Розкадрування (сторі борд). Це ескізи ключових моментів сценарію, які використовуються як план для анімації.

3. Аніматика. Попередні анімовані блоки для персонажів, які допомагають краще представляти рух і відчуття часу.

4. Продакшн. На цьому етапі відбувається моделювання, розміщення об’єктів на сцені, накладення фонів, освітлення та безпосередньо анімація.

5. Рендеринг. Процес, при якому готові анімовані 3D сцени прораховуються в послідовність картинок.

6. Пост-продакшн. Останній етап, де відбувається редагування відеоматеріалу, монтаж, поєднання візуальних ефектів і персонажів, акторська озвучка.

Звук.

Етап “Звук” у розробці відеогри є критично важливим для створення атмосфери та занурення гравця у ігровий світ. Ось ключові аспекти цього етапу:

1. Запис та обробка звуків. Звукорежисери та звукоінженери працюють над створенням звукових ефектів, які відповідають ігровим ситуаціям.

2. Композиція музики. Композитори пишуть музичні теми та мелодії, які підкреслюють емоційний тон сцен, рівнів та ігрових моментів.

3. Інтеграція звуку у гру. Програмісти інтегрують звукові файли у гру, забезпечуючи їх відтворення у відповідних моментах.

4. Звуковий дизайн. Визначається, як звуки взаємодіють з ігровим середовищем, наприклад, як вони змінюються залежно від дій гравця.

5. Озвучування персонажів. Актори озвучування записують діалоги та інші вокальні елементи, які додають глибину персонажам.

6. Тестування та налагодження звуку. Ретельне тестування для забезпечення якості звуку та його відповідності до ігрових подій.

Маркетинг.

Етап “Маркетинг” у розробці відеогри є вирішальним для її успіху на ринку. Ось ключові аспекти цього етапу:

1. Стратегія маркетингу. Розробка маркетингової стратегії, яка включає визначення цільової аудиторії, позиціонування гри, вибір каналів просування та бюджетування.

2. Брендинг та ідентичність. Створення унікальної ідентичності гри, включаючи логотип, стиль, характеристики бренду, які будуть використовуватися у всіх маркетингових матеріалах.

3. Рекламні кампанії. Запуск рекламних кампаній, які можуть включати онлайн та офлайн рекламу, соціальні медіа, інфлюенсер-маркетинг, партнерства та спонсорство.

4. Публічні відносини (PR). Робота з медіа та блогерами для створення позитивного іміджу гри та залучення уваги до неї через новини, інтерв'ю, статті та інші PR-заходи.

5. Промо-матеріали. Створення промоційних матеріалів, таких як трейлери, демо-версії, банери, постери та інші візуальні матеріали для привернення уваги до гри.

6. Цифровий маркетинг. Використання цифрових каналів, таких як електронна пошта, контент-маркетинг, SEO, SEM та аналітика для залучення трафіку та конверсії.

7. Запуск та підтримка. Планування дати випуску гри, організація заходів запуску, а також підтримка гри після випуску з допомогою оновлень, додаткового контенту та акцій.

Тестування.

Етап "Тестування" у розробці відеогри є одним з найважливіших, оскільки він забезпечує якість та стабільність кінцевого продукту. Ось ключові аспекти цього етапу:

1. Планування тестування. Визначення стратегії тестування, включаючи види тестів, які будуть проведені, та ресурси, які будуть задіяні.

2. Створення тестових сценаріїв. Розробка детальних тест-кейсів, які відображають різні ситуації в грі, щоб перевірити всі можливі варіанти геймплею.

3. Автоматизація тестування. Використання інструментів автоматизації для проведення повторюваних тестів, що дозволяє збільшити ефективність та швидкість тестування.

4. Функціональне тестування. Перевірка відповідності гри встановленим функціональним вимогам та специфікаціям.

5. Тестування продуктивності. Оцінка швидкості, стабільності та ефективності гри на різних платформах та конфігураціях обладнання.

6. Тестування на сумісність. Переконання, що гра працює коректно на різних пристроях, операційних системах та у різних мережевих умовах.

7. Тестування безпеки. Забезпечення захисту гри від зовнішніх загроз та вразливостей.

8. Бета-тестування. Залучення реальних користувачів для тестування гри в “дикій природі”, що допомагає виявити непередбачені помилки та зібрати зворотний зв’язок.

9. Виправлення помилок. Аналіз та усунення виявлених проблем, багів та інших недоліків, виявлених під час тестування.

10. Регресивне тестування. Перевірка, що виправлення помилок не призвели до нових проблем в інших частинах гри.

11. Звітування та аналіз. Підготовка звітів про результати тестування та аналіз даних для вдосконалення процесів розробки та тестування.

Існує багато видів платформ на яких можна грати, а саме:

- персональні комп’ютери (Windows, Macos, Linux і т.д.);
- мобільні пристрої (Android, Ios, Windows mobile);
- портативні ігрові системи (Playstation, Nintendo Switch, Xbox і т.д.);
- web-платформи (Google Chrome, Opera і т.д.);
- телевізори (TVOS);
- аркадні автомати.

Ігровий рушій — це програмний компонент, який відіграє ключову роль у розробці відеоігор. Він відповідає за технічну сторону гри, забезпечуючи рендеринг, фізику, звук, анімацію, штучний інтелект, мережевий код та інші

аспектиОсновна перевага ігрових рушіїв полягає в можливості створення багатоплатформових ігор – для ПК, консолей та мобільних пристроїв. Наприклад, Grand Theft Auto III та Burnout використовують рушій RenderWare, а MMORPG Lineage II базується на Unreal Engine 2.

Ігровий рушій надає багато ігрових механік одними з основних є фізичний рушій, який відповідає за симуляцію фізичних законів у грі, таких як гравітація, колізії, рух об’єктів тощо. Звуковий рушій, обробляє звукові ефекти, музику та діалоги. Відтворює звукові події в грі. Система скриптів, дозволяє розробникам створювати логіку гри за допомогою скриптів. Анімація, відповідає за рух персонажів, об’єктів та інших елементів гри. Включає в себе ключові кадри, скелетну анімацію та інші техніки. Ігровий штучний інтелект, реалізує поведінку комп’ютерних персонажів. Відповідає за прийняття рішень, навігацію та взаємодію з гравцем.

Ігрові рушії дозволяють створювати ігри для різних платформ, таких як ПК, PS4 та Xbox One. Це дозволяє ефективно використовувати один рушій для створення декількох різних ігор. Наприклад, розробники можуть використовувати один рушій для створення ігор різних жанрів або для різних платформ. Це спрощує процес розробки та забезпечує більшу ефективність.

Також існує багато різних ігрових рушіїв, які використовуються в різних жанрах ігор. Наприклад, вільні рушії, такі як Doom engine, Build Engine, Quake engine, id Tech 2, id Tech 3, id Tech 4, Cube, Godot та інші. Комерційні рушії, такі як Unreal Engine, Unity, CryEngine, Frostbite, Source, Gamebryo та інші, також широко використовуються. Кожен з них має свої особливості, переваги та недоліки, тому вибір рушія залежить від конкретних потреб та цілей проекту.

Unity - це крос-платформний інструмент для створення відеоігор і додатків, що працюють на різних пристроях, а також рушій, на якому вони базуються. Програми, розроблені за допомогою Unity, можуть працювати на ПК, мобільних пристроях, ігрових консолях і у середовищі віртуальної чи доповненої реальності, надаючи можливість створення як дво-, так і тривимірної графіки.

1.2. Призначення розробки та галузь застосування

Ігровий застосунок «Energy Ckicker» призначений для розвитку логічного мислення, посидючості та зниження рівня стресу. Гру планується використовувати у той час, коли потрібно відпочити, або коли людина відчуває надмірний стрес, нудьгуєте або має проблеми з концентрацією уваги. Гра допоможе розслабитися та відволіктися від повсякденних турбот.

Розроблений застосунок дозволяє наступне.

1. Відкрити меню магазину. купувати частини реактору, видаляти частини реактору.
2. Відкрити меню покращень. покращувати частини реактору.
3. Відкрити меню налаштувань. змінювати загальну гучність, змінювати гучність музики, змінювати гучність звуків.
4. Відкрити меню прискорення часу. змінювати швидкість перебігу часу.

1.3. Підстави для розробки

Завершення освітнього процесу в Національному технічному університеті «Дніпровська політехніка» передбачає виконання студентами кваліфікаційної роботи.

Тема кваліфікаційної роботи, була узгоджена з керівником проекту та випускаючою кафедрою, а також затверджена наказом ректора. Ця тема відображає актуальність та необхідність розробки сучасних ігрових рішень, що відповідають вимогам ринку та інтересам користувачів.

Основні підстави для розробки даного проекту включають:

- освітня програма 122 «Комп'ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;

- завдання на кваліфікаційну роботу на тему «Розробка мобільної гри з використанням рушія Unity».

1.4. Постановка завдання

Завданням даної роботи є «Розробка мобільної гри з використанням рушія Unity» на основі технологічного стеку Unity/C#. Для додатку буде використана платформа Android.

Додаток повинен реалізувати наступні функції:

- Розміщувати компоненти реактора;
- Купувати частини реактора;
- Видаляти частини реактора;
- Покращувати частини реактора;
- Змінювати загальну гучність;
- Змінювати гучність музики;
- Змінювати гучність звуків;
- Змінювати швидкість перебігу часу.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Функціональність. Гра має бути стабільною та надійною. Всі ігрові процеси, включаючи управління реактором, навігацію по інтерфейсу, робота елементів реактора, повинні працювати без збоїв. Це означає, що гравці не зіткнуться з непередбаченими помилками чи випадковими виходами з гри.

Графіка та звук. Графічні елементи, такі як текстури та анімації, мають бути чіткими та деталізованими. Звукові ефекти та музичний супровід повинні бути якісними та відповідати загальній атмосфері гри, підсилюючи ігровий досвід.

Геймплей. Геймплей має бути інтуїтивно зрозумілим та легким для освоєння новачками, але в той же час пропонувати достатній рівень складності для досвідчених гравців. Гра повинна мотивувати гравців продовжувати грати завдяки цікавим місіям та рівням.

Керування. Інтерфейс керування має бути оптимізованим для сенсорних екранів, забезпечуючи точне та плавне управління.

Стабільність та оптимізація. Гра має бути оптимізована для різних типів мобільних пристроїв, забезпечуючи плавну роботу незалежно від технічних характеристик. Оптимізація також має включати ефективне використання батареї, щоб гра не споживала зайвої енергії.

Функціональні вимоги до фінального продукту:

– Інтерфейс. Має бути чітким та зрозумілим, з легким доступом до всіх необхідних функцій;

– Збереження даних. Гра повинна автоматично зберігати ігровий прогрес та налаштування користувача, щоб можна було легко продовжити гру з місця зупинки;

– Контролі. Підтримка управління за допомогою одного пальця.

1.5.2. Вимоги до інформаційної безпеки

Мобільні ігрові додатки можуть використовувати дані користувачів для низки завдань, включаючи:

Створення та вхід у систему: Щоб забезпечити безперервний ігровий досвід, ігрові додатки вимагають від гравців реєстрації та авторизації. Під час цього процесу гравці вводять свої особисті дані, такі як ім’я, електронна адреса та пароль, що дозволяє іграм зберігати ігровий прогрес, виконувати транзакції та надавати індивідуальні пропозиції.

Збереження ігрового процесу: Ігрові додатки зберігають інформацію про рівні, які гравець пройшов, його досягнення, статистику та налаштування гри.

Це може відбуватися на серверах або локально на пристрої. Такий підхід дозволяє гравцям легко повернутися до гри після перерви або зміни пристрою.

Мультиплеєр та соціальні можливості: Ігри часто включають мультиплеєрні режими, які дозволяють гравцям змагатися або співпрацювати один з одним онлайн. Для цього ігри використовують особисті дані для підключення гравців. Соціальні функції, такі як інтеграція з соціальними мережами, дозволяють гравцям ділитися своїми успіхами та моментами з гри.

Аналіз та індивідуалізація: Розробники використовують зібрані дані для аналізу ігрових звичок та переваг гравців. Це допомагає їм вдосконалювати ігри, робити рекомендації та надавати персоналізовані пропозиції, що підвищує задоволеність гравців.

Цей ігровий додаток не збирає інформацію про особистий пристрій користувача, тому додаткові заходи для забезпечення інформаційної безпеки не потрібні. Тому нічого для забезпечення інформаційної безпеки робити не треба.

1.5.3. Вимоги до складу та параметрів технічних засобів

Відповідні характеристики необхідні для роботи додатку наведено нижче.

Для ОС Android:

1. Операційна система. Android 5.0 (Lollipop) або новіша версія.
2. Процесор. Багатоядерний процесор з мінімальною частотою 1.2 GHz.
3. Оперативна пам’ять (RAM). Мінімум 2 ГБ, але для більш вимогливих ігор рекомендується 4 ГБ або більше.
4. Вільне місце на пристрої. Залежно від гри, може знадобитися від 100 МБ до декількох ГБ.
5. Графічний процесор (GPU). Відеокарта з підтримкою OpenGL ES 2.0 або вище.
6. Екран. Рекомендується мати екран з роздільною здатністю 1280x720 пікселів або вище.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для забезпечення оптимальної роботи та широкої сумісності, ігровий додаток повинен бути створений за допомогою мови програмування C# та ігрового движка Unity. Це дозволить ігровому застосунку легко інтегруватися з різноманітними операційними системами, що є актуальним для сучасного різноплатформенного геймінгу. Однак, конкретно цей додаток розрахований на роботу виключно з операційною системою Android, що вимагає специфічної підтримки та оптимізації під цю платформу.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Результатом виконання даної кваліфікаційної роботи є розроблений мобільний ігровий додаток «Energy Clicker» із використанням технологічного стеку Unity/C#.

Під час роботи програми необхідно дотримуватися правил гри та забезпечувати зрозумілий для гравців ігровий процес. Після завершення роботи програми повинні бути збережені всі налаштування користувача та стан ігрового процесу.

Основним призначенням додатку є використання у розважальних цілях. Гра сприяє зниженню рівня стресу та розвитку посидючості.Розроблений додаток реалізує наступні функції:

- розміщувати компоненти реактора;
- купувати частини реактора;
- видаляти частини реактора;
- покращувати частини реактора;
- змінювати загальну гучність;
- змінювати гучність музики;
- змінювати гучність звуків;
- змінювати швидкість перебігу часу.

2.2. Опис застосованих математичних методів

У цій грі використовуються алгоритм розподілу та прості арифметичні операції, здебільшого з бібліотеки Mathf.

При розробці додатку були використанні такі математичні методи: Pow (возведення у ступінь), Min (Повертає найменше із двох або більше значень),

Clamp (обмежує значення між максимальним та мінімальним значеннями), Lerp (лінійна інтерполяція).

Алгоритм розподілу - рівномірно розподіляє число між сховищами з певною швидкістю, в грі використовується для реалізації розподілу тепла у трубах

2.3. Опис використаних технологій та мов програмування

Unity дозволяє створювати ігри, додатки та іммерсивні проекти для різних платформ. Особливо важливою є його мультиплатформенність, яка дозволяє створити один проект який потім можна швидко перенести на інші платформи. Інтерфейс ігрового рушія Unity на платформі Android зображено на рис. 2.1.

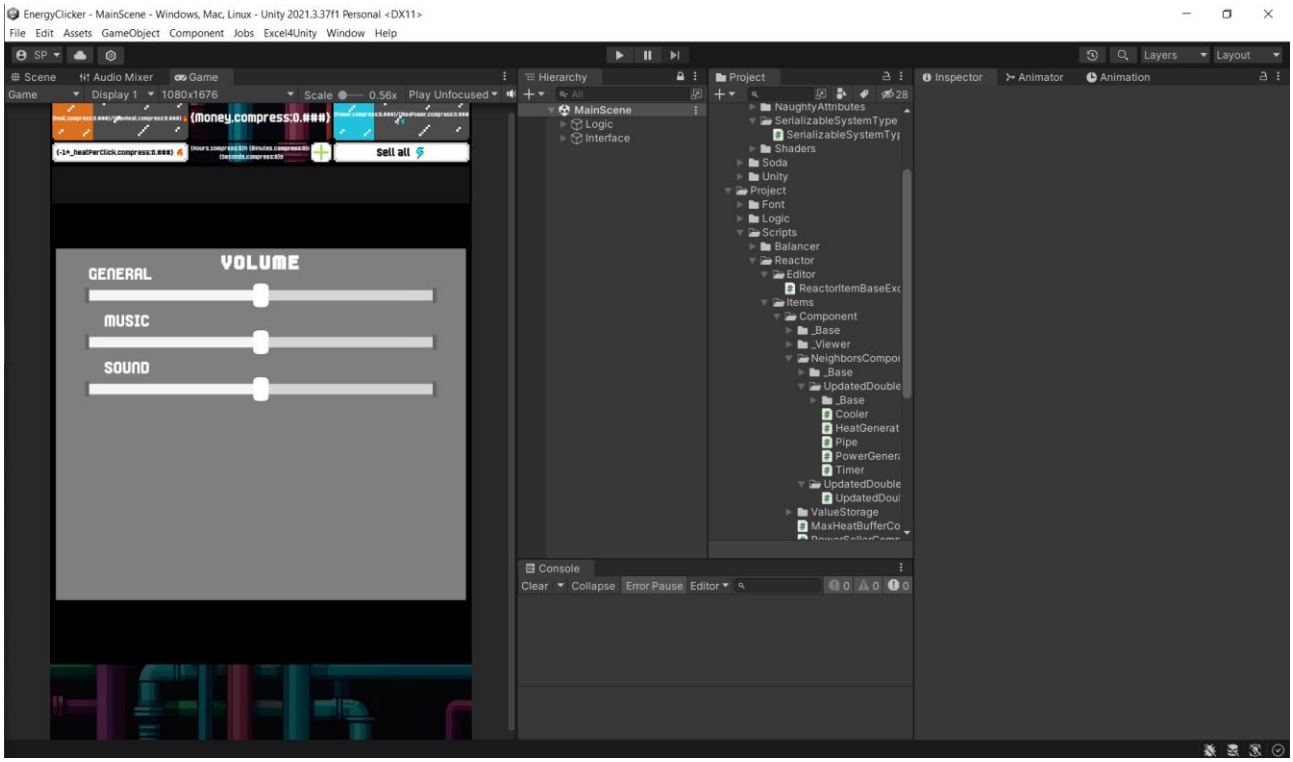

Рис. 2.1. Інтерфейс рушія Unity

Рушій Unity використовує C# як основну мову сценаріїв. Цей ігровий рушій має низький вхідний рівень і плавну криву навчання. Але в той же час він досить суворий і має багато потужних функцій для написання стабільного та ефективного коду. Все це робить технології Unity ідеальними для

початківців та зрілих програмістів. Існує безліч IDE для C# з відмінною підтримкою Unity, наприклад, Visual Studio, Visual Studio для Mac і Rider.

Магазин ресурсів Unity було запущено у 2010 році для полегшення процесу розробки ігор. За цей час спільнота розробників зробила багато ассетів, які надають готові рішення для багатьох задач.

Налагодження в реальному часі полегшує процес розробки. Плавне налагодження процесу кодування та одночасного процесу візуалізації гри, сприяє безперешкодному налагодженню під час виконання. Завдяки Unity Android Game Development можна переглядати зміни у проекті на ходу, не створюючи кожен раз як було змінено проект файл установки, щоб переглянути як буде виглядати готова програма на телефоні.

C# – мова програмування з об’єктно-орієнтованим підходом, яку вперше випустила Microsoft у 2000 році як частину .NET Framework. Вона була розроблена з метою розробки десктопних програм, веб-додатків, мобільних застосунків і ігор для операційних систем Windows.

Об’єктно-орієнтоване програмування (ООП) є стратегією розробки, яка використовує об’єкти та класи для моделювання даних та поведінки в програмах. Ця методологія є фундаментальною у програмуванні та сприяє ефективності роботи над об’ємними та складними проектами. ООП орієнтоване на декілька базових принципів, які спрощують процес розробки та підтримки програмного забезпечення, роблячи його більш інтуїтивно зрозумілим та легким у управлінні.

Мультиплатформеність, також відома як кросплатформенність, є важливою характеристикою програмного забезпечення, яка дозволяє йому функціонувати на різних програмних та апаратних платформах. Ця властивість є ключовою для розробників, оскільки вона забезпечує широку сумісність та доступність програм для користувачів на різних пристроях та операційних системах.

Мова C# вважається мовою високого рівня, оскільки її синтаксис нагадує людську мову. Іншими словами він має високий рівень абстракції від

машинного коду, тому нам необхідно скомпілювати код, написаний на C#, щоб обладнання розуміло його команди. Мови високого рівня вигідні для розробників, оскільки вони мають простіший для розуміння та управління синтаксис, на відміну від мов низького рівня, таких як C.

Інтегроване середовище розробки (IDE) створює умови для програмістів, де вони можуть писати код, тестувати його, виправляти помилки та компілювати, що є ключовими етапами розробки програмного забезпечення.

Visual Studio має ряд переваг, які роблять її відмінним вибором для IDE:

- Універсальний робочий простір для розробки програм на .NET;
- Підтримка різноманітних мов програмування, включаючи Visual Basic, C, C++ та JSON;
- Інтеграція необхідних інструментів для розробки, таких як компілятор, відладник та збирач сміття;
- Гнучке налаштування середовища та інтуїтивно зрозумілий інтерфейс користувача;
- Вбудовані можливості для командної роботи через GIT, дозволяючи командам спільно працювати над кодом і проводити код-рев'ю.

Visual Studio визнана стандартом у галузі IDE, використовується для створення програм на .NET у C# чи інших підтримуваних мовах. Також вона дозволяє працювати з іншими мовами, пов'язаними з C, та їхніми платформами. Інтерфейс середи розробки Visual Studio 2022 зображено на рис. 2.2.

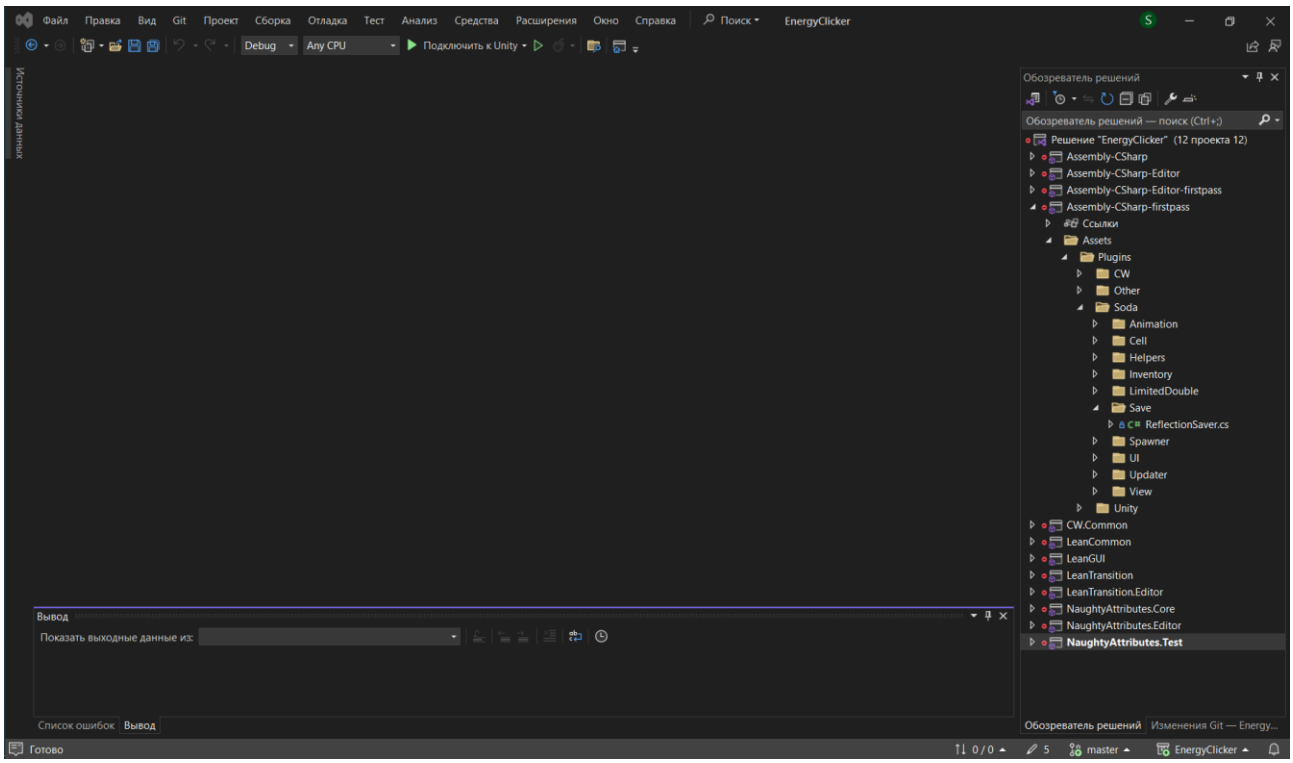

Рис. 2.2. Інтерфейс середи розробки Visual Studio 2022

JSON – легкий формат обміну даними, що широко використовується для передачі інформації між системами. Давайте розглянемо його переваги та недоліки:

1. Легкість читання та запису. JSON використовує простий синтаксис, зрозумілий як людям, так і машинам. Це робить його зручним для обміну даними.

2. Легкість передачі через мережу. JSON – компактний формат, який можна легко передавати через мережу. Це особливо важливо при обміні даними між клієнтом та сервером.

3. Незалежність від платформи JSON підтримується практично всіма сучасними мовами програмування. Це означає, що ви можете використовувати його незалежно від платформи, на якій працює ваш додаток.

4. Самоопис. JSON є форматом, що самоописується. Це означає, що дані JSON можна легко зрозуміти без додаткової документації.

5. Простота аналізу та генерації. JSON легко аналізувати та генерувати. Це робить його популярним вибором для обміну даними між різними системами.

2.4. Опис структури програми та алгоритмів її функціонування

Unity - це потужний, багато платформний і універсальний інструмент для розробки ігор, а також інтерактивних додатків. Він надає майже все необхідне для створення ігор: [16]

1. Графіка та анімація. Unity використовує сучасні техніки рендерингу, що дозволяють відтворювати графіку в реальному часі. Підтримує різні види освітлення, включаючи точкове, напрямне та площинне. Розробники можуть налаштовувати матеріали, відбиття, тіні та спеціальні ефекти, такі як частки, дим та вогонь. Має потужні інструменти для створення анімацій. Можна анімувати персонажів, об’єкти та камеру. Це дозволяє створювати рухи, вирази обличчя та інші динамічні ефекти.

2. Система компонентів. Unity дозволяє розробникам створювати компоненти та застосовувати їх до об’єктів у сцені без необхідності вручну прописувати код. Це робить розробку більш гнучкою та зручною. Кожен компонент відповідає за певний аспект поведінки об’єкта, і їх можна легко комбінувати для створення складних функціональностей. Наприклад, ви можете мати компоненти для руху, візуального відображення, здоров’я персонажа, зброї та багато інших.

3. Фізика. В Unity можна використовувати вбудовані фізичні двигуни для створення реалістичних об’єктів та їх поведінки. Фізичні компоненти, для 3D-гри, компонент Rigidbody, а для 2D-гри - Rigidbody2D. Ці компоненти дозволяють налаштовувати об’єкти так, щоб вони реагували на столкнення, гравітацію та інші сили. Колізії та зіткнення можна встановлювати параметри колізій, визначати області, де об’єкти не можуть проходити один через одного, а також використовувати детектори столкнень для взаємодії об’єктів.

4. Код. Для написання компонентів використовується мова програмування C#. Завдяки то му що ця мова програмування є простою в розумінні та дуже гнучкою можна дуже швидко та ефективно розробляти ігри та програми.

5. Активне співтовариство. Unity має велику спільноту розробників. Unity Asset Store — місце куди спільнота розробників розміщує свої ассети які інші розробники можуть як платно так і безкоштовно отримати та використати у своїх проектів.

Використані паттерни программування: [17]

1. Міст — це структурний патерн проектування, який розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої.

2. Фабричний метод — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів.

3. Ланцюжок обов'язків — це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

4. Прототип — це породжувальний патерн проектування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації.

5. Декоратор — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

6. Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

Для демонстрації варіантів сценаріїв використання користувачем програмного додатку та послідовностей ігрового процесу наведені UML діаграма варіантів використання ігрового процесу зображено на рисунку 2.3:

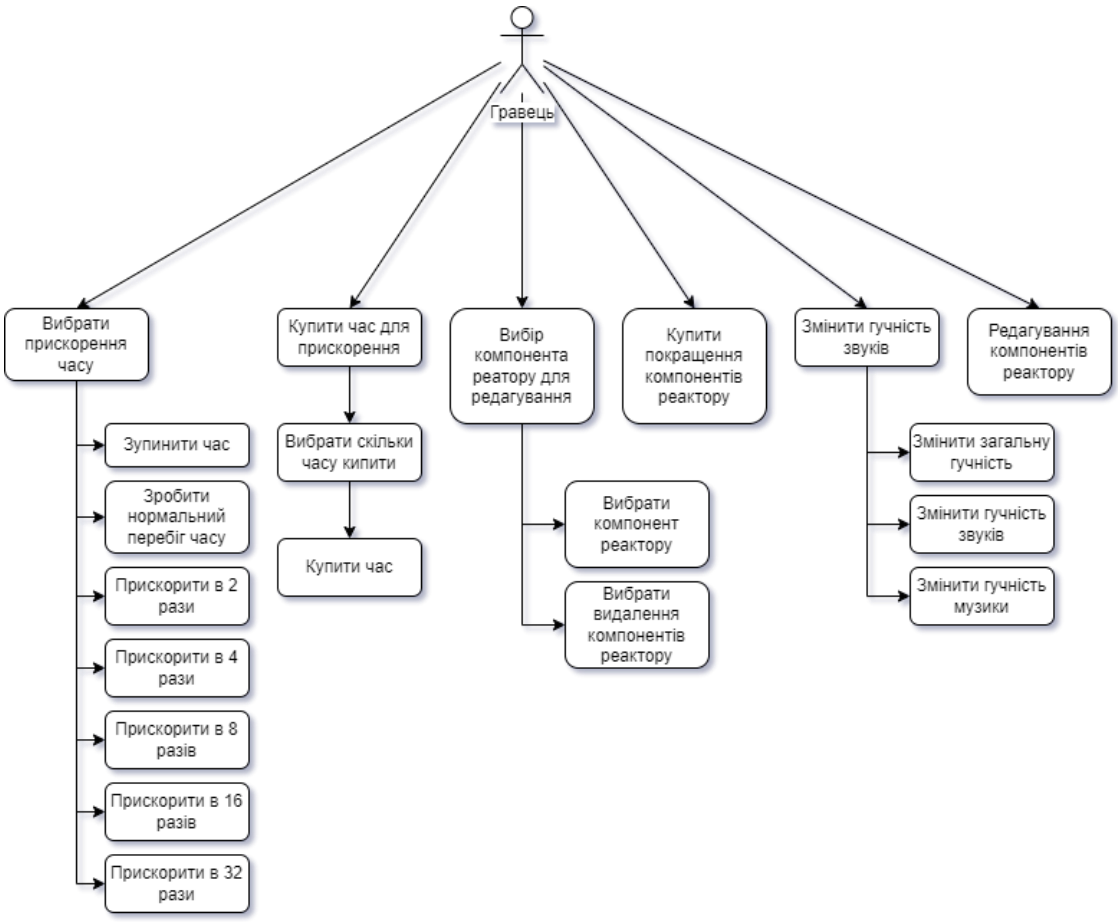

Рис. 2.3. Діаграма варіантів

Гравець з вікна «Прискорення часу» може:

- Зупинити час;
- Зробити нормальний перебіг часу;
- Прискорити час в 2 рази;
- Прискорити час в 4 рази;
- Прискорити час в 8 разів;
- Прискорити час в 16 разів;
- Прискорити час в 32 рази.

Гравець з вікна «Магазин» може:

- Вибрати компонент реактору який буде купуватися при редагуванні сітки реактору;
- Вибрати режим видалення компонентів реактору.

Гравець з вікна «Реактор» може:

- Редагувати сітку реактору.

Гравець з вікна «Магазин покрашень» може:

- Купувати покращення для компонентів реактору.

Гравець з вікна «Налаштування» може:

- Змінити загальну гучність;
- Змінити гучність звуків;
- Змінити гучність музики.

Гравець з вікна «Магазин часу» може:

- Купувати «час».

Склад файлової системи розробленого проекту зображено на рис. 2.4:

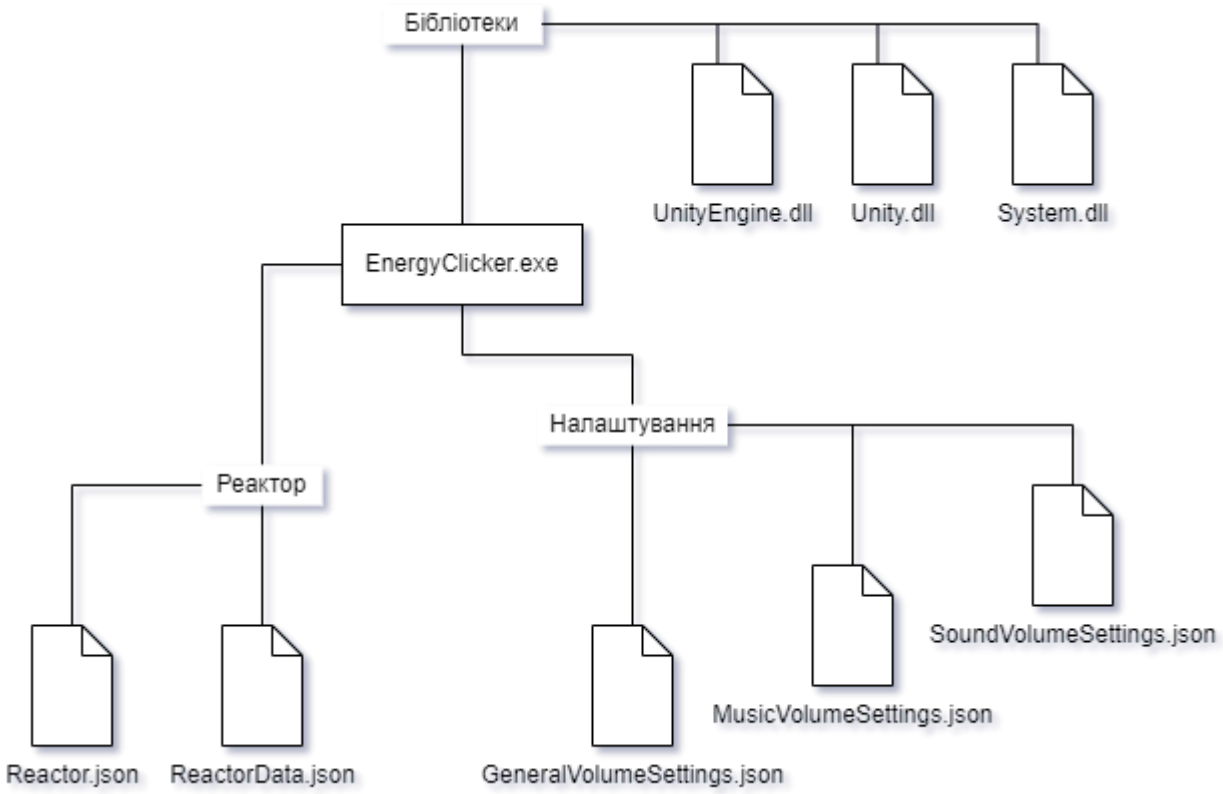

Рис. 2.4. Файлова система проекту

Цикл подій для функцій обробки подій (Event functions) в Unity

Event functions (функції обробки подій) - це вбудовані події, до яких скрипти MonoBehaviour можуть підписатися, реалізуючи відповідні методи, часто називають їх зворотними викликами (callbacks). Ці зворотні виклики відповідають подіям в основних підсистемах Unity, таких як фізика, рендеринг

та введення користувача, або стадіям життєвого циклу самого скрипта, таким як його створення, активація, оновлення залежно від кадрів та незалежно від кадрів, а також знищення. Коли відбувається подія, Unity викликає відповідний зворотний виклик у вашому скрипті, надаючи можливість реалізувати логіку відповідно до події.

Unity викликає ці зворотні виклики в певному порядку, який документований. Важливо розуміти цей порядок виконання, щоб не намагатися використовувати один зворотний виклик для виконання роботи, яка залежить від іншого зворотного виклику, який ще не був викликаний. Проте слід пам’ятати, що деякі зворотні виклики призначені для подій, таких як ті, що викликаються від користувача, і можуть відбуватися в будь-який час під час роботи вашої гри. Блок-схема, яка узагальнює порядок виконання та повторення функцій обробки подій в Unity протягом життя скрипта вказано на рисунку 2.5.

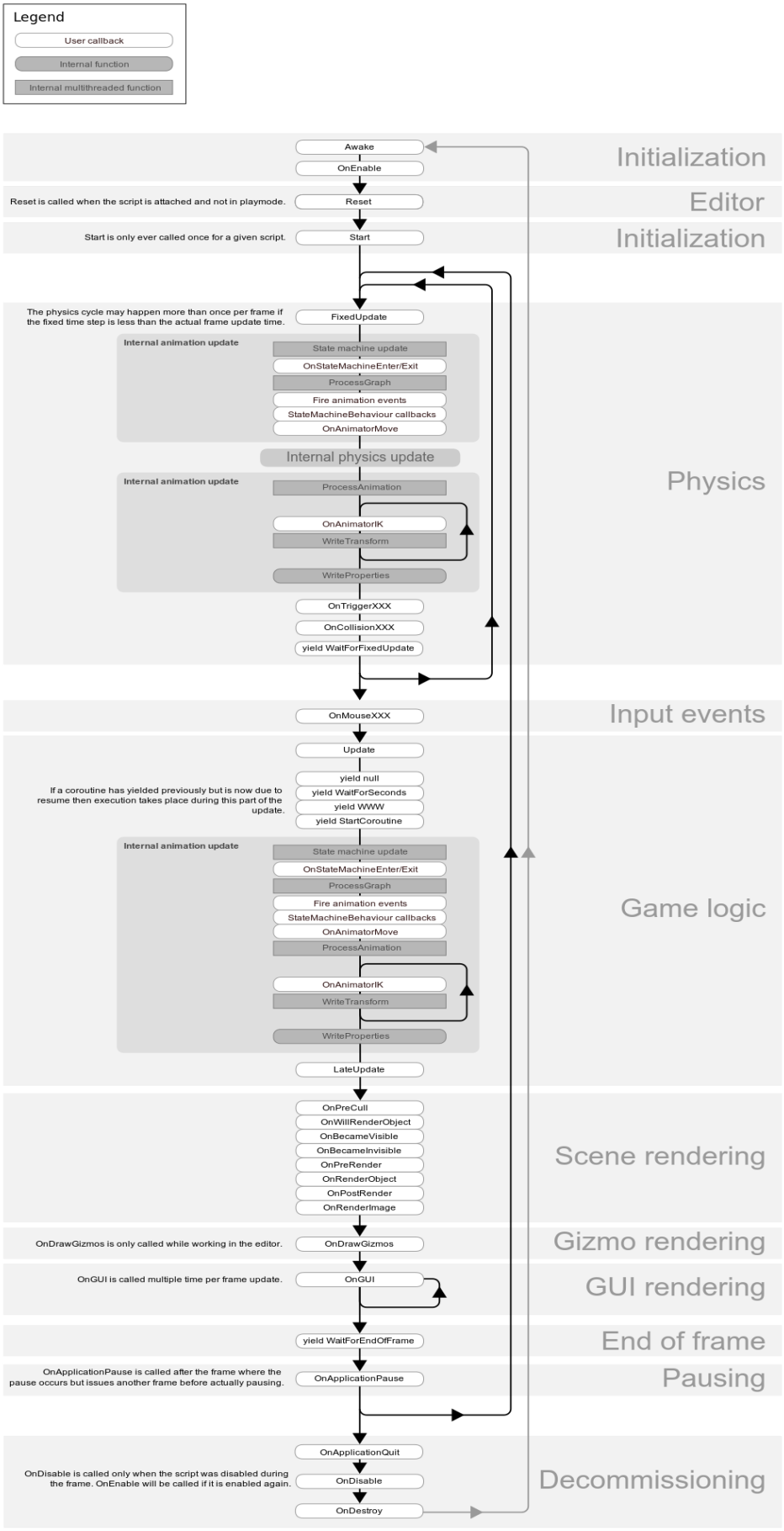

Рис. 2.5. Блок-схема життєвого циклу сценарію

В Unity кожен об’єкт представляє собою GameObject, який містить атрибути користувацьких та влаштованих компонентів. Один з таких стандартних компонентів, який не можна видалити, це Transform (рис. 2.6). В нього є такі параметри:

1. Position (Позиція): Визначає положення об’єкта в просторі за трьома осями (X, Y, Z) відносно початку координат.

2. Rotation (Поворот): Визначає орієнтацію об’єкта в просторі. Поворот задається углами навколо кожної з трьох осей (X, Y, Z).

3. Scale (Розмір): Визначає масштаб об’єкта. Значення Scale вказує, наскільки об’єкт збільшений або зменшений вздовж кожної з трьох осей.

Рис. 2.6. Компонент «Transform»

Для відображення двомірних зображень в грі використовується компонент «Image» (рис. 2.7). Після того як буде вказано у властивості Source Image посилання на зображення над ним можна проводити різні маніпуляції, такі як зміна кольору, зміна матеріалу тощо.

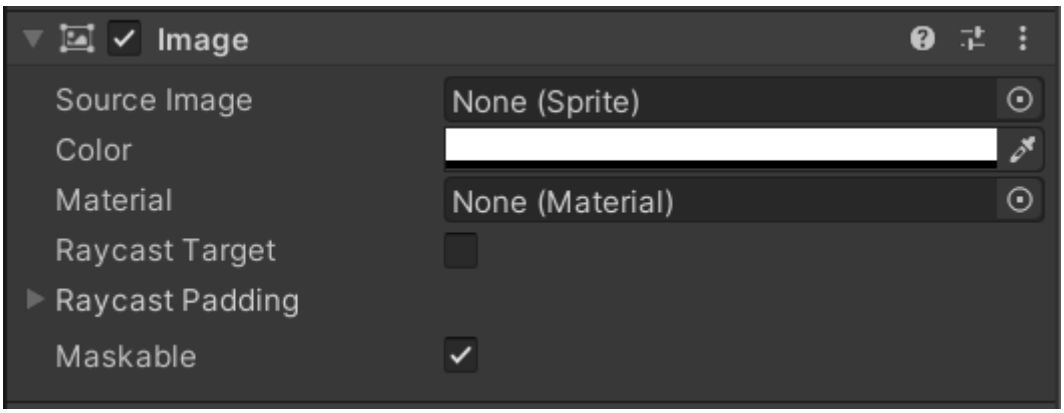

Рис. 2.7. Компонент «Image»

Для програвання анімацій в грі застосовано компонент «Animator» (рис. 2.8), щоб налаштувати роботу анімацій використовується вікно «Animation» (рис. 2.9).

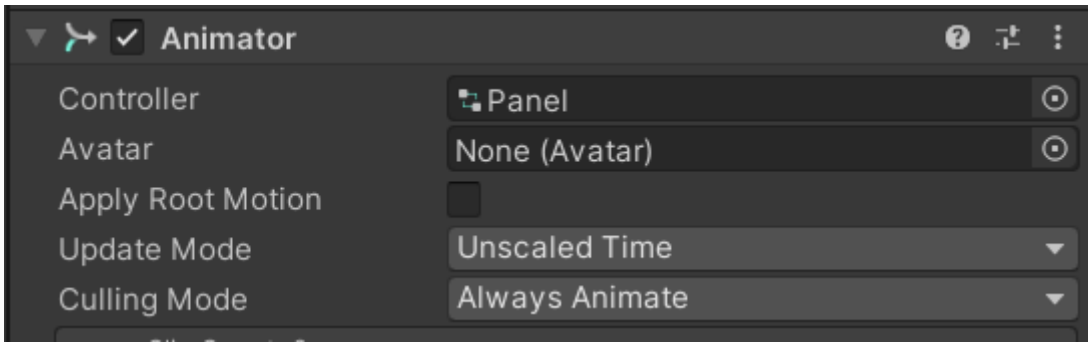

Рис. 2.8. Компонент «Animator»

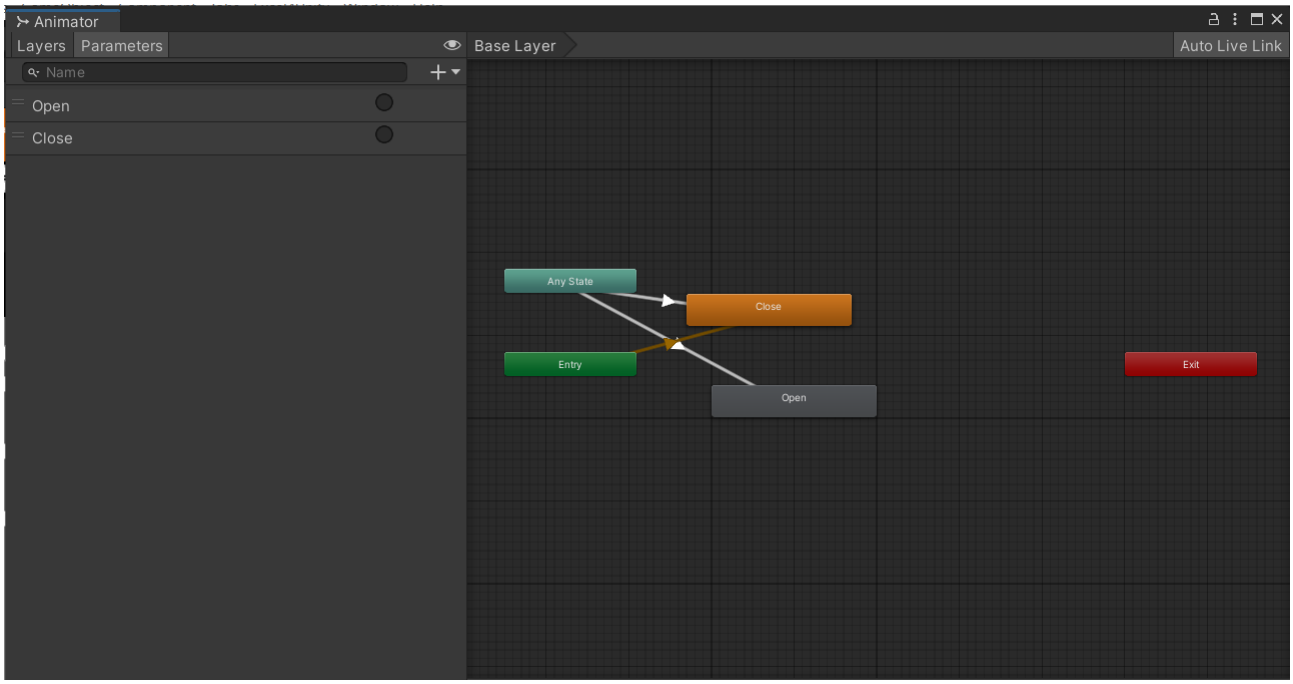

Рис. 2.9. Компонент «Animation»

Дизайн гри створений у стилі pixel-art. Зовнішній вигляд інтерфейсу тісно пов’язаний із самою грою та сутністю ігрового процесу. Приклади компонентів реактору наведено на рисунку 2.10.

Рис. 2.10. Компоненти реактору

Ігровий інтерфейс, що буде знаходитись перед гравцем весь час ігрового процесу, також стилізоване під заданий стиль. В нижній частині екрану зосереджені головні елементи керування.

Для керуючого інтерфейсу було створено наступні об’єкти і компоненти (табл. 2.1).

Таблиця 2.1

Компоненти сцени

Об’єкт	**Компонент**	**Призначення**
Reactor	Image	Візуальна картинка
	ReactorViewer	Відображати компоненти реактору
Shop	Image	Візуальна картинка
	Animator	Анімація
	ReactorShop	Магазин компонентів реактору
TimeScale	Image	Візуальна картинка
	TimeScaler	Магазин прискорення часу
Updator	Image	Візуальна картинка
	Animator	Анімація
	UpdatedBuyer	Магазин покращень компонентів

Продовження таблиці 2.1

Settings	Image	Візуальна картинка
	Animator	Анімація
	AudioMuxerVolumeSetter	Змінювати гучність аудіо міксера
TimeShop	Image	Візуальна картинка
	Animator	Анімація
	TimeShop	Магазин часу
Heat	SliderValuesChangeHelper	Переводить значення типу double у Slider
	ReflectionNumberViewer	Зчитує значення поля та відправляє його у UnityEvent
	Slider	Візуалізує число яке має максимальне значення
Power	SliderValuesChangeHelper	Переводить значення типу double у Slider
	ReflectionNumberViewer	Зчитує значення поля та відправляє його у UnityEvent
	Slider	Візуалізує число яке має максимальне значення
Money	DisplayValuesDictionaryViewer	Відображає текст та значення змінних
	ReflectionNumberViewer	Зчитує значення поля та відправляє його у UnityEvent

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Згідно з постановкою завдання у якості вхідних даних програми виступають:

- купувати або видаляти частини реактора;
- покращувати частини реактора;
- змінювати загальну гучність;
- змінювати гучність музики;
- змінювати гучність звуків;
- змінювати швидкість перебігу часу;
- керувати ресурсами реактору такими як тепло, енергія та гроші.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Розроблений додаток можна встановити на смартфон, планшет та інші пристрої з операційною системою Android але треба дотримуватись вимог до складу та параметрів технічних засобів вказаних у пункті 1.5.3.

Розробка та тестування додатку проводилось на персональному компьютері з наступними характеристиками:

- Оперативна пам'ять. 4 ГБ;
- Вбудована пам'ять. 64 ГБ;
- Операційна система. Android;
- Процесор. Qualcomm Snapdragon Snapdragon 600 Series 665, 2.0 ГГц;
- Тип ядра. Kryo 260;
- Відеоядро. Qualcomm Adreno 610;
- Роздільна здатність дисплея 2340 x 1080.

2.6.2. Використані програмні засоби

Unity надає такі можливості:

1. Пакети. Unity Asset Store — це чудовий ресурс для розробників ігор. В ньому знаходиться величезний вибір активів, інструментів та шаблонів для полегшення процесу розробки ігор. Asset Store має понад 11 000 активів з п’ятьма зірками, включаючи 3D- та 2D-моделі, SDK, шаблони та різні інструменти. Unity Package Manager — це інструмент, який дозволяє керувати пакетами в проекті Unity.

2. Аналітика. Unity Analytics — це інструмент, який допомагає збирати та аналізувати дані про продуктивність гри та поведінку гравців. Unity Profiler — це потужний інструмент для аналізу продуктивності вашої гри. Він дозволяє відстежувати, як програма використовує ресурси, такі як ЦП, пам’ять та графічний рендеринг.

3. Графічний двигун. Рендеринг — це процес перетворення тривимірної сцени на двовимірне растрове зображення за допомогою комп’ютерної програми. В Unity рендеринг відбувається за допомогою конвеєру візуалізації є декілька ковеєрів:

- Стандартний пайплайн (Built-in Render Pipeline). Використовується для більшості проектів, особливо тих, які не вимагають спеціальних ефектів або високої якості графіки. Має широку сумісність зі старими ефектами та шейдерами з Asset Store. Простий у використанні, але менше гнучкий.
- Universal Render Pipeline (URP): Це новіший конвеєр візуалізації, який замінює Lightweight Render Pipeline (LWRP). Орієнтований на більш сучасні графічні можливості та оптимізацію продуктивності. Підтримує Shader Graph для створення власних шейдерів. Має покращену підтримку 2D графіки та дозволяє використовувати Post-Processing Stack. Ідеальний вибір для нових проектів та тих, хто хоче отримати більше контролю над графікою.

4. Фізика. Unity має вбудовану фізичну систему, яка дозволяє моделювати реалістичну взаємодію між об’єктами в грі. Ця система включає симуляцію гравітації, руху та зіткнень. Можна використовувати фізичні компоненти, такі як Rigidbody, Collider та Joint, щоб створювати реалістичні ефекти в грі.

5. Аудіо. Аудіо джерела (Audio Sources) — відтворює Audio Clip у сцені.. Аудіо міксери (Audio Mixers) — надає можливість змішувати різні аудіодоріжки, налагоджувати ефекти та проводити майстеринг.

6. Анімації. Компонент Animator використовується для прив'язування анімації до об'єкта у сцені. Для роботи компонента Animator необхідно мати посилання Animator Controller, який визначає, які анімаційні кліпи використовувати, а також управляє змішуванням і переходами між ними.

7. Редагування об'єктів та сцен. В Unity є вбудовані редактори, які дозволяють створювати та редагувати об’єкти, сцени та рівні гри. Можна вибрати об’єкт у сцені та змінювати його положення, розмір та орієнтацію. Це корисно для розміщення об’єктів у просторі гри. Також можна змінювати параметри об’єктів, такі як матеріали, текстури, колір, фізика тощо. Це

дозволяє створювати різноманітні об’єкти та ефекти. Крім того, редагування сцен та рівнів гри допомагає створювати власний світ гри.

Visual Studio – це потужне інтегроване середовище розробки (IDE), яке використовується для написання, редагування, налагодження та складання коду. Вона надає інструменти для завершення коду, систему контролю версій, компілятори та розширення. За допомогою Visual Studio можна керувати всім циклом розробки в одному місці, від створення коду до розгортання програми.

Основні функції та відповідальності Visual Studio при розробці цієї гри на Unity включають:

1. Редагування коду. В Visual Studio ви можете проектувати, писати код з автозавершенням, виконувати збірку, відлагоджувати та тестувати в одному місці. Інтегрована підтримка різних мов програмування дозволяє зручно працювати зі схожими інструментами.

2. Інтеграція з Unity. Visual Studio є популярним інструментом для розробки ігор на Unity. Ви можете легко налаштувати його як основний редактор для роботи зі сценаріями C# в Unity. Це дозволяє вам зручно писати код, відлагоджувати та виконувати збірку безпосередньо в інтегрованій середовищі Visual Studio.

3. Версійний контроль Git. Visual Studio надає вбудовану підтримку Git. Ви можете керувати версіями свого коду, використовуючи команди Git, і відстежувати зміни в репозиторії. Це дозволяє вам спільно працювати з іншими розробниками та зберігати історію змін.

Git - розподілена система контролю версій, призначена для керування проектами різного розміру. Дозволяє ефективно працювати з гілками, стейджингом та різними робочими потоками. Він є набагато кращим та зручнігим вибором для контролю версій ніж такі системи як Subversion, CVS, Perforce та ClearCase. Git надає такі інструменти для роботи з контролем версій як:

1. Гілки. Ви можете створювати гілки для різних функціональностей або виправлень, а потім злити їх у головну гілку.

2. Стейджинг. Git має стейджинг-область, де ви вибираєте, які зміни включити до коміту.

3. Відстеження історії. Git зберігає всю історію змін, що дозволяє вам відстежувати рішення та хід виконання проекту в одному місці.

4. Підтримка нелінійного розвитку: Git дозволяє працювати з нескінченною кількістю гілок, що дозволяє розробникам ефективно вирішувати завдання.

5. Створення резервних копій: Git зберігає кожен коміт, що дозволяє вам відновити будь-яку версію коду.

GitHub - веб-сервис для збереження та поширення проектів. Надає зручні інструменти для розробки в команді. Pull requests, дозволяє співавторам легко повідомляти про зміни, які вони внесли до проекту.

Інтеграція з Visual Studio. GitHub Extension for Visual Studio дозволяє працювати з репозиторіями GitHub безпосередньо з Visual Studio. Можна клонувати репозиторії, комітити зміни та синхронізувати код з хмарою. Також є розширення com.unity.ide.visualstudio, яке дозволяє використовувати Visual Studio як редактор коду для Unity. Воно підтримує генерацію файлів csproj для підказок IntelliSense та автоматичне виявлення встановлених версій Visual Studio.

Інтеграція з Unity. Для роботи з Unity та Git рекомендують використовувати Git LFS (Git Large File Support). Він дозволяє ефективно зберігати великі бінарні файли, такі як текстури, моделі тощо. Включаючи .gitignore файл у проект, можна обрати типи файлів які не потрібно відстежувати (наприклад, файли збірки, кеш тощо).

2.6.3. Виклик та завантаження програми

Для початку роботи з грою на платформі Android, необхідно завантажити її інсталятор у вигляді apk файлу з будь-якого носія інформації або з інтернету. Інсталятор у вигляді apk файлу зображено на рисунку 2.11.

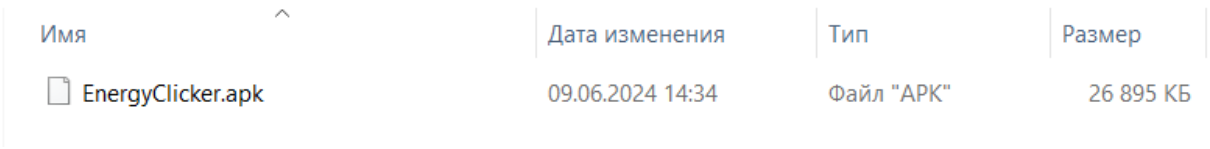

Рисунок 2.11 – Інсталятор EnergyClicker

Встановлення гри «EnergyClicker» на платформу Android показано на рисунку 2.12.

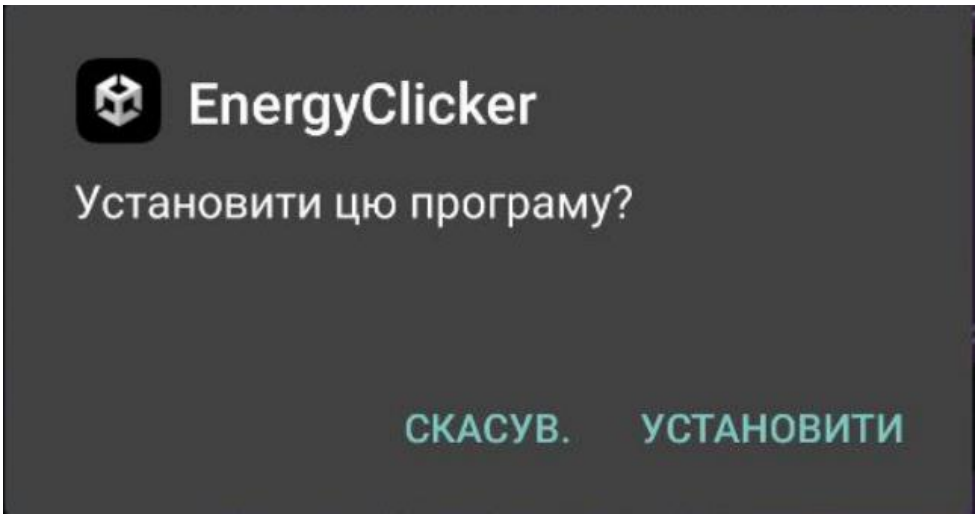

Рисунок 2.12 – Встановлення гри на андроїд

2.6.4. Опис інтерфейсу користувача

Після запуску гри, перед гравцем знаходиться вікно «Реактор» (рис. 2.17), з якого виконується редагування елементів реактору та є доступ до:

- кнопок які відкривають такі вікна як «Вікно прискорення часу», «Магазин», «Магазин покращень», «Налаштування», «Магазин часу»;
- кнопок які виконують такі дії як видалення тепла та продаж енергії;
- індикатори тепла (рис. 2.13), енергії (рис. 2.14), грошей (рис. 2.15) та часу (рис. 2.16)

Рисунок 2.13 – Індикатор тепла

Рисунок 2.14 – Індикатор енергії

Рисунок 2.15 – Індикатор грошей

Рисунок 2.16 – Індикатор тепла

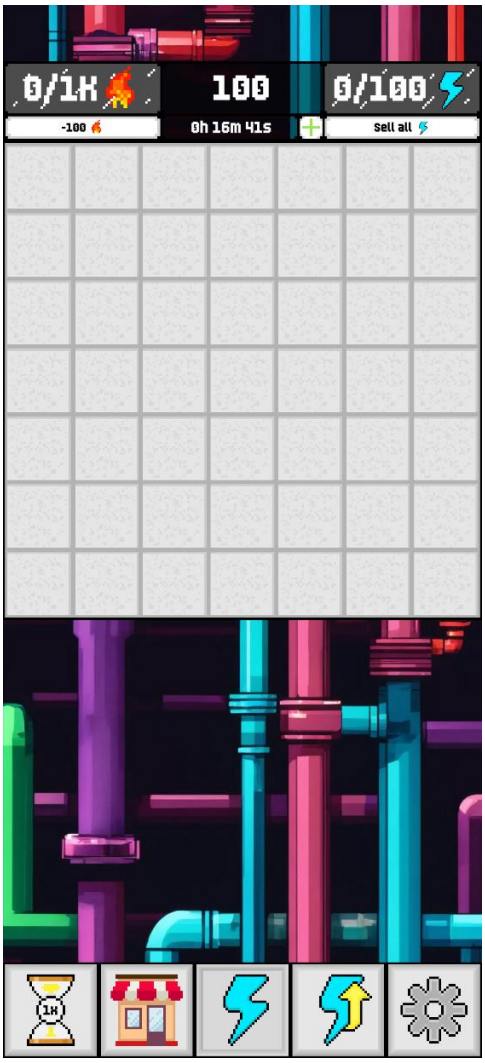

Рисунок 2.17 – Початковий екран та вікно «Реактор»

Відкрив вікно «Магазин» видно дві кнопки які відкривають розділи магазину:

1. Розділ «Енергія» в якому знаходяться компоненти реактора які виробляють тепло та «Сміттєвий бак» для видалення компонентів реактору, розділ зображений на рисунку 2.18.

2. Розділ «Охолодження» в якому знаходяться компоненти реактора які перетворюють тепло в енергію, розділ зображений на рисунку 2.24.

Рисунок 2.18 – Вікно «Енергія» вікна «Магазин»

Компонент реактору «Пальне» який відповідає за вироблення тепла є багато видів пального які відрізняються часом роботи, ціною та кількістю тепла яке виробляється (рис. 2.19).

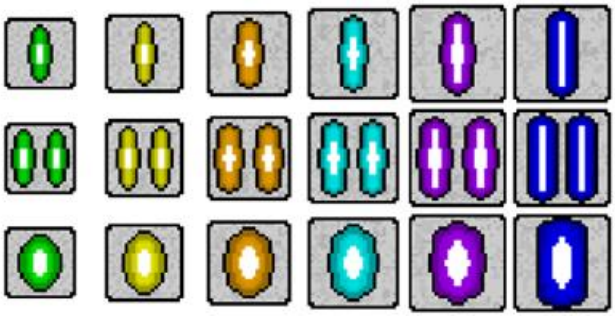

Рисунок 2.19 – Компонент реактору «Пальне»

Компонент реактору «Відбивач» який збільшує вироблення тепла у компонентів «Пальне» які знаходяться в радіусі однієї клітинки від нього (рис. 2.20).

Рисунок 2.20 – Компонент реактору «Відбивач»

Компонент реактору «Покриття» який збільшує максимальний об`єм тепла в реакторі (рис. 2.21).

Рисунок 2.21 – Компонент реактору «Покриття»

Компонент реактору «Конденсатор» який збільшує максимальний об`єм енергії в реакторі (рис. 2.22).

Рисунок 2.22 – Компонент реактору «Конденсатор»

Компонент реактору «Сміттєвий бак» який видаляє компонент в реакторі (рис. 2.23).

Рисунок 2.23 – Компонент реактору «Сміттєвий бак»

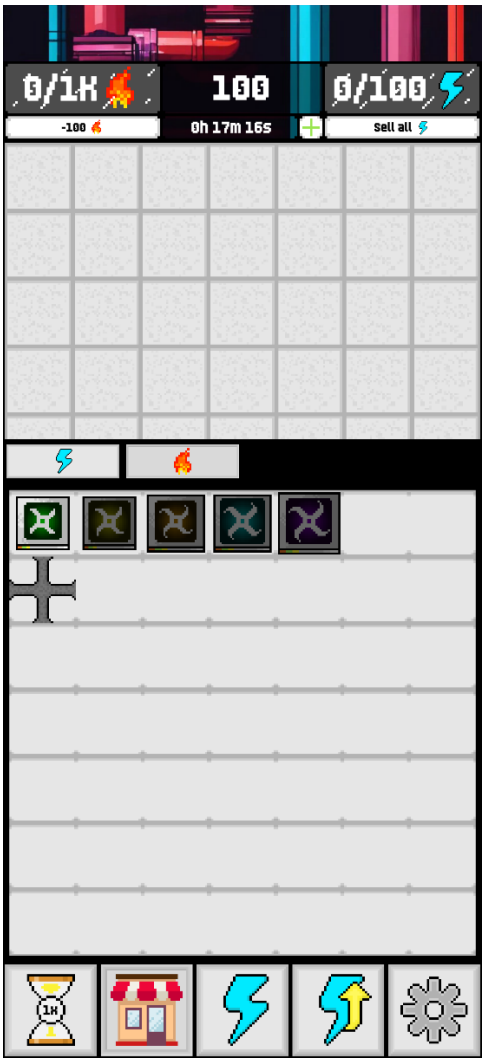

Рисунок 2.24 – Вікно «Охолодження» вікна «Магазин»

Компонент реактору «Охолоджувач» який перетворює тепло в енергію є багато видів охолоджувачів які відрізняються об’ємом переробки тепла (рис. 2.25).

Рисунок 2.25 – Компонент реактору «Охолоджувач»

Компонент реактору «Труба» який переміщує тепло між трубами в охолоджувачі (рис. 2.26).

Рисунок 2.26 – Компонент реактору «Труба»

Вікно «Прискорення часу» містить кнопки (рис. 2.27):

1. Кнопка «0x Time Scale» – безкоштовно зупиняє час
2. Кнопка «1x Time Scale» – нормалізує плив часу
3. Кнопка «2x Time Scale» – прискорює час в 2 рази при цьому використовує 2 секунди «часу» за кожну прискорену секунду
4. Кнопка «4x Time Scale» – прискорює час в 4 рази при цьому використовує 4 секунди «часу» за кожну прискорену секунду
5. Кнопка «8x Time Scale» – прискорює час в 8 рази при цьому використовує 8 секунди «часу» за кожну прискорену секунду
6. Кнопка «16x Time Scale» – прискорює час в 16 рази при цьому використовує 16 секунди «часу» за кожну прискорену секунду
7. Кнопка «32x Time Scale» – прискорює час в 32 рази при цьому використовує 32 секунди «часу» за кожну прискорену секунду

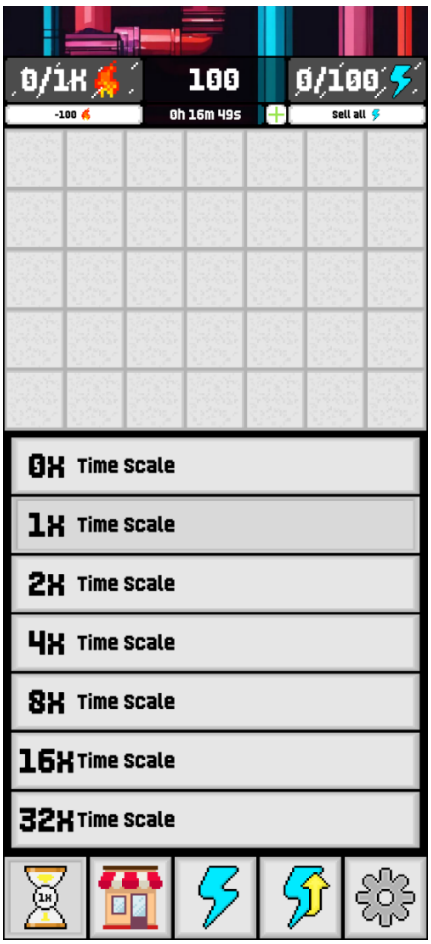

Рисунок 2.27 – Вікно «Прискорення часу»

Вікно «Магазин покращень» (рис. 2.28) в якому знаходяться покращення для всіх видів палива які збільшують час роботи та вироблення тепла. У кожного покращення є рівні покращень, чим більше рівень тим більше він коштує та більше дає бонус також є максимальний рівень покращення після якого більше не можна купувати покращення. Всі покращення зберігаються при виході з програми у файлі формату json у файлі який зберігається в постійному шляху даних.

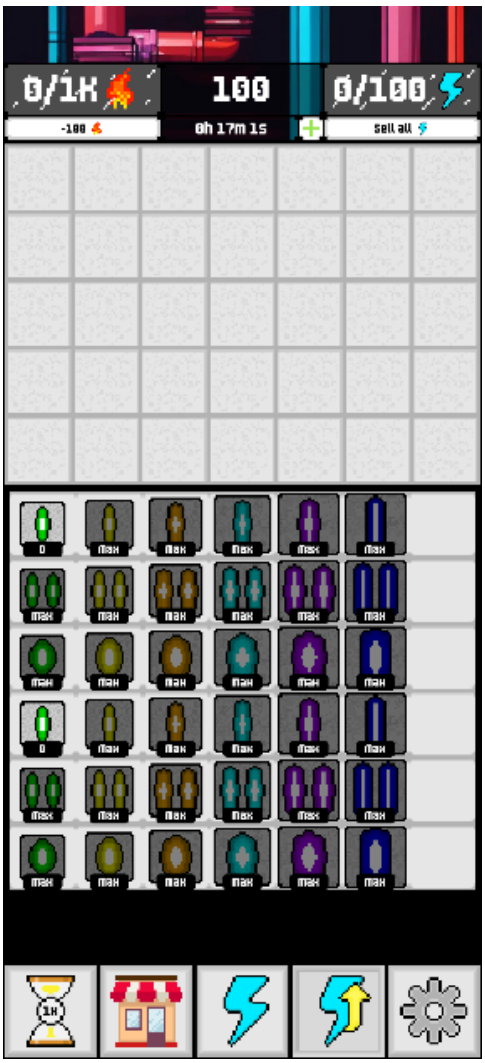

Рисунок 2.28 – Вікно «Магазин покращень»

Вікно «Налаштувань» (рис. 2.29) який містить повзунки які відповідають за зміну гучності музики, звуків та загалом. Всі налаштування які гравець зробить в цьому вікні будуть одразу збережені у файлі формату json у файлі який зберігається в постійному шляху даних.

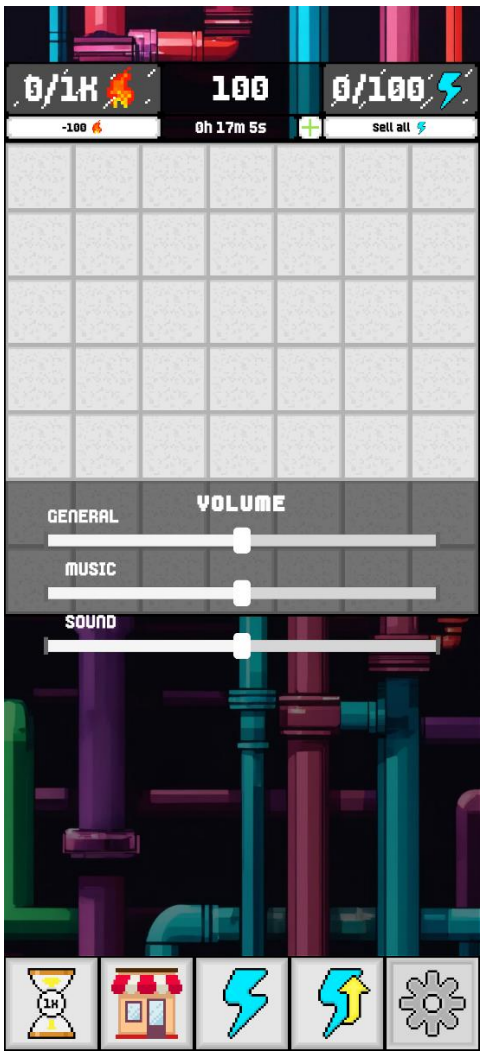

Рисунок 2.29 – Вікно «Налаштувань»

Вікно «Магазин часу» (рис. 2.30) містить панель вибору кількості часу для купівлі, тестовий індикатор ціни часу, кнопка купівлі та кнопка виходу з цього вікна.

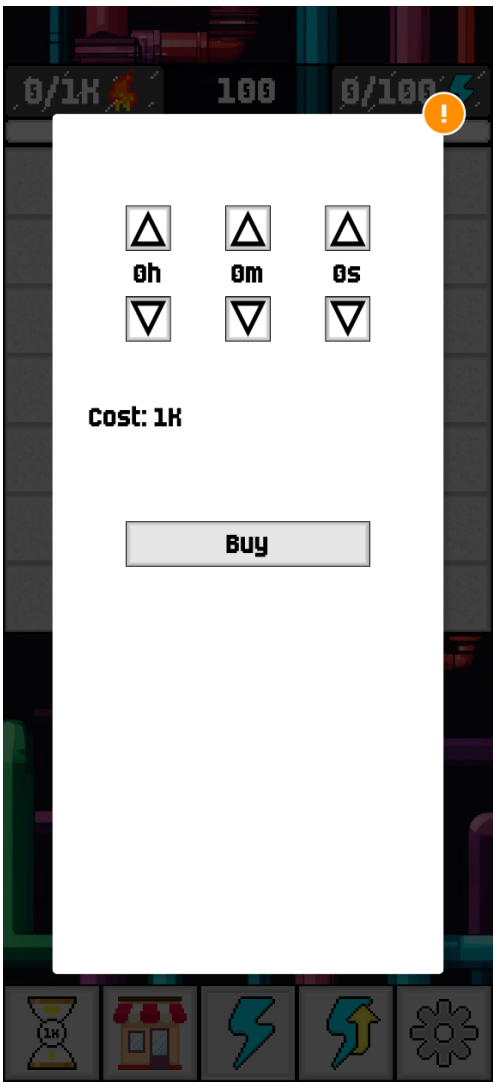

Рисунок 2.30 – Вікно «Магазин часу»

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного забезпечення

Початкові дані:

1. Передбачуване число операторів програми – 2483;
2. Коефіцієнт складності програми – 1,5;
3. Коефіцієнт корекції програми в ході її розробки – 0,1;
4. Годинна заробітна плата розробника на рушії Unity– 505 грн/год.[18]
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. Коефіцієнт кваліфікації програміста, залежний від стажу роботи – 1,2;
7. Вартість машино-години ЕОМ – 1,33 грн/год. Для розробки гри для кваліфікаційної роботи було використано ноутбук, який споживає електроенергію та використовує постійне підключення до інтернету. Щомісячна плата за інтернет від провайдера «Київстар» становить 200 грн/місяць. Ноутбук при роботі споживає 50Вт. На розробку гри було витрачено близько 360 годин впродовж двух місяців. Згідно тарифного плану кВт/год коштує 4,32 грн з урахуванням ПДВ. Маючи всі дані можна порахувати, що під час роботи ноутбук за годину споживав 50 Вт і по тарифу це виходить 0,22 грн/год. Вартість інтернету на годину становить 1,11 грн/год, яку було отримано поділивши місячну плату за інтернет на кількість годин проведених за роботою в одному місяці. Загальна вартість машино-години вийшла 1,33 грн/год.[19-20]

Трудомісткість розробки програмного забезпечення можна розрахувати за допомогою формули:

$$t = t_0 + t_u + t_а + t_n + t_{отл} + t_д \,, \qquad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ - витрати праці на дослідження алгоритму рішення задачі;

$t_а$ - витрати праці на розробку блок-схеми алгоритму;

$t_п$ - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p)\ , \qquad (3.2)$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Після підстановки значень в формулу (3.2) було отримано такий результат:

$$Q = 2483 \cdot 1{,}5 \cdot (1 + 0{,}1) = 4097$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\ , \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

Після підстановки значень в формулу (3.3) було отримано такий результат:

$$t_u = \frac{4097 \cdot 1{,}3}{76 \cdot 1{,}2} = 58 \text{ , людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_{\text{а}} = \frac{Q}{(20..25) \cdot \text{k}}, \tag{3.4}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Після підстановки значень в формулу (3.4) було отримано такий результат:

$$t_{\text{а}} = \frac{4097}{21 \cdot 1{,}2} = 163 \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі визначається за формулою:

$$t_n = \frac{Q}{(20..25) \cdot \text{k}}, \tag{3.5}$$

Після підстановки значень в формулу (3.5) було отримано такий результат:

$$t_n = \frac{4097}{23 \cdot 1{,}2} = 148 \text{ , людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5) \cdot \text{k}}, \tag{3.6}$$

Після підстановки значень в формулу (3.6) було отримано такий результат:

$$t_{\text{отл}} = \frac{4097}{4 \cdot 1{,}2} = 854 \text{ , людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 \cdot t_{\text{отл}} \text{ ,} \quad (3.7)$$

Після підстановки значень в формулу (3.7) було отримано такий результат:

$$t_{\text{отл}}^{k} = 1{,}5 \cdot 854 = 1281 \text{ , людино-годин.}$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}} \text{ ,} \quad (3.8)$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису;

$$t_{\text{др}} = \frac{Q}{(15..20) \cdot \text{k}} \text{ ,} \quad (3.9)$$

Після підстановки значень в формулу (3.9) було отримано такий результат:

$$t_{\text{др}} = \frac{4097}{16 \cdot 1{,}2} = 213 \text{ , людино-годин.}$$

де $t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\text{до}} = 0{,}75 \cdot t_{\text{др}} \text{ ,} \quad (3.10)$$

Після підстановки значень в формулу (3.10) було отримано такий результат:

$$t_{\text{до}} = 0{,}75 \cdot 213 = 160 \text{ , людино-годин.}$$

В результаті за формулою (3.8) було отримано такий результат:

$$t_{\text{д}} = 213 + 160 = 373 \text{ , людино-годин.}$$

Повертаючись до формули (3.1), було отримано повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 58 + 163 + 148 + 854 + 373 = 1646 \text{ , людино-годин.}$$

За результатами розрахунків, загальна трудомісткість розробки даного програмного застосунку складає 1646 людино-годин.

3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ ,} \quad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР} \text{ ,} \quad (3.12)$$

де t - загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

Після підстановки значень в формулу (3.12) було отримано такий результат:

$$З_{ЗП} = 1646 \cdot 505 = 831230 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_{мч}, \quad (3.13)$$

де $t_{отл}$– трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ – вартість машино-години ЕОМ, грн/год.

Після підстановки значень в формулу (3.13) було отримано такий результат:

$$З_{МВ} = 854 \cdot 1{,}33 = 1136 \cdot \text{грн.}$$

Звідси за формулою (3.11) розраховуємо витрати на створення програмного забезпечення:

$$K_{ПО} = 831230 + 1136 = 832366 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \quad (3.14)$$

де B_k – число виконавців (один розробник був відповідальний за розробку);

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 176 годин).

Після підстановки значень в формулу (3.14) було отримано такий результат:

$$T = \frac{1646}{1 \cdot 176} = 9{,}35 \text{ міс.}$$

Висновок: Для розробки гри «EnergyClicker» за підрахунками знадобиться 1646 людино-години. З урахуванням середньої заробітної плати програміста, на момент написання роботи, для розробки програмного забезпечення знадобиться 832366 грн. Приблизний час, який знадобиться для розробки складає 9,35 місяців при роботі одного програміста, зі стажем роботи 4 роки та тривалістю робочого часу приблизно 40 годин на тиждень.

ВИСНОВКИ

У цій роботі було розглянуто процес розробки двомірної гри «Energy Clicker» в жанрі «Інкрементальна гра» за допомогою рушію Unity. Були проаналізовані всі етапи розробки гри.

Використання ігрового рушія Unity сприяло ефективності та швидкості при створенні візуальних сцен, керуванні процесами, анімацією та аудіо. Розробка була детально описана з акцентом на ключові моменти програмування ігор. Також були розглянуті найбільш ефективні підходи до оптимізації та управління ресурсами для забезпечення високої продуктивності гри.

Як результат, була розроблена функціональна та візуально приваблива двомірна інкрементальна гра, яка включає базові елементи геймплею та пропонує можливості для подальшого розширення та поліпшення. Дипломна робота демонструє значення та можливості використання Unity у створенні високоякісних ігрових проектів, а також освітлює важливі аспекти створення двомірних інкрементальних ігор.

Розроблене програмне забезпечення дозволяє:

- купувати або видаляти частини реактора;
- покращувати частини реактора;
- змінювати загальну гучність;
- змінювати гучність музики;
- змінювати гучність звуків;
- змінювати швидкість перебігу часу;
- керувати ресурсами реактору такими як тепло, енергія та гроші.

Програма реалізована на мові програмування C# з використанням ігрового рушія Unity, середи розробки Visual Studio 2022, редактору зображень Adobe Photoshop.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ігор Бібічков. "Як розробляють ігри?". [Електронний ресурс] URL: https://lemon.school/blog/yak-rozroblyayut-igry (дата звернення: 11.05.2024)
2. Unity documentation. [Електронний ресурс] URL: https://docs.unity3d.com/Manual (дата звернення: 09.01.2024)
3. StackOverflow. [Електронний ресурс] URL: https://stackoverflow.com (дата звернення: 10.02.2024)
4. GitHub Documentation. [Електронний ресурс] URL: https://docs.github.com (дата звернення: 11.03.2024).
5. Photoshop Documentation. [Електронний ресурс] URL: https://helpx.adobe.com/photoshop/user-guide.html (дата звернення: 18.02.2024).
6. C# documentation. [Електронний ресурс] URL: https://learn.microsoft.com/en-us/dotnet/csharp (дата звернення: 04.02.2024).
7. Design patterns. [Електронний ресурс] URL: https://refactoring.guru/design-patterns (дата звернення: 24.01.2024).
8. Олег Топорков «Як створити свою власну гру?». [Електронний ресурс] URL: http://i.nure.ua/tekhnologiji/1886-yak-stvoriti-svoyu-vlasnu-gru (дата звернення: 30.05.2024).
9. «Як розробляються ігри для телефонів». [Електронний ресурс] URL: https://foxminded.ua/yak-stvoryty-hru-na-telefon (дата звернення: 18.02.2024).
10. Unity (game engine). Wikipedia. [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Unity_(game_engine) (дата звернення: 13.03.2024).
11. Smith G. Basic Math for Game Development with Unity 3D / G. Smith, K. Sung., 2018.
12. Ferrone H. Learning C# by Developing Games with Unity 2019: Code in C# and build 3D games with Unity, 4th Edition / Harrison Ferrone., 2019.
13. Halpern J. Developing 2D Games with Unity: Independent Game Programming with C# / Jared Halpern., 2018.

14. Хокінг Д. М. Unity в дії. Мультиплатформенна розробка на практиці. / Д. М. Хокінг, 2016.

15. Buckland Mat. Programming Game AI by Example – Texas, Wordware Publishing. – 2004.

16. Ігровий движок Unity: що це таке та на що він здатний?. [Електронний ресурс] URL: https://itproger.com/ua/news/igrovoy-dvizhok-unity-razbiraemsya-chto-k-chem (дата звернення: 29.05.2024).

17. Каталог патернів проектування. [Електронний ресурс] URL: https://refactoring.guru/uk/design-patterns/catalog (дата звернення: 29.05.2024).

18. Скільки отримує Unity developer в Україні? [Електронний ресурс] URL: https://ua.jooble.org/salary/unity-developer (дата звернення: 06.06.2024).

19. Офіційний курс гривні щодо долара США. Національний банк України. [Електронний ресурс] URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD (дата звернення: 11.06.2024).

20. Тариф на електроенергію. Yasno. [Електронний ресурс] URL: https://yasno.com.ua/b2c-tariffs (дата звернення: 11.06.2024).

ДОДАТОК А

КОД ПРОГРАМИ

Reactor.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.Mathematics;
using UnityEngine;
using UnityEngine.Events;

public class Reactor : Inventory
{
    [SerializeField] private string _fileNameReactor;
    [SerializeField] private float _timeTick;
    [SerializeField] private double _heat;
    [SerializeField] private double _maxHeat;
    [SerializeField] private double _heatPerClick;
    [SerializeField] private double _power;
    [SerializeField] private double _maxPower;
    [SerializeField] private double _powerPerUpdate;
    [SerializeField] private double _money;
    [SerializeField] private float _time;
    [SerializeField] private float _timePerUpdate;

    public Dictionary<int2, ReactorItem> Items = new Dictionary<int2, ReactorItem>();
    public List<ReactorComponent> Queue = new List<ReactorComponent>();

    public double Heat { get => _heat; set { _heat = Math.Max(0, value); HeatChanged?.Invoke(); } }

    public double MaxHeat { get => _maxHeat; set { _maxHeat = value; HeatChanged?.Invoke(); } }

    public double Power { get => _power; set { _power = Math.Clamp(value, 0, _maxPower); PowerChanged?.Invoke(); } }

    public double MaxPower { get => _maxPower; set => _maxPower = value; }

    public double Money { get => _money; set { _money = value; MoneyChanged?.Invoke(); } }

    public float Time { get => _time; set { _time = value; TimeChanged?.Invoke(); } }

    public float Hours => Mathf.Floor(Time / 60 / 60);

    public float Minutes => Mathf.Floor(Time / 60) - Hours * 60;

    public float Seconds => Time - Hours * 60 * 60 - Minutes * 60;

    public float TimePerUpdate { get => _timePerUpdate; set => _timePerUpdate = value; }

    protected override bool _invoced => false;

    private void Start()
```

```
{
    StartCoroutine(TickUpdate());
    Load();
}

private void OnApplicationQuit()
{
    Save();
}

public void SellAllPower()
{
    Money += Power;
    Power = 0;
    Updated?.Invoke();
}

public void RemoveHeat()
{
    Heat -= _heatPerClick;
    Updated?.Invoke();
}

private IEnumerator TickUpdate()
{
    while (true)
    {
        for (int i = 0; i < Queue.Count; i++)
        {
            Queue[i].Update();
        }
        if (Heat >= MaxHeat)
        {
            foreach (var item in Items.Keys.ToList())
            {
                Remove(Items[item]);
            }
        }
        _time += _timePerUpdate;
        Updated?.Invoke();
        yield return new WaitForSeconds(_timeTick);
    }
}

public bool TryGetItem(int2 position, out ReactorItem reactorItem)
{
    return Items.TryGetValue(position, out reactorItem);
}

public override int Add(SavedObject value)
{
    var index = base.Add(value);
    if (index >= 0)
    {
        if (value is ReactorItem reactorItem)
```

```
        {
            if (Items.ContainsKey(reactorItem.Position))
            {
                Remove(Items[reactorItem.Position]);
                return Add(value);
            }
            else
            {
                _savedObjects.Add(value);
                reactorItem.Reactor = this;
                Items.Add(reactorItem.Position, reactorItem);
                for (int i = 0; i < reactorItem.Components.Count; i++)
                {
                    Queue.Add(reactorItem.Components[i]);
                    reactorItem.Components[i].Start();
                }
                Queue = Queue.OrderBy(k => k.OrderUpdate).ToList();
                Added?.Invoke(value);
                Changed?.Invoke();
            }
        }
    }
    return index;
}

public override int Remove(SavedObject value)
{
    var index = base.Remove(value);
    if (index >= 0)
    {
        if (value is ReactorItem reactorItem)
        {
            if (Items.TryGetValue(reactorItem.Position, out var item))
            {
                Items.Remove(reactorItem.Position);
                int i = 0;
                while (i < Queue.Count)
                {
                    if (ReferenceEquals(Queue[i].reactorItem, item))
                    {
                        Queue[i].OnDestroy();
                        Queue.RemoveAt(i);
                    }
                    else
                    {
                        i++;
                    }
                }
            }
        }
    }
    return index;
}

public override void Load()
```

```
    {
        Items.Clear();
        Queue.Clear();
        var a = new ReactorSaveData();
        if (SaveHelper.TryLoad(SaveHelper.GetFilePathJSON(nameof(Inventory), _fileNameReactor), ref a))
        {
            _heat = a.Heat;
            _power = a.Power;
            _money = a.Money;
            _time = a.Time;
        }
        base.Load();
    }

    public override void Save()
    {
        var a = new ReactorSaveData()
        {
            Heat = _heat,
            Power = _power,
            Money = _money,
            Time = _time
        };
        SaveHelper.Save(SaveHelper.GetFilePathJSON(nameof(Inventory), _fileNameReactor), a);
        base.Save();
    }

    public struct QueueKay : IComparable<QueueKay>
    {
        public int Order;
        public int Index;

        public int CompareTo(QueueKay other)
        {
            return Order.CompareTo(other.Order);
        }
    }

    [Serializable]
    private struct ReactorSaveData
    {
        public double Heat;
        public double Power;
        public double Money;
        public float Time;
    }

    public UnityEvent Updated;
    public UnityEvent HeatChanged;
    public UnityEvent PowerChanged;
    public UnityEvent MoneyChanged;
    public UnityEvent TimeChanged;
}
```

ReactorComponent.cs

```
using System;

[Serializable]
public abstract class ReactorComponent : SavedObject
{
   [NonSerialized] public ReactorItem reactorItem;
   public int OrderUpdate;

   public virtual void Start()
   {
      Updated?.Invoke();
   }

   public virtual void OnDestroy()
   {
      Updated?.Invoke();
   }

   public virtual void Update()
   {
      Updated?.Invoke();
   }

   public Action Updated;
}
```

UpdaterBuyer.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UIElements;
using static UnityEngine.AdaptivePerformance.Provider.AdaptivePerformanceSubsystemDescriptor;

public class UpdaterBuyer : MonoBehaviour
{
   [SerializeField] private GridSpawner _updaters;
   [SerializeField] private Reactor _reactor;
   [SerializeField] private Vector2Int _position;
   [SerializeField] private RectTransform _selectViewer;

   private RectTransform _spawnedSelectViewer;
   private UpdaterViewer _updaterViewer;

   private RectTransform SpawnedSelectViewer => _spawnedSelectViewer ??= Instantiate(_selectViewer, _updaters.CellSpawner.transform);

   public Vector2Int Position
   {
      get => _position;

      set
      {
         _position = value;
```

```
            if (_updaters.Items.TryGetValue(value, out var item))
            {
                if (item.TryGetComponent<UpdaterViewer>(out var viewer))
                {
                    if (viewer.Updater.IsMaxLevel == false)
                    {
                        _updaterViewer = viewer;
                        _updaters.CellSpawner.SetCell(value.x, value.y, SpawnedSelectViewer);
                        CanBuy?.Invoke();
                        DescriptionChanged?.Invoke(viewer.Updater.Description);
                        return;
                    }
                }
            }
            _updaterViewer = null;
            Close();
        }
    }

    public void Buy()
    {
        if (_updaterViewer)
        {
            _reactor.Money -= _updaterViewer.Updater.Cost;
            _updaterViewer.Updater.Level++;
            _updaterViewer.UpdateDictionary();

            if (_reactor.Money >= _updaterViewer.Updater.Cost)
            {
                if (_updaterViewer.Updater.IsMaxLevel == false)
                {
                    CanBuy?.Invoke();
                    DescriptionChanged?.Invoke(_updaterViewer.Updater.Description);
                    return;
                }
            }
        }
        Close();
    }

    public void Close()
    {
        Destroy(SpawnedSelectViewer.gameObject);
        _spawnedSelectViewer = null;
        NotCanBuy?.Invoke();
        _position = -Vector2Int.one;
    }

    public UnityEvent CanBuy;
    public UnityEvent NotCanBuy;
    public UnityEvent<Dictionary<string, string>> DescriptionChanged;
}
```

Решта коду додається окремо на CD-диску.

ДОДАТОК Б

ВІДГУК

КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

на кваліфікаційну роботу бакалавра на тему:

«Розробка мобільної гри з використанням рушія Unity»

Студента групи 122-20-3 Лисенка Андрія Олеговича

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Лисенко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Лисенко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Лисенко.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Лисенко.ppt	Презентація кваліфікаційної роботи