Полтавський університет економіки і торгівлі

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ГРИ З ЕЛЕМЕНТАМИ ГЕЙМІФІКАЦІЇ ДЛЯ ВИВЧЕННЯ ДИСЦИПЛІНИ ІНФОРМАЦІЙНІ МЕРЕЖІ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Кущ Максим Андрійович
_____«____»_____ 202_ р.
(підпис)

Науковий керівник к. ф.-м. н., доцент, Ольховська Олена Володимирівна
_____«____»_____ 202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Терміни	Пояснення термінів
Проєктування програмного забезпечення	Етап розробки, на якому визначається архітектура, логіка взаємодії між C#-скриптами, інтерфейс користувача. Метою проєктування є створення кросплатформеної, оптимізованої навчальної гри.
Гейміфікація навчання	Метод інтеграції ігрових механік у освітній процес для підвищення залученості у вивчення дисципліни серед студентів.
Ігровий рушій	Спеціалізоване програмне забезпечення, яке надає базові технології для розробки ігор.
Canvas UI	Компонент системи графічного інтерфейсу Unity, який виступає в ролі простору, де розміщуються усі елементи керування грою.

ВСТУП

У сучасному світі цифрових технологій інформаційні мережі відіграють важливу роль у житті кожної людини. Найбільш актуальними являються програми, які призначені для покращення навчання серед студентів, оскільки ефективне навчання є одним із ключових показників успішності серед студентів.

Традиційні методи навчання мають низку недоліків: висока ймовірність помилок, відсутність мотивації, складність у розумінні нового матеріалу. У зв'язку з такими проблемами виникає потреба у створенні програмного забезпечення, яке забезпечить студентам надійну та ефективну можливість вивчати інформаційні мережі не боячись допускати помилки.

Проєктування програмного забезпечення для вивчення дисципліни "Інформаційні мережі" передбачає в собі комфортний користувацький інтерфейс, покрокове підвищення рівня складності та можливість перезапуску рівня в разі невдачі. Важливим аспектом є можливість подальшого розширення функціонала гри. [1]

Мета роботи – розробка навчальної гри з дисципліни "Інформаційні мережі", спрямованої для підвищення мотивації та залученості студентів до вивчення даної дисципліни.

Об'єкт розробки – програмне забезпечення для гейміфікації навчання з дисципліни "Інформаційні мережі".

Предмет розробки – методи, алгоритми та архітектурні рішення проєктування навчальної гри, включаючи логіку, структуру рівнів, комфортний інтерфейс та кросплатформеність.

Методи дослідження – застосовувався аналіз предметної області для вивчення специфіки дисципліни "Інформаційні мережі" та визначення вимог до гейміфікації навчального матеріалу. Метод системного аналізу дозволив розглянути гру як цілісну систему, що включає в себе

взаємопов’язані компоненти. Моделювання використовувалось для побудови логіки взаємодії з мережевими об’єктами та архітектури ігрових рівнів. Проєктування програмного забезпечення забезпечило виначення структуру гри та сценарії користувача. Під час безпосередньої реалізації застосовано метод об’єктно-орієнтованого програмування мовою C# в середовищі розробки Unity, що забезпечило гнучкість в кодуванні. Окремий метод аналізу використовувався для інтеграції окремих компонентів гриу фінальну кросплатформенну збірку. [6]

Актуальність розробки – у сучасному розвитку інформаційних технологій традиційні методи навчання часто демонструють неефективне залучення студентів. Дисципліна “Інформаційні мережі” є однією з найважливіших, але водночас і складною для сприйняття через великий обсяг теоретичного матеріалу. Застосування гейміфікації дозволяє замінити нудний освітній процес, підвищити мотивацію та стимулювати інтерес у студентів. Розробка такого програмного забезпечення відбувається у вигляді навчальної 2D-гри дає змогу візуалізувати складні мережеві процеси.

1.ПОСТАНОВКА ЗАДАЧІ

Метою даної роботи є розробка навчальної 2D-гри для вивчення дисципліни “Інформаційні мережі” для підвищення мотивації та ефективне засвоєння знань серед студентів. Для досягнення цієї мети необхідно вирішити такі завдання:

1. Провести аналіз предметної області, дослідити специфіку гейміфікації та визначити основні вимоги до ігрового застосунку.
2. Розробити логічну структуру ігрових рівнів, що відобража.ть концепції та принципи інформаційних мереж.
3. Спроєктувати архітектуру програмного забезпечення, що забезпечує взаємодію між ігровим рушієм, діями користувача, логікою прогресу та інтерфейсу.
4. Реалізувати гру мовою C# з використанням ООП в середовищі розробки Unity.
5. Забезпечити кросплатформеності для можливості його стабільного запуску в операційних системах Windows та Linux.
6. Розробити гнучкий та інтуїтивно зрозумілий інтерфейс користувача для забезпечення комфортного використання програмного забезпечення.
7. Провести тестування програмного забезпечення для конкретності роботи, надійності та відповідності поставленим вимогам.

Виконання визначених завдань дозволить створити функціональну та кросплатформену навчальну гру, що допоможе підвищити залученість серед студентів до вивчення інформаційних мереж.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд розробленої навчальної 2D-гри

Насамперед потрібно інсталювати гру. Для цього потрібно завантажити архів з відповідною версією гри, залежить від системи. В доступі є наявні дві збірки під системи Windows та Linux. Вибраний архів слід розпакувати у зручний для використання каталог.

Запуск гри відбувається за допомогою відповідного файлу в розпакованому архіві:

1. Для ОС Windows - запуск файлу з розширенням .exe;
2. Для ОС Linux – запуск файлу з розширенням .x86_64.

Навчання в додатку спрямоване на формування у студентів на розумінні побудови мережевих поток та маршрутизації даних. Гравець має можливість взаємодіяти з ігровим полем, розташовувати та обертати блоки-відбивачі для того, щоб спрямувати відповідний потік інформації під передавача до кінцевої мережі (рисунок 2.1).

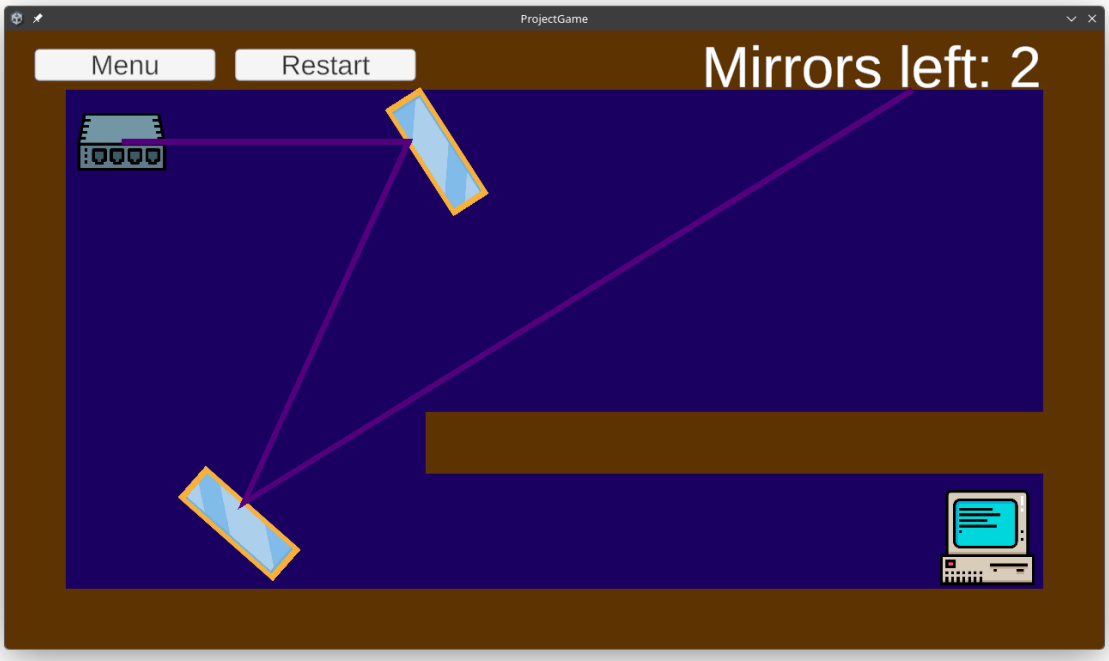

Рисунок 2.1. – Інтерфейс міні-гри “Відбивання променю даних”

На даному скриншоті (рисунок 2.1) продемострovано робоче вікно першої міні-гри. У лівому верхньому кутку екрану розташований комутаційний пристрій (маршрутизатор), якйи виступає джерелом генерації променю даних. Мета студента в даній грі довести даний промінь до кінцевої точки (комп’ютера), що знаходиться у правому нижньому кутку, уникаючи перешкод на рівні.

Для навігації під час гри вгорі додано графічний інтерфейс для користувача. Кнопки “Menu” для виходу у головне меню та “Restart” для перезапуску поточного рівня. У правому верньому кутку відображається лічильник “Mirrors left”, який вказує на кількість доступних відбивачів, що обмежують максимальну кількість кроків та змушує студента шукати найбільш короткий і оптимальний маршрут.

Друга міні-гра у навчальному додатку це гра “З’єднання дротів”, логіка гри заключається в процесі побудови кабельних систем. Вигляд робочого простору міні-гри наведено в рис. 2.2.

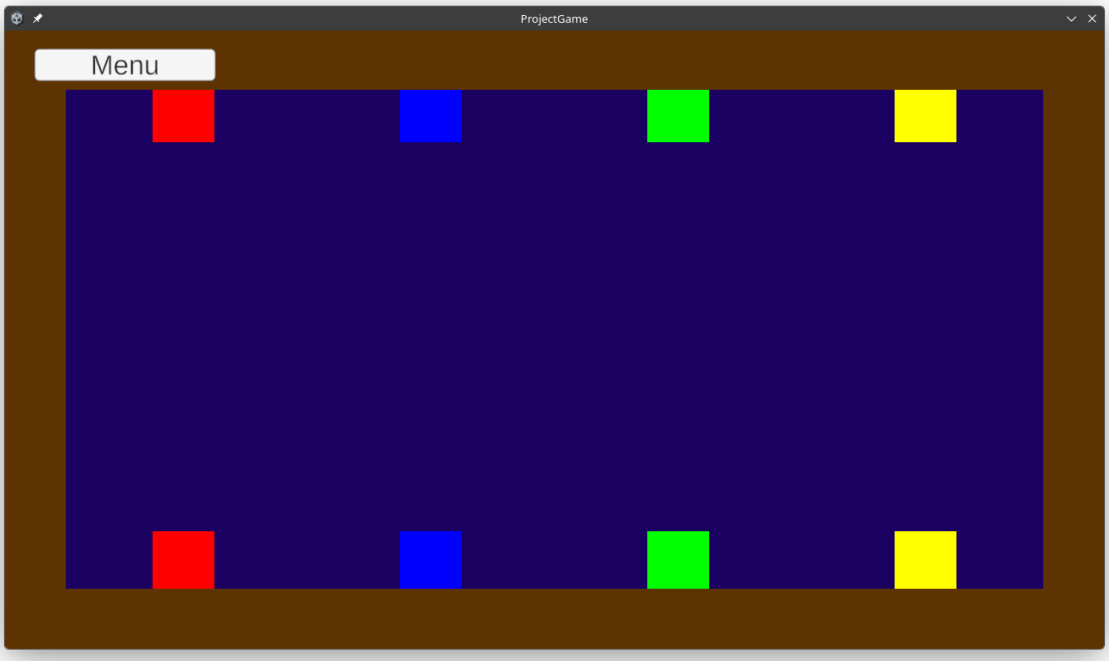

Рисунок 2.2. – Інтерфейс міні-гри “З’єднання дротів”.

На скриншоті (Рисунок 2.2) зображено початкову сцену міні-гри, де у верхній та нижній частинах розташовані порти підключення, що мають відповідні кольори. Завдання студента полягає у з’єднанні дротів відповідних кольорів.

З’єднання дротів виконується за допомогою мишки шляхом утримання та перетягування кабеля від початкової точки до кінцевої відповідного кольору. Для зручності у лівому верхньому кутку екрану додано кнопку “Menu”, яка дозволяє повернутись до головного меню гри.

2.2. Огляд навчальної гри-головоломки “7 Billion Humans”

Розглянемо начальну гру-головоломку “7 Billion Humans”. [2]

Насамперед гру потрібно інсталювати. Для цього потрібно знайти гру під назвою 7 Billion Humans в крамниці “Steam” (рис. 2.3), або перейти за відповідним посилання в браузері.

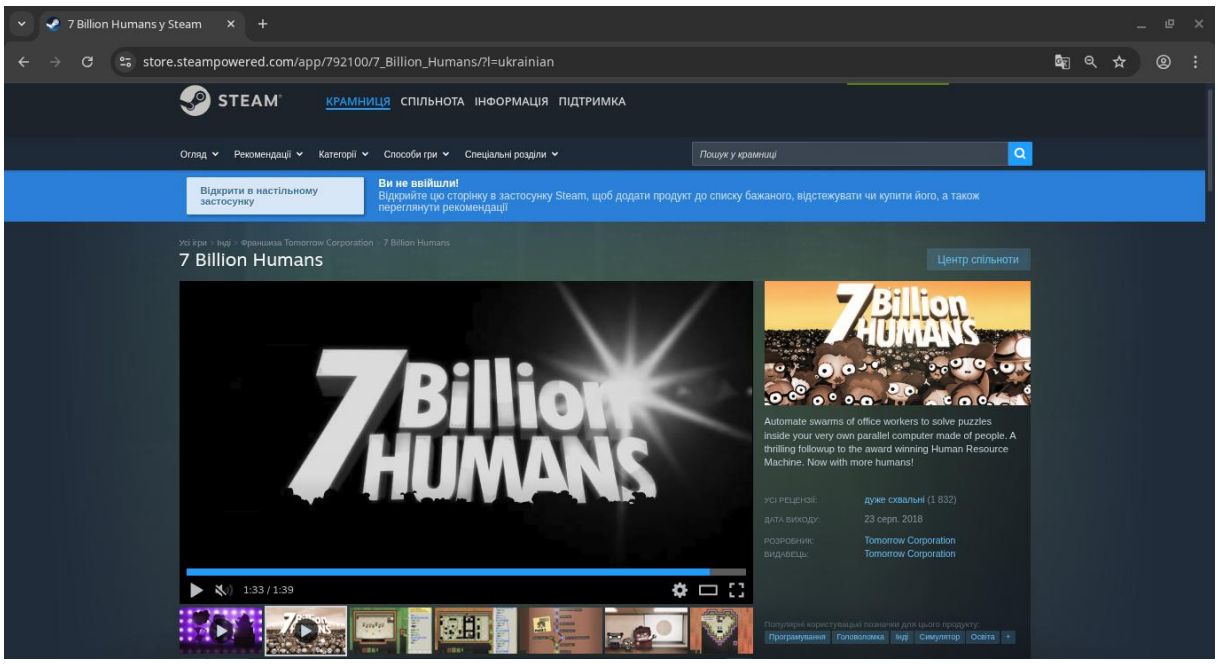

Рисунок 2.3 – Сторінка гри в крамниці Steam

У грі передбачені зручні інструменти для виконання рівнів, що допомога уникнути можливості виникнення критичних помилок. Користувач може задавати логічні умови, зокрема перевірку наявності перешкод, маршрутів завдяки зазначенню конкретних параметрів взаємодії (Рисунок 2.4).

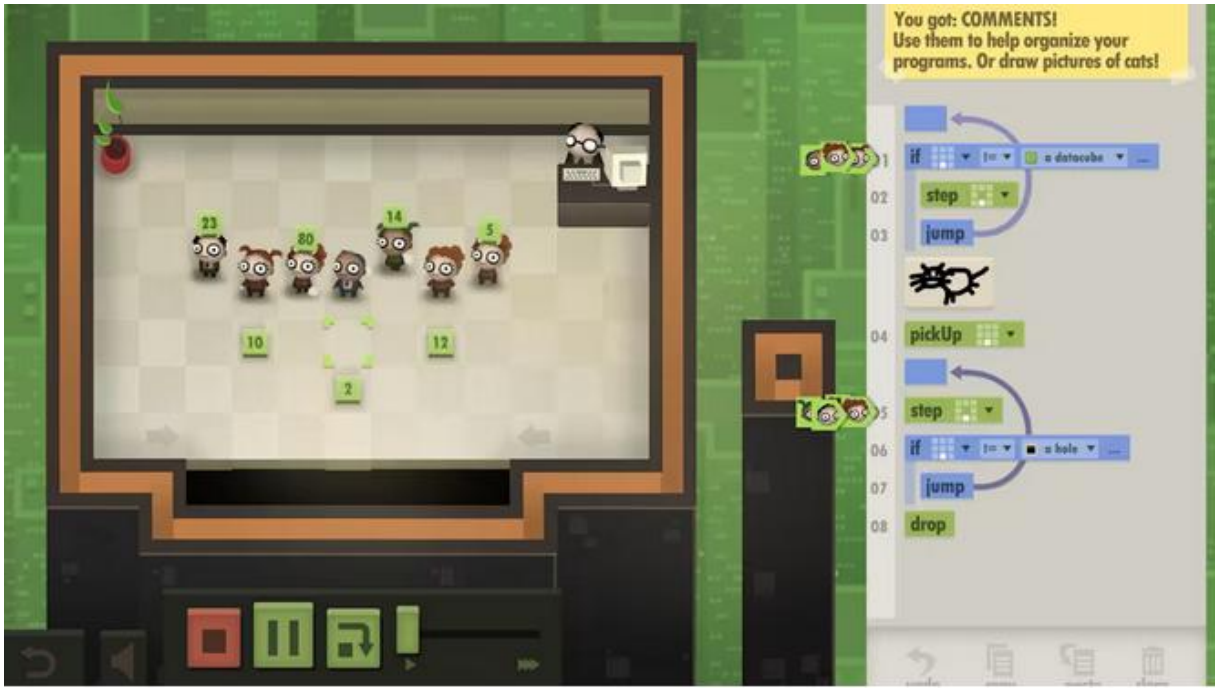

Рисунок 2.4 – Приклад взаємодії користувача з грою

У грі є візуалізація пройдених рівнів (Рисунок 2.5), а також поступове підвищення рівня складності, що відіграє важливу роль у розумінні структури та правильності дій користувача.

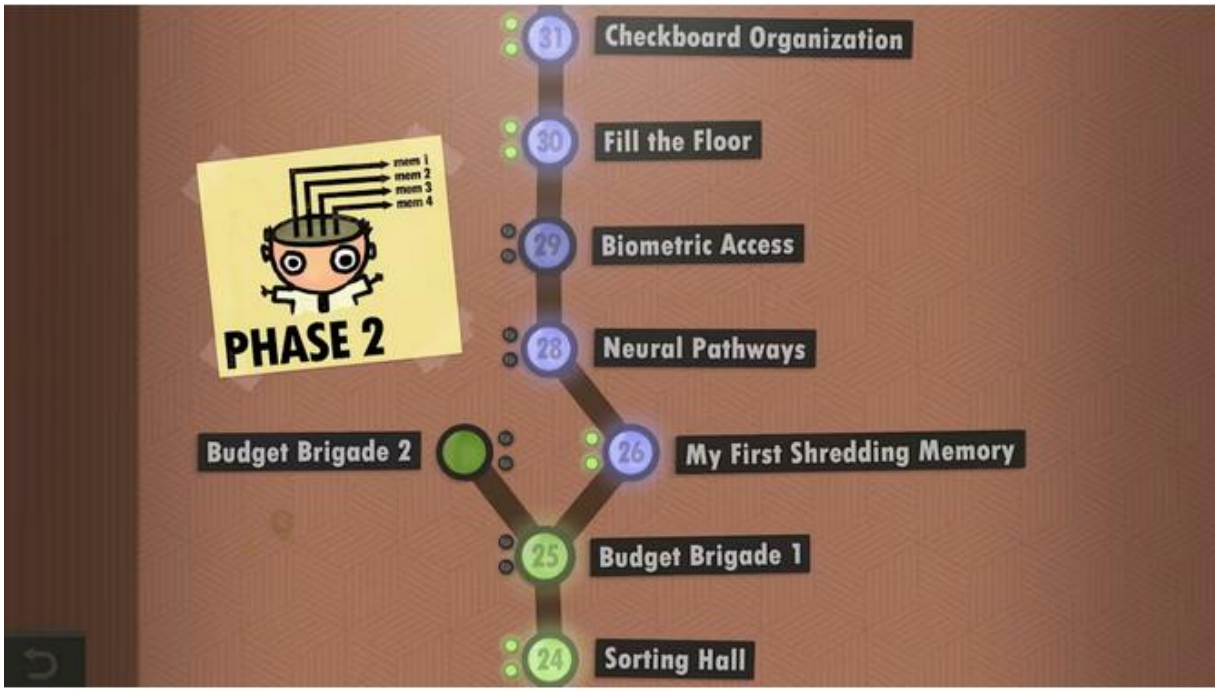

Рисунок 2.5 – Дерево рівнів

У грі передбачено також меню коли рівень був успішно пройдений. В даному меню вказано за скільки ходів можна оптимально справитись з даним завданням, у разі виконання цієї умови можна отримати додаткові бали, або просто продовжити гру (Рисунок 2.6).

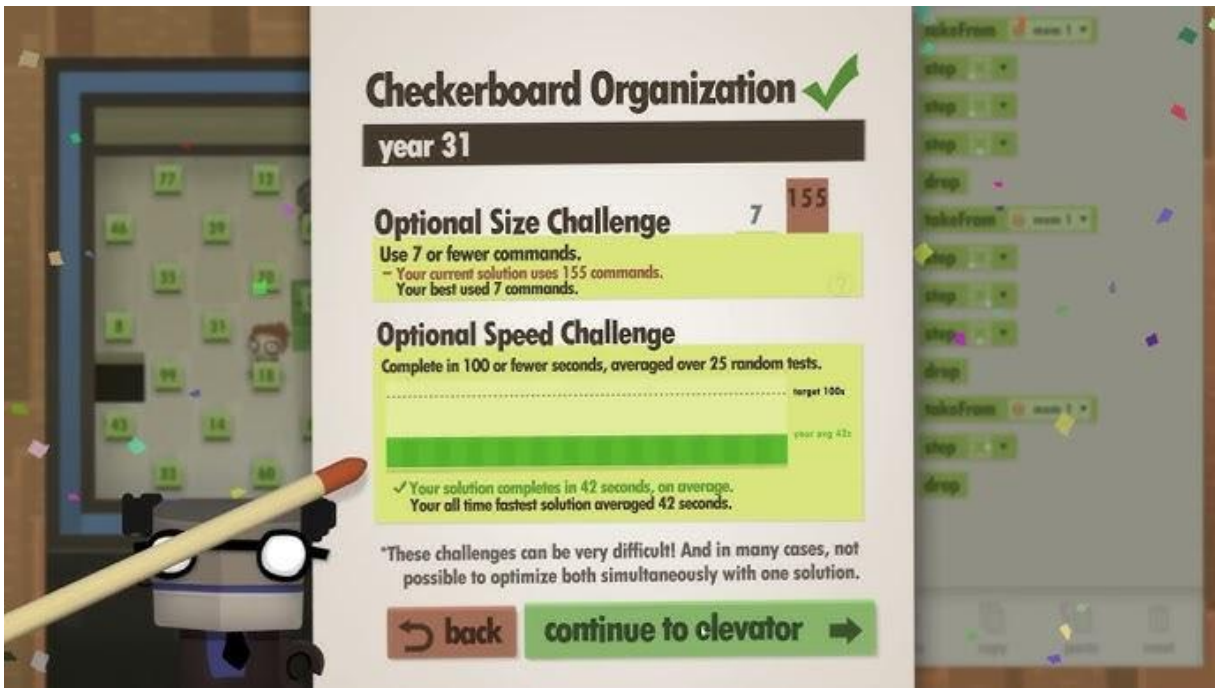

Рисунок 2.6 – Меню пройденого рівня

Також крім навчальних рівнів є розважаючі, які слугують невеличким відпочинком, зазвичай такі рівні або мають мінімальну кількість команд, або і зовсім спроєктовані у повностю розважальном характері (Рисунок 2.7).

Рисунок 2.7 – Рівень для відпочинку

Переваги гри:

1. Дозволяє детально контролювати детальні зв’язки, планувати рішення рівня, що робить додаток зручним в межах навчання логіки.
2. Забезпечує можливість взаємодіяти з великою кількістю об’єктів одночасно, фіксувати циклами повторювані операції, а також розділяти роботу між персонажами гри.
3. Присутня інтерактивна мапа рівнів, запам’ятовування пройдених рівнів, опрацювання побудованої користувачем логіки.
4. Підтримка кросплатформеності, стабільна робота на різниї операційних системах, в тому числі і мобільних телефонах.

Недоліки гри:

1. Вимагає придбання платної ліцензії.
2. Відсутня можливість перегляду проходження рівня іншими користувачами.
3. Відсутність підсказок під час проходження важких етапів гри.

Гра 7 Billion Humans є доволі непоганою в сфері навчальних ігор. ЇЇ основними перевагами є візуалізація виконання коду, логічно структуровані завдання, покрокова перевірка складеного алгоритма, а також що дуже важливо це оригінальний підхід до автоматизації процесів. Загалом ця гра добре підходить для знайовста з концепцією багатопотокових обчислень та розвитку алгоритмічного мислення.

3.ТЕХНІЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1. Архітектура та керування об’єктами в середовищі Unity

Для того щоб гра стабільно працювала у проєкті було реалізовано систему керування об’єктами. Оскільки гра задумувалася для запуску на навчальних комп’ютерах, тому замісти використання важких даних уся логіка була побудована вимключно на базі компонентів Unity: менеджері сцен, префабів та вбудованих класів.

Взаємодія користувача з середовищем у кожній із двох міні-ігор реалізовано завдяки цим трьом основним процесам: ініціалізації рівнів, зчитування дій та динамічному оновленні. Кожен із них являє собою наступне:

1. Ініціалізація рівнів – архітектура завантаження рівнів, розташування джерел та цільових вузлів.
2. Зчитування дій – відстеження взаємодії з відбивачами, натискання кнопок інтерфейсу та перетягування дротів мишею.
3. Динамічне оновлення – щокадровий перерахунок траєкторії потоку даних та щокадрове відмальовування дротів.

Технічно гра розділена на дві частини. За розрахунки та логіку відповідають скрипти на C#, а за взаємодію студента з об’єктами – інтуїтивно зрозумілий інтерфейс.

Керування ігровим процесом у розробленій грі побудований на взаємодії кількох частин коду. Під час старту будь-якого рівня система автоматично зчитує конфігурацію обраного рівня та запускає алгоритм підготовки сцени. У цей момент для першої міні-гри активується скрипт RayEmitter, який відповідає за випуск променю даних із маршрутизатора та встановлює ліміт доступних відбивачів, тоді як друга міні-гра з дротами створює на полі порти з відповідними кольорами.

Важливу роль у стабільності геймплею відіграє система відстеження зіткнень, яка постійно перевіряє стан об’єктів під час гри. За допомогою стандартних інструментів Unity система може чітко визначати чи торкнувся промінь даних встановленого відбивача під правильним кутом, або чи збігаються координати променю з кінцевим комп’ютером. Коли студент взаємодіє з елементами гри, програма в реальному часі оновлює координати та миттєво відображає зміни на екрані.

Тестування розробленої гри полягало у перевірці стабільності ігрового процесу, коректності відбивання променів та коректної фіксації дротів відповідних кольорів. Тестування проводилось запуском гри на двох ріних платформах: Windows 11 та Linux CachyOS.

Тестування ігрових функцій:

1. Тестування запуску та обмежень рівнів:
    - Перевірка коректного запуску рівнів.
    - Перевірка коректного відображення інтерфейсу при запуску рівня
    - Перевірка роботи лічильника відбивачів.

2. Тестування трасування та логіки променів:
    - Перевірка обчислення кута відбиття променю при взаємодії з відбивачем.
    - Перевірка поведінки променю при зіткненні з перешкодами.

3. Тестування з’єднання дротів:
    - Перевірка плавності слідування дроту за курсором миші.
    - Тестування логіки з’єднання дротів.

4. Тестування стабільності кросплатформеності:
    - Перевірка відсутності затримок у двох версіях гри, або вильотів.
    - Перевірка коректності у роботі файлів двох збірок.

3.2. Покроковий алгоритм роботи коду

Запуск гри зустрічає нас мінімалістичним головним меню (Рисунок 3.1), де користувачеві доступні дві кнопки: “Play” для початку гри та “Exit” для виходу з гри. Якщо натиснута кнопка “Exit” гра завершує свою роботу. При натисканні на “Play” запускається скрип менеджеру сцен SceneManagent і перекидує користувача до наступного вікна вибору рівнів (Рисунок 3.2).

Рисунок 3.1 – Головне меню

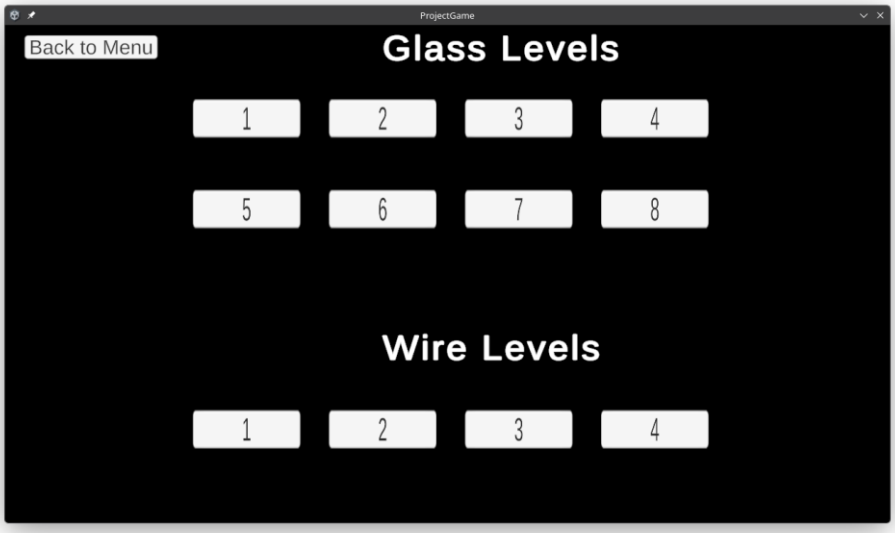

Рисунок 3.2 – Меню вибору рівнів

На рис. 3.2 відображаються доступні рівні. Як тільки користувач натискає на один із дванадцяти доступних рівнів уся подальша логіка передається скрипту LevelManager. Скрипт автоматично считує дані обраного рівня і додає необхідні елементи рівня на ігрове поле.

У межах першої міні-гри “Відбивання променю даних” запускається скрипт RayEmitter, який прораховує рух променю за допомогою трасування Physics2D.Raycast. Одночасно з цим на інтерфейсі оновлюється лічильник доступних відбивачів “Mirrors left”. При натисканні на пустому місці на робочому вікні з’являється відбивач променю і студент може обертати його змінюючи напрямок променю. Якщо промінь досягає кінцевої точки в мережі (комп’ютера) рівень вважається пройденим і можна перейти до наступного рівня натисканням кнопки “Next Level” (Рис. 3.3). Якщо користувач припустився помилки, то натискання кнопки “Restart” перезапускає поточний рівень у початковий стан.

Рисунок 3.3 – Успішно завершений рівень

У межах другої міні-гри “З’єднання дротів” на старті ініціалізуються порти з відповідними кольорами. Скрипт WireController починає

відстежувати події миші. Якщо буде утримана ліва кнопка миші на одному із початкових кольорових портів запускається алгоритм динамічного малювання лінії за допомогою інструмента LineRenderer. Гра щокадрово зчитує місце знаходження курсора і оновлює точки. У момент відпускання кнопки миші система перевіряє чи сходяться кольори портів, якщо кольори збігаються з'єднання фіксується, а якщо студент припустився помилки, то лінія автоматично видаляється.

Крок 1. Запуск гри

При відкритті гри:

- Ініціалізуються елементи інтерфейсу головного меню.
- Візуалізуються дві основні кнопки "Play" та "Exit"
- При насканні "Play" відкривається вибір рівня.

Крок 2. Ініціалізація ігрової сцени.

Гра налаштовує параметри:

- Завантажує префаби джерел даних, приймачів та комутаторів.
- Виставля стартові координати префабів.
- Оновлює інтерфейс користувача.

Крок 3. Взаємодія користувача з полем.

Користувач виконує дії:

- Змінює кути нахилу відбивачів.
- Утримує та перетягує дроти від початкого дроту до кінцевого.

Крок 4. Динамічний перерахунок та трасування.

Щокадрово виконується:

- Запуск променю через метод Physics2D.Raycast.
- Обчислення точок зіткнення та відбиття.
- Візуалізація з'єднаннь через LineRenderer.

Крок 5. Перевірка логічних умов проходження.

Система аналізує фінальні результати:

- Промінь данних досяг комп’ютера – рівень пройдено.
- Дроти однакових кольорів з’єднані – рівень пройдено.

Крок 6. Рестарт рівня.

При натисканні кнопки “Restart”:

- Лінії та об’єкти стираються з пам’яті.
- Всі елементи повертаються до початкових значеннь.

Крок 7. Завершення роботи.

При натисканні кнопки “Exit” у головному меню гра закривається.

Реалізований алгоритм успішно вирішує завдання моделювання процесів та динамічного оновлення ігрового екрану. Завдяки використанню механізму швидкої перевірки тегів додаток працює стабільно і без збоїв. Проєкт не перевантажує комп’ютер, що гарантує надійне функціонування на Windows і Linux-платформах.

3.3. Загальна UML діаграма роботи навчальної гри

Загальна UML діаграма роботи навчальної гри (рис. 3.4).

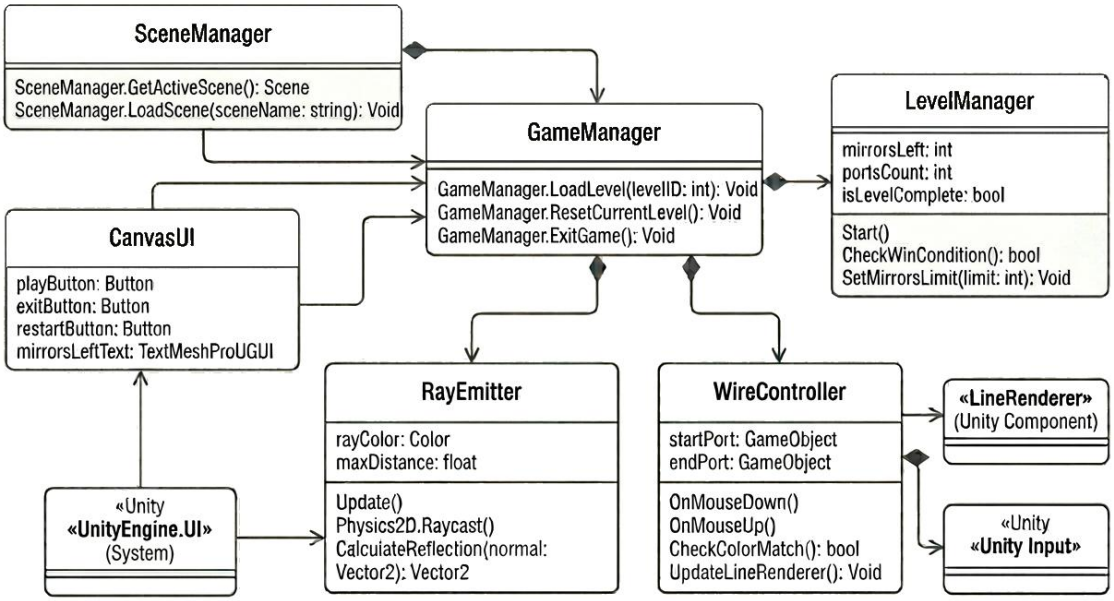

Рисунок 3.4 – UML діаграма навчальної гри

–

4.ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1. Обґрунтування вибору програмних засобів

Для розробки навчальної гри з елементами гейміфікації було обрано мову програмування C# у поєднанні з ігровим рушієм Unity, що обумовлено рядом технічних переваг. По-перше, C# є об’єктно-орієнтованою мовою програмування, що дозволяє створювати чітко структуровану архітектуру проекту з високим рівнем повторного використання коду через класи, можливістю наслідування, інкапсуляції. У комплекті з ігровим рушієм Unity це надає змогу у легкому керуванні окремими компонентами рівнів, де кожен із елементів може функціонувати як незалежний екземпляр. Такий підхід спрощує в подальшому безпроблемна додавати нові рівні, або механіки.

По-друге, мова C# разом із ігровим рушієм Unity надає доступ до вбудованих фізичних та графічних бібліотек. Зокрема, компонент Physics2D дозволя реалізовувати точне трасування променів за допомогою Raycast та миттєво рахувати кут відбиття. Також одна із чудових вбудованих фішок в Unity це рендеринг ліній Line Renderer та управління сценами Scene Management це дозволяє суттєво скоротити час розробки, при цьому гарантуючи високу продуктивність завдяки оновленню щокадровому в реальному часі.

По-третє, використання Unity дозволяє ефективно створювати сучасний та зрозумілий графічний інтерфейс за допомогою Unity UI. Якісний візуальний інтерфейс є одним з ключових факторів для комфортного користування програмою.

Важливим етапом в розробці є використання інтегрованого середовища, яке надає засоби для написання та налагодження коду, покращення загальної продуктивності та відстеження використання

оперативної пам’яті комп’ютера. Це дало змогу виявляти та усувати помилки ще на етапі тестування.

Крім того, Unity та C# забезпечують бездоганну кросплатформеність проєкту. Створена програма демонструє високу оптимізацію розрахунків, що забезпечує стабільну роботу як на Windows, так і на Linux-системах.

Таким чином, вибір мови програмування C# та ігровий рушій Unity обґрунтований поєднанням простоти розробки та гнучкості налаштувань. Використання такого поєднання дозволило реалізувати гейміфікацію для вивчення інформаційних мереж, а також створити умови для подальшого розвитку та модернізації гри.

4.2. Опис програмної реалізації

Навчальну гру для дисципліни “Інформаційні мережі” розроблено на базі мови програмування C# та ігрового рушія Unity [7, 10]. Для написання скриптів керування було використано текстовий редактор Kate (Рисунок 4.1), який є вбудованим інструментом операційних систем на базі Linux в середовищі KDE Plasma. Вибір цього текстового редактора зумовлений його швидкодією та наявністю непоганого інструментарію для розробки. Редактор підтримує підсвічування синтаксису мови C#, підтримує інтеграцію з терміналом, а також зручну роботу з декількома скриптами одночасно.

Рисунок 4.1 – Текстовий редактор Kate

Проєктна реалізація гри починається зі створення сцени головного меню та розміження UI-елементів, серед яких є кнопки “Play” для початку гри та “Exit” для виходу з гри (Рисунок 4.2).

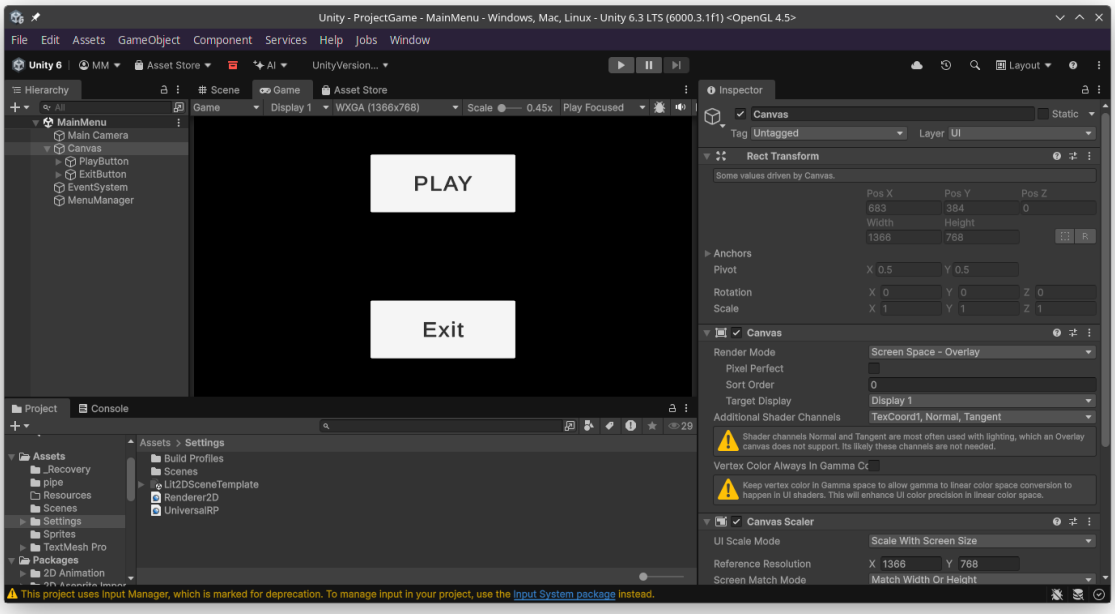

Рисунок 4.2 – Дизайн головного меню

Після розміщення елементів інтерфейсу в Canvas необхідно запрограмувати логіку натискання через обробку події On Click (Рисунок 4.3).

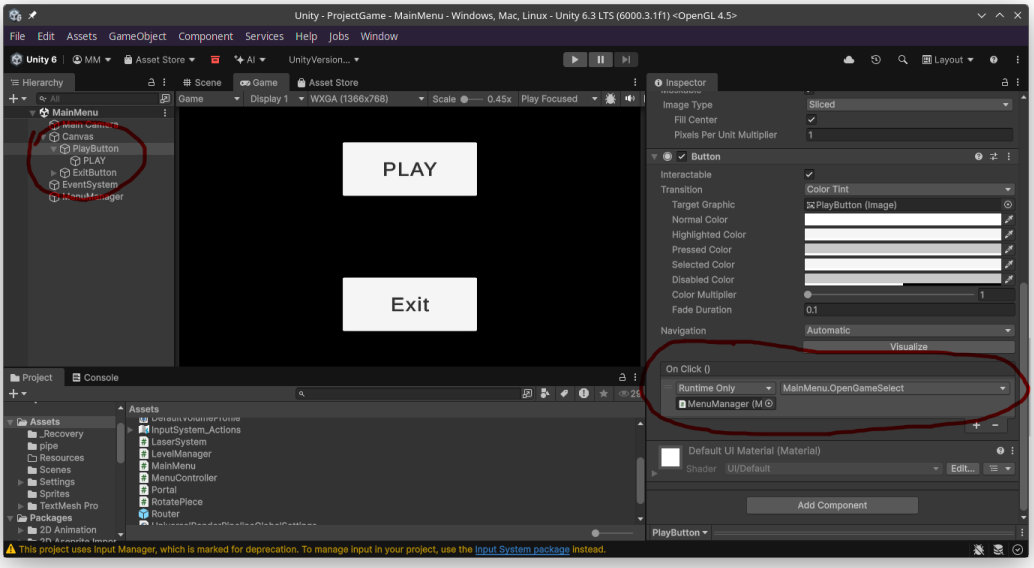

Рисунок 4.3 – Взаємодія з UI-елементами

Після натискання кнопки “Play” метод On Click викликає алгоритм OpenGameSelect, який завантажує екран вибору рівнів, також на цьому екрані є можливість повернення в меню (Рисунок 4.4).

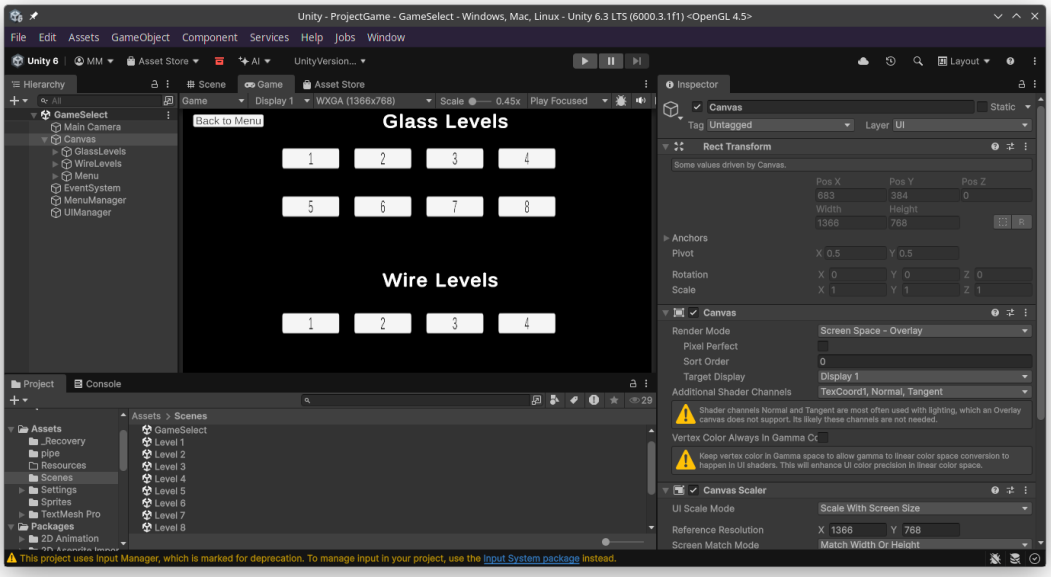

Рисунок 4.4 – Меню вибора рівнів

Після створення меню вибору рівнів потрібно прописати логіку для LevelManager, який відповідає за динамічне налаштування обраного рівня (Рисунок 4.5).

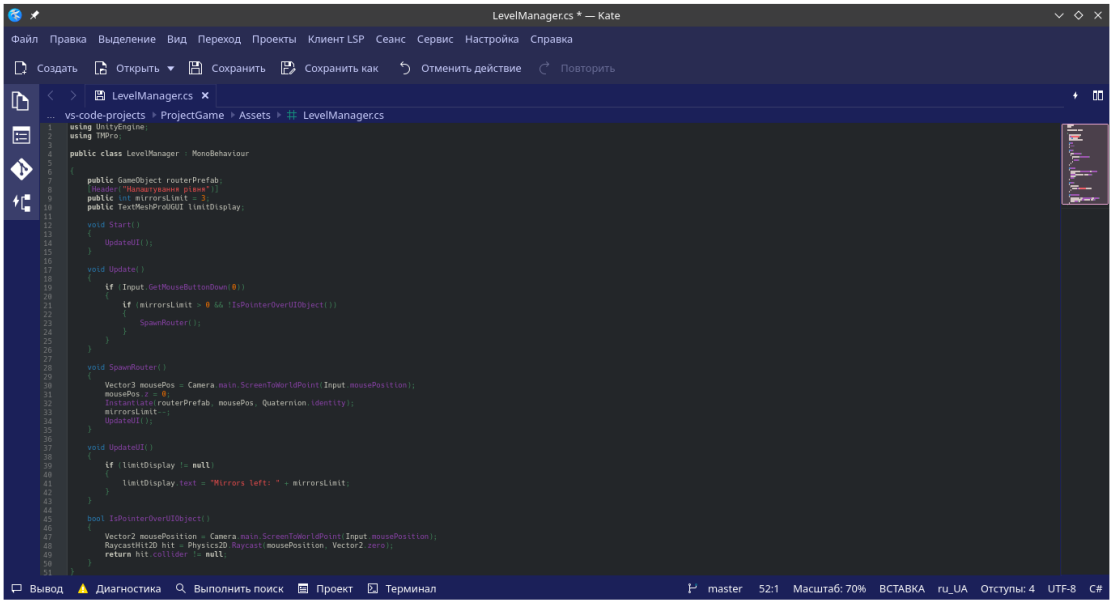

Рисунок 4.5 – Код реалізації менеджеру рівнів

Для першої міні-гри у коді реалізовано роботу скрипту RayEmitter, який за допомогою Physics2D.Raycast встановлює обмежену кількість відбивачів (Рисунок 4.6).

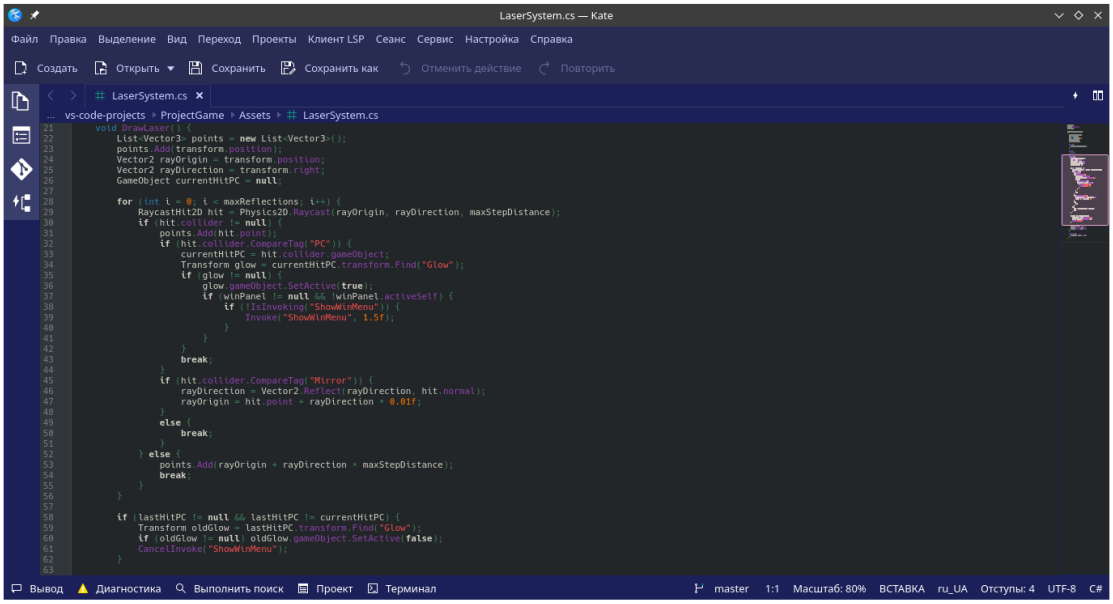

Рисунок 4.6 – Код реалізації відбивачів

Для другої міні-гри розроблено алгоритм, який перевіряє порти і підключає лише порти відповідних кольорів між собою (Рисунок 4.7).

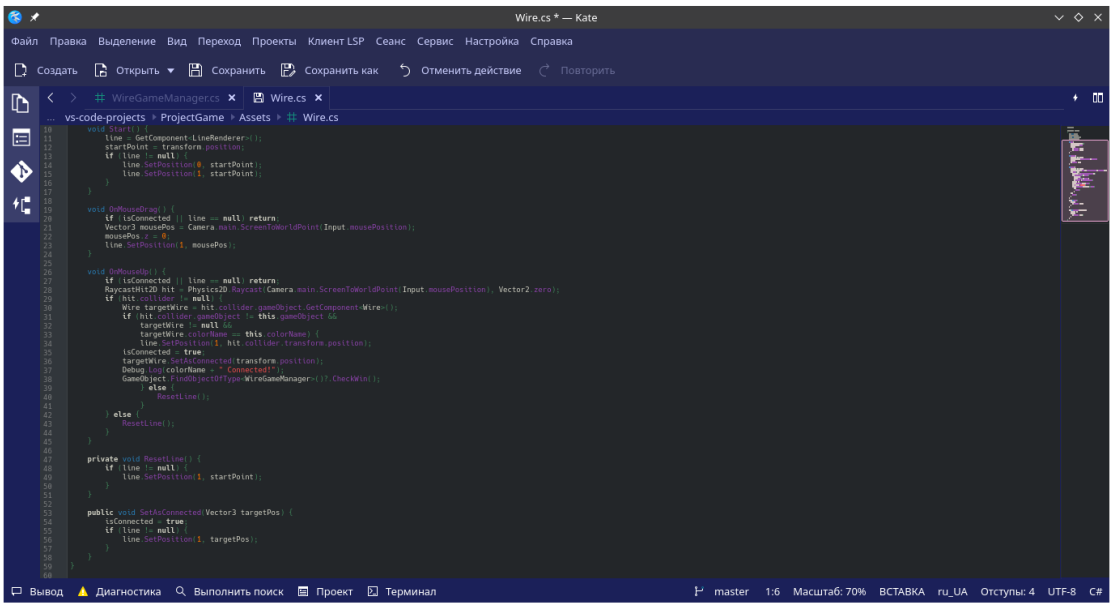

Рисунок 4.7 – Код реалізації логіки з’єднань дротів

Для фіксації перемоги було реалізовано тригери, які обробляють надходження даних, чи потрапив промінь даних до комп’ютера і цим

самим зараховується перемога (Рисунок 4.7). Таким чином реалізоване і з’єднання дротів, коли усі дроти з’єднані спрацьовує скрипт завершення рівню (Рисунок 4.8).

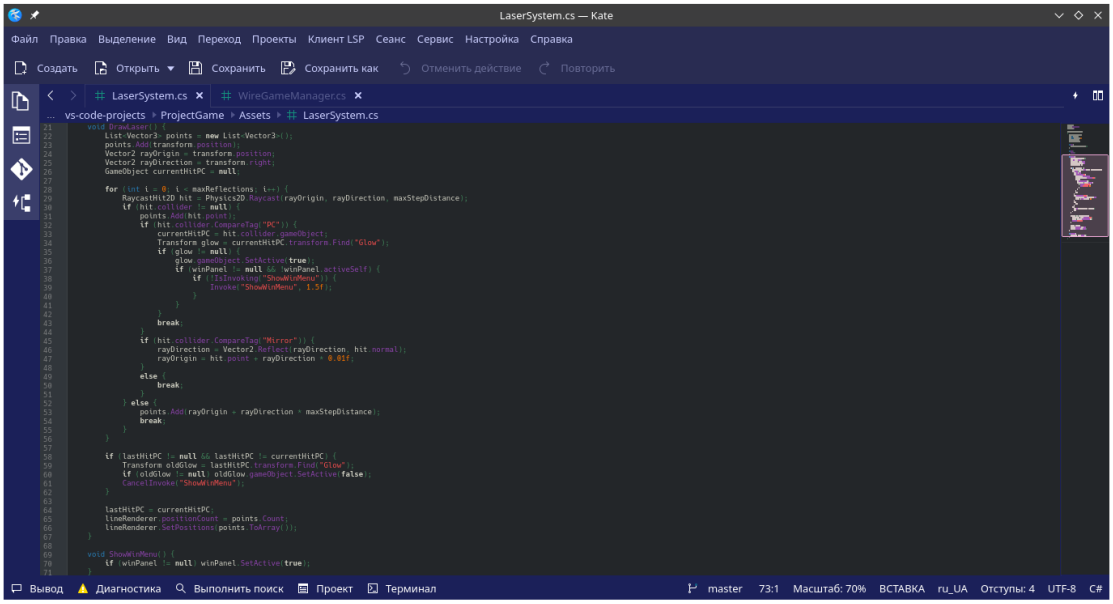

Рисунок 4.7 – Код реалізації перемоги у відбиванні променю

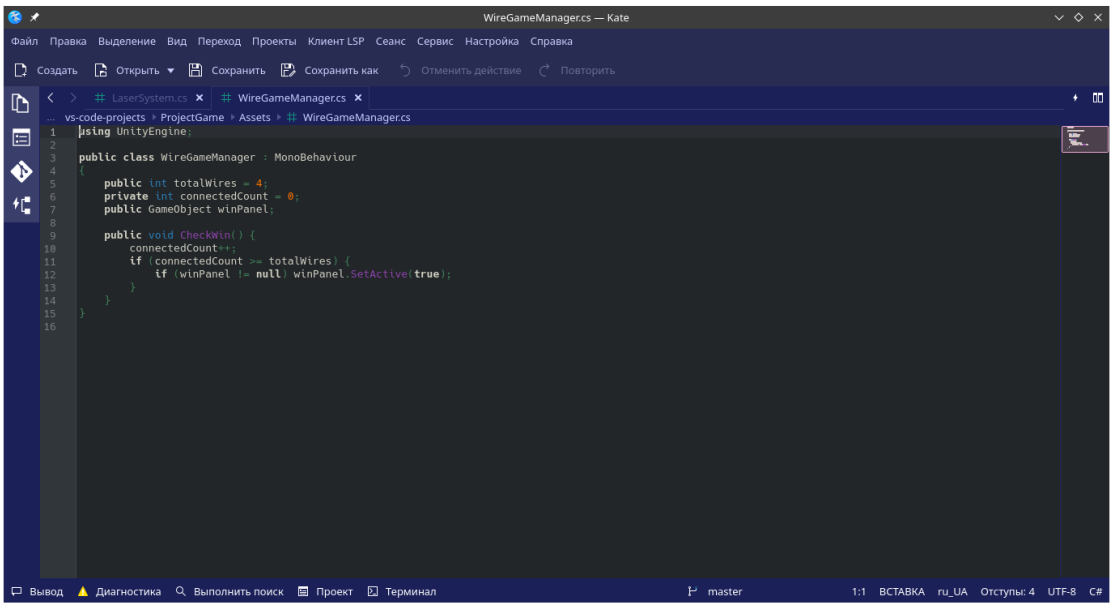

Рисунок 4.8 – Код реалізації перемоги у з’єднанні дротів

Для зручності користувача було додано можливість виходу в меню під час гри (Рисунок 4.9), а також додано кнопку рестарта, яка перезапускає поточну сцену.

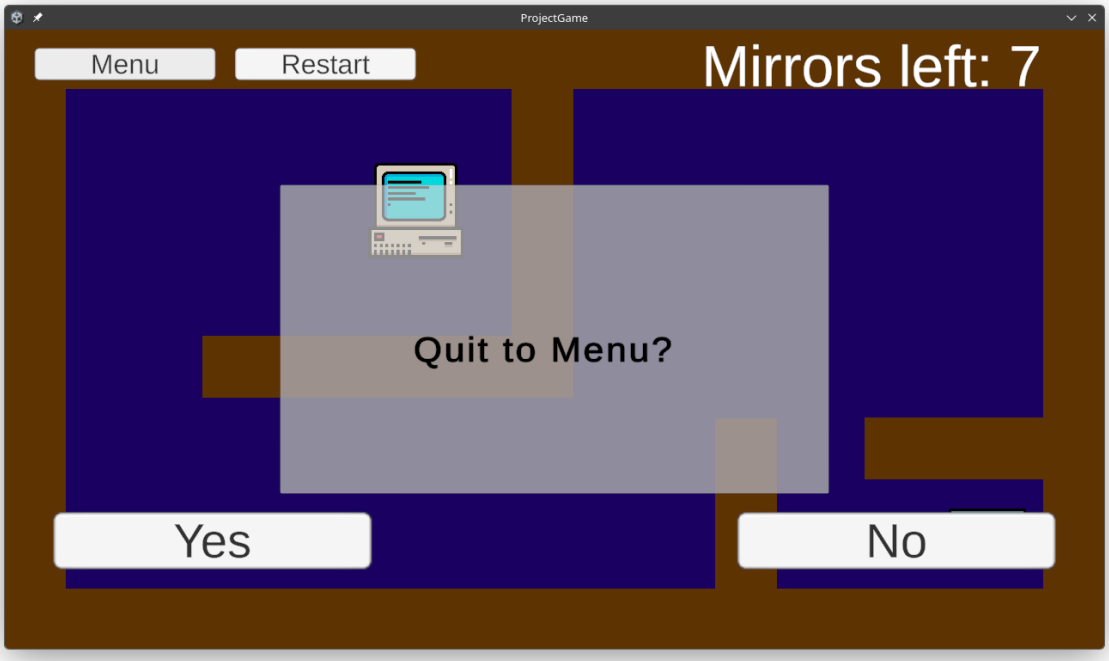

Рисунок 4.9 – Вікно виходу в головне меню

Реалізація рестарту рівня відбувається таким чином, якщо рівень стає неможливо пройти (Рисунок 4.10), то при натисканні кнопки рестарт рівень повертається до початкової сцени рівня (Рисунок 4.11).

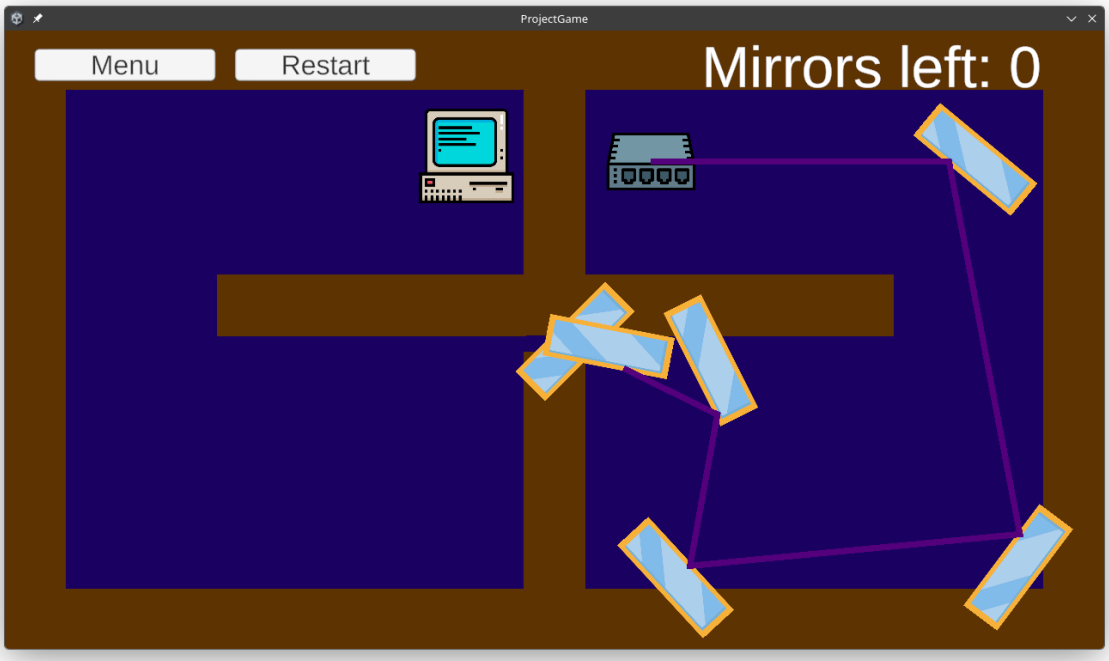

Рисунок 4.10 – Приклад неможливого завершення рівня

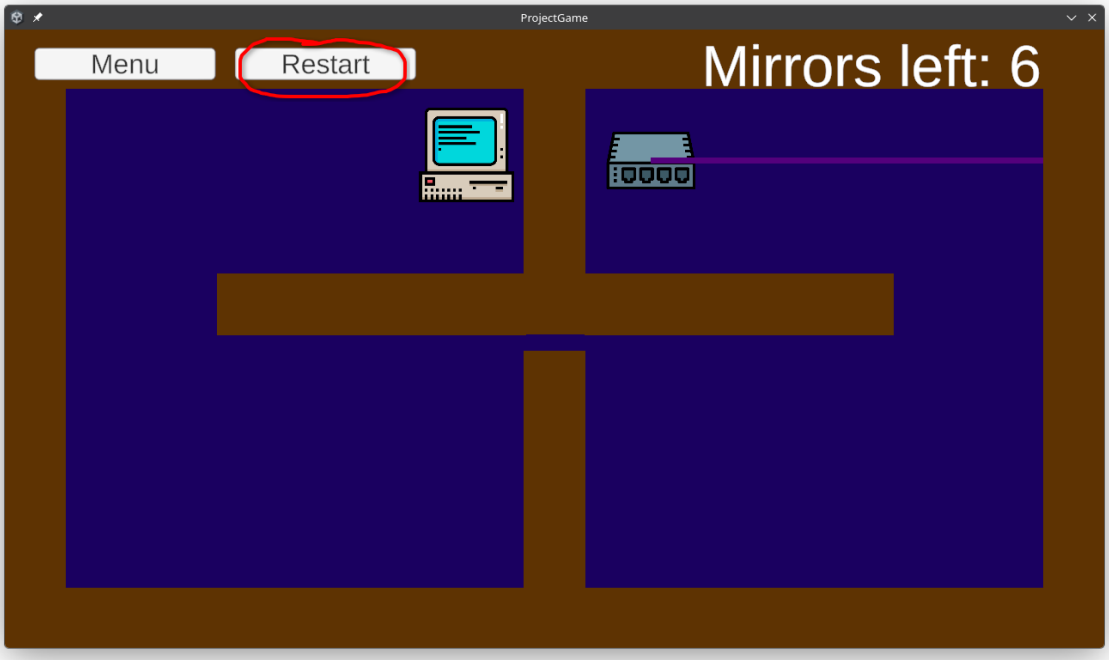

Рисунок 4.11 – Перезапуск рівня після натискання “Restart”

Весь код проєкту був створений за допомогою текстового редактору Kate з компіляцією в ігровому рушії Unity. Рушій в свою чергу забезпечив комфортний інструментарій для проєктування інтерфейсу користувача,

автоматичний прорахунок дій, а також надав можливість відслідковувати поведінку користувача за допомогою Unity Input. Також були використані UnityEngine.UI для відтворення візуальних елементів та UnityEngine.SceneManagment для керування сценами, що дозволило коректно генерувати рівні залежно їх послідовності.

–

ВИСНОВКИ

У ході виконання роботи було проведено комплексний аналіз предметної області, що дозволило визначити основні вимоги до програмного забезпечення, призначеного для вивчення дисципліни “Інформаційні мережі”. Було визначено, що впровадження в навчальний процес елементів гейміфікації з інтуїтивно зрозумілим графічним інтерфейсом суттєво підвищує сприйняття інформації та підвищує рівень залучесності серед студентів..

Навчальну гру реалізовано за допомогою мови програмування C# на рушії Unity з використанням об’єктно-орієнтованого програмування. У грі реалізовані 12 повноцінних рівнів, розділених на дві повноцінні міні-гри.

Проведене тестування підтвердило продуктивність гри, коректність роботи тригерів. Створено інтуїтивно зрозумілий інтерфейс користувача, що підвищує рівень взаємодії з користувачем.

Таким чином, створена гра підтверджує, що впровадження ігрових механік у навчальний процес є чудовим рішенням. Використання рушія Unity та мови C# забезпечило стабільне та оптимізоване функціонування гри під двома операційними системами Windows та Linux.

СПИСОК ЛІТЕРАТУРИ

1. Ольховська О.В., Черненко О.О. методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма "Комп'ютерні науки" ступеня бакалавра / О.О. Черненко, Ольховська О.В. – Полтава : ПУЕТ, 2025. - 58 с.
2. Огляд навчальної гри-головоломки "7 Billion Humans" [Електронний ресурс]: https://tomorrowcorporation.com/7billionhumans
3. Хокінг Дж. Unity в дії. Мультиплатформна розробка на C# / Дж. Хокінг. – СПб.: Пітер, 2019. - 352 с.
4. Шел Дж. Мистецтво ігрового дизайну: Книга лінз / Дж. Шел. – М.: Альпіна, 2019 - 540 с.
5. Бублик В.В. Об'єктно-орієнтоване програмування : підручник. Київ : ІТ-книга, 2019. - 432 с.
6. Васильєв О.Н. Програмування мовою C# : навчальний посібник. Київ : Центр навчальної літератури, 2021. - 312 с.
7. Трофименко О.Г., Прокоп Ю.В. C#. Створення додатків у середовищі Unity : лабораторний практикум. Одеса : Фенікс, 2022. - 180 с.
8. Бондарчук Ю.В. Гейміфікація як іноваційний підхід в сучасній освіті. Актуальні питання гуманітарних наук. 2022. Вип. 49, т1. - 182-188 с.
9. Introduction to Unity UI. Unity Technologies. [Електронний ресурс]:https://docs.unity3d.com/Manual/index.html
10. C# Documentation. Microsoft Learn. [Електронний ресурс]: https://learn.microsoft.com/en-us/dotnet/
11. Ольховський Д.М., Кошова О.П., Воробйов І.О., Стусь С.М. Створення інструменту машинного навчання для інтерактивного

аналізу та представлення табличних даних у реальному часі. Information Technology: Computer Science, Software Engineering and Cyber Security, Вип. 1, 2026, С.198-205. https://doi.org/10.32782/IT/2026-1-23

12. Kapp K. M. The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education. 2nd ed. Hoboken : Wiley, 2022. 480 p.
13. Werbach K., Hunter D. For the Win: The Power of Gamification and Game Thinking in Business, Education, Government, and Social Impact. Revised ed. Philadelphia : Wharton School Press, 2020. 168 p.
14. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2023. 816 p.
15. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2020. 880 p.
16. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 9th ed. Boston : Pearson, 2024. 864 p.

ДОДАТОК А. КОД ПРОГРАМИ

```
// LASER SYSTEM//

using UnityEngine;
using System.Collections.Generic;
using UnityEngine.UI;

public class LaserSystem : MonoBehaviour
{
    private LineRenderer lineRenderer;
    public int maxReflections = 5;
    public float maxStepDistance = 200;

    public GameObject winPanel;
    private GameObject lastHitPC;

    private List<Portal> activePortals = new List<Portal>();

    void Start() {
        lineRenderer = GetComponent<LineRenderer>();
    }

    void Update() {
        DrawLaser();
    }

    void DrawLaser() {
        List<Vector3> points = new List<Vector3>();
        points.Add(transform.position);

        Vector2 rayOrigin = transform.position;
        Vector2 rayDirection = transform.right;

        GameObject currentHitPC = null;

        activePortals.Clear();

        for (int i = 0; i < maxReflections; i++) {
            RaycastHit2D hit = Physics2D.Raycast(rayOrigin, rayDirection, maxStepDistance);

            if (hit.collider != null) {
                points.Add(hit.point);
                if (hit.collider.CompareTag("PC")) {
                    currentHitPC = hit.collider.gameObject;
                    Transform glow = currentHitPC.transform.Find("Glow");
                    if (glow != null) {
                        glow.gameObject.SetActive(true);
                        if (winPanel != null && !winPanel.activeSelf) {
                            if (!IsInvoking("ShowWinMenu")) {
```

```
                        Invoke("ShowWinMenu", 1.5f);
                    }
                }
            }
            break;
        }

        if (hit.collider.CompareTag("Mirror")) {
            rayDirection = Vector2.Reflect(rayDirection, hit.normal);
            rayOrigin = hit.point + rayDirection * 0.01f;
        }

        else if (hit.collider.GetComponent<Portal>() != null) {
            Portal portal = hit.collider.GetComponent<Portal>();
            portal.ActivateSecondaryBeam();
            activePortals.Add(portal);
            break;
        }
        else {
            break;
        }
    } else {
        points.Add(rayOrigin + rayDirection * maxStepDistance);
        break;
    }
}

foreach (Portal p in Object.FindObjectsByType<Portal>(FindObjectsSortMode.None)) {
    if (!activePortals.Contains(p)) {
        p.DeactivateBeam();
    }
}

if (lastHitPC != null && lastHitPC != currentHitPC) {
    Transform oldGlow = lastHitPC.transform.Find("Glow");
    if (oldGlow != null) oldGlow.gameObject.SetActive(false);
    CancelInvoke("ShowWinMenu");
}

lastHitPC = currentHitPC;
lineRenderer.positionCount = points.Count;
lineRenderer.SetPositions(points.ToArray());
}

void ShowWinMenu() {
    if (winPanel != null) winPanel.SetActive(true);
}
}
```

```
//LEVEL MANAGER//

using UnityEngine;
```

```
using TMPro;

public class LevelManager : MonoBehaviour
{
    public GameObject routerPrefab;

    [Header("Налаштування рівня")]
    public int mirrorsLimit = 3;
    public TextMeshProUGUI limitDisplay;

    void Start()
    {
        UpdateUI();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (mirrorsLimit > 0 && !IsPointerOverUIObject())
            {
                SpawnRouter();
            }
        }
    }

    void SpawnRouter()
    {
        Vector3 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        mousePos.z = 0;
        Instantiate(routerPrefab, mousePos, Quaternion.identity);
        mirrorsLimit--;
        UpdateUI();
    }

    void UpdateUI()
    {
        if (limitDisplay != null)
        {
            limitDisplay.text = "Mirrors left: " + mirrorsLimit;
        }
    }

    bool IsPointerOverUIObject()
    {
        Vector2 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        RaycastHit2D hit = Physics2D.Raycast(mousePosition, Vector2.zero);
        return hit.collider != null;
    }
}
```

```
//MAIN MENU//
```

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{

    public void OpenGameSelect()
    {

        SceneManager.LoadScene("GameSelect");
    }

    public void StartGlassGame()
    {

        SceneManager.LoadScene("Level 1");
    }

    public void StartWiresGame()
    {

        SceneManager.LoadScene("WireLevel 1");
    }

    public void BackToMainMenu()
    {
        SceneManager.LoadScene(0);
    }

    public void ExitGame()
    {
        Debug.Log("Гра закрилася");
        Application.Quit();
    }

    public void LoadNextLevel()
    {
        int nextSceneIndex = SceneManager.GetActiveScene().buildIndex + 1;
        if (nextSceneIndex < SceneManager.sceneCountInBuildSettings)
        {
            SceneManager.LoadScene(nextSceneIndex);
        }
        else
        {
            SceneManager.LoadScene("GameSelect");
        }
    }
}
```

```
//MENU CONTROLLER//
```

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameUIController : MonoBehaviour
{
    [Header("UI Panels")]
    public GameObject confirmationPanel;

    public void OpenConfirmation()
    {
        confirmationPanel.SetActive(true);
        Time.timeScale = 0f; // Пауза
    }

    public void CloseConfirmation()
    {
        confirmationPanel.SetActive(false);
        Time.timeScale = 1f;
    }

    public void BackToMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }

    public void RestartLevel()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    public void LoadSpecificLevel(string sceneName)
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(sceneName);
    }
}
```

```
//ROTATE PIECE//

using UnityEngine;

public class RotatePiece : MonoBehaviour
{
    private bool isDragging = false;

    void OnMouseDown()
    {
        isDragging = true;
    }
```

```
    void OnMouseUp()
    {
        isDragging = false;
    }

    void Update()
    {
        if (isDragging)
        {

            Vector3 mouseWorldPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0;
            Vector3 direction = mouseWorldPos - transform.position;
            float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
            transform.rotation = Quaternion.Euler(0, 0, angle);
        }
    }
}
```

```
//WIRE//

using UnityEngine;

public class Wire : MonoBehaviour
{
    public string colorName;
    private LineRenderer line;
    private bool isConnected = false;
    private Vector3 startPoint;

    void Start() {
        line = GetComponent<LineRenderer>();
        startPoint = transform.position;

        if (line != null) {
            line.SetPosition(0, startPoint);
            line.SetPosition(1, startPoint);
        }
    }

    void OnMouseDrag() {

        if (isConnected || line == null) return;

        Vector3 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        mousePos.z = 0;
        line.SetPosition(1, mousePos);
    }

    void OnMouseUp() {
        if (isConnected || line == null) return;
```

```
        RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.mousePosition), Vector2.zero);

        if (hit.collider != null) {
            Wire targetWire = hit.collider.gameObject.GetComponent<Wire>();

            if (hit.collider.gameObject != this.gameObject &&
                targetWire != null &&
                targetWire.colorName == this.colorName) {

                line.SetPosition(1, hit.collider.transform.position);
            isConnected = true;
            targetWire.SetAsConnected(transform.position);
            Debug.Log(colorName + " Connected!");
            GameObject.FindObjectOfType<WireGameManager>()?.CheckWin();
                } else {
                    ResetLine();
                }
        } else {
            ResetLine();
        }
    }

    private void ResetLine() {
        if (line != null) {
            line.SetPosition(1, startPoint);
        }
    }

    public void SetAsConnected(Vector3 targetPos) {
        isConnected = true;

        if (line != null) {
            line.SetPosition(1, targetPos);
        }
    }
}
```

```
//WIRE GAME MANAGER//

using UnityEngine;

public class WireGameManager : MonoBehaviour
{
    public int totalWires = 4;
    private int connectedCount = 0;
    public GameObject winPanel;

    public void CheckWin() {
        connectedCount++;
        if (connectedCount >= totalWires) {
            if (winPanel != null) winPanel.SetActive(true);
```

```
        }
    }
}
```