**Форма № Н-6.01у**

Кременчуцький національний університет імені Михайла Остроградського
(повне найменування вищого навчального закладу)
Кафедра автоматизації та інформаційних систем
(повна назва кафедри, циклової комісії)

# КУРСОВИЙ ПРОЄКТ (РОБОТА)

з дисципліни «Сучасні мови об'єктно-орієнтованого програмування»
(назва дисципліни)

на тему Ком'ютерна відеогра «The Brave Man»

Студента 2 курсу КН–23–1 групи
Ступінь вищої освіти «Бакалавр»
(бакалавр, магістр)
Спеціальність 122 – «Комп'ютерні науки»
Освітньо-професійна програма
«Комп'ютерні науки»
Полинько І. М.
(прізвище та ініціали)
Керівник старший викладач кафедри АІС
Бельська В. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____. Оцінка: ЄКТС ____

Члени комісії _____. В. Ю. Бельська
(підпис) (ініціали та прізвище)
_____ Н. М. Істоміна
(підпис) (ініціали та прізвище)
_____ О. В. Кліменко
(підпис) (ініціали та прізвище)

м. Кременчук 2024 рік

**Форма № Н-9.01у**

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра автоматизації та інформаційних систем

Дисципліна «Сучасні мови об'єктно-орієнтованого програмувння»

Освітній ступінь «Бакалавр»

Спеціальність 122 – «Комп'ютерні науки»

Освітня програма «Комп'ютерні науки»

Курс 2 група КН-23-1 семестр 3

# ЗАВДАННЯ
# НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Полинько Ігор Миколайович
(прізвище, ім'я, по-батькові)

1. Тема роботи: Ком'ютерна відеогра «The Brave Man»
2. Термін здачі студентом роботи: 5 грудня 2024 р
3. Вихідні дані до роботи: *.png, *.jpg, *.anim, *.cs, *.mp3, *.png, *jpeg
4. Зміст пояснювальної записки: (перелік питань, що підлягають розробці): постановка завдання, аналіз технічного завдання, розробка інтерфейсу, графіки та створення основного програмного коду гри, структура програмного забезпечення.
5. Перелік графічного матеріалу:
6. Дата видачі завдання: 09.09. 2024 р.

# КАЛЕНДАРНИЙ ПЛАН

| № пор. | Назва етапів курсового проекту | Терміни виконання етапів проекту | Вказівки та зауваження викладача (з зазначенням дати консультації) | Оцінювання етапів проекту | | |
|---|---|---|---|---|---|---|
| | | | | за національною шкалою | за шкалою ЄКТС | кількість балів |
| 1 | Етап 1<br>Аналіз предметної області | 05.10.24-<br>25.10.24 | | | | |
| 2 | Етап 2<br>Створення моделі даних | 16.10.24-<br>28.10.24 | | | | |
| 3 | Етап 3<br>Розробка інтерфейсу, графіки та створення основного програмного коду гри | 29.10.24-<br>24.11.24 | | | | |
| 4 | Етап 4<br>Тестування програмного коду | 16.10.24-<br>29.11.24 | | | | |
| 5 | Етап 5<br>Оформлення пояснювальної записки | 25.11.24-<br>31.11.24 | | | | |
| 6 | Етап 9<br>Захист | 05.12.24 | | | | |
| | Разом | 8 тижнів | | | | |

Студент _____
(підпис)

Керівник _____ _____В. Ю. Бельська____
(підпис) (ініціали та прізвище)

«1» жовтня 2024 р.

# РЕФЕРАТ

Курсова робота містить 55 сторінок, 2 розділи, 25 рисунків, 1 таблиця, 6 використаних джерел.

Об’єкт розробки – Відеогра «The Brave Man».

Мета: створення комп’ютерної 2D відеогри «The Brave Man» на рушії Unity.

Під час виконання завдання, поставленого на курсову роботу, було виконано аналіз алгоритму гри «The Brave Man». Визначено основні функціональні та не функціональні вимоги до гри, що створюється. Побудована модель даних. Створити функціонал, що дозволить вирішити наступні задачі: розробка алгоритму роботи інтерфейсу, персонажів, організації ігрових подій, досягнень, розблокування наступних подій, розробка системи роботи анімацій, розробка системи нанесення та отримання урону, створення графіки для інтерфейсу та самих ігрових подій.

В якості мови програмування для створення відеогри буде використовуватися скриптовий діалект мови C# та ігровий рушій Unity. Для створення візуальної частини відеогри буде використовуватися додаток для редагування Aseprite.

Результатом виконання всіх етапів є 2D відеогра, яка призначена для розваг та відпочинку, а також тренування стратегічних та моторних навичок гравців, поліпшення логічного мислення.

ВІДЕОГРА, АЛГОРИТМ, ГРАФІКА, ПОДІЇ

# ЗМІСТ

# ВСТУП

В сучасному світі комп’ютерні технології використовуються майже у всіх сферах нашого життя. Однією з провідних галузей застосування є розважальна галузь. Разом із появою комп’ютерів з’явилися і комп’ютерні ігри, які одразу ж знайшли масу шанувальників. В 2024 році відеоігри вже давно перестали існувати лише для розваг та безцільного витрачення часу, нині ігри – це потужний інструмент для навчання в невимушеній формі.

Відеогра — це електронна розважальна гра, де гравець використовує ігрову платформу та взаємодіє з нею за допомогою ігрового контролера, такого як джойстик, клавіатура, геймпад або сенсорний екран.

На сьогоднішній день ігрова індустрія розвивається неймовірно стрімко, кількість активних гравців зростає з кожним роком, як і прибутки від продажу ігор.

З кожним роком ігри стають все кращими з технічної сторони та все більше приваблюють інвесторів, які бачать в відеоіграх нові золоті гори. Відеоігри стають комерційними продуктами, особливо ті, що є головними та найдорожчими на ринку. В них усе менше вкладено душі розробників та все більше методів заробітку на гравцях, особливо якщо це «Free to play» відеоігри. Цінники нових відеоігор зростають, але якість інколи може лише знижуватися, бо студії, що розробляють їх готові жертвувати репутацією заради коштів.

Користь від комп’ютерної гри полягає у розвитку уваги, моторики, швидкості реакції, сприйняття та креативності. Комп’ютерні ігри допомагають набувати спеціальні знання та розвивають когнітивні навички, що входять у загальну структуру життєдіяльності. Ігровий досвід розширює і поліпшує низку умінь і навичок, важливих для подальшого навчання, для професійної діяльності та соціальної взаємодії. Динаміка сучасних комп’ютерних ігор навчає швидко сприймати незнайомі ситуації й адаптуватися до них. В умовах сучасного суспільства, що бурхливо змінюється, розвинена інтелектуальна гнучкість забезпечить пристосування до нових та несподіваних реалій життя.

Відеоігри сприяють розвитку навичок людей. Наприклад, стратегічні ігри вимагають від гравців аналітично мислити, оцінювати різноманітні ситуації та їх комбінації, а також приймати важливі рішення, що можуть бути корисними у реальному житті. Відеоігри-головоломки розвивають логічне мислення, відеоігри-шутери покращають точність та реакцію.

Unity — це один із найпопулярніших рушіїв для розробки ігор, який відомий своєю гнучкістю, зручністю використання та потужним набором інструментів. Він підтримує створення ігор для широкого спектра платформ, включаючи ПК, консолі, мобільні пристрої та навіть віртуальну реальність. Unity вирізняється середовищем для розробки, яке поєднує візуальний редактор з можливістю програмування на C#. Його популярність зумовлена невисоким порогом входу для початківців, активною спільнотою розробників та великим маркетплейсом, де можна знайти готові ресурси, плагіни та інструменти. Unity часто використовують як для інді-проєктів, так і для комерційних AAA-ігор, завдяки чому рушій став ключовим інструментом у сучасній ігровій індустрії.

Мобільні ігри дозволяють грати в будь-який час і в будь-якому місці, не заважаючи і не займаючи багато місця, допомагають «вбити час» або повноцінно насолодитися гарною відеогрою. Кросс-платформеність відеоігр дозволяє насолоджуватися ними ще більшому кругу людей.

Отже, відеоігри популярні з багатьох причин і загалом приносять користь більшості осіб, включаючи гравців, розробників і тих, хто заробляє на іграх. Вони дарують чудовий досвід та соціальну взаємодію. Ніщо не є ідеальним, тож і відеоігри можуть завдати шкоди тим, хто не вміє зупинятися.

Метою даного курсового проекту є проектування та розробка комп’ютерної гри «The Brave Man».

# 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Делегати. Стандартні делегати. Для чого використовуються.

Делегати в мові програмування C# є особливим типом, який дозволяє зберігати посилання на методи з певною сигнатурою. Це дозволяє програмам динамічно викликати різні методи під час виконання, не прив'язуючись до їхніх імен. Делегати часто використовуються у тих випадках, коли необхідно передавати методи як параметри, викликати їх у відповідь на певні події або змінювати поведінку програми залежно від ситуації.

Делегати дозволяють значно спростити реалізацію динамічної поведінки в програмі. Вони визначають тип даних, що відповідає сигнатурі методів, які можуть бути викликані.

Делегати забезпечують:

- Гнучкість у програмуванні, адже методи можуть бути прив'язані до делегата під час виконання програми.
- Можливість роботи з методами без прямого посилання на них.
- Відсутність необхідності знати точну реалізацію методу для його використання.

C# має вбудовані стандартні делегати, які спрощують розробку:

1. Action: використовується для методів, які не повертають значення, але можуть приймати параметри. Прототип: Action<int, string>. Даний Action може приймати два параметри: число і текст, але не повертає значення.
2. Func: підходить для методів, які повертають значення та можуть мати параметри. Прототип: Func<int, string, bool>. Цей Func приймає два параметри (число та текст) і повертає логічне значення. Може використовуватися, наприклад, для перевірки, чи виконав ворог певну дію успішно.
3. Predicate: застосовується для методів, які повертають значення типу bool та використовуються для перевірки умов. Прототип: Predicate<string>.

Predicate приймає один параметр типу string (наприклад, ім'я ворога) і повертає bool, що може використовуватися для перевірки умов, таких як "чи активний ворог?".

У відеоіграх делегати відіграють важливу роль у створенні динамічних систем. Наприклад:

- Обробка подій: делегати є основою механізму подій у C#, що дозволяє реалізувати системи взаємодії в грі, наприклад, обробку кліків мишкою, натискання клавіш, зіткнення об’єктів або запуск сценаріїв під час виконання.
- Гнучке управління ігровими об'єктами: завдяки делегатам можна динамічно змінювати поведінку персонажів або інших елементів гри, наприклад, у відповідь на дії гравця або події в ігровому світі.
- Організація архітектури: делегати сприяють модульності та масштабованості коду, дозволяючи відокремлювати логіку різних компонентів і спрощувати взаємодію між ними.
- Обробка станів: у складних ігрових механіках делегати можуть використовуватися для управління переходами між станами гри, як-от зміна рівнів, активація бонусів чи обмеження дій гравця.

Прикладом використання делегатів може виступати наступна ситуація: у грі є кілька видів ворогів, які повинні реагувати на звук тривоги. Кожен тип ворога має свою поведінку: одні атакують, інші тікають, а ще деякі ховаються. За допомогою делегата можна зберігати методи для кожної поведінки та викликати їх динамічно залежно від ситуації. Наприклад, делегат може бути прив’язаний до відповідного методу для конкретного типу ворога, який виконається після активації тривоги.

Таким чином, делегати є важливим інструментом у розробці відеоігор, забезпечуючи гнучкість, зручність і легкість у створенні інтерактивних і динамічних механік. Їх використання дозволяє спрощувати складні сценарії та зменшувати залежність між компонентами програми, що сприяє побудові надійної архітектури гри.

## 1.2 Аналіз технічного завдання на роботу

Мета курсового проєкту – створення комп’ютерної 2D відеогри «The Brave Man» на рушії Unity.

Під час розробки відеогри були висунуті наступні вимоги:

### 1.2.1 Функціональні вимоги

1. Система гри має містити декілька рівнів;
2. Геймплей передбачає покращення досвіду гравця, збільшення складності та розблокування навичків герою;
3. Створити сюжет, що містить елементи геймплею;
4. Звукове та графічне супроводження;
5. Наявність налаштувань ігрового процесу.
6. Наявність досягнень та меню.
7. Інформація про гру та розробника. Наявність лого гри.

### 1.2.2 Нефункціональні вимоги

1. Звукове та графічне супроводження мають визначати сеттінг гри та допомагати гравцю погрузитися і зрозуміти те, що відбувається під час ігрового процесу. Для створення графіки у грі буде використаний Aseprite — популярний редактор піксельної графіки, який спеціалізується на створенні спрайтів, анімацій та ретро-артів. Він підтримує експортування в різні формати, зокрема спрайт-листи й анімовані GIF-файли, що робить його ідеальним інструментом для створення графіки в стилі піксель-арт. Програма має великий набір інструментів для малювання, таких як пензлі, заливки та ефекти, а також підтримує налаштування робочого простору під потреби користувача;
2. Ігровий процес має передбачати розвиток герою, що включає поліпшення умінь головного герою та гарно поставлений дизайн рівнів, завдяки

яким, гравець самостійно з кожної спроби буде все краще розуміти та використовувати арсенал вмінь для супротиву більш складним ворогам.

3. Сумісність з великою кількістю комп'ютерних збірок, невеликі системні потреби;

4. Усі поля введення повинні бути захищені від некоректного введення.

### 1.3 Опис алгоритму основних задач/підзадач у роботі

На рисунку 1.1 зображене загальне представлення алгоритму основних задач у додатку:

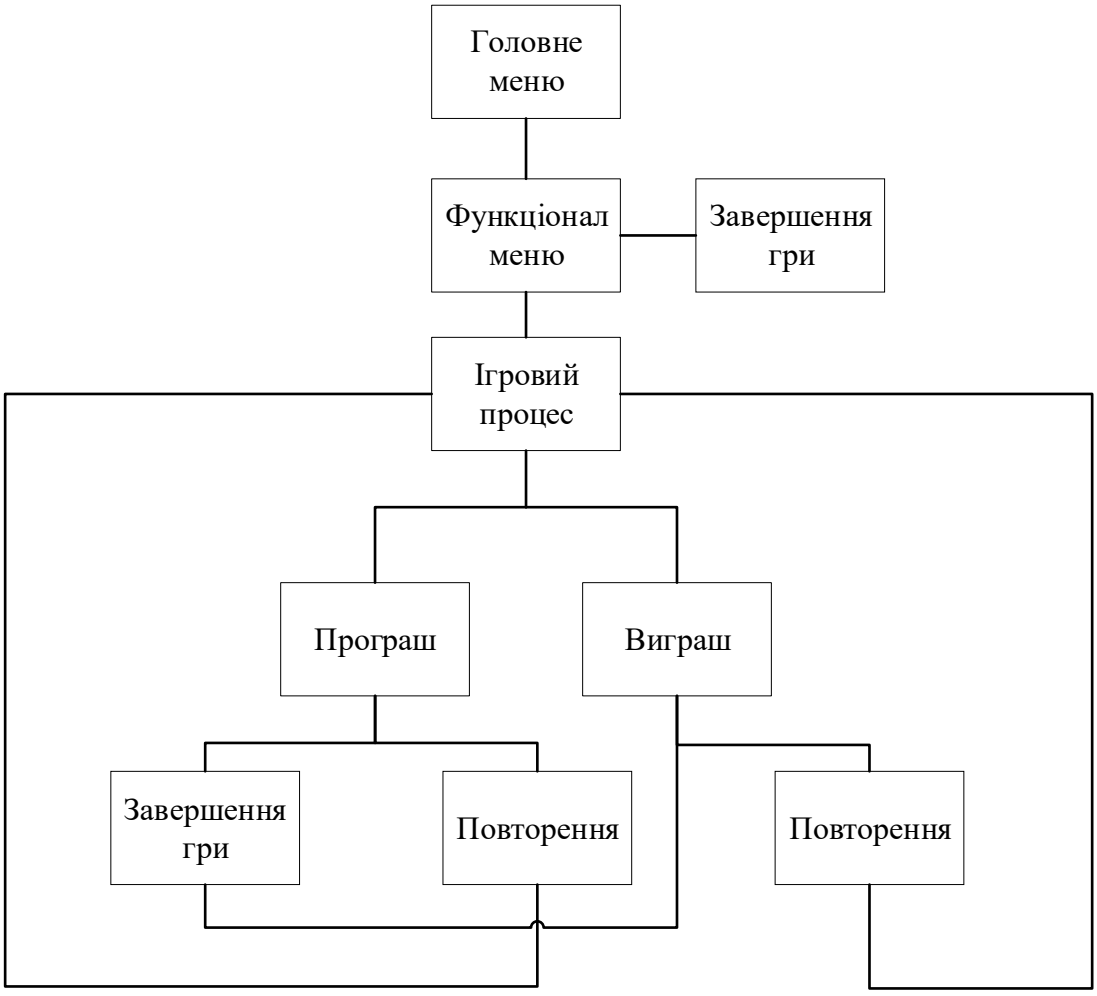


Рисунок 1.1 – Діаграма загальньного представлення алгоритму основних задач

Гра має складатися з декількох сцен меню: «Головне меню», «Налаштування», «Вибір рівня», «Досягнення», «Як грати» та «Розробник».

У головному меню гравець повинен мати змогу переходити до будь-якої частини гри, а також переглядати відомості про розробника та про саму гру.

Меню вибору рівня має надавати можливість вибору одного з трьох рівнів до проходження, а також кнопки для читання записок з детальним описом сюжету.

Меню налаштувань має дозволяти змінювати гучність усієї гри, передивлятися управління, а також скидати увесь прогрес проходження гри, враховуючи здобуті комбо-атаки, пройдені рівні та зароблені досягнення. Всі налаштування та прогрес у грі мають зберігатися в конфігурації гри та відтворюватися після кожного запуску гри поки не будуть видалені гравцем у меню налаштувань.

Кнопка вихід у головному меню повинна надати можливість вийти з гри.

**Висновки до розділу**

Було розглянуто основні аспекти використання делегатів у мові програмування C#, включаючи їхню роль у створенні динамічної поведінки та гнучкості в коді. Проаналізовано стандартні делегати (Action, Func, Predicate) та їхнє практичне застосування в різних сценаріях, зокрема в ігрових проєктах, де делегати використовуються для обробки подій, управління ігровими об'єктами та організації архітектури програми.

Також проаналізовано технічне завдання для створення комп’ютерної 2D відеогри «The Brave Man» на рушії Unity. Розглянуто функціональні й нефункціональні вимоги, зокрема, підтримку розвитку персонажа, інтерактивність геймплею, якісний графічний та звуковий супровід, а також сумісність із різними системами. Для розробки графіки передбачено використання редактора Aseprite. Крім того, описано структуру гри та алгоритми основних задач, які забезпечують зручність ігрового процесу та налаштувань для користувача.

# 2 ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Структура програмного забезпечення

Застосунок створений за допомою рушію Unity на базі мови C#.

На рисунку 2.1 зображено оглядач рішень програмного застосунку.

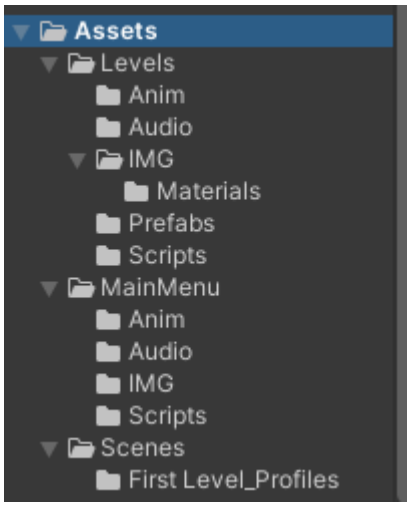


Рисунок 2.1 – Оглядач рішень

В оглядачі рішень створено трі основні папки: рівні, головне меню та сцени. У папках сцени містяться усі сцени гри. Папки рівні та головне меню мають однакову структуру, але поділені між собою для зручної роботи при розробці додатку.

Під час розробки гри було створено 7 сцен, які зображені на рисунку 2.2.

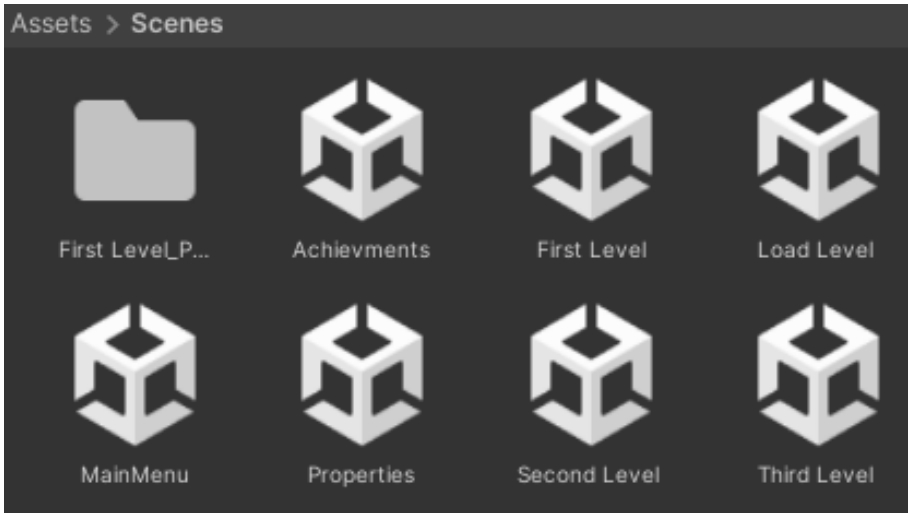


Рисунок 2.2 – Сцени гри

Усі скрипти гри зображено на рисунку 2.3 і рисунку 2.4.


Рисунок 2.3 – Скрипти для меню


Рисунок 2.4 – Скрипти для рівнів

В таблиці 2.1 представлено опис скриптів проекту курсової роботи

Таблиця 2.1

| **Назва скрипта** | **Призначення скрипта** |
|---|---|
| ExitGame.cs | Скрипт, що дозволяє покинути гру. |
| MainMenuButtons.cs | Скрипт, що відповідає за перехід між сценами головного меню і призначається на різні кнопки, з вказанням потрібного номеру сцени для переходу на кожній з кнопокю |
| SoundManager.cs | Скрипт, який виступає менеджером звуків та музики. |
| SceneController.cs | Скрипт, який здійснює перехід між сценами через анімацію затухання. Працює разом з Canvas панеллю на яку накладена анімація. |
| GlobalVolumeControl.cs | Прирівнює усім звукам гри встановлене гравцем значення гучності у налаштуваннях. |
| VolumeSlider.cs | Скрипт, що відноситься до слайдеру гучності усіх звуків гри, який розташований у меню налаштувань. Зчитує дані слайдеру і робить зміни до GlobalVolumeControl.cs. |
| AchievmentManager.cs | Група скриптів до кожного із досягнень, що регулюють відображення стану досягнення у меню досягнень. Приймає значення про виконане досягнення від групи скриптів AchievmentCompleted.cs, що працюють на рівнях гри. |
| LevelLoad.cs | Скрипт, що завантажує рівень. На відміну від SceneController працює без анімації переходу, щоб уникнути неможливості проведення стратегічного проходження рівню. |

Продовження таблиці 2.1

| Назва модуля | Призначення модуля |
|---|---|
| OpenZapus.cs | Скрипт, що контролює відкриття сюжетних записок по мірі проходження рівнів. |
| ShowingLockedLvlIMG.cs | Скрипт, що відображає картинку заблокованого рівня. Коли гравець проходить минулий рівень, цей скрипт вимикає картинку. |
| ShowLvlIMG.cs | Скрипт, що відображає картинку разблокованого рівня. Картинка разблокованого рівня містить скрипт LoadLevel.cs. |
| EndWindow.xaml | Вікно закінчення гри |
| AchievmentCompleter.cs | Група скриптів, що перевіряють досягнення. |
| UnlockLvl.cs | Група скриптів до кожного другого і третього рівнів, що розблоковують до них доступ при завершенні минулого. |
| AchievmentIMG.cs | Група скриптів, що відображають повідомлення про виконане досягнення. |
| TheBraveMan.cs | Скрипт головного герою, що містить багато функцій, завдяки яким можна керувати героєм. |
| Dart.cs | Скрипт, що керує об’єктом «Дротик», який викликає гравець натисканням на відповідну клавішу. |
| Enemy.cs | Скрипт злодію ближнього бою, що містить багато функцій. |
| Enemy2.cs | Скрипт злодію далекого бою, що містить інші функції. |
| Pause.cs | Скрипт, що заморожує час у грі и викликає панель паузи. При повторному натисканні клавіші відновлює гру. |

Продовження таблиці 2.1

| Назва модуля | Призначення модуля |
|---|---|
| PauseStopper.cs | Скрипт, що відновлює гру. Накладається на кнопку. |
| RefreshTime.cs | Скрипт, що відновлює час при натиснені гравцем кнопки «Повторити спробу» на панелі програшу. |
| LevelManager.cs | Група скриптів до кожного рівня, які створюють події і дають можливість завершити рівень. |
| HumanCircle.cs | Скрипт, що вмикає червоне коло на другому рівні коли гравець бере на руки потерпілу жертву і вимикає його при донесенні потерпілого до меж кола. |

Фрагменти скриптів для ворогів у додатку А, механіки герою у додатку Б, внутрішні функції гри у додатку В, менеджери рівнів гри у додатку Г.

### 2.2 Опис роботи програми

Після запуску гри відкривається головне меню, з якого можна перейти у наступні вікна: вибір рівню, досягнення, налаштування, про розробника/гру, а також вийти з гри.

У меню досягнень можна передивитися отримані гравцев досягнення.

У меню налаштувань можна змінити гучність гри, видалити прогрес, а також передивится керування гри.

У меню вибору рівнів можна ознайомуватися з сюжетними записами по мірі проходження рівнів, а також запустити доступний рівень.

На рівні гравець керує героєм, що може бігати ліворуч\праворуч, стрибати двійним стрибком та битися рукою і ногою. Гра має систему комбо, прийоми якої можна побачити, читаючи сюжетні записи. Також герой може стріляти

обмеженою кількістю дротиків, що наносять дуже високий урон. Їх кількість зображена у лівому нижньому кутку екрану.

Кожен рівень складається з 2D коридору, на якому при виконанні дій будуть з’являтися нові вороги, а після перемоги над усіма – відкривається маркер завершення рівню.

На рівні є два типи ворогів: вороги ближнього бою та далекого бою. Перший тип спроможний йти до героя і атакувати його, якщо той знаходиться у полі зору першого. Другий тип не може рухатися та стріляє по герою кожні декілька секунду, якщо той також знаходиться у полі зору.

Також на деяких рівнях є інтерактив, відмінний від бою, такий як рятування людей на другому рівні.

З кожного ворога при перемозі над ним випадає серце, що додає десять до здоров’я гравця, яке вказано в лівому верхньому кутку екрану.

При виконання будь-якого з досягнень – гра попереджає про це панелькою у правому нижньому кутку.

Якщо здоров’є гравця закінчується – він програє і має можливість почати спочатку. Якщо виграє – може перейти до меню вибору рівнів, де буде розблокований новий рівень і нова сюжетна записка.

Також можна вставити рівень на паузу через клавішу «Esc».

Інтерфейс гри має різноманітні підсвічування об’єктів взаємодії у меню, а також плавні переходи між меню. Кожна сцена супроводжується музикою та звуками. Усі дії гравця і ворогів мають звукове сопроводження.

Візуальний стиль виконаний самостійно та виражає кожний елемент гри для зрозумілості того, що відбувається на екрані та передавання стилю бачення атмосфери і сюжету гри. Гра містить багато анімацій для усіх елементів рівнів.

На рисунку 2.5 наведена робота програми:

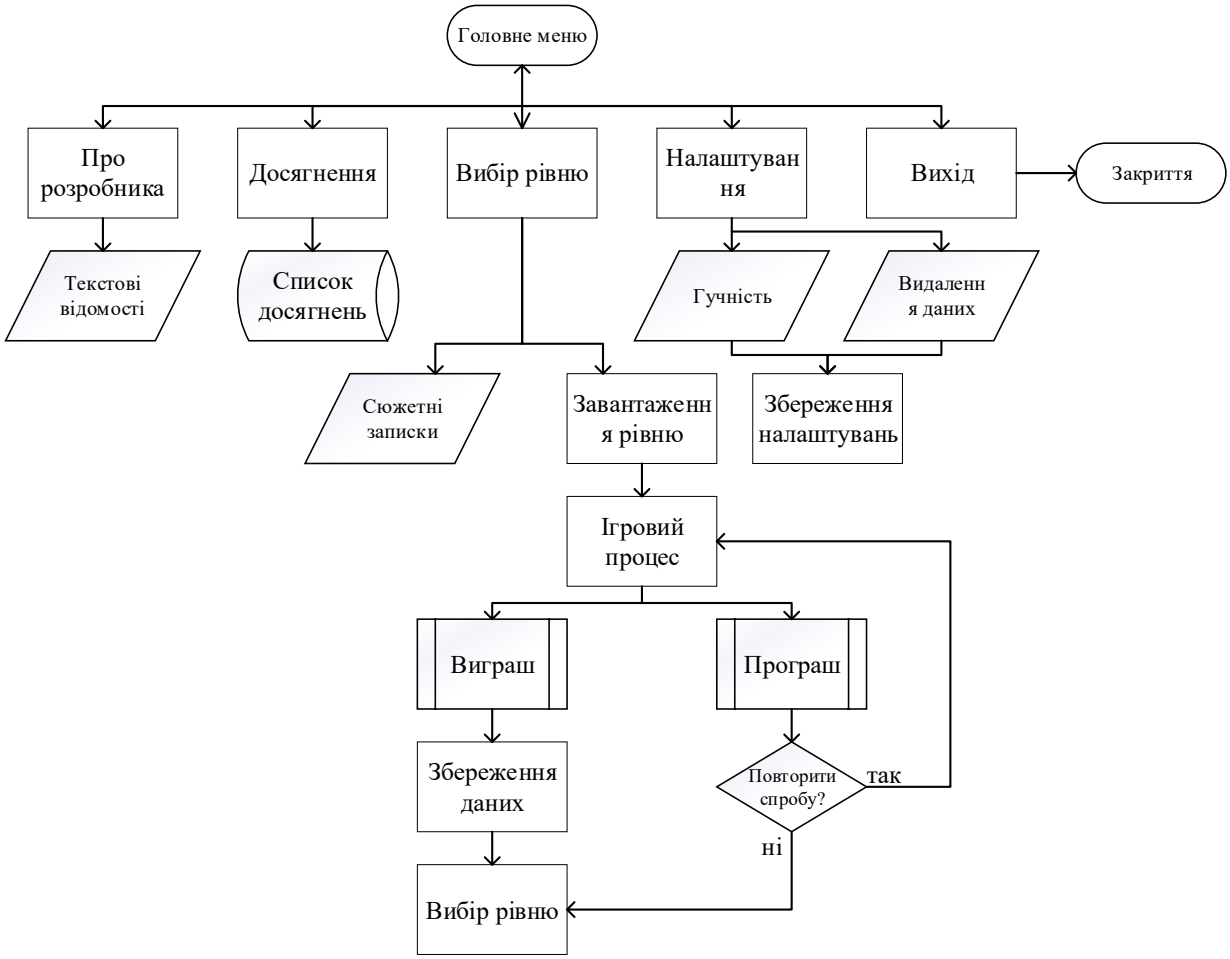


Рисунок 2.5 – Діаграма роботи програми

## 2.3 Функціональна схема програми

На рисунку 2.6 показано функціональну схему програми, основні процеси, що відбуваються при роботі додатку.

Як видно з рисунку 2.6 після запуску гри першим відображається вікно з головним меню, де є можливість перейти у меню налаштувань, у меню досягнень, покинути гру, а також перейти до вибору рівнів. Меню налаштувань містить свої прецеденти в якості зміни гучності та видалення прогресу. Окрім гравця, діаграма містить актора «Система». Вона організовує роботу рівнів та збереження даних скриптами.

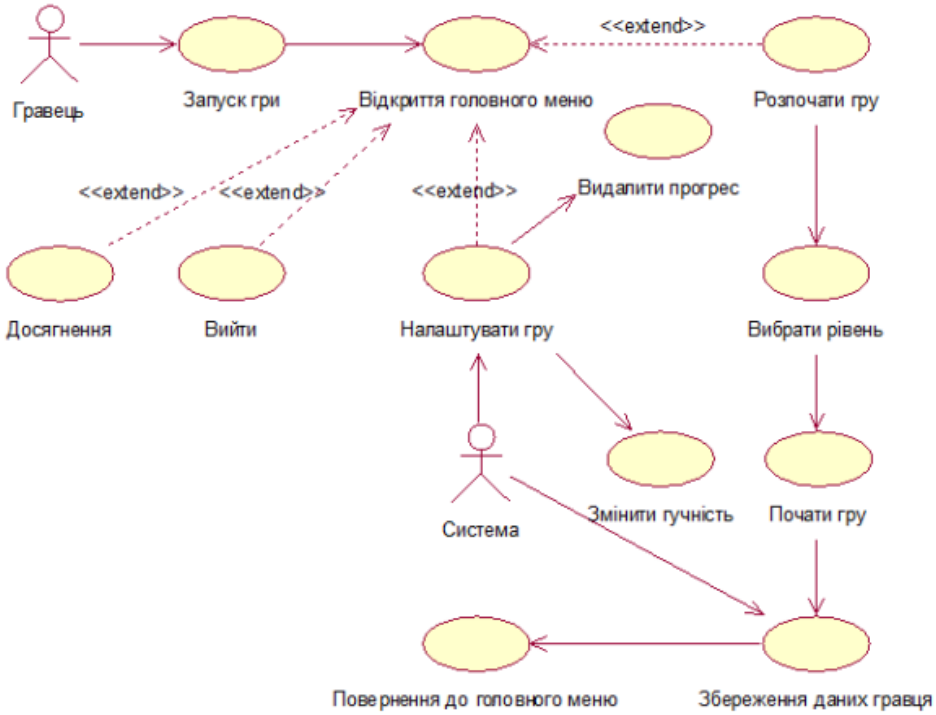


Рисунок 2.6 – Функціональна схема програми

На рисунку 2.7 зображена діаграма класів проекту.

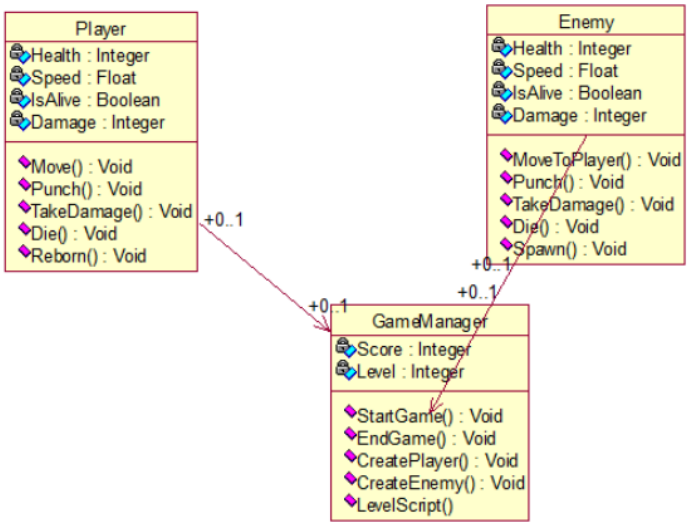


Рисунок 2.7 - Діаграма класів проекту

Як видно на рисунку 2.7 модель даних містить три основних класи додатку. Перший клас – гравець, що містить власні ігрові параметри та функції. Клас ворогів схожий з класом гравця, але має свої особливості та залежить від дій гравця на сцені. Клас ігрового менеджеру об’єднує обидва класи та маніпулює подіями на сцені.

## 2.4 Опис інтерфейсу програми

Відеогра є однокористувацькою грою на платформі ПК. Відеогра розроблялася під роздільну здатність екрану 1920 × 1080.

Значок запуску програми показано на рисунку 2.8.

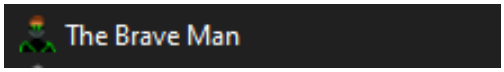


Рисунок 2.8 – Іконка гри

Запустивши відеогру з’являється заставка та логотип рушію гри, а після - головне меню. Головне меню зображене на рисунку 2.9. З усіх наступних меню можна повертатися до головного, натискаючи на кнопки «Закрити» або «Повернутися».

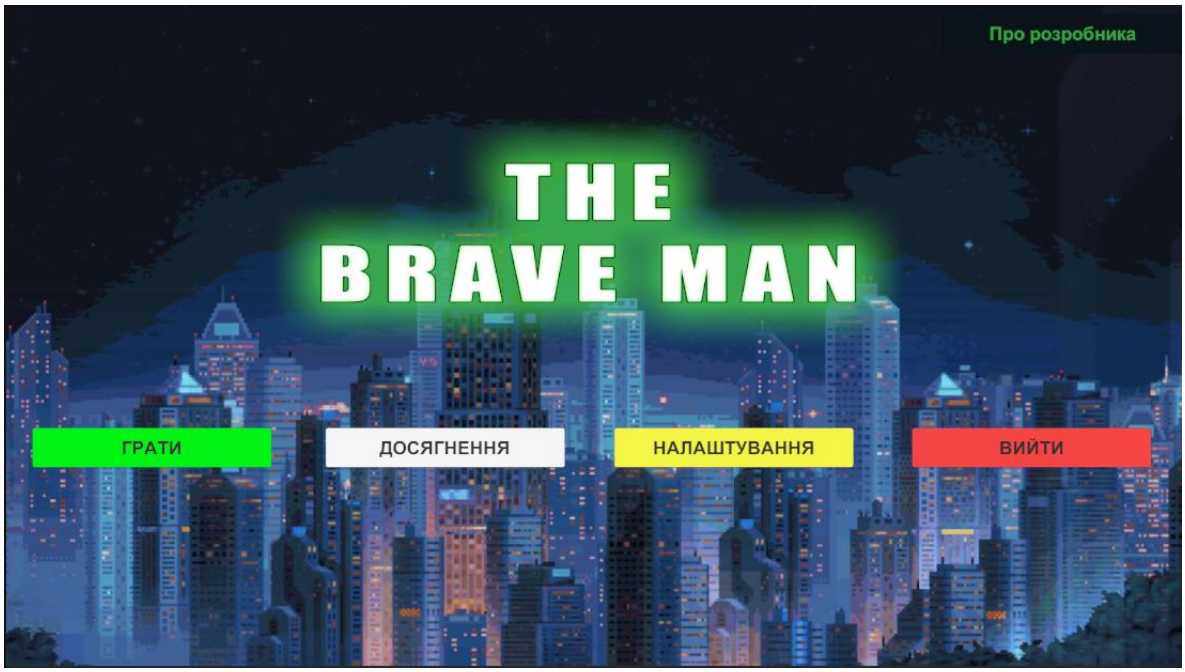


Рисунок 2.9 – Головне меню

Для переходу до меню «Про розробника» потрібно натиснути на кнопку «Про розробника» (рисунок 2.10).

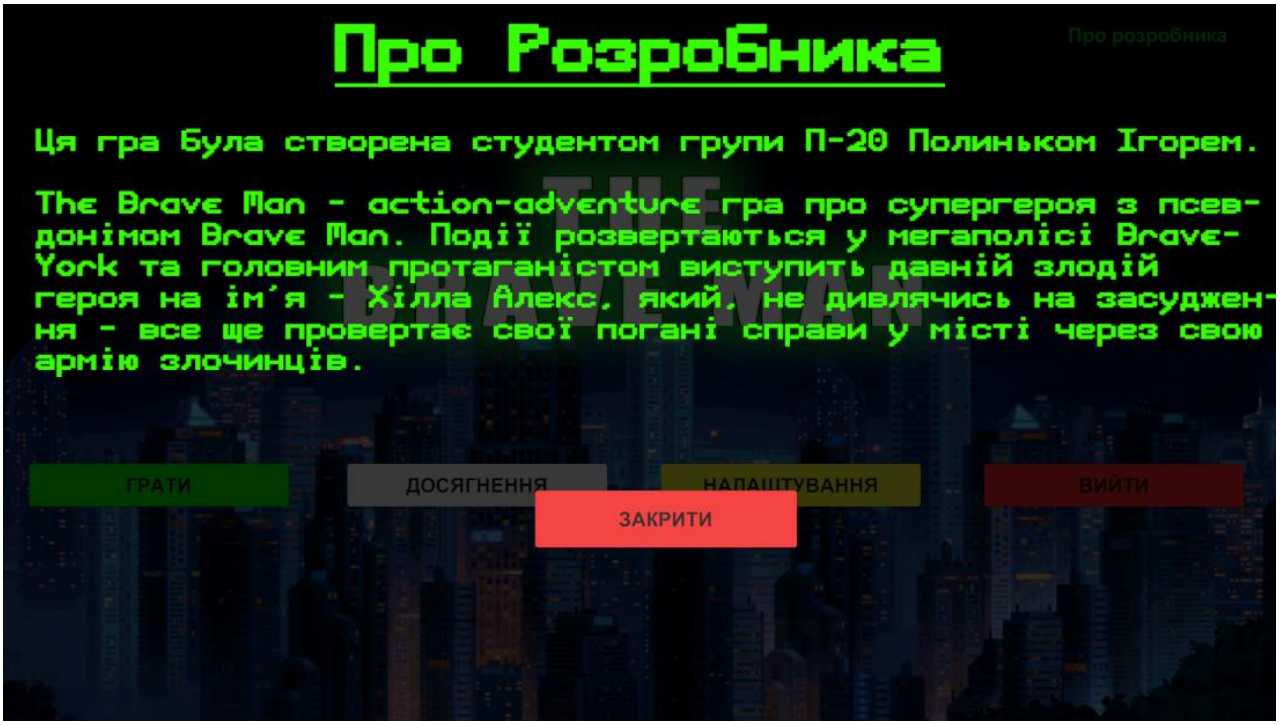


Рисунок 2.10 – Про розробника

Для переходу у меню вибору рівнів потрібно натиснути на кнопку з написом «ГРАТИ». Меню вибору рівнів представлено на рисунку 7 у додатках Д.

Для перевірки досягнень потрібно натиснути кнопку «ДОСЯГНЕННЯ». Меню досягнень представлено на рисунку 2 у додатках Д.

Після натискання кнопки з надписом «НАЛАШТУВАННЯ» переходимо до меню налаштувань, де можна змінити гучність та скинути прогрес гри. Меню налаштувань виглядає так, як показано на рисунку 5 у додатках Д.

При виборі розділу «Управління» отримуємо інформацію про призначення клавіш керування гри (рисунок 2.11).

Якщо гравець хоче почати гру, то йому потрібно вибрати один з рівнів у меню вибору рівня. Після вибору рівня він потрапить безпосередньо у сам ігровий процес, який зображений на рисунку 2.12.

Взаємодію між героєм та одним зі злодіїв за допомогою атаки можна побачити на рисунку 1 у додатках Д.

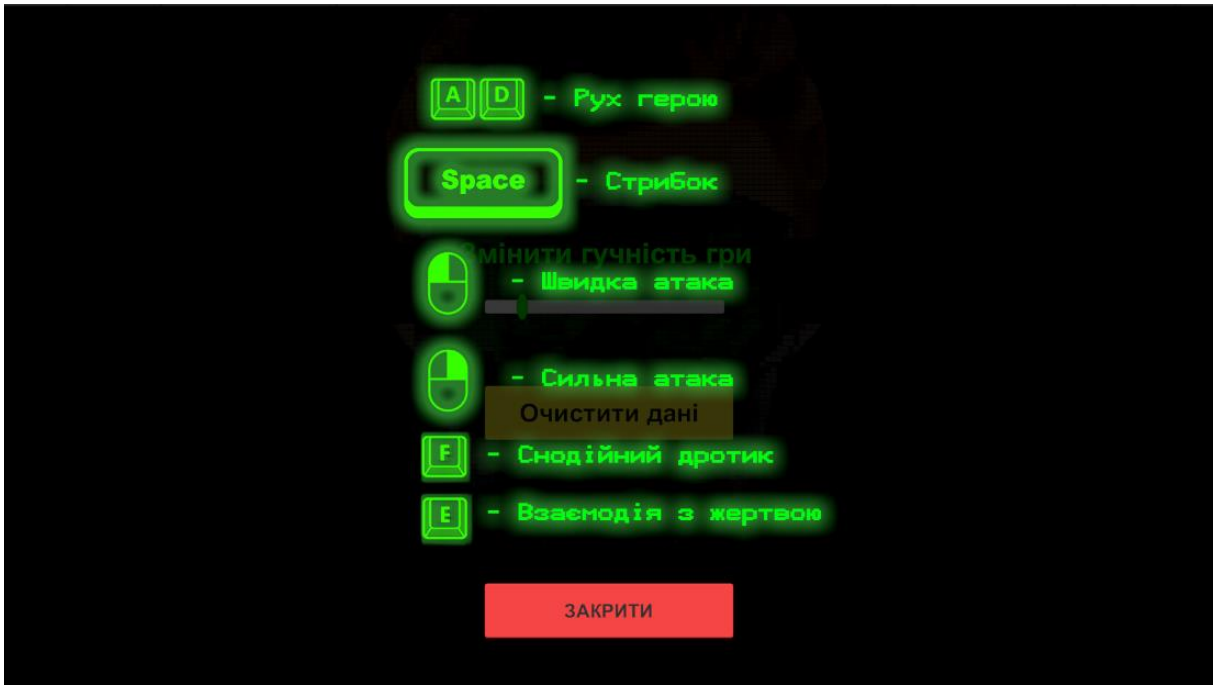


Рисунок 2.11 – Призначення клавіш управління


Рисунок 2.12 – Перший рівень

Другий тип злодію виглядає так, як зображено на рисунку 2.13.

Якщо при ігровому процесі натиснути клавішу «Esc», то з’явиться меню паузи, що дозволяє вийти в меню чи повернутися до ігрового процесу.

Для того, щоб повернутися до меню вибору рівнів потрібно натиснути кнопку «Вийти у меню».

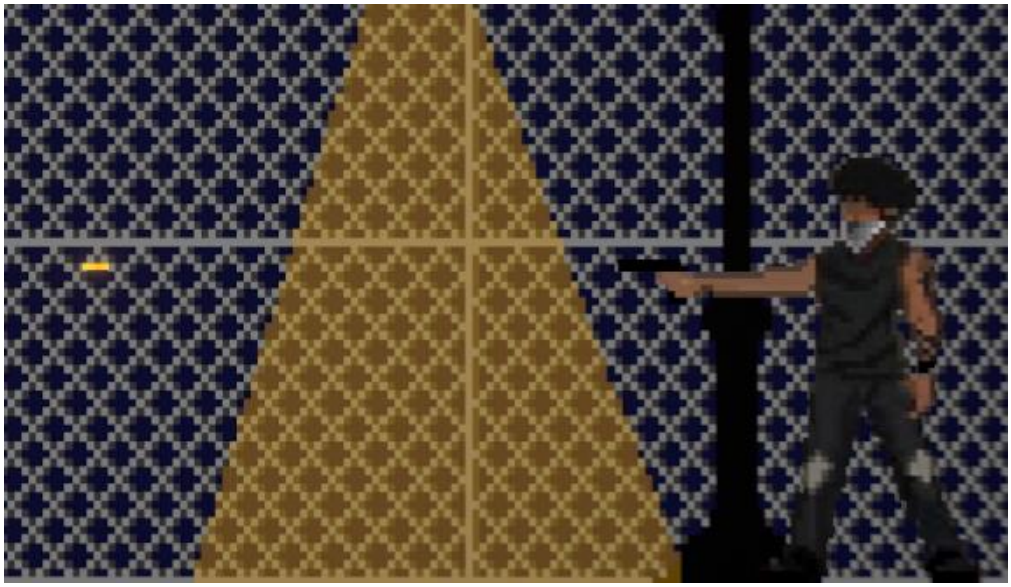

Рисунок 2.13 – Другий тип злодіїв

Для того, щоб повернутися до меню вибору рівнів потрібно натиснути кнопку «Вийти у меню». Для того, щоб вийти з цього меню, потрібно натиснути кнопку «Повернутися до гри» або повторно натиснути клавішу «Esc». Меню паузи зображено на рисунку 2.14.

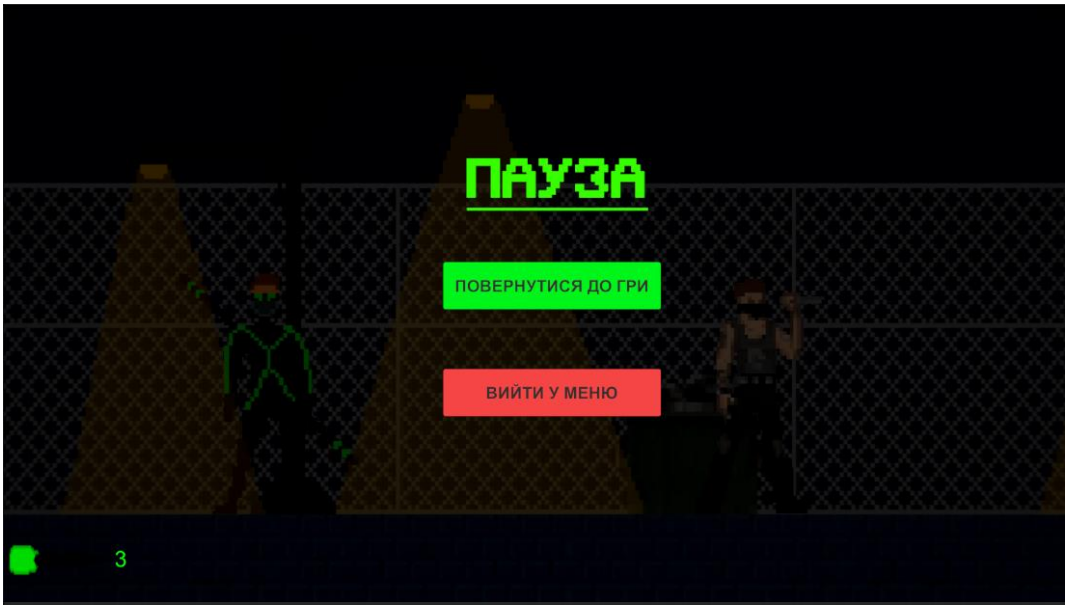


Рисунок 2.14 – Меню паузи

Для виходу з гри потрібно натиснути червону кнопку з надписом «ВИЙТИ».

Повідомлення про перемогу продемонстроване на рисунку 8 у додатках Д.

На рисунку 2.15 зображене серце, що збільшує здоров’я герою та з’являється після перемоги над ворогом.


Рисунок 2.15 – Серце здоров’я

Постраждалі люди, яких треба рятувати зображені на рисунку 2.16.


Рисунок 2.16 – Жертви

**Висновки до розділу**

Застосунок має зручну структуру папок та добре організовані скрипти, що забезпечують логічний розподіл функціоналу та зручність у процесі розробки. Усі створені сцени, скрипти та графічні елементи розроблені для досягнення максимального занурення користувача в ігровий процес. Ігровий інтерфейс відрізняється інтуїтивною зрозумілістю, а кожен компонент ретельно продуманий, щоб забезпечити комфортну взаємодію для гравця.

# ВИСНОВКИ

У рамках курсового проекту було розроблено 2D відеогру під назвою «The Brave Man» для ПК з операційною системою Windows. На рушії Unity із застосуванням мови програмування C#. У рамках проєкту було виконано всебічний аналіз алгоритмів, необхідних для реалізації гри, зокрема: організації ігрового інтерфейсу, взаємодії персонажів, ігрових подій, системи досягнень та розблокування нових рівнів.

Визначено основні функціональні та нефункціональні вимоги до гри. Розроблено алгоритми, які забезпечують якісну взаємодію гравця з інтерфейсом, анімаціями та системою бойових механік. Особливу увагу приділено графічному супроводу, для якого використовувався редактор Aseprite.

Структура застосунку вирізняється логічною організацією папок і скриптів, що забезпечує легкість у розробці та супроводі гри. Результатом роботи є відеогра, яка не лише розважає, але й допомагає розвивати стратегічне мислення, моторні навички та логіку.

Таким чином, курсова робота підтверджує можливість успішного використання сучасних інструментів розробки, таких як Unity та C#, для створення інтерактивних ігор, що відповідають вимогам сучасних користувачів.

# СПИСОК ЛІТЕРАТУРИ

1. Офіційна документація Unity. URL: https://docs.unity.com/ (дата звернення 05.10.2024).

2. Unity Asset Store. URL: https://assetstore.unity.com (дата звернення 05.10.2024).

3. Документація про Microsoft Visual Studio Wikipedia. URL: https://uk.wikipedia.org/wiki/ Microsoft_Visual_Studio (дата звернення 05.10.2024).

4. Офіційна сторінка застосунку Aseprite в Steam. URL: https://store.steampowered.com/app/431730/ Aseprite/?l=ukrainian (дата звернення 05.10.2024).

5. Пенні де Біл. "Цілісна розробка ігор з Unity: Повний посібник з впровадження ігрової механіки, мистецтва, дизайну та програмування." CRC Press, 2017. – 476 с.

6. Джеремі Гібсон Бонд. "Вступ до дизайну ігор, прототипування та розробки: Від концепції до гри на Unity і C#." Addison-Wesley Professional, 2017. – 944 с.

## Додаток А

## C# механіки ворогів

## Скрипт Enemy1.cs:

```
// Скрипт логіки злодію ближнього бою

public class Enemy : MonoBehaviour
{
    void Start()
    {
        enemyKilledAchievement = PlayerPrefs.GetInt("enemyKilledAchievement", 0) == 1;
        currentHealth = maxHealth;

        player = GameObject.FindGameObjectWithTag("Player").transform;
        animator = GetComponent<Animator>();
        theBraveMan = FindObjectOfType<TheBraveMan>();
        audioSource = GetComponent<AudioSource>();
    }

    private bool isMoving = false;

    void Update()
    {
        if (attacking)
        {
            attackTimer += Time.deltaTime;
            if (attackTimer >= 1f)
            {
                attacking = false;
                attackTimer = 0f;
            }
        }

        if (player != null && !death)
        {
            float distanceToPlayer = Vector2.Distance(transform.position,
player.position);

            if (distanceToPlayer < detectionRadius && !attacking)
            {
                Vector2 direction = (player.position - transform.position).normalized;
                transform.Translate(direction * speed * Time.deltaTime);

                isMoving = true;
                animator.SetBool("IsMoving", isMoving);

                moveVector = direction;
            }
            else
            {
                isMoving = false;
                animator.SetBool("IsMoving", isMoving);
            }
        }

        if (death)
        {
            deathtimer += Time.deltaTime;

            if (deathtimer >= 2.0f)
            {
                Instantiate(hp, transform.position, transform.rotation);
                Destroy(gameObject);
            }
        }
```

```
        Reflect();

        if (stopTime <= 0)
        {
            speed = normalspeed;
        }
        else
        {
            speed = 0;
            stopTime -= Time.deltaTime;
        }
    }

    void Reflect()
    {
        if (player != null && !death)
        {
            float distanceToPlayer = Vector2.Distance(transform.position,
player.position);

            if (distanceToPlayer < detectionRadius)
            {
                Vector3 directionToPlayer = player.position - transform.position;
                if ((directionToPlayer.x > 0 && !faceRight) || (directionToPlayer.x <
0 && faceRight))
                {
                    transform.localScale = new Vector3(-transform.localScale.x,
transform.localScale.y, transform.localScale.z);
                    faceRight = !faceRight;
                }
            }
        }
    }

    public void TakeDamage(int damage)
    {
        stopTime = startStopTime;
        currentHealth -= damage;
        animator.SetTrigger("Get Damage");
        Debug.Log("HP Enemy: " + currentHealth);

        if (currentHealth <= 0)
        {
            Die();
        }
    }

    public void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            if (timeBtwAttack <= 0)
            {
                if (!attackSoundplayed)
                {
                    audioSource.PlayOneShot(attackSound);
                    attackSoundplayed = true;
                }
                animator.SetTrigger("Attacking");
            }
            else
            {
                attackSoundplayed = false;
                timeBtwAttack -= Time.deltaTime;
            }
```

```
        }
    }

    public void OnEnemyAttack()
    {
        if (!death)
        {
            timeBtwAttack = starttimeBtwAttack;
            attacking = true;
            theBraveMan.health -= damage;
            Debug.Log("HP The Brave Man: " + theBraveMan.health);

            if (theBraveMan.health > 0)
            {
                theBraveMan.PlayGetDamageAnimation();
            }
        }
    }

    void Die()
    {
        if (!enemyKilledAchievement)
        {
            enemyKilledAchievement = true;
            PlayerPrefs.SetInt("enemyKilledAchievement", 1);
            PlayerPrefs.Save();
        }

        animator.SetTrigger("Death");
        death = true;
    }
}
```

## Скрипт Enemy2.cs:

```
// Скрипт логіки злодію далекого бою
public class Enemy2 : MonoBehaviour
{
    void Start()
    {
        enemyKilledAchievement = PlayerPrefs.GetInt("enemyKilledAchievement", 0) == 1;
        currentHealth = maxHealth;

        player = GameObject.FindGameObjectWithTag("Player").transform;
        animator = GetComponent<Animator>();
        theBraveMan = FindObjectOfType<TheBraveMan>();
        audioSource = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (attacking)
        {
            attackTimer += Time.deltaTime;
            if (attackTimer >= 1f)
            {
                attacking = false;
                attackTimer = 0f;
            }
        }

        if (player != null && !death)
        {
```

```
            float distanceToPlayer = Vector2.Distance(transform.position,
player.position);

            if (distanceToPlayer < detectionRadius)
            {
                Reflect();
                if (!attacking)
                {
                    if (timeBtwAttack <= 0)
                    {
                        animator.SetTrigger("Attacking");
                        OnEnemyAttack();
                        attackplaying = 0f;
                    }
                    else
                    {
                        timeBtwAttack -= Time.deltaTime;
                    }
                }
                else
                {
                    if (attackplaying >= 1f)
                    {
                        animator.SetTrigger("Idle");
                    }
                }
            }
        }

        if (death)
        {
            deathtimer += Time.deltaTime;

            if (deathtimer >= 2.0f)
            {
                Instantiate(hp, transform.position, transform.rotation);
                Destroy(gameObject);
            }
        }
    }

    public void OnEnemyAttack()
    {
        if (!death)
        {
            audioSource.PlayOneShot(attackSound);
            attacking = true;
            Shoot();
            timeBtwAttack = starttimeBtwAttack;
        }
    }

    void Shoot()
    {
        if (bulletSpawnPoint != null && BulletPrefab != null)
        {
            GameObject newBullet = Instantiate(BulletPrefab,
bulletSpawnPoint.position, Quaternion.identity);
            Destroy(newBullet, 3f);
        }
    }
}
```

## Скрипт Bullet.cs:

```
// Скрипт, що відповідає за пулю ворога
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Bullet : MonoBehaviour
{
    public float moveSpeed = 5f;
    public SpriteRenderer spriteRenderer;
    public int attackBulletDamage = 30;
    Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        spriteRenderer = GetComponent<SpriteRenderer>();

        GameObject player = GameObject.FindGameObjectWithTag("Enemy2");
        if (player != null)
        {
            Enemy2 enemy2 = player.GetComponent<Enemy2>();
            if (enemy2 != null)
            {
                Vector2 moveDirection = enemy2.faceRight ? Vector2.right :
Vector2.left;
                rb.velocity = moveDirection * moveSpeed;

                if (!enemy2.faceRight)
                {
                    spriteRenderer.flipX = true;
                }
            }
        }

        Destroy(gameObject, 3f);
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        TheBraveMan playerScript = other.GetComponent<TheBraveMan>();

        if (playerScript != null)
        {
            playerScript.GetBulletDamage(attackBulletDamage);

            Destroy(gameObject);
        }
    }
}
```

**Додаток Б**

**C# механіки герою**

## Скрипт TheBraveMan.cs:

```
// Скрипт головного героя

public class TheBraveMan : MonoBehaviour
{
    void Start()
    {
        foreach (Transform child in transform)
        {
            if (child.CompareTag("DartSpawnPoint"))
            {
                dartSpawnPoint = child;
                break;
            }
        }

        rb = GetComponent<Rigidbody2D>();
        bulletScript = GameObject.FindObjectOfType<Bullet>();
        audioSource = GetComponent<AudioSource>();
        walkAudioSource = gameObject.AddComponent<AudioSource>();
        backgroundMusicSource = Camera.main.GetComponent<AudioSource>();
    }

    void Update()
    {
        humansSavedCompletedAchievement =
PlayerPrefs.GetInt("humansSavedCompletedAchievement", 0) == 1;

        if (helping)
        {
            humantimer += Time.deltaTime;
        }
        if (notdead && !lvlended)
        {
            Walk();
            Jump();

            ComboTimer();
            if (attack1Playing)
            {
                attack1PlayingTimer += Time.deltaTime;

                if (attack1PlayingTimer > 0.5f)
                {
                    attack1Playing = false;
                    attack1PlayingTimer = 0f;
                }
            }
            else if (Time.time >= nextAttack1Time)
            {
                Reflect();
                if (Input.GetMouseButtonDown(0) && !attackBlock && !helping)
                {
                    Attack1();
                    attack1Playing = true;
                    nextAttack1Time = Time.time + 2f / attackRate;
                }
            }
            if (Time.time >= nextAttack2Time)
            {
                if (Input.GetMouseButtonDown(1) && !attackBlock && !helping)
                {
                    Attack2();
                    attack2Playing = true;
```

```
                nextAttack2Time = Time.time + 3f / attackRate;
            }
        }

        if (Input.GetKeyDown(KeyCode.F) && dartscount >= 1 && !helping)
        {
            ShootDart();
            dartscount -= 1;
            combo2part = true;
        }

        if (maxhealth < health)
        {
            health = maxhealth;
        }
        if (minhealth > health)
        {
            health = minhealth;
        }
    }
    else
    {
        timetodeath += Time.deltaTime;
        if (timetodeath > 2.5f && !lvlended)
        {
            losePanel.SetActive(true);
            StopBackgroundMusic();
            if (!playedLoseSound)
            {
                PlaySound(loseSound);
                Time.timeScale = 0;
                playedLoseSound = true;
            }
        }
    }

    if (isCollidingWithHuman && Input.GetKeyDown(KeyCode.E))
    {
        helping = true;
        Destroy(humanObject);
        GameObject humanCircleObject = HumanCircle;
        if (humanCircleObject != null)
        {
            humanCircleObject.SetActive(true);
        }
    }
    if (humantimer >= 1f && Input.GetKeyDown(KeyCode.E) && helping)
    {
        humantimer = 0f;
        helping = false;
        GameObject human = Instantiate(humanPrefab, dartSpawnPoint.position,
Quaternion.identity);
    }

    Die();
}

void Walk()
{
    moveVector.x = Input.GetAxis("Horizontal");
    rb.velocity = new Vector2(moveVector.x * speed, moveVector.y);

    if (moveVector.x != 0)
    {
        if (!walksoundplayer && walksoundplayerGround)
```

```
            {
                PlayWalkSound(walkSound);
                walksoundplayer = true;
            }
            if (!helping)
            {
                speed = 6f;
                animator.SetTrigger("Run");
            }
            else
            {
                speed = 4f;
                animator.SetTrigger("Move");
            }
        }
        else if (moveVector.x == 0)
        {
            walkAudioSource.Stop();
            walksoundplayer = false;
            if (!helping)
            {
                animator.SetTrigger("TheBraveManIdle");
            }
            if (helping)
            {
                animator.SetTrigger("IdlewithHuman");
            }
        }
        if (!walksoundplayerGround)
        {
            walkAudioSource.Stop();
        }
    }

    public bool faceRight = true;
    void Reflect()
    {
        if ((moveVector.x > 0 && !faceRight) || (moveVector.x < 0 && faceRight))
        {
            transform.localScale *= new Vector2(-1, 1);
            faceRight = !faceRight;
        }
    }

    void Attack1()
    {
        PlaySound(attackSound);

        if (combo1part1 && combo1part2)
        {
            attack3Playing = true;
            animator.SetTrigger("Attack3");

            Collider2D[] hitEnemies =
Physics2D.OverlapCircleAll(attack3point.position, attack3Range, enemyLayers);

            foreach (Collider2D enemy in hitEnemies)
            {
                Enemy enemyScript = enemy.GetComponent<Enemy>();
                Enemy2 enemy2Script = enemy.GetComponent<Enemy2>();

                if (enemyScript != null && enemyScript.lastHitTime + 0.05f <
Time.time)
                {
                    enemyScript.TakeDamage(attack3Damage);
```

```
                    enemyScript.lastHitTime = Time.time;
                }
                if (enemy2Script != null && enemy2Script.lastHitTime + 0.05f <
Time.time)
                {
                    enemy2Script.TakeDamage(attack3Damage);
                    enemy2Script.lastHitTime = Time.time;
                }
            }
        }
        else
        {
            attack1Playing = true;
            animator.SetTrigger("Attack1");

            Collider2D[] hitEnemies =
Physics2D.OverlapCircleAll(attack1point.position, attack1Range, enemyLayers);

            foreach (Collider2D enemy in hitEnemies)
            {
                Enemy enemyScript = enemy.GetComponent<Enemy>();
                Enemy2 enemy2Script = enemy.GetComponent<Enemy2>();

                if (enemyScript != null && enemyScript.lastHitTime + 0.05f <
Time.time)
                {
                    enemyScript.TakeDamage(attack1Damage);
                    enemyScript.lastHitTime = Time.time;
                }
                if (enemy2Script != null && enemy2Script.lastHitTime + 0.05f <
Time.time)
                {
                    enemy2Script.TakeDamage(attack1Damage);
                    enemy2Script.lastHitTime = Time.time;
                }
            }

            combo1part1 = true;
        }
    }

    void Attack2()
    {
        if (combo2part && humansSavedCompletedAchievement)
        {
            PlaySound(attack4Sound);
            attack4Playing = true;
            animator.SetTrigger("Attack4");

            Collider2D[] hitEnemies =
Physics2D.OverlapCircleAll(attack4point.position, attack4Range, enemyLayers);

            foreach (Collider2D enemy in hitEnemies)
            {
                Enemy enemyScript = enemy.GetComponent<Enemy>();
                Enemy2 enemy2Script = enemy.GetComponent<Enemy2>();

                if (enemyScript != null && enemyScript.lastHitTime + 0.05f <
Time.time)
                {
                    enemyScript.TakeDamage(attack4Damage);
                    enemyScript.lastHitTime = Time.time;
                }
                if (enemy2Script != null && enemy2Script.lastHitTime + 0.05f <
Time.time)
```

```
                {
                    enemy2Script.TakeDamage(attack4Damage);
                    enemy2Script.lastHitTime = Time.time;
                }
            }
        }
        else
        {
            PlaySound(attackSound);
            attack2Playing = true;
            animator.SetTrigger("Attack2");

            Collider2D[] hitEnemies =
Physics2D.OverlapCircleAll(attack2point.position, attack2Range, enemyLayers);

            foreach (Collider2D enemy in hitEnemies)
            {
                Enemy enemyScript = enemy.GetComponent<Enemy>();
                Enemy2 enemy2Script = enemy.GetComponent<Enemy2>();

                if (enemyScript != null && enemyScript.lastHitTime + 0.05f <
Time.time)
                {
                    enemyScript.TakeDamage(attack2Damage);
                    enemyScript.lastHitTime = Time.time;
                }
                if (enemy2Script != null && enemy2Script.lastHitTime + 0.05f <
Time.time)
                {
                    enemy2Script.TakeDamage(attack2Damage);
                    enemy2Script.lastHitTime = Time.time;
                }
            }

            if (combo1part1)
            {
                combo1part2 = true;
                nextAttack1Time = 0f;
            }
        }
    }

    public void GetBulletDamage(int damage)
    {
        if (notdead)
        {
            health -= damage;
            Debug.Log("HP The Brave Man: " + health);
            PlayGetDamageAnimation();
            PlaySound(damageSound);
        }
    }

    void ShootDart()
    {
        if (dartSpawnPoint != null && dartPrefab != null)
        {
            PlaySound(dartSound);
            animator.SetTrigger("Dart");
            GameObject newDart = Instantiate(dartPrefab, dartSpawnPoint.position,
Quaternion.identity);
            Destroy(newDart, 3f);
        }
    }
```

```
public void PlayGetDamageAnimation()
{
    animator.SetTrigger("GetDamage");
}

void Jump()
{
    if (jumpcount < 2 && Input.GetKeyDown(KeyCode.Space) && !helping)
    {
        PlaySound(jumpSound);
        attackBlock = true;
        animator.SetTrigger("Jump");

        rb.AddForce(Vector2.up * jumpForce);
        walksoundplayerGround = false;

        jumpcount++;
    }

    if (jumpcount >= 2)
    {
        jumptimer += Time.deltaTime;
        if (jumptimer >= 1.3f)
        {
            jumpcount = 0;
            jumptimer = 0f;
        }
    }
}

void OnDrawGizmosSelected()
{
    if (attack1point == null)
        return;

    Gizmos.DrawWireSphere(attack1point.position, attack1Range);

    if (attack2point == null)
        return;

    Gizmos.DrawWireSphere(attack2point.position, attack2Range);

    if (attack3point == null)
        return;

    Gizmos.DrawWireSphere(attack3point.position, attack3Range);
}

public void Heal()
{
    health += amount;
}

void OnCollisionEnter2D(Collision2D Stolknovenie)
{
    if (Stolknovenie.gameObject.tag == "HP")
    {
        Destroy(Stolknovenie.gameObject);
        amount = 10;
        Heal();
        Debug.Log("HP The Brave Man: " + health);
    }
    if (Stolknovenie.gameObject.tag == "Ground")
    {
        attackBlock = false;
```

```
            animator.SetTrigger("Idle");
            jumpcount = 0;
            walksoundplayerGround = true;
        }
        if (Stolknovenie.gameObject.tag == "Finish")
        {
            lvlended = true;
            animator.SetTrigger("End");
            StopBackgroundMusic();
            PlaySound(winSound);
            winPanel.SetActive(true);
        }
        if (Stolknovenie.gameObject.tag == "Human")
        {
            isCollidingWithHuman = true;
            humanObject = Stolknovenie.gameObject;
        }
    }

    void OnCollisionExit2D(Collision2D Stolknovenie)
    {
        if (Stolknovenie.gameObject.tag == "Human")
        {
            isCollidingWithHuman = false;
            humanObject = null;
        }
    }

    void ComboTimer()
    {
        if (combo1part1)
        {
            comboTimer += Time.deltaTime;
            if (comboTimer >= 3.0f)
            {
                combo1part1 = false;
                combo1part2 = false;
                comboTimer = 0.0f;
            }
        }
        if (combo2part)
        {
            combo2Timer += Time.deltaTime;
            if (combo2Timer >= 1.0f)
            {
                combo2part = false;
                combo2Timer = 0.0f;
            }
        }
    }

    void OnGUI()
    {
        GUIStyle style = new GUIStyle();
        style.fontSize = 40;
        style.normal.textColor = Color.green;

        GUI.Label(new Rect(200, Screen.height - 100, 200, 20), "" + dartscount,
style);

        GUI.Label(new Rect(200, 60, 200, 20), "" + health, style);
    }

    void PlaySound(AudioClip clip)
    {
```

```
        audioSource.PlayOneShot(clip);
    }

    void StopBackgroundMusic()
    {
        if (backgroundMusicSource != null)
        {
            backgroundMusicSource.Stop();
        }
    }

    void PlayWalkSound(AudioClip clip)
    {
        walkAudioSource.PlayOneShot(clip);
    }
}
```

## Додаток В

## C# внутрішні функції гри

## Скрипт Pause.cs:

```
// Скрипт паузи
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pause : MonoBehaviour
{
    public GameObject pausePanel;

    private bool isPaused = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            TogglePause();
        }
    }

    public void TogglePause()
    {
        isPaused = !isPaused;

        if (isPaused)
        {
            Time.timeScale = 0;
            pausePanel.SetActive(true);
        }
        else
        {
            Time.timeScale = 1;
            pausePanel.SetActive(false);
        }
    }
}
```

## Скрипт GlobalVolumeControl.cs:

```
//Скрипт для зміни гучності всієї гри
public class GlobalVolumeControl : MonoBehaviour
{
    public static float globalVolume = 1.0f;

    private void Awake()
    {
        if (PlayerPrefs.HasKey("GlobalVolume"))
        {
            globalVolume = PlayerPrefs.GetFloat("GlobalVolume");
        }
        else
        {
            PlayerPrefs.SetFloat("GlobalVolume", globalVolume);
        }

        DontDestroyOnLoad(gameObject);
    }

    public static void SaveVolume()
    {
        PlayerPrefs.SetFloat("GlobalVolume", globalVolume);
        PlayerPrefs.Save();
```

```
    }

    public static void SetAllAudioSourcesVolume()
    {
        AudioSource[] allAudioSources = FindObjectsOfType<AudioSource>();

        foreach (var audioSource in allAudioSources)
        {
            if (audioSource != null)
            {
                audioSource.volume = globalVolume;
            }
        }
    }
}
```

## Скрипт AchievmentCompleter.cs:

```
// Скрипт для виконання одного з досягнень
public class FirstAchievmentCompleter : MonoBehaviour
{
    void Start()
    {
        levelCompletedAchievment = true;

        hasShownAchievement = PlayerPrefs.GetInt("HasShownAchievement", 0) == 1;

        if (levelCompletedAchievment && !hasShownAchievement)
        {
            ShowAchievementCompletedImage();

            hasShownAchievement = true;

            PlayerPrefs.SetInt("HasShownAchievement", hasShownAchievement ? 1 : 0);
            PlayerPrefs.Save();
        }

        DontDestroyOnLoad(gameObject);
    }

    private void ShowAchievementCompletedImage()
    {
        achievementCompletedImage.gameObject.SetActive(true);
        Invoke("HideAchievementCompletedImage", achievmenttimer);
    }
}
```

## Скрипт AchievmentManager.cs:

```
// Скрипт для перевірки одного з досягнень
public class FirstAchievmentManager : MonoBehaviour
{
    public Image achievementImage;
    public Sprite achievementUnlockedSprite;
    public Sprite achievementBlockedSprite;

    public bool achievementUnlocked = false;

    void Start()
    {
        achievementUnlocked = PlayerPrefs.GetInt("FirstAchievementUnlocked", 0) == 1;
        UpdateAchievementUI();
```

```
    }

    void Update()
    {
        CheckAchievement();
    }

    private void CheckAchievement()
    {
        if (FirstAchievmentCompleter.levelCompletedAchievment)
        {
            UnlockAchievement();

            UpdateAchievementUI();

            FirstAchievmentCompleter.levelCompletedAchievment = false;
        }
    }

    private void UpdateAchievementUI()
    {
        if (achievementImage != null && achievementUnlockedSprite != null)
        {
            achievementImage.sprite = achievementUnlocked ?
achievementUnlockedSprite : achievementBlockedSprite;
        }
    }

    private void UnlockAchievement()
    {
        achievementUnlocked = true;

        PlayerPrefs.SetInt("FirstAchievementUnlocked", achievementUnlocked ? 1 : 0);
        PlayerPrefs.Save();
    }
}
```

## Скрипт SceneController.cs:

```
// Скрипт для плавного переходу між сценами

public class SceneController : MonoBehaviour
{
    void Start()
    {
        Invoke("DisableFadeOut", 1f);
    }

    void DisableFadeOut()
    {
        GameObject fadeOutPanel = GameObject.Find("Fade Out");
        if (fadeOutPanel != null)
        {
            fadeOutPanel.SetActive(false);
        }
    }
}
```

## Скрипт SettingsManager.cs:

```
// Скрипт для видалення прогресу гри
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SettingsManager : MonoBehaviour
{
    public void DeleteProgressButtonOnClick()
    {
        PlayerPrefs.DeleteAll();
        PlayerPrefs.Save();

        DestroyAllDontDestroyOnLoadObjects();
    }

    public void DestroyAllDontDestroyOnLoadObjects()
    {
        var tempScene = SceneManager.CreateScene("TempScene");

        var dontDestroyOnLoadObjects = GetDontDestroyOnLoadObjects();
        foreach (var obj in dontDestroyOnLoadObjects)
        {
            SceneManager.MoveGameObjectToScene(obj, tempScene);
        }

        SceneManager.UnloadSceneAsync(tempScene);
    }

    private List<GameObject> GetDontDestroyOnLoadObjects()
    {
        List<GameObject> dontDestroyOnLoadObjects = new List<GameObject>();
        GameObject temp = null;
        try
        {
            temp = new GameObject();
            DontDestroyOnLoad(temp);
            var tempScene = temp.scene;
            SceneManager.MoveGameObjectToScene(temp, SceneManager.GetActiveScene());
            dontDestroyOnLoadObjects.AddRange(tempScene.GetRootGameObjects());
        }
        finally
        {
            if (temp != null)
            {
                Destroy(temp);
            }
        }
        return dontDestroyOnLoadObjects;
    }
}
```

## Скрипт SoundManager.cs:

```
// Скрипт, що додається на об'єкт, щоб змінити гучність усіх AudioSource на об'єкті
using UnityEngine;

public class SoundManager : MonoBehaviour
{
    private AudioSource[] audioSources;

    void Start()
    {
        audioSources = GetComponentsInChildren<AudioSource>();

        foreach (AudioSource audioSource in audioSources)
```

```
        {
            audioSource.volume = GlobalVolumeControl.globalVolume;
        }

        PlaySounds();
    }

    void PlaySounds()
    {
        foreach (AudioSource audioSource in audioSources)
        {
            audioSource.Play();
        }
    }
}
```

## Скрипт VolumeSlider.cs:

```
// Скрипт для роботи слайдеру зміни гучності у налаштуваннях
using UnityEngine;
using UnityEngine.UI;

public class VolumeSlider : MonoBehaviour
{
    public Slider volumeSlider;

    private void Start()
    {
        volumeSlider.value = GlobalVolumeControl.globalVolume;
        volumeSlider.onValueChanged.AddListener(OnVolumeChanged);
    }

    private void OnVolumeChanged(float value)
    {
        GlobalVolumeControl.globalVolume = value;
        GlobalVolumeControl.SaveVolume();
        GlobalVolumeControl.SetAllAudioSourcesVolume();
    }

    private void SaveVolume()
    {
        PlayerPrefs.SetFloat("GlobalVolume", GlobalVolumeControl.globalVolume);
        PlayerPrefs.Save();
    }
}
```

## Скрипт ExitGame.cs:

```
// Скрипт для виходу з гри
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ExitGame : MonoBehaviour
{
    public void StopGame()
    {
        Application.Quit();
    }
}
```

## Додаток Г

## C# менеджери рівнів гри

## Скрипт FirstLevelManager.cs:

```
// Скрипт логіки першого рівня
public class FirstLevelManager : MonoBehaviour
{
    private int enemiesDestroyed = 0;

    [SerializeField] GameObject Finish;
    public GameObject EnemyPrefab;
    public GameObject Enemy2Prefab;
    public Transform spawnPoint1;
    public Transform spawnPoint2;
    public int spawned2 = 0;
    public int spawned4 = 0;

    void Start()
    {
        Time.timeScale = 0;
    }

    void Update()
    {
        Enemy[] enemies = FindObjectsOfType<Enemy>();
        Enemy2[] enemies2 = FindObjectsOfType<Enemy2>();

        foreach (Enemy enemy in enemies)
        {
            if (enemy.death && !enemy.counted)
            {
                enemy.counted = true;
                enemiesDestroyed++;
            }
        }

        foreach (Enemy2 enemy in enemies2)
        {
            if (enemy.death && !enemy.counted)
            {
                enemy.counted = true;
                enemiesDestroyed++;
            }
        }

        if (enemiesDestroyed >= 2)
        {
            if (spawned2 < 2)
            {
                Instantiate(EnemyPrefab, spawnPoint1.position, spawnPoint1.rotation);
                spawned2++;
            }
        }

        if (enemiesDestroyed >= 4)
        {
            if (spawned4 < 1)
            {
                Instantiate(Enemy2Prefab, spawnPoint1.position, spawnPoint1.rotation);
                Instantiate(Enemy2Prefab, spawnPoint2.position, spawnPoint2.rotation);
                spawned4++;
            }
        }

        if (enemiesDestroyed >= 6)
        {
            GameObject finishObject = Finish;
```

```
            if (finishObject != null)
            {
                finishObject.SetActive(true);
            }
        }
    }
}
```

## Скрипт SecondLevelManager.cs:

```
// Скрипт логіки другого рівня
public class SecondLevelManager : MonoBehaviour
{
    public int enemiesDestroyed = 0;

    [SerializeField] GameObject Finish;
    public GameObject EnemyPrefab;
    public GameObject Enemy2Prefab;
    public Transform spawnPoint1;
    public Transform spawnPoint2;
    public int spawned1 = 0;
    public int spawned2 = 0;
    public int spawned1sec = 0;

    private Enemy[] enemies;
    private Enemy2[] enemies2;

    void Start()
    {
        Time.timeScale = 0;
    }

    void Update()
    {
        int savedHumans = HumanCircle.GetTotalSaved();

        enemies = FindObjectsOfType<Enemy>();
        enemies2 = FindObjectsOfType<Enemy2>();

        foreach (Enemy enemy in enemies)
        {
            if (enemy.death && !enemy.counted)
            {
                enemy.counted = true;
                enemiesDestroyed++;
            }
        }

        foreach (Enemy2 enemy in enemies2)
        {
            if (enemy.death && !enemy.counted)
            {
                enemy.counted = true;
                enemiesDestroyed++;
            }
        }

        if (enemiesDestroyed >= 2 && savedHumans == 1)
        {
            if (spawned1 < 1)
            {
                Instantiate(Enemy2Prefab, spawnPoint2.position, spawnPoint2.rotation);
                spawned1++;
```

```
            }
        }

        if (enemiesDestroyed >= 3)
        {
            if (spawned2 < 1)
            {
                Instantiate(EnemyPrefab, spawnPoint1.position, spawnPoint1.rotation);
                Instantiate(EnemyPrefab, spawnPoint1.position, spawnPoint1.rotation);
                spawned2++;
            }
        }

        if (savedHumans == 2)
        {
            if (spawned1sec < 1)
            {
                Instantiate(EnemyPrefab, spawnPoint1.position, spawnPoint1.rotation);
                spawned1sec++;
            }
        }

        if (savedHumans == 3 && enemiesDestroyed >= 6)
        {
            GameObject finishObject = Finish;
            if (finishObject != null)
            {
                finishObject.SetActive(true);
            }
        }
    }

    public void SetAllNull()
    {
        enemiesDestroyed = 0;
        HumanCircle.ResetSavedHumans();
        enemies = FindObjectsOfType<Enemy>();
        enemies2 = FindObjectsOfType<Enemy2>();

        foreach (Enemy enemy in enemies)
        {
            Destroy(enemy.gameObject);
        }

        foreach (Enemy2 enemy in enemies2)
        {
            Destroy(enemy.gameObject);
        }

        spawned1 = 0;
        spawned2 = 0;
        spawned1sec = 0;
    }
}
```

## Скрипт ThirdLevelManager.cs:

```
// Скрипт логіки третього рівня
public class ThirdLevelManager : MonoBehaviour
{
    public int enemiesDestroyed = 0;
```

```
[SerializeField] GameObject Finish;
public GameObject EnemyPrefab;
public GameObject Enemy2Prefab;
public Transform spawnPoint1;
public Transform spawnPoint2;
public Transform spawnPoint3;
public int spawned2 = 0;
public int spawned3 = 0;
public int spawned2sec = 0;

private Enemy[] enemies;
private Enemy2[] enemies2;

void Start()
{
    Time.timeScale = 0;
}

void Update()
{
    enemies = FindObjectsOfType<Enemy>();
    enemies2 = FindObjectsOfType<Enemy2>();

    foreach (Enemy enemy in enemies)
    {
        if (enemy.death && !enemy.counted)
        {
            enemy.counted = true;
            enemiesDestroyed++;
        }
    }

    foreach (Enemy2 enemy in enemies2)
    {
        if (enemy.death && !enemy.counted)
        {
            enemy.counted = true;
            enemiesDestroyed++;
        }
    }

    if (enemiesDestroyed >= 2)
    {
        if (spawned3 < 1)
        {
            Instantiate(Enemy2Prefab, spawnPoint1.position, spawnPoint1.rotation);
            Instantiate(EnemyPrefab, spawnPoint1.position, spawnPoint2.rotation);
            Instantiate(EnemyPrefab, spawnPoint3.position, spawnPoint3.rotation);
            spawned3++;
        }
    }

    if (enemiesDestroyed >= 4)
    {
        if (spawned2 < 1)
        {
            Instantiate(Enemy2Prefab, spawnPoint2.position, spawnPoint2.rotation);
            spawned2++;
        }
    }

    if (enemiesDestroyed >= 6)
    {
        if (spawned2sec < 1)
        {
```

```
                Instantiate(EnemyPrefab, spawnPoint3.position, spawnPoint3.rotation);
                Instantiate(EnemyPrefab, spawnPoint3.position, spawnPoint3.rotation);
                spawned2sec++;
            }
        }

        if (enemiesDestroyed >= 8)
        {
            GameObject finishObject = Finish;
            if (finishObject != null)
            {
                finishObject.SetActive(true);
            }
        }
    }

    public void SetAllNull()
    {
        enemiesDestroyed = 0;
        HumanCircle.ResetSavedHumans();

        enemies = FindObjectsOfType<Enemy>();
        enemies2 = FindObjectsOfType<Enemy2>();

        foreach (Enemy enemy in enemies)
        {
            Destroy(enemy.gameObject);
        }

        foreach (Enemy2 enemy in enemies2)
        {
            Destroy(enemy.gameObject);
        }

        spawned3 = 0;
        spawned2 = 0;
        spawned2sec = 0;
    }
}
```

## Додаток Д

## Тестування додатку

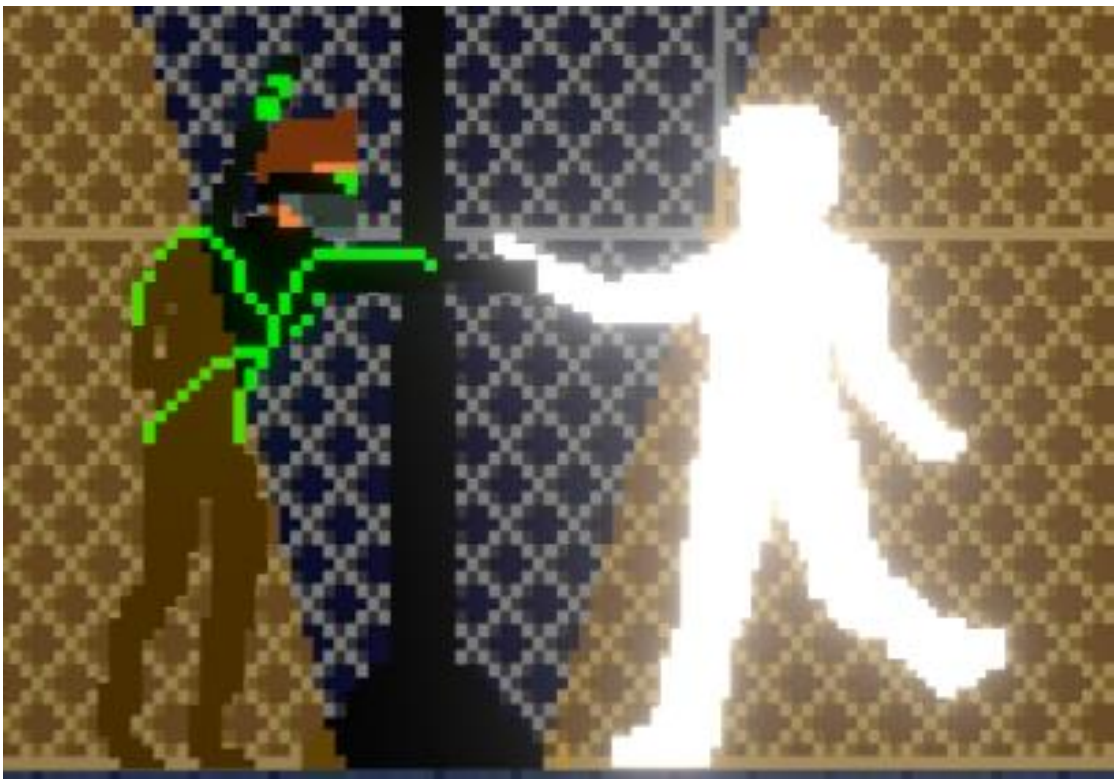

Рисунок 1 – Отримання ворогом урону

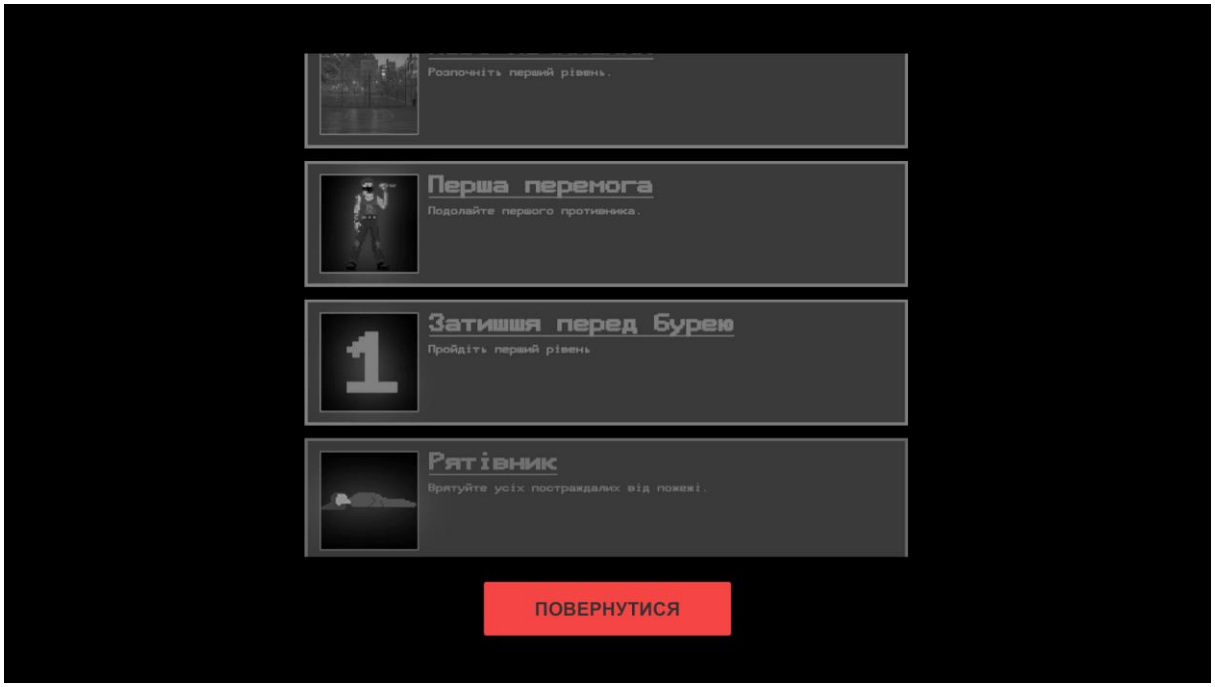


Рисунок 2 – Відсутні досягнення на початку гри


Рисунок 2 – Деякі виконані досягненя

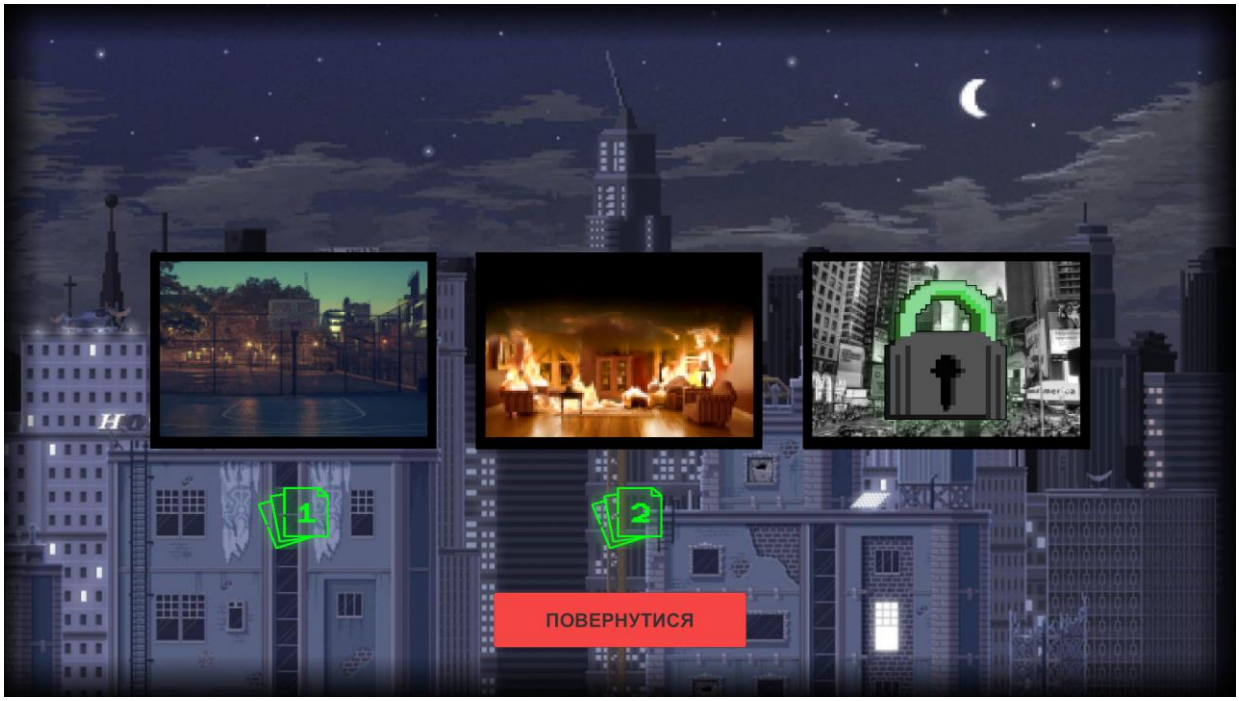


Рисунок 4 – Розблокування рівнів та сюжетних записок


Рисунок 5 – Очищення даних

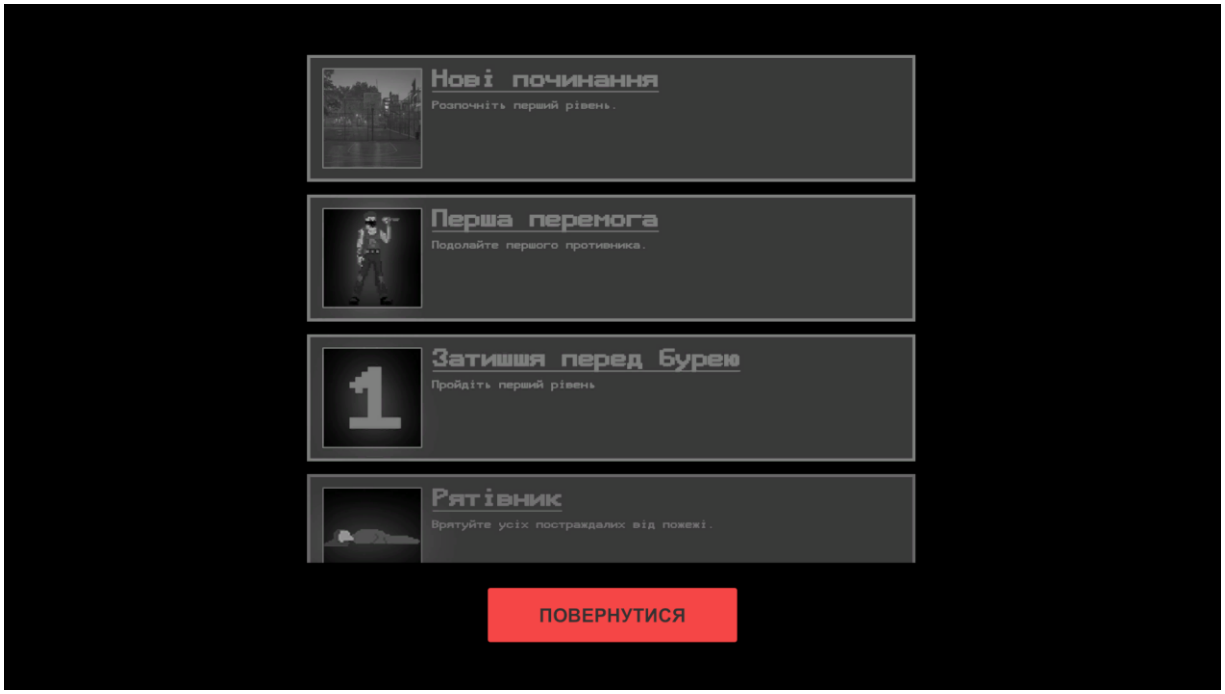


Рисунок 6 – Очищені досягнення

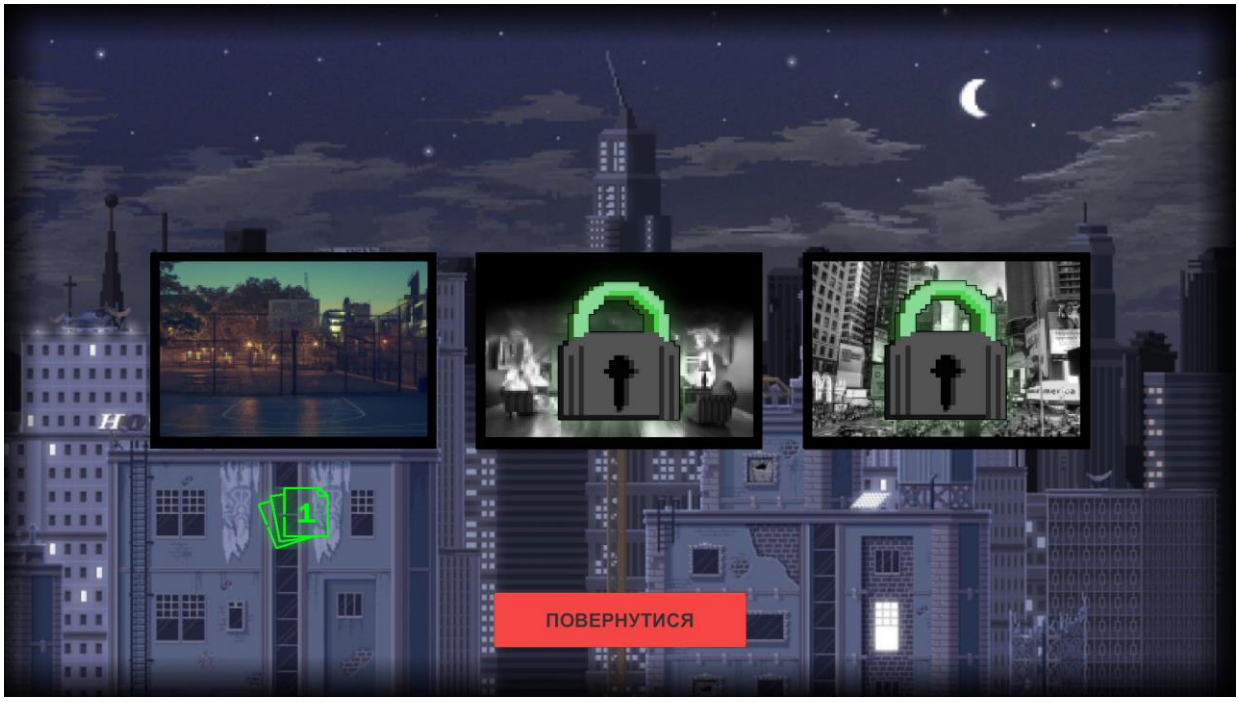


Рисунок 7 – Заблоковані рівні

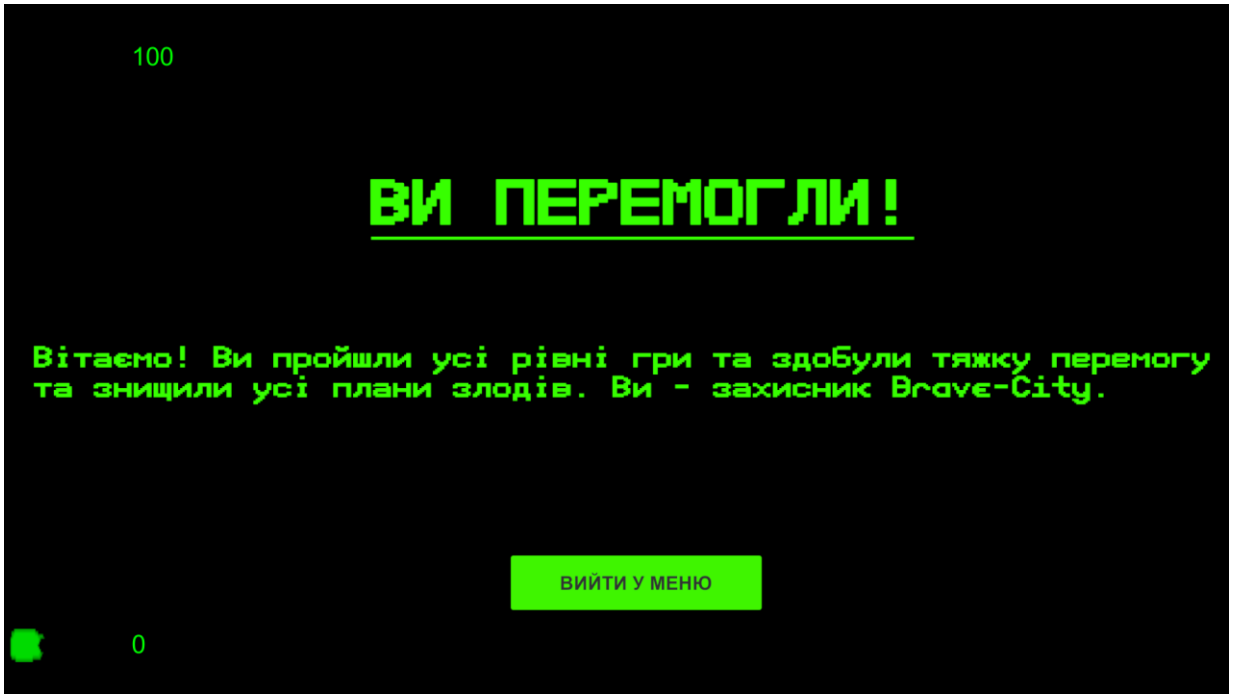


Рисунок 8 – Кінець гри