Вінницький національний технічний університет
(повне найменування вищого навчального заклалу)

Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

«Комп'ютерна гра Tech Campus з гейміфікацією навчального процесу»

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Гончарук А.В
(прізвище та ініціали)

Керівник: асистент кафедри КН
Малініч І. П.
(прізвище та ініціали)

«04» червня 2025 р.

Рецензент: к.т.н., проф., доц каф АІІТ
Паламарчук Є.А.
(прізвище та ініціали)

«05» червня 2025 р.

Допущено до захисту
Зав. кафедри
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра Комп'ютерних наук

Рівень вищої освіти – перший (бакалаврський)

Галузь знань – 12 «Інформаційні технології»

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

проф., д.т.н. Яровий А. А.

«05» травня 2025р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гончаруку Андрію Віталійовичу

1. Тема роботи: Комп'ютерна гра Tech Campus з гейміфікацією навчального процесу

керівник роботи: Малініч І.П. асистент каф. КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу "20" березня 2025 року №80

2. Термін подання студентом роботи: 30 травня 2025р

3. Вихідні дані до роботи: Кількість гравців: 1; Кількість складностей гри: 3; час відгуку не повинен перевищувати 500 мілісекунд;

4. Зміст текстової частини (перелік питань, які потрібно розкрити):Дослідження предметної області. Визначення інструментів програмної реалізації. Розробка алгоритму для гри Tech Campus. Розробка графічної частини. Аналіз результатів тестування. Розробка інструкції користувача.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): UML-схеми алгоритмів, графічний інтерфейс програми, результати роботи програми.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	Малініч І.П., асистент каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	05.05.25-07.05.25	
2	Аналіз існуючих рішень	08.05.25–11.05.25	
3	Розробка алгоритму для гри Tech Campus	11.05.25-17.05.25	
4	Реалізація гри Tech Campus	17.05.25-27.05.25	
5	Тестування гри Tech Campus	27.05.25–01.06.25	
6	Оформлення пояснювальної записки	02.06.25 -04.06.25	

Студент Гончарук Гончарук А.В
(підпис) *(прізвище та ініціали)*

Керівник роботи Малініч І.П.
(підпис) *(прізвище та ініціали)*

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 90 сторінок формату А4, містить 21 рисунок, список використаних джерел налічує 22 найменування.

У цій бакалаврській роботі розроблено алгоритми та програмну архітектуру навчальної гри “Tech Campus” з елементами гейміфікації для засвоєння основ IP-мереж. Вдосконалено графічний інтерфейс, реалізовано різні рівні складності, а також досліджено методи підвищення ефективності навчального процесу за допомогою інтерактивних ігрових механік. Для реалізації гри використано сучасний ігровий рушій Unity та мову програмування C#. Результатом дослідження є програмний додаток “Tech Campus”, який може використовуватись для навчання та самоперевірки навичок у галузі мережевого адміністрування.

Ключові слова: Tech Campus, гейміфікація, IP-мережі, Unity, навчальна гра.

ABSTRACT

The bachelor's thesis consists of 90 A4 pages, contains 21 figures, and the list of references includes 22 sources.

In this bachelor's thesis, algorithms and software architecture for the educational game "Tech Campus" with gamification elements for mastering the basics of IP networking have been developed. The graphical interface has been improved, multiple levels of difficulty have been implemented, and methods for increasing the effectiveness of the learning process through interactive game mechanics have been explored. The game was developed using the modern Unity game engine and the C# programming language. The result of the research is the "Tech Campus" application, which can be used for training and self-assessment of network administration skills.

Keywords: Tech Campus, gamification, IP networking, Unity, educational game.

ЗМІСТ

ВСТУП

Актуальність досліджень. На сьогоднішній день, існує багато методологій та практик для навчання нових фахівців їх спеціалізації, багато традиційних із них, є досить нудними із-за чого нові фахівці втрачають мотивацію, і прагнення здобути бажану спеціальність. Багато навчальних закладів почали працювати над підвищенням рівня мотивації їх студентів різними методами – надання стипендій, грантів та працевлаштування активно використовується ще з 20 століття, проте у 21 столітті – активно використовуються різного виду спрощення та персоналізація матеріалу що подається.

Одним із найефективніших методів – є застосування гейміфікації. Використання гейміфікованих підходів у навчанні дозволяє перетворити нудне традиційне навчання на цікавий інтерактивний досвід.

Одним із найважчих для засвоєння розділ мережевих технологій, де потрібно опанувати поняття IP підключень.

Для гейміфікування теми IP підключення, буде розроблено гру «Tech Campus» - головоломку на основі практичних та теоретичних навичок IP підключень.

Метою дослідження є покращення ефективності навчального процесу IP мереж.

Задачі дослідження:

1) Дослідити теоретичні основи гейміфікації та визначити основні мотиваційні елементи.

2) Провести аналіз існуючих рішень гейміфікованих ігровиз застосунків.

3) Спроектувати модель гри «Tech Campus».

4) Розробити алгоритм динамічних ігрових сесій та рівнів складності.

5) Здійснити програмну реалізацію у середовищі рушія Unity.

6) Протестувати програмний модуль гри та порівняти результати зміни навичок.

7) Розробити інструкцію користувача програмного модулю комп’ютерної гри.

Об’єктом дослідження є процес гейміфікації навчального процесу IP підключень.

Предметом дослідження є гейміфікаційні механіки та алгоритмічні рішення для підвищення ефективності навчання IP підключень.

Практична цінність. Розробка гри «Tech Campus» з елементами гейміфікації забезпечує краще засвоєння навчального матеріалу, зменшує навантаження на студента, і має можливість інтеграції у модуль навчального процесу як інструмент для викладачів, який дозволить відстежувати якість опанування навичок, та рівень залучення студентів до опанування навичок.

Публікації. За результатами бакалаврської дипломної роботи опубліковані тези доповіді на конференції "LIV Всеукраїнська науково-технічна конференція підрозділів ВНТУ" (м. Вінниця, Україна) [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІГОР З ЕЛЕМЕНТАМИ ГЕЙМІФІКАЦІЇ

1.1 Аналіз сучасного підходу до розробки ігор із навчальним ефектом

Використання ігор у навчальному процесі має глибокі історичні корені. Ще з давніх часів вважалося, що навчання через гру є одним із найефективніших способів засвоєння знань, оскільки воно стимулює мислення, емоційне залучення та підвищує мотивацію до навчання. Ще у школах давньої Греції існувало твердження що для розвити грамотну та сильну особистість при навчанні без радості – неможливо.

Термін «гейміфікація» виник на початку 21ст. і означає використання ігрових механік та дизайну для залучення гравців у неігрові процеси. Гейміфікація в освіті передбачає впровадження таких елементів як система балів, рівнів, нагород, рівнів, сюжетів, рейтингів, викликів для стимулювання мотивації гравця, до навчання [2].

Із розвитком педагогічних наук в 19 та 20ст. гейміфікація навчального процесу масово використовувалась у початковій школі, для розвитку уваги, інтересу, пам’яті, логіки та творчості дітей.

Справжній прорив гейміфікація зробила із розвитком комп’ютерних технологій. Комп’ютеризація відкрила нові горизонти для гейміфікованих методик, починаючи від простих програм-тренажерів закінчуючи точними симуляторами, які моделюють реальні системи, ситуації та процеси.

Яскравим прикладом ранніх гейміфікованих застосунків є «Козаки», що навчала правильному плануванню ресурсів та прийняттю рішень.

У традиційному навчанні зазвичай спеціалісти обмежені вивченням віртуальних схем у симуляторах на зразок Cisco PT чи GNS3. Однак такі симулятори зазвичай самі по собі є важкими для опанування, ще не доходячи до самої суті побудови віртуальних мереж, і не завжди мають актуальний або повний інструментарій відповідно до ринку технологічного забезпечення. В

результаті надмірної складності та обмежень частина студентів не має інтересу заглиблюватись у матеріал, та обмежується не побудовою правильної мережі, а базовим виконанням умов поставлених у завданні викладача.

З приходом 21ст. з'явився новий напрям ігор який назвали «Сер'йозні ігри», сер'озні ігри поєднували у собі розважальний геймплей із навчальним контентом. Сер'озні ігри заклали грунт для цілої низки навчальних методик що базуються на принципах гейміфікації (рис. 1.1).

Рисунок 1.1 – елемент геймплею з гри «Козаки»

На сьогоднішній день ігри використовують не лише для формування знань, але й для розвитку професійних компетентностей, особливо актуальним стало створення ігор з нахилос на засвоєння технічних компетентностей, де необхідне не лише теоретичне підгрунтя, але й вміння застосовувати ці знання на практиці [3].

На основі цих даних розроблено гру «Tech Campus», яка на основі гейміфікованих методик дозволяє розвити навички з IP-підключень, та портового менеджменту – базових умінь для спеціалістів у сфері мережевих технологій.

Гра «Tech Campus» безпосередньо використовує ці переваги, враховано мотивацію, дизайн та безпосередньо навчальний процес, завдяки обмеженню часу та системі очок грати у гру стає просто весело.

Особливу увагу було приділено балансу складності та кінцевими цілями, з однієї сторони завдання достатньо складні щоб засвоїти матеріал, а з іншої достатньо реалістичні щоб виконати цілі гри [4].

Ефективність ігор як інструменту для навчання пояснюється їх психологічним впливом на механізми мотивації. Правильне розуміння цих механізмів дозволяє правильно будувати навчальні ігри, щоб вони не лише були джерелом нових знань але й джерелом мотивації і задоволення.

За основу для психологічного підгрунтя гри «Tech Campus» було взято теорію самодетермінації, розроблену Е.Десі та Р.Раяном у ранній період заснування гейміфікації як окремого напряму. Відповідно цієї теорії необхідно врахувати 3 основні пункти – автономія, компетентність та соціальну залученість [5].

У грі «Tech Campus» гравець може самостійно обрати з якого підключення йому розпочати проходження рівня, що дає йому автономію.

Складність завдання у грі «Tech Campus» кидає виклик гравцю, відповідно після успішного проходження рівня гравець буде відчувати компетенцію.

Очки за проходження рівня зберігаються у таблиці результатів, відповідно при проходжені гри декількома гравцями – вони матимуть конкуренцію міє собою, що задовольнить соціальну залученість гравців.

Ігри володіють унікальною здатністю утримувати увагу гравців протягом тривалого часу. Активне залучення гравця до прийняття рішень, динамічні умови ігрового середовища, необхідність уваги до змін у ігрових викликах, усе це тримає гравця у стані активного когнітивного напруження, що в точності повторює умови необхідні для традиційного навчання.

Проте у традиційних умовах навчання студенти чи учні часто сприймають інформацію пасивно, що значно знижує ефективність сприйняття отриманої інформації. На відміну від цього формат гри як у грі «Tech Campus» вимагає від гравця постійної уваги та залученості [6].

Ігрові механізми викликають у гравця почуття досягнення та успіху, підвищують самооцінку та схиляють до прийняття викликів від більш складних рівнів, що стимулює більш глибоке оволодіння матеріалом [7].

Технічні дисципліни такі як інформатика, комп'ютерні мережі, телекомунікація вимагають від студентів не лише засвоєння знань та навичок, але й їх активне застосування у практичних сценаріях.

Особливо складною є сфера мережевих технологій, з усіма типами зв'язків , протоколами передач та фізичною складовою сфери.

Використання гейміфікованих ігор дозволить будувати освітні середовища де студент не просто буде запам'ятовувати окремі моменти IP-підключень, а буде застосовувати їх у реальному часі для виконання чітких завдань (рис. 1.2).

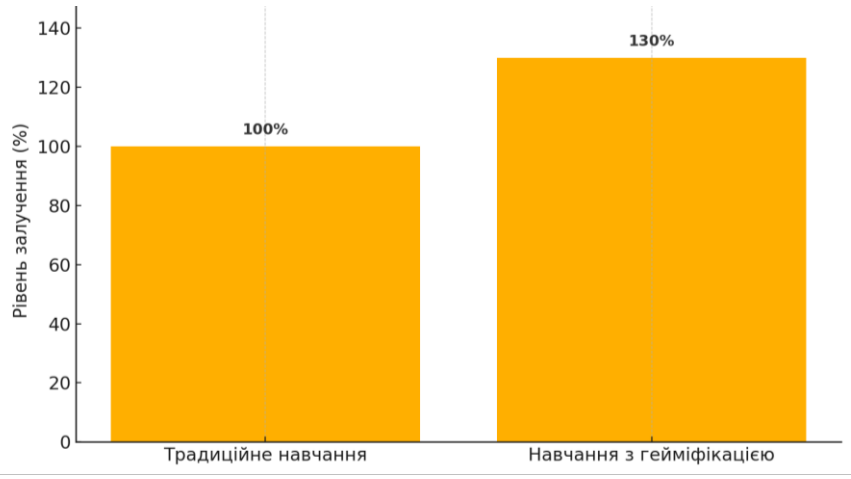


Рисунок 1.2 – рівень залучення студентів у порівнянні традиційного навчання із гейміфікованими застосунками

Окрім наукової складової ефективність гейміфікованої гри значною мірою визначається саме її ігровою складовою. На основі сучасних теорій геймдизайну необхідно виділити базові вимоги з технічної сторони такі як інтуїтивність, прогресія складності, зрозумілий зворотній зв'язок, можливість відстеження прогресу [8].

Для інтуїтивності необхідно мінімалізувати кількість активних елементів у навігації грою, що підвищить ефективність орієнтування у ігровому просторі, та мінімалізує час на освоєння управління

Для прогресії складності необхідна реалізація декількох рівнів складності від легких до важких які заставлять напружитись вже досвідчених експертів у сфері IP-підключеннь.

Зрозумілий зворотній зв’язок вимагає реалізації сповіщень про успішне або неуспішне проходження ігрового рівня, методом нарахуванням або відніманням очок прогресу, або інтерактивних сповіщень.

Можливість відстеження прогресу вимагає реалізації системи рейтингів, або значків-досягнень за успішно виконані рівні.

1.2 **Аналіз існуючих аналогів**

На сьогодні існує широкий вибір програмних засобів які дозволяють вивчати комп’ютерні мережі як і в межі традиційної освіти так і в гейміфікованому середовищі.

До першої категорії можна віднести популярні професійні інструменти Cisco Packet Tracer, GNS3, BOSON NetSim та інші. Головна їх особливість – це висока технічна деталізація, яка дозволяє будувати максимально наближені до реальних мережеві структури та архітектури. Такі програмні засоби зазвичай орієнтовані на студентів технічних спеціальностей, загальних або приватних курсів сертифікації мережевих спеціалістів. Такі програмні засоби дають змогу віртуально створити мережі, та зв’язки, окремо налаштовувати цілу низку загальноприйнятих протоколів маршрутизації VLAN, NAT, DHCP, та інші елементи мережевих структур.

Але на противагу можливостям цих програмних засобів стає їх складність. Високий поріг входу для новачків, складність інтерфейсу, ціна та програмні вимоги є важкою перепоною для користувачів які ще не мали досвіду роботи із мережевими зв’язками. Високі вимоги цих програмних засобі дуже часто знижують інтерес користувачів до навчання та роботи.

До другої категорії можна віднести гейміфіковані освітні платформи. За останніх десять років було розроблено низку справді корисних, цікавих, відкритих та інформативних платформ. Одними із найкращих представників є TryHackMe, Hack The Box, CyberStart. Орієнтація цих програмних засобів полягає у вивченню комп'ютерних мереж, кібербезпеки, етичного хакінгу та споріднених дисциплін. Відмінність від традиційного вивчення матеріалу у цих програмних застосунках полягає у вивченні матеріалу з використанням гейміфікованих елементів. Такі платформи дозволяють проводити віртуальні кібератаки, тренуватись у хакінгу та виявленню вразливостей, аналізувати пакети та віддалено підключатись до пристроїв але виконувати всі ці дії в ігровому варіанті.

Хоча гейміфіковане середовище має високу ефективність у залученні користувачів, воно також обмежене. Більшість із перерахованих гейміфікованих засобів на превелику жаль, має нахил саме на кібербезпеку та додатково вимагає від користувача певних знань та навичок.

Для того щоб відокремити плюси та мінуси розглянемо ці програмні засоби окремо.

Cisco Packet Tracer – є одним із найвідоміших програмних засобів для вивчення мережевих технологій. Цей програмний засіб створений компанією Cisco Systems спеціально для підтримки навчальної програми Cisco Network Academy, і на сьогоднішній день є одним із базових інструментів для вивчення мережевих технологій у навчальних закладах.

Основне призначення Cisco Packet Tracer полягає у проектуванні та візуалізації мереж та їх середовищ без потреби у фізичному технічному забезпеченні. Завдяки високому технологічному рівню Cisco Packet Tracer користувачі можуть не лише будувати віртуальні мережі а також безпосередньо налаштовувати їх за допомогою терміналів віртуальних складників. Реалістичність є основною перевагою Cisco Packet Tracer. Користувачі мають можливість емулювати пристрої Cisco, які матимуть поведінку їх фізичних аналогів, що дозволяє відчути досвід наближений до

роботи із справжнім фізичним обладненням і дає можливість здобути навички відповідні до практичних. Cisco Packet Tracer надає модливість емулювати розлогий вибір мереж VLAN, DHCP, NAT, OSPF, EIGRP тощо.

Проте в потужного функціоналу Cisco Packet Tracer є великий перелік недоліків, що у значній мірі обмежують його ефективність відносно нових користувачів. Передусім інтерфейс Cisco Packet Tracer є надзвичайно складним – велика кількість меню, кнопок, команд швидко приводить нового користувача у стан жаху. Навчання у Cisco Packet Tracer зазвичай обмежується однотипними лабараторними завданнями і зарання відомими сценаріями, що зводить елемент креативності на нівець. Правильне виконання завдання залежить не від пошуку варіантів рішення самої задачі, а від статичного слідування алгоритму виконання завдання. Також великим недоліком є те що Cisco Packet Tracer можливо використовувати лише у локальних обставинах, він вимагає стабільного встановлення , постійних оновлень і не завжди може працювати стабільно на більш старих пристроях.

GNS3 також є ондним із найбільших програмних засобів для моделювання мережевих структур, який навідміну від Cisco Packet Tracer є вже справжнім професійним застосунком в IT-сфері. Також навідміну від Cisco Packet Tracer - GNS3 працює з реальними образами операційних систем, що саме дозволяє запускати зпроектовані мережі у віртуальному просторі. GNS3 дозволяє створювати комплексні технології з великою кількістю вузлів, включаючи маршрутизатори, фаєрволи, комутатори та інші мережеві пристрої. Так само як і Cisco Packet Tracer, має можливість віртуального налаштування пристроїв через командний рядок, але також і дозволяє незалежно на кожний пристрій інсталювати програмне забезпечення, серівіси та робити аналіз зв’язку. Такі можливості дають змогу не лише виконувати лабараторні завдання, але є й хорошими для проведення тренінгів та моделювання складних сценаріїв з високим рівнем критичності. Гнучкість є основною перевагою GNS3, користувач може створити та конфігурувати мережу максимально наближену до реального середовища, що робить

інструмент ідеальним для побудови тестових мереж перед реальним фізичним впровадженням. Окрім цих факторів GNS3 дозволяє інтеграцію з іншими віртуальними платформами, створювати гібридні мережі, та має хмарну інтеграцію з API.

Однак, попри всі переваги GNS3 – це все ще робочий інструмент, у ньому немає підказок, навчальних завдань, мети або рейтингів. Як і всі інші програмні засоби GNS3 вимагає уже наявних знань та навичок, вимагає знання інтерфейсу програмного засобу, на доволі просунутому рівні, програма має дефекти нестабільності з елементами віртуальних конфігурацій та окремих операційних систем, програмний засіб GNS3 вимагає знання і практичні навички у роботах з підвидами операційних систем для віртуальних машин. Для новачка у мережевих технологіях вибір цієї програми для опанування стане скоріше кінцем чим початком кар'єри.

TryHackMe – це один із найбільших гейміфікованих програмних застосунків, орієнтований на розвиток навичок та вмінь у сфері кібербезпеки, адміністування систем, основ комп'ютерних мереж та білого хакінгу. Після запуску у 2018 році дуже швидко завоювала прихильність користувачів через свою чітку гейміфіковану систему. Програмний засіб надає можливість розвити навички у окремих сферах як і новачкам, так і вже досвідченим працівникам інтелектуальних технологій.

Структурована подача матеріалу є однією із ключових переваг TryHackMe. Користувачі програмного засобу TryHackMe можуть самостійно обрати гілки навчання та складність, тобто для початку навчання із застосунком TryHackMe необхідно мати лише бажання та доступ до мережі інтернет, тобто не потрібно встановлювати додаткове програмне забезпечення, налаштовувати операційні системи, опановувати інтерфейс, усі можуть розпочати із командного рядка. Так як програмний засіб TryHackMe розташований поністю у хмарі усе налаштовується автоматично, користувач отримує готове віртуальне середовище із чіткою метою для проходження

завдання. Гейміфікацію у TryHackMe можна чудово спостерігати у ігровому вигляді завдань, системі очок, бейджів, рівнів і змагального рейтингу.

TryHackMe – чудовий програмний засіб, але як і все, вона має свої обмеження. Перед усім TryHackMe – зосереджена на аспекті кібербезпеки, і мережеві технології розглянуті там у найменших деталях, тобто програмний застосунок не вчить користувача як побудувати мережу, а вчить як нею маніпулювати. Також великим негативним аспектом є те, що TryHackMe не підтримує локалізацію Українською мовою, а також використовує на певних рівнях технічний жаргон, що також є досить відчутною завадою до деяких користувачів, які тільки шукають свій шлях у мережевих технологіях.

1.3 **Постановка задачі дослідження**

Завданням цієї дипломної роботи є створення навчальної комп'ютерної гри "Tech Campus" з елементами гейміфікації, орієнтованої на покращення розуміння основ IP-мережевих з'єднань через залучення гравця до інтерактивних і візуальних ігрових ситуацій.

Метою дослідження є збільшення результативності навчального процесу з тематики IP-адресації та базових принципів створення мереж, використовуючи ігрові методи, які сприяють кращому залученню, мотивації та практичному закріпленню знань. В рамках цього дослідження головна увага приділяється розробці динамічної освітньої гри, що дозволяє студентам розвивати навички підключення пристроїв у віртуальному мережевому середовищі.

Об'єктом дослідження є процес гейміфікації навчального процесу IP-з'єднань у віртуальному просторі.

Предметом дослідження є гейміфікаційні механізми та алгоритмічні рішення, які забезпечують реалізацію навчального функціоналу на базі ігрової платформи з ефективним зворотним зв'язком та поступовим збільшенням складності.

У процесі реалізації дослідження буде створено прототип гри, що використовує гейміфіковану модель, де гравець повинен підключати віртуальні пристрої (ПК) до маршрутизатора відповідно до IP-адрес.

Гра буде складатися з послідовних рівнів з поступовим зростанням складності, обмеженнями за часом, системою балів, досягнень та візуального зворотного зв'язку. Для забезпечення реалістичності моделі буде використано матричну модель рівня, а також концепцію проміжних вузлів (хабів), що дає змогу моделювати сценарії з підмережами та портовими розгалуженнями.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- Виконати аналіз теоретичних підвалин гейміфікації, окреслити її освітній потенціал та основні складові, що впливають на студентську мотивацію.
- Вивчити наявні приклади гейміфікованих навчальних платформ та ігор, встановити їхні сильні сторони та недоліки в процесі вивчення мережевих технологій.
- Сконструювати архітектуру гри «Tech Campus», визначити її ключові компоненти, правила функціонування об'єктів, модель інтерфейсу та методику побудови рівнів.
- Сформувати алгоритми ігрових сесій, правила перевірки IP-підключень, реакції на помилки, систему прогресу, нарахування балів та інтерфейс управління (меню, перезапуск, вибір складності).
- Втілити ігровий додаток у середовищі рушія Unity, зважаючи на оптимізацію графічного зображення та забезпечення інтерактивної взаємодії.
- Здійснити тестування гри на предмет відповідності реалізації логіки, інтерфейсу, ігрової механіки та рівнів складності.
- Створити інструкцію користувача для гри «Tech Campus» з описом функціоналу та цілей.

Програма має надавати інтуїтивно зрозумілий інтерфейс, дозволяючи гравцеві виконувати операції простими маніпуляціями (перетягування, кліки) та забезпечувати негайний зворотний зв'язок. Ключовими критеріями успішності реалізації будуть рівень засвоєння знань, залученість гравця, гнучкість адаптації складності рівнів та зручність користування як для студента, так і для викладача.

У цьому розділі було окреслено коло питань, які потребують дослідження, сформульовано головну ціль роботи, визначено те, що саме буде вивчатися та на чому зосереджуватиметься увага. Додатково, представлено основні задачі, що будуть вирішені під час написання дипломної роботи.

Дослідження базується на важливості теми гейміфікації в навчальному процесі, а також на необхідності покращення якості засвоєння базових знань про IP-мережі, використовуючи ігрові елементи.

У підсумку, створення гри Tech Campus має продемонструвати можливість використання гейміфікації як ефективного методу для навчання базовим технічним навичкам у галузі мережевих технологій.

1.4 Висновок до розділу 1

У першому розділі було здійснено аналіз предметної області гейміфікації освітнього процесу та навчальних ігор у сфері комп’ютерних мереж. Встановлено, що використання гейміфікаційних підходів дозволяє підвищити мотивацію студентів, забезпечити їхнє активне залучення до навчання та сприяє кращому засвоєнню складних технічних понять.

Проаналізовано існуючі інструменти та програмні продукти для навчання комп’ютерним мережам, такі як Cisco Packet Tracer, GNS3, TryHackMe та Hack The Box. Визначено їх переваги та недоліки, серед яких основними є складність інтерфейсу для новачків, відсутність інтегрованих гейміфікаційних механік та обмеження у практичній інтерактивності для початкового рівня користувачів.

На основі проведеного огляду сформульовано актуальність розробки нової навчальної гри з елементами гейміфікації, що дозволить підвищити ефективність навчання базових принципів IP-мереж. Обґрунтовано доцільність створення програмного продукту, який би поєднував навчальні цілі із захопливим ігровим процесом.

Результати цього розділу стали підґрунтям для подальшого проектування та реалізації гри “Tech Campus”, що має на меті забезпечити сучасний, інтуїтивний і мотивуючий інструмент для формування практичних навичок у сфері мережевого адміністрування.

2 ПРОЕКТУВАННЯ ГРИ «TECH CAMPUS»

2.1 Опис структурно-функціональної організації

Розробка навчального програмного забезпечення, або ігор вимагає не лише технічного втілення, а й грамотного логічного підґрунтя для взаємодії користувача з системою. Для гри "Tech Campus", задум полягає у створенні ігрового середовища, що симулює реальні ситуації, де гравець поступово освоює основи IP-мереж, процес логічного з'єднання пристроїв до маршрутизаторів, згідно з певними вимогами щодо адресації. Виконуючи ігрові завдання, гравець закріплює та вдосконалює ключові поняття мережевого моделювання.

Ігровий рівень у Tech Campus представлений полем, структурованим у вигляді матриці [9]. У цій сітці розташовані віртуальні об'єкти - персональні комп'ютери з наперед визначеними IP-адресами, маршрутизатор із портами, до яких необхідно підключити пристрої, а також на складніших рівнях існують додаткові проміжні елементи а саме хаби або комутатори (рис. 2.1).

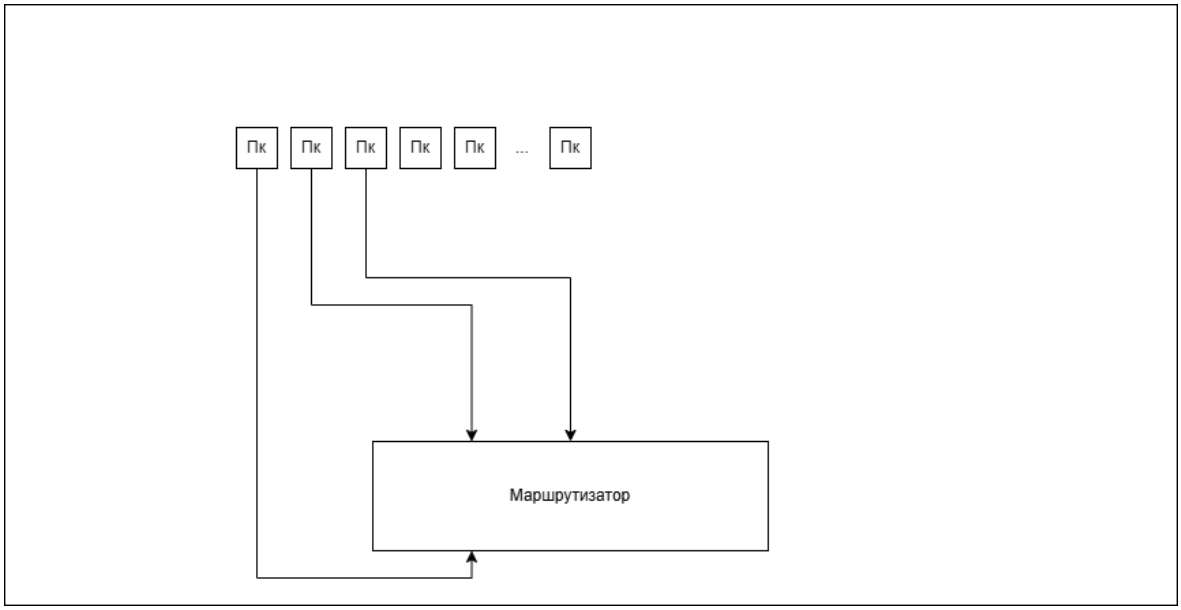


Рисунок 2.1 – Макет рівня гри комп'ютерної гри «Tech Campus»

Ці об'єкти не є статичними - користувач самостійно аналізує, до яких саме портів підключати ПК, орієнтуючись лише на наявні умови та принципи

адресації. Кожен рівень виступає як мікросимулятор реальної локальної мережі, в якій необхідно досягти правильного мережевого з’єднання [10].

Кожен комп’ютер на ігровому полі має чітко визначену IP-адресу. Візуально адреса відображається безпосередньо над об'єктом або в його контурі, що дає змогу користувачеві миттєво її ідентифікувати. Кількість комп'ютерів варіюється в залежності від складності рівня. На початкових етапах це 3–4 пристрої, що мають бути підключені до одного маршрутизатора. На подальших рівнях кількість ПК збільшується, зростає і складність логіки з’єднання, ускладнюються вимоги до коректного підбору IP, додаються випадки адресних конфліктів та умовна фільтрація [11].

Оскільки головним завданням користувача є коректне підключення віртуального пристрою з визначеною IP-адресою до порту роутера з заданими умовами, виникає потреба у точному програмному алгоритмі перевірки відповідності. У спрощеній формі, кожний комп'ютер має фіксовану IP-адресу, яку можна представити як чотири байти (IPv4): A.B.C.D. У той час як порт маршрутизатора може мати одну з декількох умов: або чітку IP-адресу, або діапазон (наприклад, 192.168.0.0 – 192.168.0.255), або умовний фільтр за останнім октетом (*.168.0.10), або позначення підмережі за допомогою CIDR-нотації (192.168.1.0/24).

Для перевірки правильності з'єднання система використовує алгоритм порівняння вхідної IP-адреси з умовами, що зберігаються в структурі кожного порту. У випадку з CIDR-маскою або діапазоном, перевірка зводиться до побітового порівняння мережевої частини IP-адреси (за маскою) або до визначення, чи входить адреса до певного інтервалу. У випадку шаблонного фільтру, «всі адреси, що закінчуються на .10», перевірка відбувається шляхом порівняння останнього байта IP. Такі перевірки виконуються в реальному часі під час спроби створити з'єднання: у випадку успіху - порт активується, у разі невідповідності - з'єднання не фіксується [12].

На більш складних рівнях гри додається додаткове обмеження, яке стосується маршрутів з’єднань. Від гравця вимагається не тільки обрати

правильну логічну пару «ПК - порт», але й прокласти кабель таким чином, щоб його маршрут не перетинався з вже створеними з'єднаннями. Це імітує реальні технічні обмеження фізичного розташування мереж у серверних кімнатах або на друкованих платах, де неприпустимі конфлікти в прокладці ліній.

Для реалізації такої перевірки використовується матрична модель рівня, де кожен піксель, блок або клітинка, яку займає кабель, позначається у внутрішній структурі даних гри як «зайнята» (рис. 2.2).

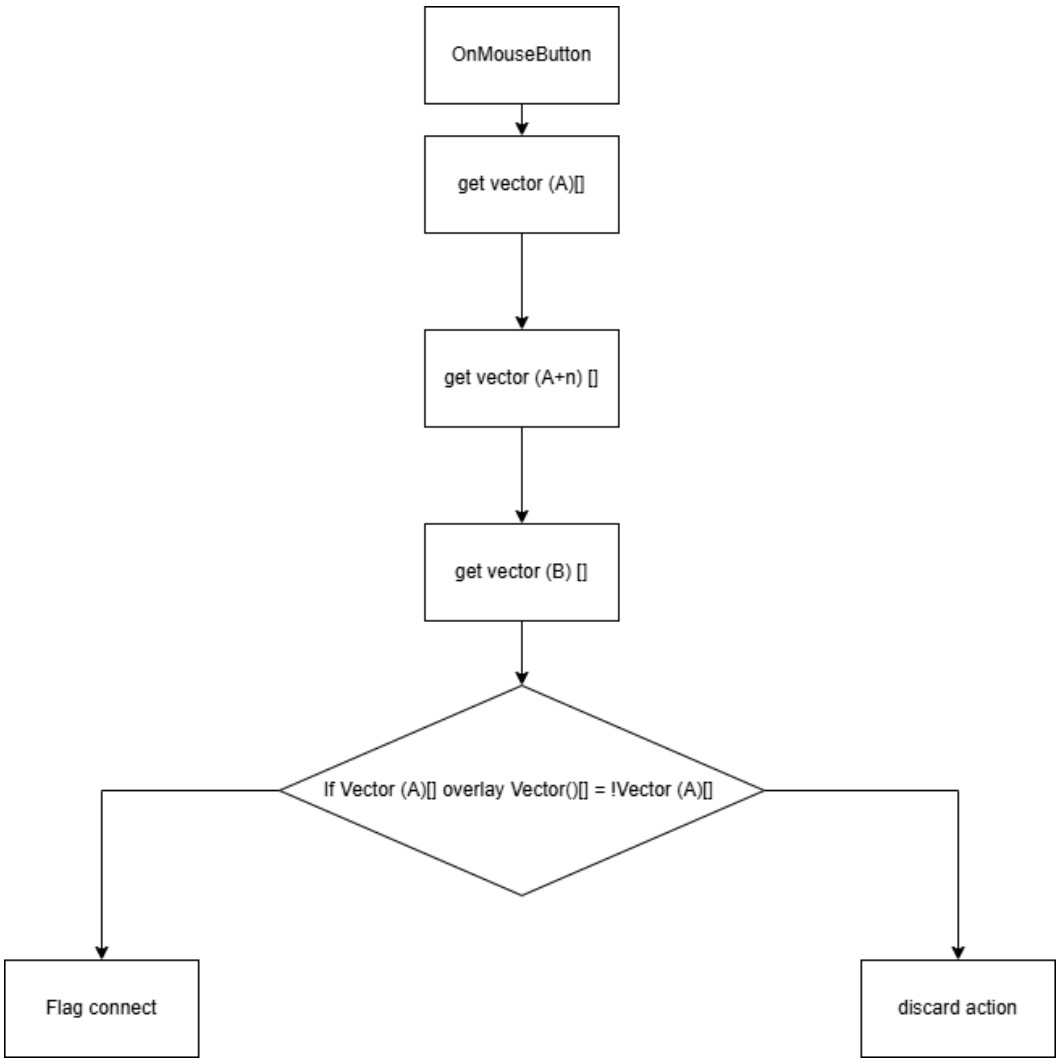


Рисунок 2.2 – Логіка перевірки з’єднань для комп’ютерної гри «Tech Campus»

Під час спроби прокладання нового з'єднання система симулює шлях від ПК до порту за певним алгоритмом маршрутизації, і перевіряє, чи не перетинає новий маршрут вже зайняті клітинки. Якщо перетин виявлено -

гравець або не зможе провести кабель, або отримає візуальне попередження про неприпустимість маршруту [13].

Цей елемент значно ускладнює геймплей, але водночас формує у гравця навички просторового мислення, стратегічного планування та орієнтування у сітковій логіці. Він спонукає не лише розв'язувати завдання на відповідність IP-адрес, але й розробляти оптимальні маршрути, уникаючи конфліктів з іншими підключеннями, що є дуже близьким до реальних задач, які стоять перед мережевими адміністраторами.

Ігрова логіка базується на тому, що гравець спочатку ознайомлюється зі структурою рівня: вивчає розміщення комп'ютерів, маршрутизаторів, умови підключення. Потім, на основі отриманої інформації, гравець планує логічний маршрут для кожного ПК. Йому необхідно визначити, який комп'ютер відповідає якій умові на маршрутизаторі, а також спроектувати траєкторію кабелю, якщо рівень передбачає уникнення перетинів з іншими лініями. На ранніх етапах кабелі можуть перетинатися без обмежень, проте у складних рівнях прокладання має враховувати сіткову модель маршруту та уникати колізій [14].

Процес з'єднання здійснюється через інтуїтивну взаємодію: користувач або переміщує візуальний кабель від комп'ютера до порту маршрутизатора, або обирає послідовність дій через кліки (вибір пристрою , вибір порту). У випадках, коли рівень містить проміжні елементи (наприклад, хаби), гравець може підключити кілька комп'ютерів до одного вузла, а потім з'єднати цей вузол з портом маршрутизатора. Така структура дозволяє моделювати підмережі, а також додає додаткову гнучкість у розв'язанні задачі.

Після налагодження взаємодії між об’єктами, програма здійснює перевірку: зіставляє IP-адресу комп’ютера з умовами, заданими на порті маршрутизатора. Якщо з'єднання правильне, інтерфейс візуально підтверджує успіх (наприклад, підсвічуванням зеленим або іншим індикатором). У випадку помилки з'єднання чи невідповідності IP, з'єднання не зберігається, і гравець може самостійно проаналізувати та скоригувати свій вибір. Система не

містить автоматичних підказок чи штрафів за помилки – це зроблено свідомо, аби стимулювати самостійне мислення та уважність.

На кожному рівні гравець має виконати всі з'єднання – тобто заповнити всі порти маршрутизатора відповідно до встановлених умов. Успішне проходження рівня можливе тільки тоді, коли всі логічні пари «ПК–порт» будуть з'єднані правильно. По завершенні рівня система може показати узагальнені результати: кількість правильних з'єднань, час проходження, кількість помилок (якщо це необхідно для статистики). Гравець має змогу перейти до наступного рівня, перезапустити поточний або змінити складність гри через основне меню.

Меню гри виконує допоміжну роль, забезпечуючи базові функції: перезапуск рівня, зміна рівня складності та вихід з гри. Усі дії реалізовані через стандартні UI-елементи, щоб не відволікати від основної навчальної механіки. Отже, інтерфейс гри навмисно мінімалістичний, аби утримувати фокус користувача на аналітичній частині завдань (рис. 2.3).

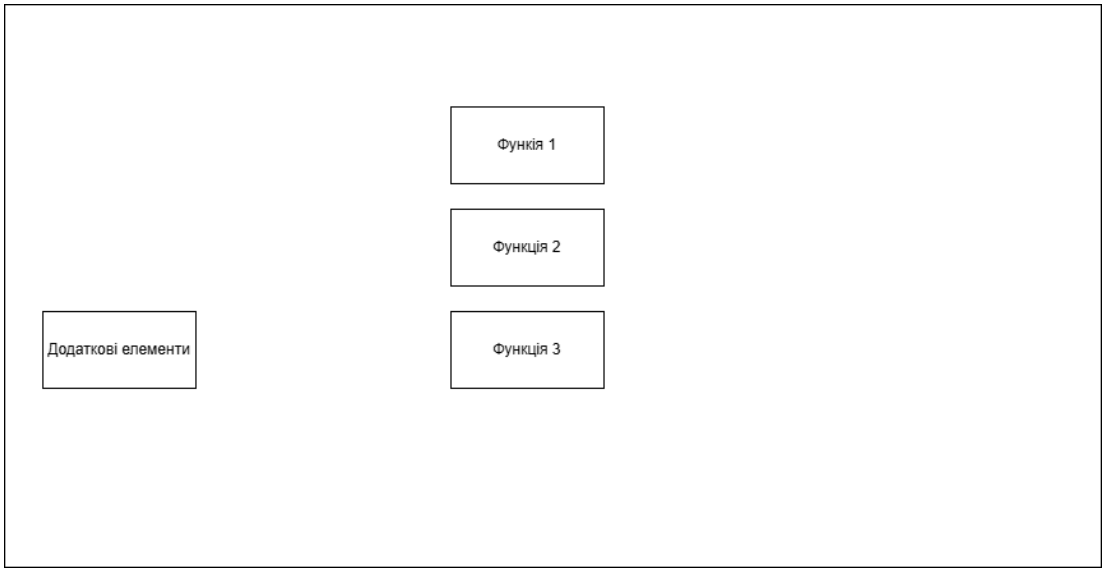


Рисунок 2.3 – Макет меню комп’ютерної гри «Tech Campus»

Особлива увага в процесі розробки гри приділяється варіативності складності. У Tech Campus впроваджено три умовні рівні: «легкий», «середній» та «складний». На легкому рівні гравець стикається з простим

зіставленням між IP-адресами та портами. Середній рівень додає додаткові елементи - хаби, діапазони IP, а також початкові обмеження на перетин ліній. Складний рівень включає вже комплексну логіку: IP з різних підмереж, фейкові адреси, порти з шаблонними умовами та повну заборону на перетин з'єднань. Кожен рівень сконструйовано так, аби формувати у гравця послідовне розуміння, що поступово ускладнюється і вимагає більшої уваги та аналітичних навичок.

Таким чином, логіка гри «Tech Campus» ґрунтується на симуляції реальної задачі з побудови IP-мережі, що передбачає логічне мислення, практичну увагу до адресації та вміння працювати в системі обмежень. Відсутність підказок та автоматичного навчання робить гру інструментом активного мислення, а її структура дозволяє легко масштабувати як контент, так і складність рівнів в рамках освітнього процесу.

Ініціалізація ігрової сцени у грі «Tech Campus» - це ключовий момент, що передує безпосередньому взаємодію користувача з віртуальним середовищем. На цій стадії система створює всі необхідні параметри для успішного проходження рівня, включно з створенням ігрових об'єктів, їх просторовим розміщенням, наданням властивостей (зокрема, IP-адрес) та визначенням правил взаємодії між ними. Ініціалізація проводиться централізовано за допомогою керуючого компонента – класу GameManager, який виступає основою архітектури гри.

Першим кроком ініціалізації є створення віртуального ігрового поля. Поле представляє собою двовимірну матрицю, кожна клітинка якої може бути зайнята об'єктом (наприклад, ПК, портом, хабом або кабелем) або залишатися вільною для прокладення з'єднань. Така структура дає змогу реалізовувати алгоритми перевірки перетинів, оптимізувати обчислення траєкторій та створює умови для масштабування рівня складності без змін загальної логіки гри.

Наступним етапом відбувається генерація та розміщення персональних комп'ютерів. Кожному ПК надається унікальна IP-адреса, що генерується

випадковим чином або відповідно до заданої логіки рівня (наприклад, фіксований діапазон, CIDR-підмережа тощо). IP-адреса відображається в інтерфейсі користувача – над відповідним об'єктом на полі або в боковій панелі. Після створення, кожен ПК зберігається як об'єкт класу PCUnit, який містить необхідні дані (адресу, координати, стан підключення) та методи взаємодії [15].

Далі відбувається створення маршрутизатора з набором портів, кожен з яких має власні правила для прийняття підключень. Кількість портів і типи їх правил (пряма відповідність IP, шаблонна перевірка, діапазон, CIDR) залежать від складності рівня. Кожен порт реалізовано як об'єкт класу RouterPort, в структурі якого закладено логіку перевірки, статус активності та взаємодія з відповідним з'єднанням. Для кожного порту також визначається його розташування на полі, що враховується при побудові кабелів.

У складніших рівнях додаються проміжні вузли – хаби або комутатори, які моделюють підмережі. Хаби дозволяють об'єднати декілька ПК до одного проміжного вузла, який, в свою чергу, підключається до порту маршрутизатора. Вони реалізовані як окремі об'єкти класу Hub, які містять перелік підключених ПК, обмеження на кількість підключень та методи управління відображенням з'єднань.

Після генерації всіх елементів, GameManager здійснює логічне розташування об'єктів на ігровій сітці: комп'ютери, зазвичай, розташовуються в нижній або лівій частині поля, порти маршрутизатора – у протилежній, а хаби – між ними. Це забезпечує логічне візуальне сприйняття мережі, спрощує орієнтацію та дозволяє гравцю сконцентруватися на побудові маршруту підключень.

Завершальним етапом ініціалізації є виведення інтерфейсних елементів, таких як ігрове меню (перезапуск, складність, вихід), індикатори часу та, за потреби, статистичні лічильники підключень. Ці компоненти реалізовані через клас UIController і повністю ізольовані від логіки перевірки з'єднань.

Таким чином, ініціалізація ігрової сцени представляє собою чітко визначений, багаторівневий процес, який слугує "стартовою точкою" для кожного рівня. Від її коректності залежить не тільки стабільність роботи гри, але й можливість правильно сформувати навчальне навантаження та забезпечити поступове ускладнення завдань відповідно до цілей навчального модуля.

Процес починається з ініціалізації, коли система створює об'єкти комп'ютерів, присвоює їм IP-адреси, формує набір портів маршрутизатора з визначеними умовами, а також (за потреби) додає хаби. Після підготовки управління переходить до гравця.

Далі ігрова сесія переходить до етапу взаємодії з користувачем. Гравець аналізує IP-адреси пристроїв, умови підключення, обирає правильні пари та прокладає з'єднання. Кожна спроба з'єднання активує логічний модуль перевірки: IPValidator перевіряє правильність IP-адреси згідно з умовами на порту.

У разі успішного з'єднання система фіксує зв'язок і оновлює статус порту. Якщо виявлено помилку, з'єднання скасовується, і гравець може спробувати знову. На складних рівнях додатково залучається блок перевірки перетинів маршрутів, що блокує прокладання кабелю через вже зайняті координати.

Після підключення всіх портів маршрутизатора відбувається фінальна перевірка, яка послідовно аналізує всі встановлені з'єднання. Якщо всі умови виконані, гра показує підсумкову інформацію (час, кількість спроб, стан порту) та пропонує перейти на наступний рівень, повторити поточний або вийти з гри.

Після запуску гри та ініціалізації ігрової локації, гравець отримує доступ до ігрового поля, де постає перед завданням. Головною відмінністю гри "Tech Campus" є активна пізнавальна взаємодія гравця з елементами змодельованої мережевої інфраструктури. Завдання полягає не лише у фізичному з'єднанні

пристроїв, а й у формуванні цілісного логічного ланцюга ухвалення рішень, ґрунтуючись на розумінні основ мережевої адресації.

На самому початку взаємодії користувач може оглянути всі об'єкти на екрані. Він візуально сприймає розміщення пристроїв, таких як персональні комп'ютери (ПК), лінії з'єднання та порти маршрутизатора. Кожен ПК має унікальну IP-адресу, яка відображається безпосередньо на екрані у вигляді підпису. Одночасно, кожен порт маршрутизатора містить вимогу підключення – або у вигляді точної IP-адреси, або у вигляді шаблону, CIDR-маски чи діапазону. Відтак, гравець одразу має всі вхідні дані для розв'язання задачі.

Далі гравець проводить логічний аналіз: порівнює IP-адреси ПК з умовами портів маршрутизатора. На цьому етапі формується попереднє розуміння, які пристрої повинні бути з'єднані. Це вимагає від користувача вміння розпізнавати структури IP-адрес, аналізувати підмережі, застосовувати умовні оператори (наприклад, "всі адреси з останнім октетом = 10") та визначати належність адрес до конкретних діапазонів або масок.

Після завершення аналітичного етапу користувач переходить до фізичного з'єднання. Залежно від реалізації, підключення може здійснюватися або шляхом перетягування (drag & drop), або через покрокове натискання (спершу вибір ПК, потім вибір порту або хабу). Якщо рівень містить проміжні вузли – хаби, то процес складається з двох етапів: ПК , хаб, а потім хаб , маршрутизатор. Отже, гравець будує з'єднання, враховуючи як логіку адресації, так і топологію мережі.

Коли з'єднання створено, система негайно перевіряє його правильність, що базується на логіці, описаній в наступному розділі. Якщо з'єднання відповідає умовам – воно зберігається. У протилежному випадку – скидається, і гравець може змінити свій вибір. Унікальністю гри є відсутність автоматичних підказок або повідомлень про помилки – це спонукає гравця самостійно шукати причини невдачі.

Користувач продовжує цей цикл: аналіз , підключення , перевірка, поки всі порти маршрутизатора не будуть задіяні. Після підключення останнього

з'єднання запускається загальна перевірка відповідності всіх пар "ПК - порт". Якщо всі пари коректні – рівень пройдено (рис. 2.4).

Рисунок 2.4 – Логіка комп’ютерної гри «Tech Campus»

У процесі гри гравець також повинен враховувати просторові обмеження: на складних рівнях заборонено перетин кабелів, а також є обмеження на кількість доступних портів. Це додає в гру елемент просторового планування та оптимізації мережевої структури. Такі виклики

змушують користувача приймати рішення не лише з огляду на відповідність IP, а й з позиції економії місця та топологічної ефективності.

Таким чином, алгоритм дій гравця в грі "Tech Campus" являє собою багаторівневий процес, який поєднує аналітичне мислення, просторову уяву та послідовне виконання дій у віртуальному середовищі. Його структура точно відображає реальний цикл прийняття технічного рішення у сфері мережевого адміністрування – від аналізу завдання до практичної реалізації.

В основі освітнього процесу в грі "Tech Campus" лежить оцінка знань гравця щодо будови IP-адресації та принципів з'єднання пристроїв у мережу. Для гарантії достовірності навчального ефекту система реалізує програмну перевірку правильності з'єднань, виконаних гравцем. Ця перевірка запускається щоразу, коли гравець намагається встановити з'єднання між ПК та портом маршрутизатора або, якщо потрібно, через проміжний хаб. Процес верифікації включає в себе не лише оцінку формальної відповідності адреси, а й поглиблений аналіз структури IP з урахуванням контексту складності завдання [16].

Фундаментом верифікації є модуль IPValidator, що виконує автоматизовану логічну перевірку відповідності між IP-адресою пристрою та умовою, заданою на порту маршрутизатора. Для цього в грі підтримуються різні типи умов:

Пряма відповідність: порт чекає на конкретну IP-адресу, наприклад, 192.168.0.10. Перевірка здійснюється шляхом побайтового порівняння IP-адреси ПК з умовою. Такий тип перевірки використовується здебільшого на початкових етапах.

Діапазон IP-адрес: порт приймає будь-яку адресу з визначеного інтервалу, наприклад, 192.168.0.10–192.168.0.50. У цьому випадку IP-адреса розглядається як 32-бітне ціле число, і перевірка виконується арифметично.

CIDR-маска (Classless Inter-Domain Routing): наприклад, 192.168.1.0/24. Тут адреса розбивається на мережеву та хостову частини. Валідація

проводиться шляхом побітового маскування, після чого результати порівнюються з базовою адресою підмережі.

Шаблон або умовний фільтр: порт приймає адреси, що закінчуються на певний октет або мають фіксовану частину, наприклад: "всі адреси, що закінчуються на .5" або "всі з другим октетом 168". Цей фільтр дозволяє створити ігрові завдання, що вимагають часткового аналізу IP-структури.

Модуль IPValidator втілює методи перевірки кожного з цих варіантів у формі універсального інтерфейсу. Це дозволяє портам маршрутизаторів зберігати посилання на об'єкт-умову незалежно від її типу, при цьому система зберігає масштабованість – у майбутньому можна буде додавати нові типи перевірок без зміни загальної логіки.

Після того, як гравець створює з'єднання, метод перевірки активується автоматично. У разі позитивного збігу з умовою, об'єкт порту активується, і в інтерфейсі користувача він отримує відповідне візуальне позначення (наприклад, зелений індикатор або замикання лінії). Якщо виникає помилка, з'єднання не реєструється, а візуальний елемент повертається до вихідного стану. Система при цьому не відображає сповіщення про тип помилки, що є продуманою педагогічною стратегією: гравець повинен самостійно проаналізувати та виправити недоліки, розвиваючи логічне мислення.

Важливою характеристикою валідаційного механізму є його реактивність у реальному часі. Це означає, що перевірка здійснюється не наприкінці рівня, а безпосередньо в момент спроби підключення. Такий підхід сприяє формуванню зв’язку між дією гравця та наслідками, закріплюючи знання через практичну взаємодію.

Окрім цього, реалізована перевірка унеможливлює одночасне підключення кількох пристроїв до одного порту – кожен порт маршрутизатора здатний прийняти лише одне з’єднання, і тільки за умови його валідності. Така вимога моделює принципи справжньої фізичної інфраструктури, де порт є єдиною точкою доступу.

Таким чином, логіка перевірки з'єднань у грі "Tech Campus" є ключовим елементом ігрового навчального процесу. Вона базується на класичних алгоритмах IP-верифікації, адаптованих до гнучких ігрових умов, і виконує не тільки функцію контролю, але й функцію навчального зворотного зв'язку, що є ключовим принципом у гейміфікованому освітньому середовищі.

На більш просунутих рівнях гри «Tech Campus», гравцеві потрібно не лише вірно встановити, який комп'ютер до якого порту маршрутизатора має бути під'єднаний, але й фізично прокласти шлях сполучення в межах ігрового поля таким чином, щоб він не перетинався з вже прокладеними кабелями. Цей механізм відтворює обмеження, властиві для реального проектування мережевої інфраструктури, де фізичні перетини є неприпустимими, особливо у випадку кабелів із схильними до шумів каналами чи специфічними технічними вимогами до прокладки ліній.

Для реалізації цієї функціональності у грі застосовується матрична модель поля, де кожна клітинка відповідає конкретному координатному блоку на сцені. Коли гравець створює з'єднання між двома об'єктами (комп'ютером та портом, або комп'ютером , хабом , портом), система за допомогою об'єкта WireConnection формує маршрут, що проходить клітинками ігрової матриці. Цей маршрут може бути створений вручну (за допомогою взаємодії гравця) або напівавтоматично – із використанням алгоритмів побудови шляху (наприклад, A* або прямолінійне прокладання по горизонталі й вертикалі).

Кожна клітинка, яку займає кабель, позначається у внутрішній структурі даних як "зайнята", і при спробі прокласти новий маршрут система здійснює перевірку: чи збігаються координати нового з'єднання з координатами вже відзначених активних з'єднань. Якщо збігу немає – з'єднання дозволяється; у випадку перетину – гравець отримує візуальне попередження (наприклад, підсвічування маршруту червоним кольором), або система блокує прокладку кабелю на рівні внутрішньої логіки.

Ця перевірка відбувається до моменту завершення з'єднання, що дає змогу попередити конфлікти до того, як система збереже з'єднання як валідне.

Це має значення з погляду продуктивності, оскільки вимагає менше виправлень у вже збереженій структурі маршрутизованої сітки. Таким чином, перевірка перетинів не є постфактум-корекцією, а складовою частиною активного контролю при плануванні.

З технічної точки зору, перевірка здійснюється на основі координатних позначок, які можуть бути представлені у вигляді: однотипних векторів (дві точки початку та кінця кожного з’єднання – якщо з’єднання пряме), або повного переліку клітинок, через які проходить кабель (у разі маршрутної побудови).

Окрім базового порівняння, в системі можуть бути реалізовані оптимізаційні алгоритми, що дають можливість визначити не лише факт перетину, але й які з альтернативних маршрутів матимуть мінімальну кількість перешкод. Це відкриває шлях для реалізації додаткових сценаріїв, де гравець має не тільки правильно підключити, але й обрати найбільш ефективний маршрут, наприклад, за кількістю поворотів, довжиною шляху або кількістю перетинів, які необхідно обійти.

Реалізація перевірки на перетини має не тільки технічне, але й дидактичне значення. Вона формує у гравця практичне уявлення про обмеження фізичного простору, спонукає планувати не тільки з огляду на IP-адресацію, а й враховуючи реальне розташування об'єктів. Це особливо важливо під час підготовки до роботи з обладнанням в реальних умовах, наприклад, під час прокладання кабельних трас у серверних кімнатах.

Отже, алгоритм перевірки перетинів у грі «Tech Campus» є одним із важливих компонентів логіки високого рівня складності. Він об'єднує аналіз топології з математичною обробкою просторових координат, що надає ігровому процесу реалізму та практичної користі.

Фінальним етапом кожної ігрової сесії у грі «Tech Campus» є перевірка повноти виконаного завдання, формування кінцевих результатів та обробка переходу до наступного рівня або повернення в головне меню. Завершення рівня - це не просто умовний перехід між ігровими сценами, а повноцінна

логічна процедура, що вимагає оцінки низки критеріїв та реакції системи на дії користувача.

Після того як гравець завершив прокладання усіх з'єднань, система активує внутрішній цикл підсумкової перевірки. В рамках цього циклу GameManager звертається до кожного об'єкта RouterPort, перевіряючи, чи: Порт має активне з'єднання. З'єднання встановлено з об'єктом PCUnit. IP-адреса підключеного ПК відповідає умові порту.

Якщо всі порти маршрутизатора заповнено коректно, і всі логічні перевірки пройдено успішно, система визначає, що рівень завершено. Якщо хоча б один порт залишився незаповненим, або IP не відповідає умовам - гравцю пропонується перевірити з'єднання, і рівень не вважається пройденим. Таким чином, обов'язковою умовою завершення є повна відповідність всіх логічних пар «ПК - порт».

У момент завершення рівня GameManager ініціює підрахунок результатів, що може включати такі компоненти:

Час проходження рівня - загальний час від моменту старту до завершення останнього з'єднання. Він використовується для порівняння ефективності користувача між спробами або для відображення у підсумковому вікні.

Кількість коректних підключень - дає змогу визначити ефективність гравця в рамках окремої сесії.

Кількість помилкових спроб - якщо гравець не з першої спроби під'єднав пристрій до правильного порту (враховується лише на складному рівні або за вимогою викладача).

Рейтинг або оцінка (опціонально) - може бути реалізована у вигляді умовної шкали, наприклад: "Відмінно", "Добре", "Потребує перегляду", або у вигляді кількості зірок (1–3), що виступають гейміфікованим стимулом до повторного проходження.

Після підрахунку система виводить екран результату, що реалізується через модуль UIController. Він відображає зібрану інформацію, надає короткий

огляд дій гравця та пропонує вибір подальших дій: Перейти до наступного рівня. Повторити рівень (з повним скиданням з'єднань) У складніших реалізаціях результати можуть бути збережені у внутрішній базі даних (наприклад, через PlayerPrefs або зовнішній JSON-файл), що дозволяє відстежувати динаміку навчання, зберігати прогрес і підключати аналітичний модуль для викладача або адміністратора системи.

Інтерфейс фінального етапу оформлено максимально лаконічно, без відволікаючих елементів. Це відповідає загальній концепції гри, яка не передбачає надмірної візуальної гейміфікації, а концентрується на змістовному зворотному зв'язку. Особливістю реалізації є відсутність автоматичних підказок або виправлень - гравець самостійно приймає рішення: перейти далі, чи ще раз спробувати побудувати оптимальні маршрути та знайти правильні відповідності IP.

Таким чином, процедура завершення рівня у грі «Tech Campus» виконує дві функції: з одного боку - формальну, пов'язану із завершенням задачі та переходом до наступного рівня, а з іншого - дидактичну, дозволяючи гравцеві рефлексивно оцінити власний результат і прийняти рішення щодо повторного навчання(рис. 2.5).

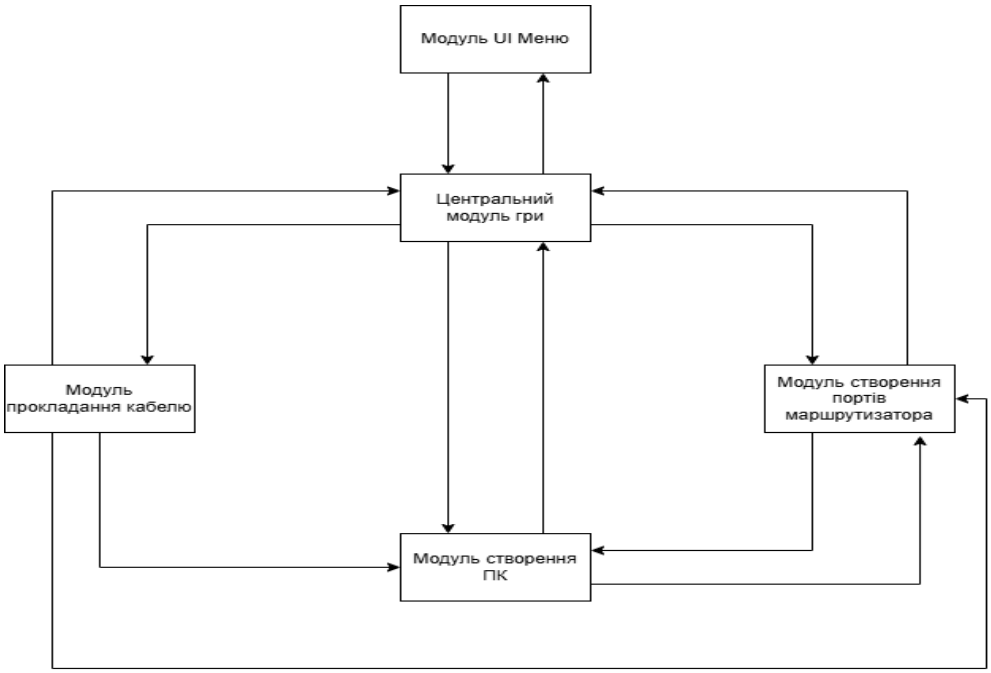


Рисунок 2.5 – Взаємодія всіх модулів комп’ютерної гри «Tech Campus»

. Це посилює ефект гейміфікації як засобу навчання, орієнтованого на результат.

2.2 **UML діаграми класів**

У процесі розробки програмного забезпечення, особливо комп'ютерних ігор, використання візуального моделювання системи за допомогою стандартів UML (Unified Modeling Language) є надзвичайно важливим. Діаграми UML дають можливість на етапі проектування визначити ключові компоненти системи, їхні функції, взаємозалежності, взаємодії та шляхи передачі даних. Це особливо актуально для складних програмних середовищ, де необхідна чітка модульна структура з логічно взаємодіючими класами [16].

Діаграма класів – це потужний інструмент для розробки програмного забезпечення, який допомагає зрозуміти структуру системи, документувати її та створити код, що відповідає цій структурі. Використання діаграм класів допомагає розробникам візуалізувати структуру системи, зрозуміти взаємозв’язки між класами та створити код, що відповідає цій структурі.

Діаграма класів програмного продукту зображена на рис. 2.6

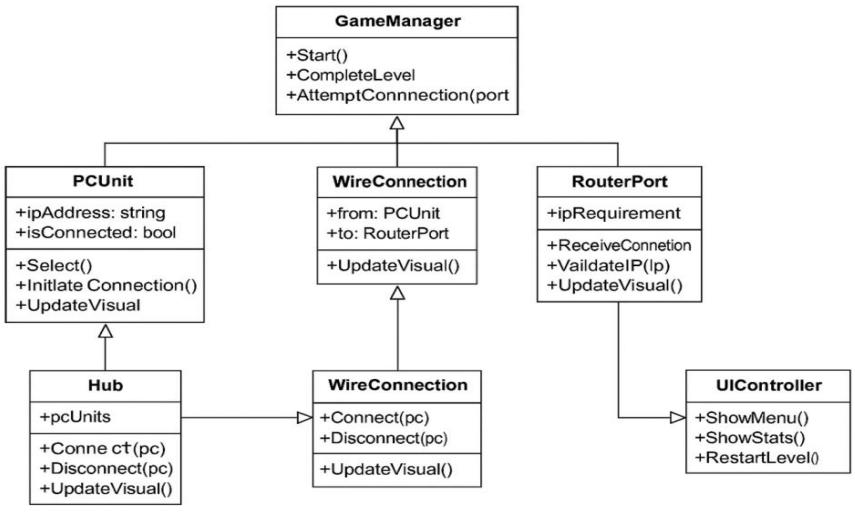


Рисунок 2.6 – UML-діаграма класів програмного модулю комп’ютерної гри «Tech Campus»

З діаграми видно, яким саме чином побудовані класи, які відображають сутності, а також відношення між класами в програмному продукті.

В рамках проєкту «Tech Campus» була розроблена UML-діаграма класів, яка відображає структуру ядра гри, основні функціональні об’єкти, відповідальність кожного модуля, а також передбачає масштабованість та гнучку модифікацію системи в майбутньому. Гра має модульну архітектуру, де кожен клас відповідає за певну функціональність: від управління ігровим станом до перевірки IP-адрес та відображення з'єднань [17].

Центральним елементом структури є клас GameManager, який виконує функцію контролера рівня. Саме він ініціалізує сцену, розміщує об’єкти на ігровому полі, відстежує завершення рівня, обробляє спроби підключення та керує переходами між станами гри. GameManager взаємодіє з іншими класами через делегати та прямі посилання на об'єкти сцени, координуючи роботу всієї системи.

Кожен ПК у грі представлений окремим екземпляром класу PCUnit. Цей клас містить інформацію про IP-адресу пристрою, його позицію на полі, а також методи взаємодії: вибір, ініціація з’єднання, графічне оновлення. Клас також містить прапори стану (підключено / не підключено) та пов’язаний з класом WireConnection, який створює візуальну лінію (кабель) між ПК і портом.

Порти маршрутизатора реалізовані через клас RouterPort. Він зберігає логіку перевірки відповідності IP: містить об’єкт-умову, яка може задаватись як конкретна IP-адреса, діапазон, шаблон або CIDR-маска. Також він відповідає за прийом підключення, зміну свого візуального стану та валідацію з’єднання за допомогою допоміжного модуля IPValidator.

Клас IPValidator - це ізольований функціональний блок, що реалізує алгоритми логічної перевірки IP-адрес. Його методи дозволяють перевірити відповідність IP до умови порту, враховуючи всі можливі формати: пряме співпадіння, діапазон, шаблон і CIDR. Таким чином, валідація винесена в

окрему одиницю, що забезпечує її багаторазове повторне використання та тестування [18].

У випадках, коли на рівні присутні хаби, їхня поведінка описується у класі Hub. Цей клас підтримує зберігання кількох підключених ПК, містить логіку розподілу з’єднань, обмеження на кількість портів і може взаємодіяти з RouterPort через WireConnection. Таким чином, Hub є своєрідним проміжним вузлом, який дозволяє відобразити складну топологію мережі.

Клас WireConnection відповідає за візуалізацію кабелю на ігровому полі. Він містить координати початку та кінця, методи малювання (на базі LineRenderer або його аналога в UI), а також прапори для перевірки перетинів з іншими кабелями. Для складних рівнів в ньому реалізовано логіку перевірки маршруту на наявність конфліктів за допомогою внутрішньої сіткової моделі поля.

Для обслуговування інтерфейсу гри використовується клас UIController, який відповідає за відображення меню, кнопок перезапуску, зміни складності, а також загальних статистичних елементів (таймер, очки, стан порту). Клас також забезпечує зв’язок між GameManager і Unity-сценами, надаючи доступ до основних елементів управління.

Усі ці класи взаємопов’язані через логіку подій, інтерфейси або безпосередні зв'язки. UML-діаграма класів дозволяє чітко простежити, які об’єкти взаємодіють між собою, які методи відповідають за які функції, та як забезпечується незалежність модулів. Це критично важливо для майбутнього масштабування, рефакторингу коду та забезпечення модульності.

Завдяки чітко розподіленим зонам відповідальності та правильно побудованим UML-зв’язкам, програмна структура гри «Tech Campus» є логічною, підтримуваною та відкритою до розширення. Впровадження UML-підходу на етапі проектування дозволило зменшити ризики дублювання логіки, покращити взаємодію компонентів та зосередити увагу на ізольованій реалізації ключових механік гри. Залежність: Один клас використовує інший, але не має прямого зв’язку ним

Асоціація: Об’єкти двох класів пов’язані між собою, дозволяючи переходити від одного до іншого

Агрегація: Один клас складається з інших класів, але вони можуть існувати окремо.

Композиція: Один клас складажться з інших класів, і вони не можуть існувати окремо

Діаграми класів застосовуються для візуалізації структури щоб розуміти з який частин складається система, документування щоб зафіксовувати архітектуру системи для майбутніх розробок та конструювання для створення коду системи, виходячи з діаграми.

2.3 Висновок

У другому розділі подано розгорнутий опис архітектурних засад роботи гри «Tech Campus», яка постає як гейміфікований засіб навчання для опанування азів IP-мереж та методів з'єднання пристроїв в локальній інфраструктурі. Розглянуто загальні механізми взаємодії користувача з ігровим середовищем, алгоритми старту рівня, перевірки IP-адрес, контролю перетинів кабелів та умови завершення сесії.

Детально описано структуру навчальної взаємодії, що реалізується через циклічну послідовність: аналіз умов, формування з'єднань, перевірка правильності, адаптація до просторових обмежень та підсумкова оцінка результатів. Чітке розмежування системних і призначених для користувача дій дозволяє грі поступово збільшувати складність задач, при цьому не обтяжуючи гравця надмірними елементами інтерфейсу.

Особливу увагу приділено адаптивності системи: наявність декількох рівнів складності (від прямого з'єднання до багаторівневої маршрутизації з хабами та контролем перетинів) забезпечує широкий спектр сценаріїв. Це дозволяє використовувати гру як для індивідуального навчання, так і для аудиторних занять, де потрібен поступовий розвиток логічного мислення та аналітичних здібностей у сфері мережевого адміністрування.

Також проаналізовано принципи внутрішньої валідації IP-адрес з використанням шаблонних умов, CIDR-масок і діапазонів, що наближає логіку гри до реальних випадків конфігурації мереж. Матричне представлення ігрового простору ефективно реалізує механіку перевірки перетинів, що підвищує практичну цінність ігрового процесу.

До текстового опису додано діаграми, що візуалізують ключові аспекти архітектури гри та динаміку взаємодії. Вони надали цілісне уявлення про структуру системи й забезпечили прозорість алгоритмічної логіки, необхідної для реалізації та тестування програмного модуля.

Отже, описаний у розділі підхід до моделювання навчального процесу через ігрову форму дозволяє ефективно втілити принципи гейміфікації у технічній освіті, а також створює основу для подальшої реалізації, масштабування і впровадження гри «Tech Campus» як навчального ресурсу в IT-сфері.

3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ TECH CAMPUS

3.1 Обгрунтування вибору інструментів технічної реалізації

Сучасна розробка ігор вимагає застосування потужних інтегрованих середовищ, які здатні забезпечити цілісний цикл створення, тестування та впровадження програмного продукту. Від правильного вибору інструментарію безпосередньо залежить якість, функціональність, масштабованість і подальша підтримка гри, особливо в контексті освітніх проєктів з елементами гейміфікації. Під час планування та реалізації гри "Tech Campus" було проведено ретельний аналіз доступного інструментарію розробки, з врахуванням специфіки задачі, вимог до функціоналу, графічної складової, планованого розширення й інтеграції.

Основним середовищем розробки було обрано Unity (версія 2022.2) - один з найпоширеніших та найфункціональніших ігрових рушіїв для створення 2D- та 3D-ігор будь-якого масштабу. Ключовими факторами вибору Unity є:

Unity дозволяє розробнику генерувати білди для широкого кола операційних систем - Windows, Linux, MacOS, а також для мобільних платформ (Android, iOS) та браузерних застосунків (WebGL). Це створює умови для масштабування проєкту, його використання у різних навчальних середовищах та подальшого розвитку продукту.

Для реалізації гри "Tech Campus" було застосовано систему роботи зі спрайтами, тайлмапами, UI, що дозволило створити інтуїтивно зрозумілий, візуально привабливий та ефективний інтерфейс користувача, а також забезпечити легку інтеграцію графічних матеріалів.

C# – офіційно рекомендована мова для Unity, що має низку вагомих переваг: сучасний синтаксис, сувора типізація, потужна система об'єктно-орієнтованого програмування, тісна інтеграція з API рушія, підтримка асинхронного коду та LINQ, що дозволяє ефективно управляти взаємодіями

між компонентами, подіями, ігровими об'єктами та складними механіками гейміфікації. Використання C# сприяє підвищенню надійності коду, його розширюваності та полегшує командну роботу.

Редактор Unity – це основа праці розробника, що пропонує багатовимірне середовище, яке структурує увесь процес створення інтерактивних 2D, 3D та VR/AR проектів. Оскільки рушій Unity працює на низькому рівні, використовуючи низькорівневі оптимізації, редактор сам по собі слугує як «міст» між технічними можливостями рушія та інтуїтивним дизайном ігрового контенту. Під час запуску редактора Unity користувач опиняється в цілісній екосистемі, де зібрані всі потрібні панелі та інструменти, аби полегшити як первинне прототипування, так і професійну розробку кінцевих версій проектів. Центральне місце в інтерфейсі займає вікно Scene, що відповідає за просторовий монтаж ігрового світу. Там можна безпосередньо маніпулювати об’єктами, переміщуючи, обертаючи та змінюючи їх масштаб у тривимірному просторі, водночас відстежуючи зміни в реальному часі. Поруч із Scene розташоване вікно Game, де зображається фінальний результат з урахуванням налаштувань камери та компонентів відтворення. Це дає змогу миттєво оцінити, як саме гравець побачить сцену під час гри, адже будь-які зміни в Scene автоматично оновлюються в Game, якщо проект знаходиться в режимі Play. Зверху цих панелей часто розміщується панель інструментів, де інженер може обирати режими трансформації (переміщення, обертання, масштабування), ввімкнути режим панорами чи зуму, а також налаштовувати інструменти малювання ландшафту або розміщення розсіяного контенту на поверхнях.

Зліва від Scene та Game, як правило, знаходиться вікно Hierarchy, що відображає ієрархічну структуру всіх GameObject-ів у поточній сцені. Як і в традиційних середовищах розробки, де об’єктно-орієнтовані елементи зберігаються як деревоподібні зв’язки, Hierarchy миттєво показує, які об’єкти є батьківськими чи дочірніми, як компоненти згруповані у конгломерати. Наприклад, група моделей, налаштована як «Персонаж», може містити

складну структуру з кількох частин тіла, Collider-ів, скриптів та аніматора, підпорядкованих єдиному кореневому об’єкту. Це значно полегшує навігацію у великих сценах з сотнями та тисячами елементів, оскільки за потреби можна швидко знайти будь-який елемент або увімкнути/вимкнути видимість та активність великої групи об’єктів одним кліком.

Важливою складовою є панель Inspector, де відбувається детальне налаштування властивостей обраного об’єкта. Коли розробник вибирає GameObject у Hierarchy або безпосередньо в Scene, Inspector демонструє всі додані до цього об’єкта компоненти – від базових Transform та Mesh Renderer до кастомних скриптів на C#. Кожен компонент має свій власний набір полів, які можна змінювати «на льоту»: це можуть бути позиція, обертання, масштаб, матеріали, анімаційні контролери, параметри освітлення та інше. Багато полів підтримують drag-and-drop, що спрощує роботу з ресурсами: наприклад, аби призначити матеріал моделі, достатньо перетягнути текстуру чи шейдер з панелі Project у відповідне поле в Inspector. Окрім того, розробник може додавати нові компоненти чи видаляти старі, швидко зберігаючи гнучкість у налаштуванні поведінки об’єкта та його зовнішнього вигляду. Для зручності кастомізації існують атрибути, які програміст може застосувати в коді, щоб визначити, як поля його скрипта будуть відображатися в Inspector: це дає змогу створювати зрозумілі інтерфейси налаштувань навіть для тих, хто менш обізнаний із вихідним кодом.

Нижня частина редактора зазвичай відведена під вікно Project, де відображається весь набір ресурсів, імпортованих у проект. Файли, папки, префаби, текстури, аудіо, скрипти, шейдери та сцени організовані у вигляді ієрархічного дерева, яке показує фізичну структуру папок у файловій системі. Тут можна не тільки переглядати ресурси, але й імпортувати нові через drag-and-drop із зовнішніх каталогів, запускати налаштування імпорту 3D-моделей чи оптимізації текстур, визначати налаштування конвертації аудіо. Ця панель водночас інтегрована з механізмом Asset Database, який слідкує за змінами файлів і підтримує індексацію для швидкого пошуку. Процес імпорту може

супроводжуватися конвертацією метаданих, створенням анімованих кліпів та генерацією шейдерних варіантів, що робить роботу з ресурсами плавним та передбачуваним.

У правому нижньому кутку зазвичай розташоване вікно Console, яке використовується для відстеження повідомлень від рушія: тут з'являються помилки компіляції скриптів, попередження про потенційні проблеми (наприклад, відсутність матеріалів на моделях, конфлікти фізичних шарів тощо) та лог-повідомлення, які розробник може програмно виводити у процесі налагодження. Console підтримує фільтрацію за рівнем повідомлень (Error, Warning, Info) і дозволяє подвійним кліком перейти до відповідного рядка в коді чи сюжетному об'єкті сцени. Завдяки цьому налагодження стає дуже інтерактивним: як тільки Unity виявляє невідповідність у скрипті, він миттєво надає зворотний зв'язок і дозволяє виправити помилку, не потребуючи вручну переглядати логи у зовнішньому застосунку.

Важливою особливістю редактора є система Package Manager, яка дає змогу керувати набором готових середовищ, бібліотек та інструментів безпосередньо з інтерфейсу. Переходячи в розділ Package Manager, розробник може підключати офіційні пакети від Unity (наприклад, TextMesh Pro, Post-processing Stack, Addressables), отримувати доступ до пакетів із Unity Registry чи інсталювати сторонні рішення з Git. Підтримка версійності та сумісності пакетів з поточною версією рушія гарантує, що будь-які оновлення будуть відбуватися без раптових збоїв. Окрім того, за допомогою Package Manager можна створювати власні пакети: виділяти конкретні скрипти, шейдери чи налаштування в окремий пакет та ділитися ним між проектами чи колегою, не переносячи при цьому зайві невикористані ресурси.

Неперечною перевагою є масштабованість інтерфейсу: кожен розробник може налаштувати компоновку вікон під свої потреби, створивши кілька вкладок Scene, Game, Console чи інші панелі та розташувавши їх у зручному порядку. Позитивним аспектом є й можливість зберігати кілька робочих

просторів (Layout), що дає змогу швидко перемикатися між режимом дизайну рівнів, режимом налагодження коду та режимом художника, який займається текстуруванням і матеріалами. Це особливо корисно в командних проектах, коли один розробник фокусується на компонентній логіці, а інший – на художньому оформленні: кожен може працювати в оптимальному для себе інтерфейсі.

Серед просунутих можливостей Unity Editor варто виокремити вбудований редактор анімацій, де за допомогою вікна Animation можна створювати ключові кадри для просунутого контролю руху персонажів, змін властивостей компонентів та складних процедурних анімацій. Тут також знаходиться Animator і дерево переходів між станами анімації, що дає змогу реалізувати складні поведінкові моделі, наприклад, переключення анімацій ходьби, бігу, стрибка чи атаки з урахуванням параметрів, які змінюються в реальному часі на основі даних скриптів.

Не можна обійти увагою й інструменти освітлення та візуального оформлення, які представлені в Lighting Settings. Завдяки цим налаштуванням розробник може задавати глобальні параметри затінення, керувати картами освітленості (Lightmaps), налаштовувати динамічні й статичні джерела світла, а також використовувати глобальне освітлення (Global Illumination) для досягнення більш реалістичного візуалу. Вбудована підтримка фізично коректного рендерингу (Physically Based Rendering) дозволяє застосовувати налаштування матеріалів, які адекватно реагують на освітлення, а HDR (High Dynamic Range) дає змогу досягати багатих контрастів і насичених кольорів.

Підсумовуючи, Unity Editor постачає інструменти для профілювання продуктивності: Profiler аналізує навантаження на процесор, графічний процесор, пам’ять, мережеву взаємодію та інші підсистеми. Це дає змогу виявляти «вузькі місця» у роботі гри, усувати надлишкові виклики до фізики або непотрібні відображення об’єктів. Миттєвий доступ до цих даних допомагає оптимізувати проект ще на етапі розробки, щоб уникнути подальших проблем із продуктивністю на цільових платформах.

Unity Editor є комплексним середовищем, яке інтегрує в собі інструменти для моделювання сцен, налаштування та налагодження логіки на C#, управління ресурсами, створення анімацій, освітлення, матеріалів та профілювання. Завдяки гнучкості інтерфейсу, підтримці пакунків і кастомізації робочих просторів, розробник отримує змогу організувати процес створення гри від початкового ескізу до фінального білду, зберігаючи високу продуктивність та зручність роботи на кожному етапі. Це середовище не тільки полегшує навчання основам геймдеву, але й задовольняє вимоги професійних студій, дозволяючи швидко прототипувати, оптимізувати та випускати проекти на численні платформи з тієї самої бази коду.

Підтримка сторонніх ресурсів та розширень. Unity підтримує імпорт графіки, анімацій, зовнішніх бібліотек та плагінів, завдяки чому у грі “Tech Campus” була реалізована індивідуальна стилістика, що відповідає вимогам сучасних освітніх застосувань.

Мова C# закріпилась як ключовий інструмент для розробки ігор на рушії Unity завдяки своєму виваженому синтаксису, об’єктно-орієнтованому стилю та щільній інтеграції з самою архітектурою рушія. Своїм народженням C# завдячує Microsoft, яка в межах платформи .NET мала за мету створити мову, що поєднає зручність і знайомість з мовами, подібними до C, з надійними механізмами управління пам'яттю та безпечного програмування. З моменту появи у 2000 році, C# еволюціонувала, підтримуючи шаблони класів, лямбда-вирази, технології LINQ та асинхронне програмування. Такий постійний розвиток зробив мову все більш привабливою для розробників, а її сильна екосистема в поєднанні з рушієм Unity сформувала унікальну платформу для швидкої реалізації ігрової логіки з мінімальними затратами часу та ресурсів.

Unity, у свою чергу, базується на компонентно-орієнтованій архітектурі, де кожен ігровий об’єкт є контейнером для окремих компонентів. Саме через компоненти забезпечується модульність, розширюваність і повторне використання коду. C# стає основним засобом реалізації поведінки цих компонентів: клас, що успадковує MonoBehaviour, не лише отримує доступ до

внутрішніх API рушія, але й автоматично інтегрується у життєвий цикл Unity. Коли розробник створює скрипт на C#, Unity розміщує його екземпляр як компонент на певному GameObject. Відповідно до життєвого циклу, методи Awake, Start, Update та FixedUpdate дають можливість послідовно ініціалізувати дані, здійснювати підписку на події, виконувати щокадрові оновлення логіки та коректно обробляти фізичні взаємодії (через FixedUpdate). Така модель забезпечує чіткий розподіл етапів ініціалізації та оновлення, зменшуючи ймовірність помилок, які можуть виникнути через непередбачуване виконання коду.

Враховуючи це, розробник може писати C#-скрипти, які відіграють роль контролерів поведінки ігрових об’єктів. Наприклад, рух персонажа часто реалізується через отримання даних про введення користувача в методі Update з подальшою передачею цих даних в FixedUpdate для застосування до Rigidbody з метою точного обчислення фізичних взаємодій. Такий розподіл завдань гарантує плавний і стабільний рух у грі, уникнення непередбачуваних поведінкових артефактів, а також дотримання принципу відокремлення логіки геймплею від фізичного рушія. Універсальність C# виявляється й у тому, що код, створений для одного типу об’єкта, може бути легко адаптований під інші, оскільки шаблони класів, події та делегати сприяють слабкій зв’язності між компонентами. Замість безпосередніх звернень до інших об’єктів програміст може ініціювати власні події, такі як OnHealthChanged або OnDied, які підписані представниками UI, аудіосистеми або менеджером геймплею. Цей підхід узгоджується з принципами SOLID, зокрема з принципом інверсії залежностей, оскільки кожен компонент підписується на ті події, які його цікавлять, не зберігаючи в пам’яті прямих посилань на інші класи.

Окрім того, C# у Unity активно використовує корутини як спосіб організації асинхронних операцій у межах одного потоку виконання. У разі опису класичної шаблонної зброї навичка Reload може бути представлена як корутина: коли натискається кнопка "стріляти", активується корутина, яка здійснює постріл, затримує виконання на встановлений час (використовуючи

операцію WaitForSeconds) і лише після цього дозволяє наступний постріл. Таким чином, запобігається блокування потоку і забезпечується чітка послідовність подій. Проте розробник повинен бути обережним і не запускати декілька корутин одночасно без перевірки, оскільки існує ризик багаторазового повторення операцій перезаряджання або інших важливих процесів, що може призвести до некоректного стану гри. В такому випадку слід передбачити флаг isReloading, який інформуватиме про те, що корутина все ще працює, і заборонятиме її повторний запуск, поки попередній цикл не буде завершено.

Питання оптимізації пам'яті та продуктивності є вкрай важливим у контексті мобільних ігрових проектів. Незважаючи на те, що C# працює з автоматичним збирачем сміття, часті алокації об’єктів (наприклад, створення масивів або нових рядків у межах методу Update) можуть спричинити раптові "фризи" під час виконання, коли рушій ініціює збирання сміття. Тому сучасна практика передбачає уникнення зайвої динамічної пам’яті в критичних точках оновлення геймплею, а використання патерну Object Pooling дозволяє повторно використовувати вже створені об'єкти, що значно зменшує навантаження на GC. При цьому Unity надає механізми завантаження ресурсів через статичний каталог Resources або через сучасну систему Addressables, яка дозволяє асинхронно та дозовано завантажувати необхідні активаційні дані (текстури, моделі, аудіофайли) без підвищених пікових навантажень.

Після завершення розробки коду на C#, Unity застосовує технологію IL2CPP (Intermediate Language To C++), що перетворює проміжний код .NET в C++ перед остаточною компіляцією на цільові платформи. Такий підхід дозволяє зменшити накладні витрати на інтерпретацію IL, збільшити продуктивність виконання та захистити вихідний код від легкого зворотного інжинірингу. Водночас розробник повинен пам’ятати, що деякі можливості Reflection можуть бути обмежені або потребувати спеціальних налаштувань для коректної роботи в середовищі IL2CPP, а також дотримуватись практики мінімізації використання рефлексії в продуктивному коді.

Загалом, використання C# у Unity відкриває широкі можливості для реалізації складної ігрової логіки. Використання об’єктно-орієнтованих підходів, патернів проєктування (Single Responsibility, Observer, Factory Method, State Machine) сприяє покращенню читабельності, розширюваності та простоти тестування. Велика спільнота розробників Unity забезпечує доступ до безлічі прикладів, шаблонів і готових рішень, що полегшує вирішення типових завдань — від управління рухом персонажа та анімацією до реалізації складних мережевих механік. Розробник, який володіє основами C# і розуміє специфіку взаємодії з Unity API, може створювати високоякісний, масштабований та продуктивний код, що відповідає вимогам сучасного ігрового ринку. При цьому академічне розуміння архітектурних принципів рушія та можливостей мови допомагає уникнути типової спіралі технічного боргу, коли швидка реалізація призводить до хаотичного коду та ускладнює подальшу підтримку та розширення проєкту. На практиці це означає, що грамотне проєктування класів, чітка інкапсуляція даних, делегування відповідальності через події та правильне управління ресурсами дозволяють досягти балансу між швидкістю розробки та якістю кінцевого продукту. Саме поєднання можливостей C# та оптимізованих механізмів Unity формує потужну платформу для створення сучасних ігор у будь-якому жанрі – від простих 2D-аркад до великих 3D-проектів із розгалуженою логікою та складною фізикою. [19].

Для написання, редагування та підтримки коду використовувалось сучасне середовище Visual Studio Code з офіційним плагіном Unity Tools від Unity. Це забезпечило інтеграцію автодоповнення, навігацію по проєкту, інструменти для зручного налагодження та управління залежностями. Така інтеграція дозволила значно скоротити час на перехід від написання коду до тестування в редакторі Unity, збільшила продуктивність розробника та мінімізувала ризики технічних помилок.

Важливою складовою розробки гри “Tech Campus” стала офіційна студентська ліцензія Unity. Unity Technologies пропонує безкоштовний доступ

до повнофункціональної версії Unity Pro для студентів через спеціальний Student Plan. Цей план розроблений саме для тих, хто навчається в акредитованих середніх та вищих навчальних закладах і підтвердив свій студентський статус. Відрізнити Student Plan від інших безкоштовних варіантів Unity можна перш за все тим, що він надає повний інструментарій Unity Pro без жодних функціональних обмежень, а натомість вимагає підтвердження статусу студента через спеціальний сервіс верифікації. Після успішної верифікації студент отримує ліцензію на один рік, яку можна продовжити, якщо академічний статус залишається актуальним. Процес реєстрації досить простий: необхідно в Unity Hub увійти в обліковий запис Unity ID, вибрати опцію активації Student Plan і пройти процедуру підтвердження через сторонній сервіс, що перевіряє приналежність до навчального закладу. Після цього ліцензійний ключ надходить на електронну адресу, і у вікні керування ліцензіями Unity Hub автоматично активує всі можливості Pro-редакції.

Основною перевагою Student Plan є доступ до тих самих професійних інструментів, які використовуються в комерційній розробці. Студенти мають можливість створювати 2D- та 3D-проєкти будь-якої складності, працювати з високоякісними графічними ефектами, розширеними системами фізики та багатоплатформним розгортанням. В рамках цього плану відкривається доступ до Unity Cloud Services, включно з налаштуванням збірки проєкту в хмарі, що значно спрощує командну роботу та інтеграцію CI/CD. Окрім того, користувачі Student Plan отримують права на використання певних сторонніх інструментів з навчальною ліцензією, зокрема Odin Inspector для гнучкого налаштування інспектора та Synty Asset Bundle для швидкого прототипування з використанням готових 3D-моделей. Такий набір ресурсів дає змогу зосередитись на вивченні розробки ігор та практичному застосуванні здобутих знань, замість витрачання часу на створення базових ресурсів або придбання сторонніх інструментів.

Щодо комерційного використання, Student Plan не обмежує можливості випуску та продажу власних проєктів. Єдиною умовою є дотримання фінансового порогу: якщо прибуток від продажу ігор або додатків перевищить один мільйон доларів США за останні дванадцять місяців, розробник повинен перейти на відповідну платну ліцензію Unity Pro або Enterprise. Раніше цей ліміт був значно нижчим, але зараз його підняли до мільйона, що робить студентську ліцензію особливо привабливою для тих, хто розпочинає свій шлях у геймдеві й не готовий платити за комерційний інструментарій.

Student Plan суттєво відрізняється від безкоштовної Personal Edition тим, що остання має жорсткіше фінансове обмеження – прибуток понад сто тисяч доларів за весь час використання вимагає переходу на платний план, натомість Student Plan дозволяє заробляти більше без обмежень функціональності. Водночас існує ще й Educational Grant License, призначена не окремим студентам, а навчальним закладам, де адміністратори вишу можуть розпоряджатися кількома ліцензіями одночасно. Student Plan, навпаки, прив’язаний до конкретної людини та може використовуватись незалежно від університетської інфраструктури, наприклад, вдома або в місцях неформального навчання.

З академічної точки зору, доступ до Student Plan сприяє формуванню практичних навичок у розробці ігор та інтерактивного контенту. Студенти працюють у тій самій версії редактора, яку використовують професіонали індустрії, і це підвищує їхню конкурентоспроможність на ринку праці. Безкоштовний доступ до хмарних сервісів Unity Cloud та знижки на асети в Unity Asset Store дозволяють ефективно прототипувати ігрові концепції, експериментувати з окремими ресурсами та одночасно оволодівати практиками управління проектними артефактами. Наявність інструментів для візуального налагодження коду, готових 3D-моделей та потужних засобів оптимізації дає змогу зосередитись на розробці геймплейних механік та вивченні архітектурних патернів, не витрачаючи зайвий час на опанування базових редакторських технік або придбання дорогих плагінів.

Таким чином, Student Plan є важливим інструментом у підготовці майбутніх розробників ігор, дозволяючи без жодних фінансових перепон оволодіти всіма необхідними інструментами Unity Pro та отримати безцінний досвід роботи в професійному середовищі. Завдяки цьому студенти можуть швидше інтегруватися в індустрію, створювати якісні проєкти та паралельно здобувати знання, які будуть актуальними у подальшій кар'єрі.

Окрім основного рушія, для підготовки графічних матеріалів використовувались інструменти Adobe Photoshop (для створення та обробки спрайтів, фонів, іконок) та Aseprite (для розробки піксельної графіки й анімацій). Система контролю версій Git забезпечувала відстеження змін та надійне зберігання історії розробки проєкту.

3.2 Програмна реалізація гри Tech Campus

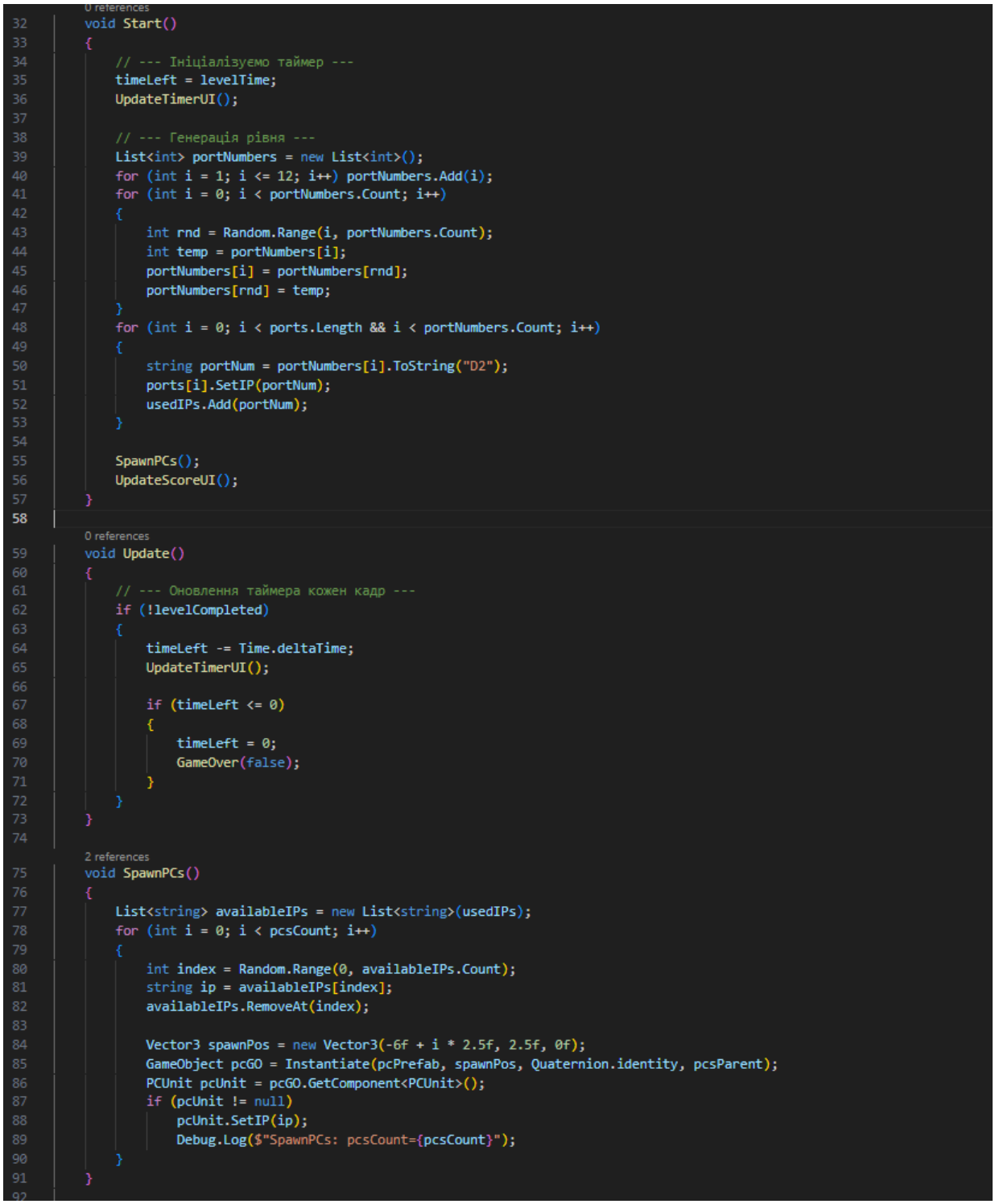
Для практичної реалізації гри «Tech Campus». було розроблено комплексну структуру головних об'єктів, які визначають функціонування всієї ігрової системи. Нижче наведено докладний опис кожного з ключових елементів програмного модуля, їхньої ролі у загальній архітектурі гри, основних функцій та алгоритмічних рішень, що забезпечують освітню, гейміфікаційну та ігрову логіку.

На першому етапі впровадження передбачено створення ключового компонента - GameManager.

GameManager реалізовано як клас-одиночка (Singleton pattern) - це досягається статичним полем instance і присвоєнням його в Awake(). Це дозволяє будь-якому іншому компоненту гри легко звертатися до GameManager та його публічних методів/змінних, забезпечуючи централізоване керування грою.

GameManager - це центральний керуючий компонент усієї ігрової логіки у проєкті «Tech Campus». Його завдання - координувати взаємодію між усіма об’єктами сцени, реалізовувати життєвий цикл рівня, керувати таймером, підрахунком очок, складністю гри, ініціалізувати і рестартувати рівень, а

також забезпечувати зв’язок між ігровим світом та користувацьким інтерфейсом (рис. 3.1).



```
0 references
void Start()
{
    // --- Ініціалізуємо таймер ---
    timeLeft = levelTime;
    UpdateTimerUI();

    // --- Генерація рівня ---
    List<int> portNumbers = new List<int>();
    for (int i = 1; i <= 12; i++) portNumbers.Add(i);
    for (int i = 0; i < portNumbers.Count; i++)
    {
        int rnd = Random.Range(i, portNumbers.Count);
        int temp = portNumbers[i];
        portNumbers[i] = portNumbers[rnd];
        portNumbers[rnd] = temp;
    }
    for (int i = 0; i < ports.Length && i < portNumbers.Count; i++)
    {
        string portNum = portNumbers[i].ToString("D2");
        ports[i].SetIP(portNum);
        usedIPs.Add(portNum);
    }

    SpawnPCs();
    UpdateScoreUI();
}

0 references
void Update()
{
    // --- Оновлення таймера кожен кадр ---
    if (!levelCompleted)
    {
        timeLeft -= Time.deltaTime;
        UpdateTimerUI();

        if (timeLeft <= 0)
        {
            timeLeft = 0;
            GameOver(false);
        }
    }
}

2 references
void SpawnPCs()
{
    List<string> availableIPs = new List<string>(usedIPs);
    for (int i = 0; i < pcsCount; i++)
    {
        int index = Random.Range(0, availableIPs.Count);
        string ip = availableIPs[index];
        availableIPs.RemoveAt(index);

        Vector3 spawnPos = new Vector3(-6f + i * 2.5f, 2.5f, 0f);
        GameObject pcGO = Instantiate(pcPrefab, spawnPos, Quaternion.identity, pcsParent);
        PCUnit pcUnit = pcGO.GetComponent<PCUnit>();
        if (pcUnit != null)
            pcUnit.SetIP(ip);
            Debug.Log($"SpawnPCs: pcsCount={pcsCount}");
    }
}
```

Рисунок 3.1 – Фрагмент коду компонента «GameManager»

ports - масив портів маршрутизатора (об’єктів класу RouterPort), які необхідно згенерувати для кожного рівня.

pcPrefab - префаб для створення ПК на сцені.

pcsParent - трансформ-контейнер, у якому групуються всі створені ПК (для зручності організації у ієрархії сцени).

pcsCount - поточна кількість ПК, що динамічно змінюється в залежності від складності.

usedIPs - список IP-адрес, які були присвоєні портам і використовуються для генерації ПК.

score, scoreText - поточна кількість очок гравця і UI-елемент для їх відображення.

levelTime, timeLeft, timerText - тривалість рівня, залишок часу і відповідний UI-елемент.

levelCompleted - прапорець, який сигналізує про завершення рівня (для блокування дій після фінішу).

Основні функції та їх алгоритмічне наповнення:

1. Awake()

Ініціалізує Singleton, забезпечуючи доступ до GameManager з будь-якого іншого класу.

2. Start()
    1) Ініціалізує таймер.
    2) Генерує рівень: створює портові адреси (випадковим чином), розподіляє їх по об’єктах RouterPort, зберігає використані IP, спавнить ПК із відповідними IP.
    3) Оновлює інтерфейс (очки, таймер).
3. Update()
    1) Відповідає за щосекундне оновлення таймера.
    2) Якщо рівень не завершено - віднімає час і оновлює відображення; якщо час вичерпано - завершує рівень і виводить повідомлення.
4. SpawnPCs()
    1) Зі списку згенерованих IP створює потрібну кількість ПК у визначених координатах.

2) Для кожного ПК призначає свою IP-адресу та додає у батьківський об’єкт на сцені.
3) Додає дебаг-лог для контролю роботи спавну.

5. AddScore(), UpdateScoreUI()
   1) Оновлює значення очок при кожній дії, відображає зміни у UI.
6. UpdateTimerUI()
   1) Виводить поточний залишок часу у UI.
7. CheckLevelComplete()
   1) Перевіряє, чи всі ПК підключені (визначає завершення рівня).
   2) Якщо так - викликає GameOver з ознакою успіху.
8. GameOver(bool win)
   1) Виводить фінальне повідомлення: у разі перемоги - додає бонус за залишок часу (10 очок за кожну секунду), інакше - повідомляє про завершення через брак часу.
9. RestartGameSession()

   Повністю скидає стан сцени:
   1) Видаляє всі ПК.
   2) Очищає масив IP-адрес.
   3) Знову генерує адреси портів і ПК.
   4) Оновлює очки та таймер, повертає прапорець рівня у початковий стан.
   5) Перезапускає спавн ПК.
10. ApplyDifficulty(int newPcsCount, float newLevelTime)
    1) Дозволяє змінити складність гри “на льоту”, приймаючи з MenuManager або іншого класу нові параметри (кількість ПК, час), після чого перезапускає рівень з оновленими даними.

Без GameManager проєкт був би набором розрізнених об'єктів, не здатних до координованої роботи, підрахунку результатів чи реалізації складніших сценаріїв.

MenuManager відповідає за організацію роботи всіх меню: головного, меню складності, паузи, а також за коректний перехід між ігровими режимами.

Основні функції:

Start() - Відкриває головне меню при запуску гри, переводить гру у стан паузи.

ShowMainMenu(), HideAllMenus() - Активує чи деактивує відповідні панелі, керує Time.timeScale для коректної зупинки геймплею.

StartGame() - Закриває меню, переводить гру у режим взаємодії зі сценою.

OpenDifficulty() - Відкриває меню вибору складності.

SetDifficultyEasy/Medium/Hard() - Передає GameManager відповідні параметри (кількість ПК, час) через ApplyDifficulty.

PauseGame(), ResumeGame() - Ставить гру на паузу чи повертає в ігровий режим (активує/деактивує меню).

MenuManager ізолює інтерфейс від геймплею, спрощує супровід і розвиток системи меню, дає змогу легко розширювати функціонал без зміни основної логіки (рис. 3.2).

```
0 references
void Update()
{
    // ESC відкриває/закриває меню під час гри
    if (Input.GetKeyDown(KeyCode.Escape) && !mainMenuPanel.activeSelf && !difficultyPanel.activeSelf)
    {
        PauseGame();
    }
}

// ГОЛОВНЕ МЕНЮ
1 reference
public void ShowMainMenu()
{
    if (mainMenuPanel != null) mainMenuPanel.SetActive(true);
    if (difficultyPanel != null) difficultyPanel.SetActive(false);
    Time.timeScale = 0f;
    gameIsPaused = true;
}

5 references
public void HideAllMenus()
{
    if (mainMenuPanel != null) mainMenuPanel.SetActive(false);
    if (difficultyPanel != null) difficultyPanel.SetActive(false);
    Time.timeScale = 1f;
    gameIsPaused = false;
}
```

Рисунок 3.2 – Фрагмент коду компонента «MenuManager»

Клас CableDrawer - відповідає за механіку прокладання кабелів між ПК та маршрутизатором.

CableDrawer забезпечує взаємодію користувача з ігровим полем через механіку “drag-and-drop” - малювання та підключення кабелів від ПК до портів.

Update() - Відслідковує події миші під час малювання кабелю, оновлює точки та LineRenderer.

StartDrawingCable(PCUnit pc) - Запускає процес малювання кабелю від обраного ПК, ініціалізує LineRenderer, задає колір.

CableIntersectsExisting(List<Vector3> newCable) - перевіряє, чи новий кабель перетинає хоча б один із вже прокладених (алгоритм перевірки відрізків).

LinesIntersect(Vector2 A, Vector2 B, Vector2 C, Vector2 D) - Геометричний алгоритм визначення перетину відрізків.

Завершення малювання (OnMouseUp) - перевіряє, чи підключено до правильного порта, аналізує перетини, нараховує очки, оновлює статуси.

Виуористання перевірки на перетин кабелів додає реалістичності та гейміфікації, дозволяє тренувати навички планування мережі, забезпечує візуальний контроль і ускладнення (рис. 3.3).

```
    // Допоміжна функція для перетину відрізків
    1 reference
    public static bool LinesIntersect(Vector2 A, Vector2 B, Vector2 C, Vector2 D)
    {
        float denominator = (B.x - A.x) * (D.y - C.y) - (B.y - A.y) * (D.x - C.x);
        if (denominator == 0) return false;

        float numerator1 = (A.y - C.y) * (D.x - C.x) - (A.x - C.x) * (D.y - C.y);
        float numerator2 = (A.y - C.y) * (B.x - A.x) - (A.x - C.x) * (B.y - A.y);

        float r = numerator1 / denominator;
        float s = numerator2 / denominator;

        return (r >= 0 && r <= 1) && (s >= 0 && s <= 1);
    }
}
```

Рисунок 3.3 – Фрагмент коду компонента «CableDrawer»

Клас PCUnit - Представлення ПК, як об’єкта на ігровому полі.

PCUnit - скрипт, що моделює окремий ПК на полі: візуалізує, зберігає адресу, запускає прокладання кабелю.

SetIP(string ip) - Встановлює IP-адресу, оновлює вивід на сцену.

OnMouseDown() - При натисканні миші ініціює прокладання кабелю через CableDrawer, якщо ПК не підключений.

Клас PCUnit забезпечує ідентифікацію вузлів мережі, підтримує навчальний підхід, дозволяє розширювати клас, для використання його на інших пристроях (рис. 3.4).

```
0 references
public class PCUnit : MonoBehaviour
{
    2 references
    public TextMeshPro ipLabel;  // Прив'язати 3D TextMeshPro (підпис над ПК)
    1 reference
    public string ipAddress;     // IP цієї машини
    1 reference
    public bool isConnected = false; // Чи вже підключено кабелем

    // Встановлення IP-адреси і підпису
    0 references
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }

    // Клік мишею по ПК — запускаємо малювання кабелю
    0 references
    private void OnMouseDown()
    {
        if (!isConnected)
        {
            // Передаємо посилання на цей ПК скрипту CableDrawer
            FindObjectOfType<CableDrawer>().StartDrawingCable(this);
        }
    }
}
```

Рисунок 3.4 – Фрагмент коду компонента «PCUnit»

Клас RouterPort - Представлення порта маршрутизатора, як об’єкта на ігровому полі.

RouterPort моделює порт маршрутизатора: зберігає адресу, відображає її у сцені, контролює статус підключення.

SetIP(string ip) - Призначає унікальну IP-адресу порту, оновлює підпис.

isConnected - Прапорець для перевірки зайнятості порта (запобігає повторному підключенню).

Моделює роботу портів реального мережевого обладнання, дозволяє відстежувати та контролювати підключення, забезпечує навчальний ефект (рис. 3.5).

```
0 references
public class RouterPort : MonoBehaviour
{
    2 references
    public TextMeshPro ipLabel;   // Прив'язати 3D TextMeshPro (підпис над портом)
    1 reference
    public string ipAddress;      // IP для цього порта
    0 references
    public bool isConnected = false; // Чи вже зайнятий порт

    // Встановлення IP-адреси і підпису
    0 references
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }
}
```

Рисунок 3.5 – Фрагмент коду компонента «RouterPort»

Взаємодія та загальна логіка у грі «Tech Campus» має наступний вид

GameManager ініціалізує рівень, генерує порти та ПК із унікальними адресами.

MenuManager управляє всіма меню, передає параметри складності GameManager-у.

Гравець клікає на ПК (PCUnit), запускається процес малювання кабелю (CableDrawer).

CableDrawer обробляє взаємодію, малює кабель, перевіряє підключення, перетини, нараховує очки, повідомляє GameManager.

RouterPort приймає підключення, оновлює свій статус.

GameManager веде облік очок, часу, завершує рівень при виконанні умов.

MenuManager забезпечує повернення у меню, зміну складності, рестарт гри.

3.3 Тестування гри Tech Campus

При тестуванні програмного додатку гри Tech Campus необхідно перевірити правильність виконання графічного інтерфейсу та ігрових механік, що були спроектовані та розроблені під час виконання кваліфікаційної роботи [20].

Тестування розпочато з відкриття головного файлу гри Tech Campus.exe, після відкриття нас зустрічає головне меню гри (рис. 3.6)

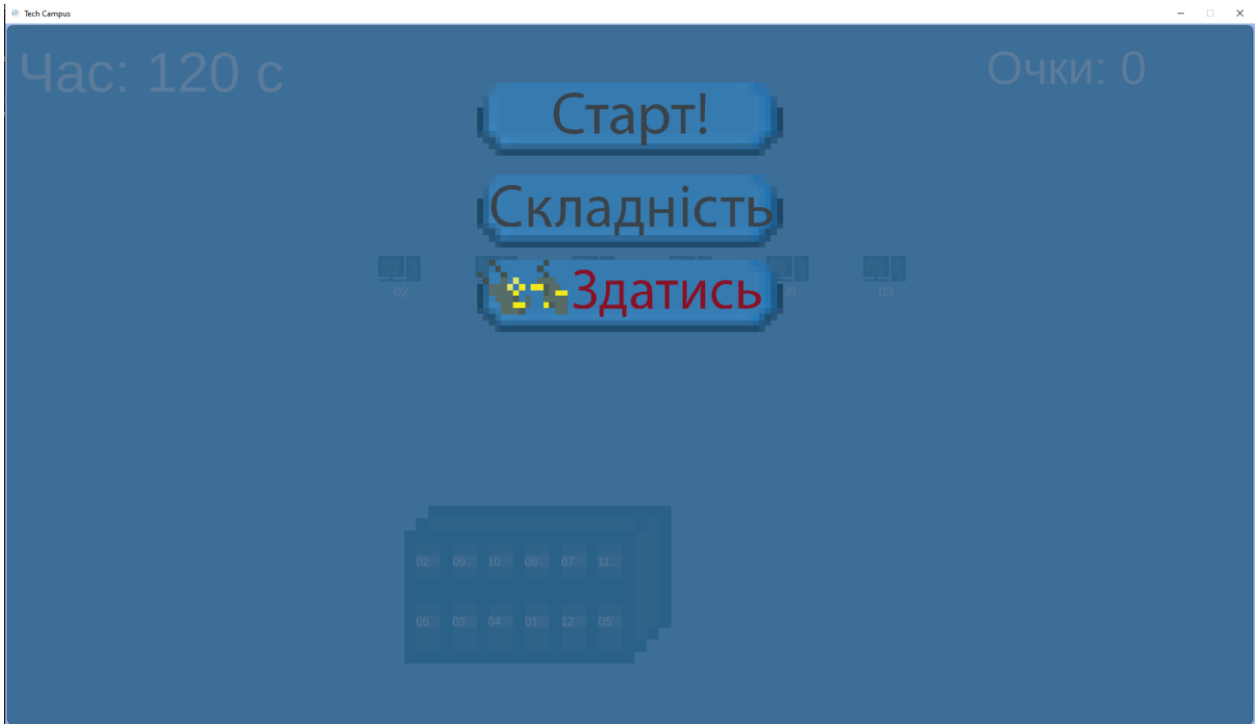


Рисунок 3.6 – Загальний вигляд головного меню меню гри Tech Campus

При старті застосунку автоматично обирається середня складність - «Цікавіше» для тестування виконаємо запуск гри в режимі «Легко», оберемо її в через функцію вибору складності (рис. 3.7) .

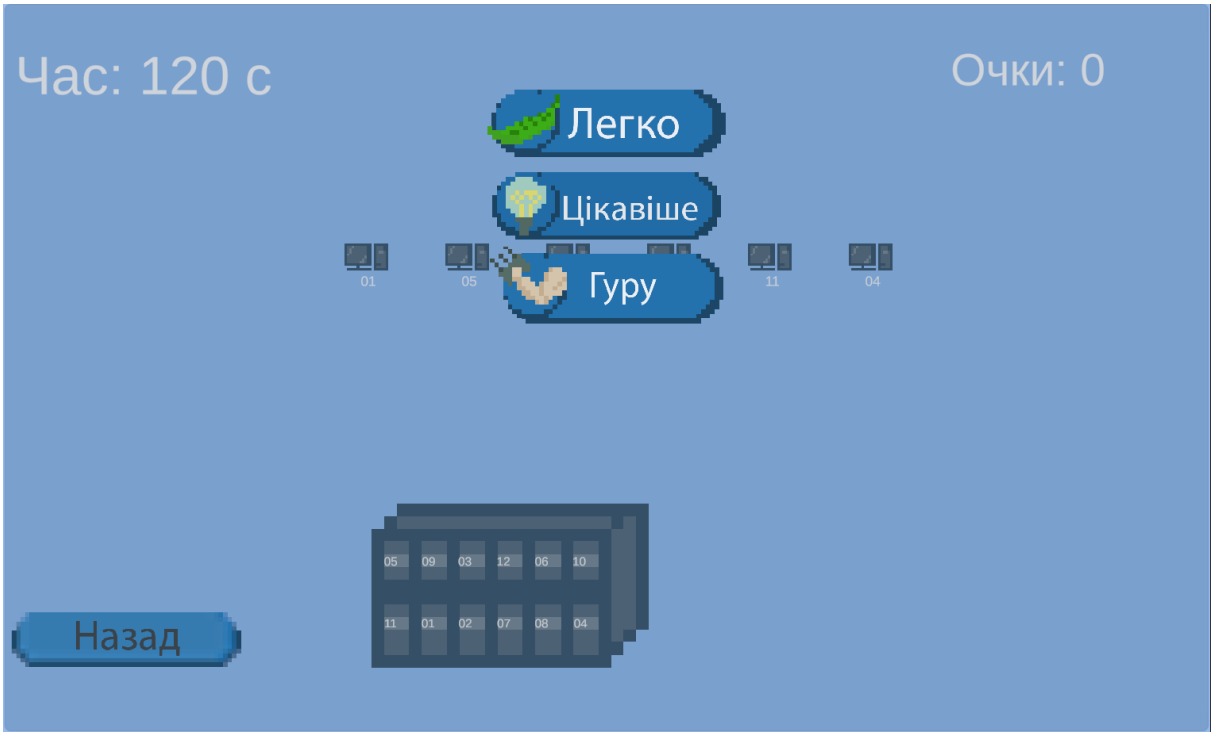


Рисунок 3.7 – Загальний вигляд меню вибору складності

Після вибору рівня складності, ігровий процес автоматично запуститься, як і при натисканні кнопки «Старт» (рис. 3.8)

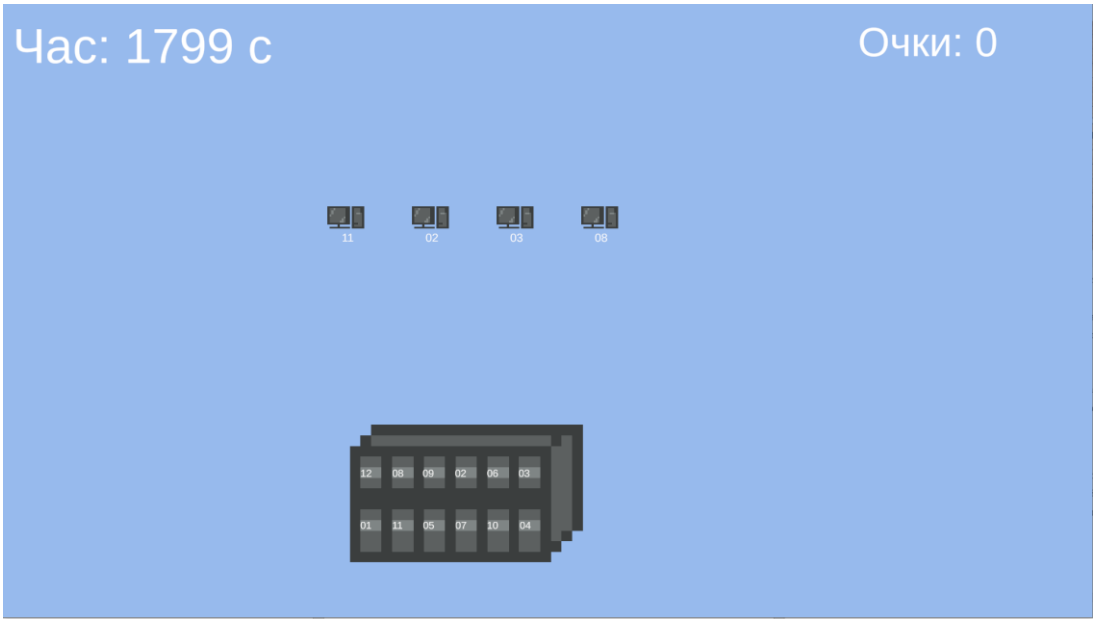


Рисунок 3.8 – Загальний вигляд ігрового рівня

Відлік часу розпочато, проведемо перевірку підключення (рис. 3.9).

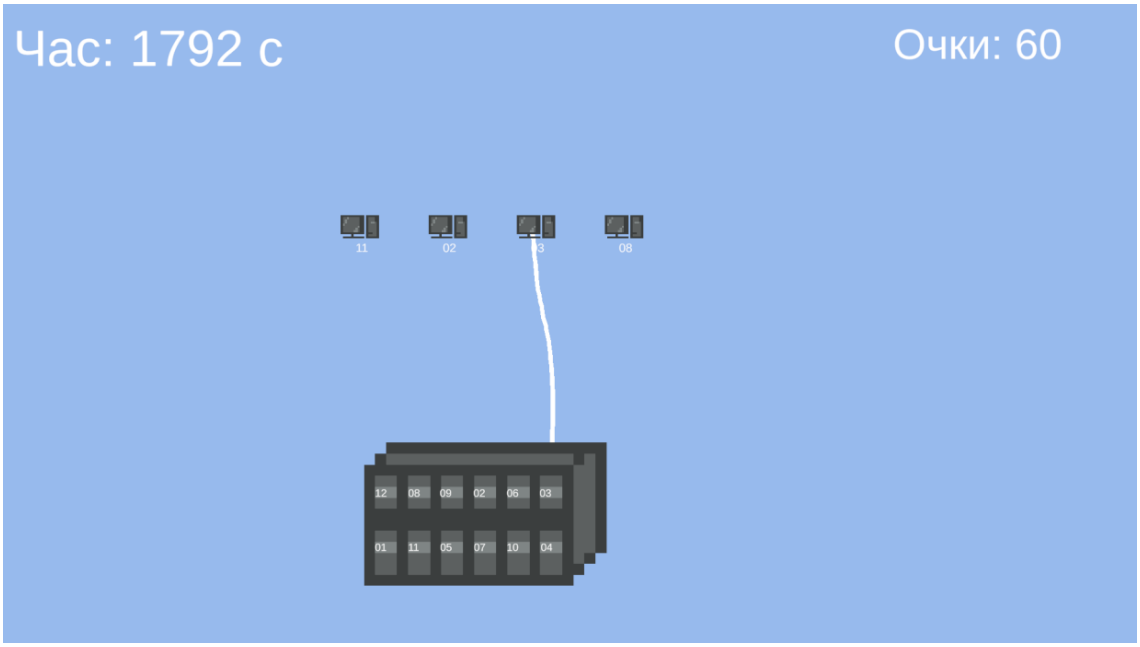


Рисунок 3.9 – Перевірка правильності підключення

Підключення зафіксоване і очки нараховані гравцеві, отже метод підключення правильний [21].

Тепер виконаємо перевірку неправильного підключення, проведемо лінію між девайсом що не відповідає порту (рис. 3.10).

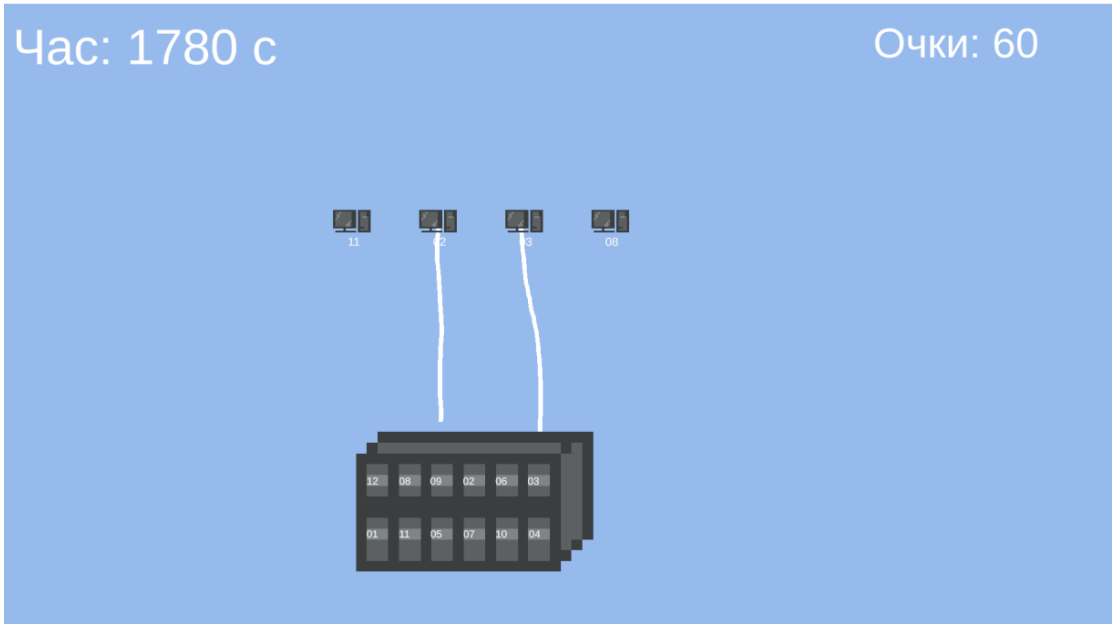


Рисунок 3.10 – Спроба провести неправильний вибір

Алгоритм відхилить з’єднання і очки не будуть нараховані гравцеві (рис. 3.11).

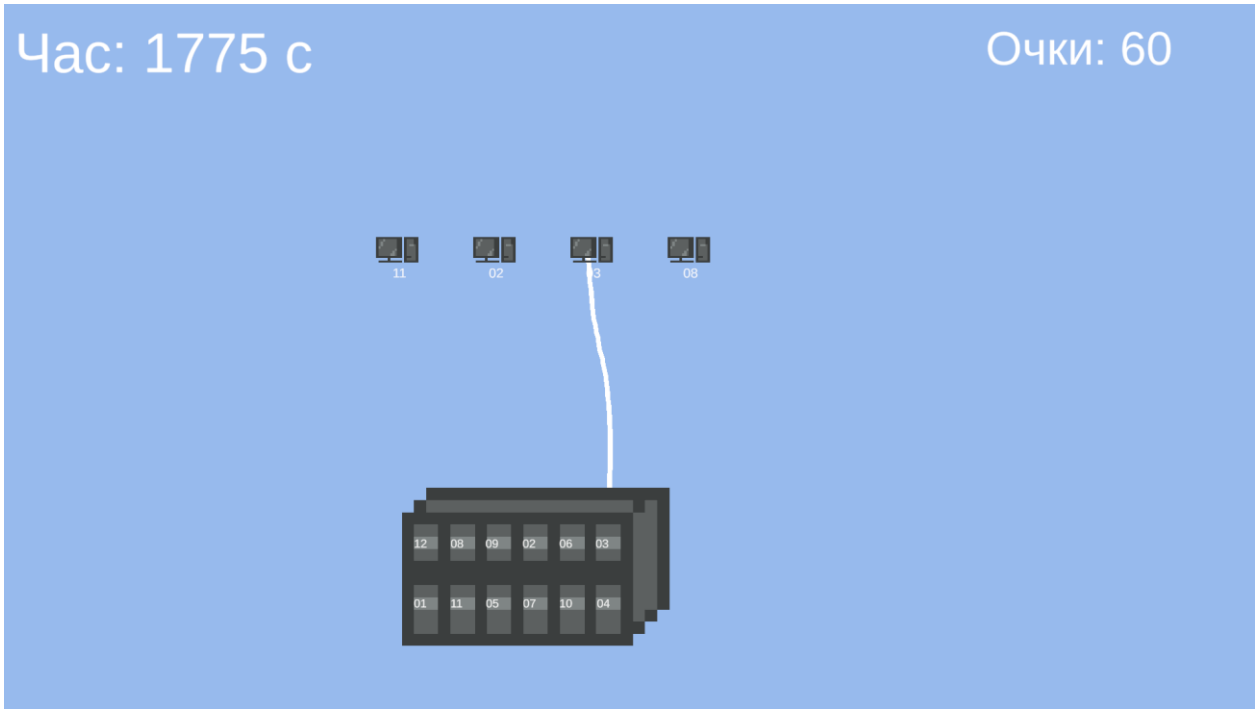


Рисунок 3.11 – Результат неправильного вибору

Перевіримо завершення рівня перемогою (рис. 3.12)

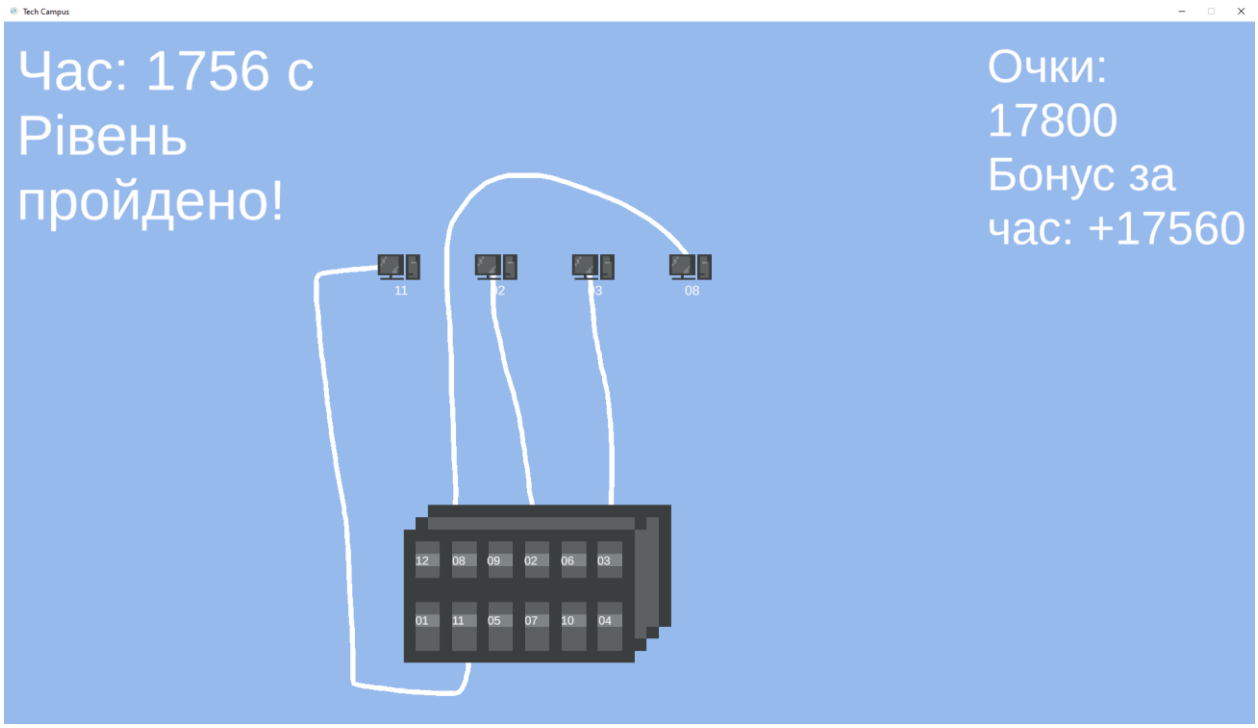


Рисунок 3.12 – Перемога гравця

Виконаємо перевірку на поразку (рис. 3.13).

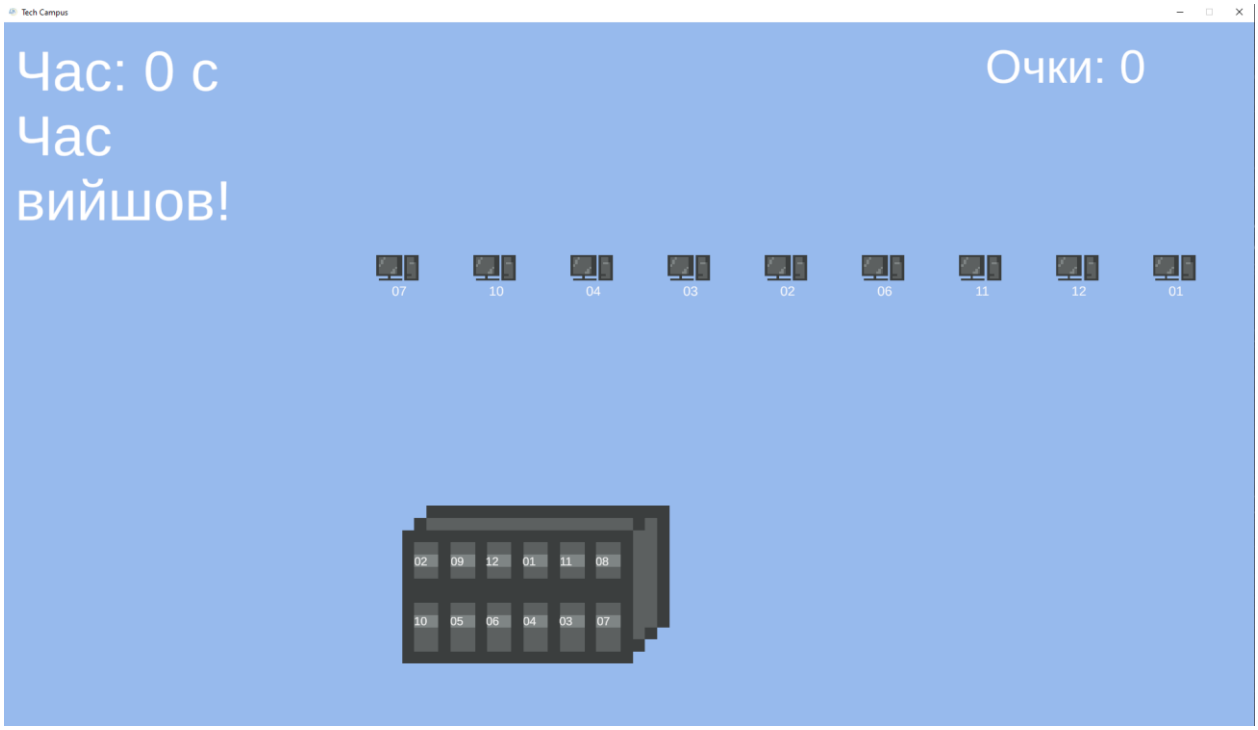


Рисунок 3.12 – Поразка гравця

Згідно з проведеними тестами, комп'ютерна гра «Tech Campus» виконує всі функції, що зазначалися в проектувані програмного додатку [22]

3.4 Висновок

У даному розділі було обгрунтовано вибір технологій реалізації, а саме мови програмування та середовища розробки. Також було описано технологію реалізації алгоритмів гри. Розглянуто програмний код та його основні функції.

Обраними програмними засобами було реалізовано гру «Tech Campus».

Було протестовано гру «Tech Campus».

ВИСНОВКИ

Поставлені перед бакалаврською дипломною роботою задачі виконано в повному обсязі, а саме:

1. Проаналізовано предметну область і визначено актуальність впровадження гейміфікації в навчання основ IP-мереж.
2. Проведено аналіз сучасних гейміфікованих рішень та ігрових застосунків у сфері комп'ютерних мереж і кібербезпеки.
3. Розроблено архітектуру програмного модуля навчальної гри "Tech Campus" з елементами гейміфікації.
4. Сформовано алгоритм функціонування динамічних ігрових сесій, з урахуванням різних рівнів складності, перевірки правильності підключень та оптимізації ігрового процесу.
5. Здійснено програмну реалізацію ігрового модуля у середовищі Unity з використанням C#.
6. Проведено тестування програмного продукту та проведено порівняльний аналіз результатів засвоєння навичок мережевого адміністрування за допомогою гри.
7. Розроблено інструкцію користувача та підготовлено гру до впровадження у навчальний процес.

У першому розділі було розглянуто сучасні підходи до гейміфікації освітніх процесів, проаналізовано приклади програмних продуктів та інструментів для вивчення комп'ютерних мереж. Також досліджено освітній потенціал використання ігор для підвищення мотивації та залученості студентів.

У другому розділі розроблено загальну архітектуру гри "Tech Campus", сформовано алгоритми генерації мережевих підключень, описано механіку взаємодії гравця з ПК та портами, а також виконано моделювання основних сценаріїв ігрового процесу. Представлено діаграми взаємодії компонентів та блок-схеми алгоритмів.

У третьому розділі проаналізовано вибір мови програмування та інструментів розробки, здійснено програмну реалізацію гри у середовищі Unity. Детально описано алгоритми, структуру класів та реалізацію гейміфікаційних механік, а також проведено тестування на функціональність та стійкість. За результатами тестування встановлено, що розроблений модуль працює коректно і відповідає поставленим вимогам.

Мета роботи — підвищення ефективності навчального процесу основ IP-мереж шляхом гейміфікації — досягнута.

Розроблена гра “Tech Campus” має значний потенціал для використання у навчальних закладах, як сучасний інтерактивний інструмент формування та перевірки практичних навичок з мережевого адміністрування, а також може бути основою для подальшого розвитку в напрямку освітніх ігор та симуляторів. В якості апробації результати роботи були представлені на Всеукраїнській науково-технічній конференції підрозділів ВНТУ (24-27 березня 2025 року).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гончарук А., Малініч І. Використання гейміфікації у першому етапі співбесід // Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ 24-27 березня 2025 року. – 2025. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23940.
2. Гейміфікація в освіті: методи, стратегії, результати / О. Б. Гузь, М. А. Шевченко // Освіта та розвиток обдарованої особистості. – 2019. – № 8. – С. 35–41.
3. Deterding S., Dixon D., Khaled R., Nacke L. From game design elements to gamefulness: defining "gamification" // Proceedings of the 15th International Academic MindTrek Conference. – 2011. – P. 9–15.
4. Anderson, C. A., & Dill, K. E. Video games and aggressive thoughts, feelings, and behavior in the laboratory and in life // Journal of Personality and Social Psychology. – 2000. – Vol. 78, No. 4. – P. 772–790.
5. Костюк Г. С., Грабовець О. А. Психологічні аспекти гейміфікації навчання // Психолінгвістика. – 2018. – № 23. – С. 88–94.
6. Hamari J., Koivisto J., Sarsa H. Does gamification work? – A literature review of empirical studies on gamification // 47th Hawaii International Conference on System Sciences. – 2014. – P. 3025–3034.
7. Кузьмінський А. І., Ткаченко О. М. Сучасні тренди розвитку інформаційних технологій в освіті // Інформаційні технології і засоби навчання. – 2020. – № 4. – С. 12–24.
8. Gee, J. P. What Video Games Have to Teach Us About Learning and Literacy // New York: Palgrave Macmillan, 2007. – 256 p.
9. Unity Technologies. Unity User Manual 2023. [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Manual/index.html
10. Палій І. О., Мозгова Н. Г. Моделювання та проектування програмних систем // Вісник НТУУ «КПІ». – 2016. – № 1. – С. 112–120.

11. Lethbridge, T. C., & Laganière, R. Object-Oriented Software Engineering: Practical Software Development using UML and Java. – 3rd ed. – London: McGraw-Hill, 2005. – 624 p.
12. Bobko A., Shmorgun L., Bobko N. Using the Unity game engine for simulation-based training systems // CEUR Workshop Proceedings. – 2019. – Vol. 2386. – P. 378–387.
13. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. – 3rd ed. – Boston: Addison-Wesley, 2004. – 208 p.
14. Nacke, L. E., & Deterding, S. The maturing of gamification research // Computers in Human Behavior. – 2017. – Vol. 71. – P. 450–454.
15. ISO/IEC/IEEE 42010:2011. Systems and software engineering — Architecture description [Стандарт]. – Geneva: ISO, 2011. – 46 p.
16. Unity Technologies. Scripting API Documentation [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/
17. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference. – 8th ed. – Sebastopol: O'Reilly Media, 2022. – 1080 p.
18. Паляниця О. В., Мариненко І. О. Тестування програмних продуктів: теорія і практика // Київ: КНУ, 2018. – 142 с.
19. Koster, R. Theory of Fun for Game Design. – 2nd ed. – Sebastopol: O'Reilly Media, 2013. – 304 p.
20. Бородіна Л. В. Практика розробки і тестування ігрових застосунків на Unity // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2020. – № 37. – С. 57–62.
21. Половко В. М., Калюжний О. В. Методи контролю якості програмного забезпечення // Київ: Ліра-К, 2017. – 320 с.
22. А. А. Яровий, О. В. Сілагін, І. В. Богач. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» Вінниця : ВНТУ, 2023. 54 с.

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Комп'ютерна гра Tech Campus з гейміфікацією навчального процесу

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,16 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Малініч І.П.
(підпис) (прізвище, ініціали, посада)

Здобувач Гончарук _____ Гончарук А.В.
(підпис) (прізвище, ініціали)

Додаток Б

ІНСТРУКЦІЯ КОРИСТУВАЧА

1. Запустити програмний додаток.
2. Відкриється меню гри.
3. Натиснути кнопку «Старт!».
4. Для встановлення з'єднання потрібно з'єднати пк з певною окетою до порту який відповідає цій океті.
5. Гра триває до моменту коли всі ПК будуть мати своє з'єднання, або час буде вичерпано.
6. При натисканні кнопки ESC можливість визвати головне меню.
7. При потребі у вкладці «Складність» головного меню можливо змінити складність рівня.
8. При потребі можна перезапустити рівень кнопкою «Здатися».

Додаток В

ЛІСТИНГ ПРОГРАМНОГО КОДУ

Вміст файлу GameManager.cs

```
using UnityEngine;
using System.Collections.Generic;
using TMPro;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public static GameManager instance;
    public RouterPort[] ports;
    public GameObject pcPrefab;
    public Transform pcsParent;
    public int pcsCount = 6;

    private List<string> usedIPs = new List<string>();

    // --- ОЧКИ ---
    public int score = 0;
    public TextMeshProUGUI scoreText;

    // --- ТАЙМЕР ---
    public float levelTime = 120f; // Тривалість рівня у секундах (наприклад, 2 хв)
    private float timeLeft;
    public TextMeshProUGUI timerText; // Додай ще один TMP у Canvas

    private bool levelCompleted = false;

    void Awake()
    {
        instance = this;
    }

    void Start()
    {
        // --- Ініціалізуємо таймер ---
        timeLeft = levelTime;
        UpdateTimerUI();

        // --- Генерація рівня ---
        List<int> portNumbers = new List<int>();
        for (int i = 1; i <= 12; i++) portNumbers.Add(i);
        for (int i = 0; i < portNumbers.Count; i++)
        {
            int rnd = Random.Range(i, portNumbers.Count);
            int temp = portNumbers[i];
            portNumbers[i] = portNumbers[rnd];
            portNumbers[rnd] = temp;
        }
        for (int i = 0; i < ports.Length && i < portNumbers.Count; i++)
        {
            string portNum = portNumbers[i].ToString("D2");
            ports[i].SetIP(portNum);
            usedIPs.Add(portNum);
        }

        SpawnPCs();
        UpdateScoreUI();
    }

    void Update()
    {
        // --- Оновлення таймера кожен кадр ---
```

```
        if (!levelCompleted)
        {
            timeLeft -= Time.deltaTime;
            UpdateTimerUI();

            if (timeLeft <= 0)
            {
                timeLeft = 0;
                GameOver(false);
            }
        }
    }

    void SpawnPCs()
    {
        List<string> availableIPs = new List<string>(usedIPs);
        for (int i = 0; i < pcsCount; i++)
        {
            int index = Random.Range(0, availableIPs.Count);
            string ip = availableIPs[index];
            availableIPs.RemoveAt(index);

            Vector3 spawnPos = new Vector3(-6f + i * 2.5f, 2.5f, 0f);
            GameObject pcGO = Instantiate(pcPrefab, spawnPos, Quaternion.identity, pcsParent);
            PCUnit pcUnit = pcGO.GetComponent<PCUnit>();
            if (pcUnit != null)
                pcUnit.SetIP(ip);
                Debug.Log($"SpawnPCs: pcsCount={pcsCount}");
        }
    }

    // --- ОЧКИ ---
    public void AddScore(int points)
    {
        score += points;
        UpdateScoreUI();
    }

    public void UpdateScoreUI()
    {
        if (scoreText != null)
            scoreText.text = "Очки: " + score;
    }

    // --- ТАЙМЕР ---
    public void UpdateTimerUI()
    {
        if (timerText != null)
            timerText.text = "Час: " + Mathf.CeilToInt(timeLeft) + " с";
    }

    // --- Перевірка кінця рівня (можна викликати з CableDrawer, коли всі підключення виконані) -
--
    public void CheckLevelComplete()
    {
        // Перевіряємо, чи всі ПК підключені
        int connectedCount = 0;
        foreach (var port in ports)
        {
            if (port.isConnected) connectedCount++;
        }

        if (connectedCount >= pcsCount && !levelCompleted)
        {
            levelCompleted = true;
            GameOver(true);
        }
    }
```

```
    // --- Завершення рівня ---
    void GameOver(bool win)
    {
        if (win)
        {
            // Додаємо бонус за час (наприклад, 10 очок за кожну секунду)
            int bonus = Mathf.CeilToInt(timeLeft) * 10;
            AddScore(bonus);
            if (scoreText != null)
                scoreText.text += "\nБонус за час: +" + bonus;
            if (timerText != null)
                timerText.text += "\nРівень пройдено!";
        }
        else
        {
            if (timerText != null)
                timerText.text += "\nЧас вийшов!";
        }
    }
    public void RestartGameSession()
{
    // 1. Знищити всіх дітей у pcsParent (ПК)
    foreach (Transform child in pcsParent)
        Destroy(child.gameObject);

    // 2. Очистити масив usedIPs
    usedIPs.Clear();

    // 3. Заново згенерувати порти
    List<int> portNumbers = new List<int>();
    for (int i = 1; i <= 12; i++) portNumbers.Add(i);
    for (int i = 0; i < portNumbers.Count; i++)
    {
        int rnd = Random.Range(i, portNumbers.Count);
        int temp = portNumbers[i];
        portNumbers[i] = portNumbers[rnd];
        portNumbers[rnd] = temp;
    }
    for (int i = 0; i < ports.Length && i < portNumbers.Count; i++)
    {
        string portNum = portNumbers[i].ToString("D2");
        ports[i].SetIP(portNum);
        usedIPs.Add(portNum);
    }

    // 4. Скинути очки і таймер
    score = 0;
    UpdateScoreUI();
    timeLeft = levelTime;
    UpdateTimerUI();

    // 5. Скинути levelCompleted
    levelCompleted = false;

    // 6. Заспавнити ПК з новими параметрами
    SpawnPCs();
}

    public void ApplyDifficulty(int newPcsCount, float newLevelTime)
{
    pcsCount = newPcsCount;
    levelTime = newLevelTime;
    Debug.Log($"ApplyDifficulty: pcsCount={pcsCount}, levelTime={levelTime}");
    RestartGameSession();
}
```

Вміст файлу MenuManager.cs

```
using UnityEngine.SceneManagement;
using UnityEngine;

public class MenuManager : MonoBehaviour
{
    public GameObject mainMenuPanel;      // Панель головного меню (затемнення+кнопки)
    public GameObject difficultyPanel;    // Панель вибору складності

    [Header("Параметри для рівнів складності")]
    public int easyPCs = 4;
    public float easyTime = 180f;

    public int mediumPCs = 6;
    public float mediumTime = 120f;

    public int hardPCs = 10;
    public float hardTime = 60f;

    private bool gameIsPaused = true;

    void Start()
    {
        ShowMainMenu();
    }

    void Update()
    {
        // ESC відкриває/закриває меню під час гри
        if (Input.GetKeyDown(KeyCode.Escape) && !mainMenuPanel.activeSelf &&
!difficultyPanel.activeSelf)
        {
            PauseGame();
        }
    }

    // ГОЛОВНЕ МЕНЮ
    public void ShowMainMenu()
    {
        if (mainMenuPanel != null) mainMenuPanel.SetActive(true);
        if (difficultyPanel != null) difficultyPanel.SetActive(false);
        Time.timeScale = 0f;
        gameIsPaused = true;
    }

    public void HideAllMenus()
    {
        if (mainMenuPanel != null) mainMenuPanel.SetActive(false);
        if (difficultyPanel != null) difficultyPanel.SetActive(false);
        Time.timeScale = 1f;
        gameIsPaused = false;
    }

    // КНОПКА "СТАРТ/ПРОДОВЖИТИ"
    public void StartGame()
    {
        HideAllMenus();
        Debug.Log("Кнопка Start натиснута!");
    }

    // КНОПКА "СКЛАДНІСТЬ"
    public void OpenDifficulty()
    {
        if (mainMenuPanel != null) mainMenuPanel.SetActive(false);
        if (difficultyPanel != null) difficultyPanel.SetActive(true);
    }

    // КНОПКА "ЗДАТИСЬ"
    public void Surrender()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    // ВИБІР СКЛАДНОСТІ
public void SetDifficultyEasy()
```

```
{
    GameManager.instance.ApplyDifficulty(easyPCs, easyTime);
    HideAllMenus();
}
public void SetDifficultyMedium()
{
    GameManager.instance.ApplyDifficulty(mediumPCs, mediumTime);
    HideAllMenus();
}
public void SetDifficultyHard()
{
    GameManager.instance.ApplyDifficulty(hardPCs, hardTime);
    HideAllMenus();
}



    // ПАУЗА/ВІДНОВЛЕННЯ
    public void PauseGame()
    {
        if (mainMenuPanel != null) mainMenuPanel.SetActive(true);
        Time.timeScale = 0f;
        gameIsPaused = true;
    }

    public void ResumeGame()
    {
        HideAllMenus();
    }
}
```

Вміст файлу CableDrawler.cs

```
using UnityEngine;
using System.Collections.Generic;

public class CableDrawer : MonoBehaviour
{
    public LineRenderer linePrefab;
    public LayerMask portLayerMask;
    public float minDistance = 0.1f;
    public Vector3 portOffset = Vector3.zero;

    private LineRenderer currentLine;
    private PCUnit selectedPC;
    private bool isDrawing = false;
    private List<Vector3> points = new List<Vector3>();

    public List<List<Vector3>> allCables = new List<List<Vector3>>(); // Зберігає всі прокладені
кабелі

    void Update()
    {
        if (isDrawing && currentLine != null)
        {
            Vector3 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            mousePos.z = 0;
            if (points.Count == 0 || Vector3.Distance(points[points.Count - 1], mousePos) >
minDistance)
            {
                points.Add(mousePos);
                currentLine.positionCount = points.Count;
                currentLine.SetPosition(points.Count - 1, mousePos);
            }
        }

        if (isDrawing && Input.GetMouseButtonUp(0))
        {
            Vector2 mouseWorld = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            RaycastHit2D hit = Physics2D.Raycast(mouseWorld, Vector2.zero, 0.1f, portLayerMask);

            if (hit.collider != null)
```

```
            {
                RouterPort targetPort = hit.collider.GetComponent<RouterPort>();
                if (targetPort != null && selectedPC != null && !targetPort.isConnected)
                {
                    if (selectedPC.ipAddress == targetPort.ipAddress)
                    {
                        // Фінальна точка з урахуванням offset
                        Vector3 portPos = targetPort.transform.position + portOffset;
                        points.Add(portPos);
                        currentLine.positionCount = points.Count;
                        currentLine.SetPosition(points.Count - 1, portPos);

                        selectedPC.isConnected = true;
                        targetPort.isConnected = true;

                        // Додаємо кабель до історії для перевірки перетинів у майбутньому
                        allCables.Add(new List<Vector3>(points));

                        // Перевірка на перетин
                        bool hasIntersection = CableIntersectsExisting(points);

                        // Нарахування очок
                        if (hasIntersection)
                            GameManager.instance.AddScore(60); // Менше очок за перетин
                        else
                            GameManager.instance.AddScore(100); // Більше очок за чисте
підключення
                         // --- стоп рівень ---
                        GameManager.instance.CheckLevelComplete();
                    }
                    else
                    {
                        Destroy(currentLine.gameObject);
                        GameManager.instance.AddScore(0); // За неправильне підключення нічого не
даємо
                    }
                }
                else
                {
                    Destroy(currentLine.gameObject);
                    GameManager.instance.AddScore(0);
                }
            }
            else
            {
                Destroy(currentLine.gameObject);
                GameManager.instance.AddScore(0);
            }

            isDrawing = false;
            currentLine = null;
            selectedPC = null;
            points.Clear();
        }
    }

    public void StartDrawingCable(PCUnit pc)
    {
        if (pc.isConnected) return;
        selectedPC = pc;
        isDrawing = true;
        points.Clear();

        currentLine = Instantiate(linePrefab);
        currentLine.positionCount = 1;
        Vector3 startPos = pc.transform.position;
        currentLine.SetPosition(0, startPos);
        points.Add(startPos);
```

```
        // Рандомний колір кабелю (наприклад, яскравий, не темний)
        Color randomColor = Random.ColorHSV(0f, 1f, 0.7f, 1f, 0.9f, 1f);
        currentLine.startColor = randomColor;
        currentLine.endColor = randomColor;
    }

    public bool CableIntersectsExisting(List<Vector3> newCable)
    {
        // Перевіряє, чи перетинається новий кабель із вже прокладеними
        foreach (var cable in allCables)
        {
            for (int i = 0; i < cable.Count - 1; i++)
            {
                for (int j = 0; j < newCable.Count - 1; j++)
                {
                    if (LinesIntersect(cable[i], cable[i + 1], newCable[j], newCable[j + 1]))
                        return true;
                }
            }
        }
        return false;
    }

    // Допоміжна функція для перетину відрізків
    public static bool LinesIntersect(Vector2 A, Vector2 B, Vector2 C, Vector2 D)
    {
        float denominator = (B.x - A.x) * (D.y - C.y) - (B.y - A.y) * (D.x - C.x);
        if (denominator == 0) return false;

        float numerator1 = (A.y - C.y) * (D.x - C.x) - (A.x - C.x) * (D.y - C.y);
        float numerator2 = (A.y - C.y) * (B.x - A.x) - (A.x - C.x) * (B.y - A.y);

        float r = numerator1 / denominator;
        float s = numerator2 / denominator;

        return (r >= 0 && r <= 1) && (s >= 0 && s <= 1);
    }
}
```

Вміст файлу RouterPort.cs

```
using TMPro;
using UnityEngine;

public class RouterPort : MonoBehaviour
{
    public TextMeshPro ipLabel;   // Прив'язати 3D TextMeshPro (підпис над портом)
    public string ipAddress;      // IP для цього порта
    public bool isConnected = false; // Чи вже зайнятий порт

    // Встановлення IP-адреси і підпису
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }
}
```

Вміст файлу PCUnit.cs

```
using TMPro;
using UnityEngine;

public class PCUnit : MonoBehaviour
{
    public TextMeshPro ipLabel;  // Прив'язати 3D TextMeshPro (підпис над ПК)
```

```
    public string ipAddress;     // IP цієї машини
    public bool isConnected = false; // Чи вже підключено кабелем

    // Встановлення IP-адреси і підпису
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }

    // Клік мишею по ПК — запускаємо малювання кабелю
    private void OnMouseDown()
    {
        if (!isConnected)
        {
            // Передаємо посилання на цей ПК скрипту CableDrawer
            FindObjectOfType<CableDrawer>().StartDrawingCable(this);
        }
    }
}
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«Комп'ютерна гра Tech Campus з гейміфікацією навчального процесу»

Виконав: студент 4 курсу, групи 2КН-21б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)
Гончарук Гончарук А.В.
(прізвище та ініціали)
Керівник: асистент каф. КН
Малініч І. П.
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Рисунок Г.1 – елемент геймплею з гри «Козаки»

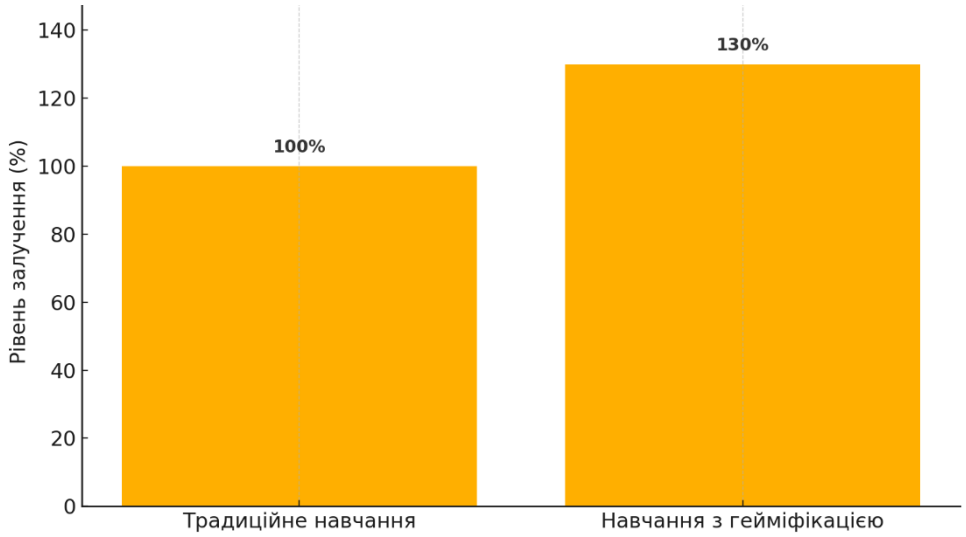


Рисунок Г.2 – рівень залучення студентів у порівнянні традиційного навчання із гейміфікованими застосунками

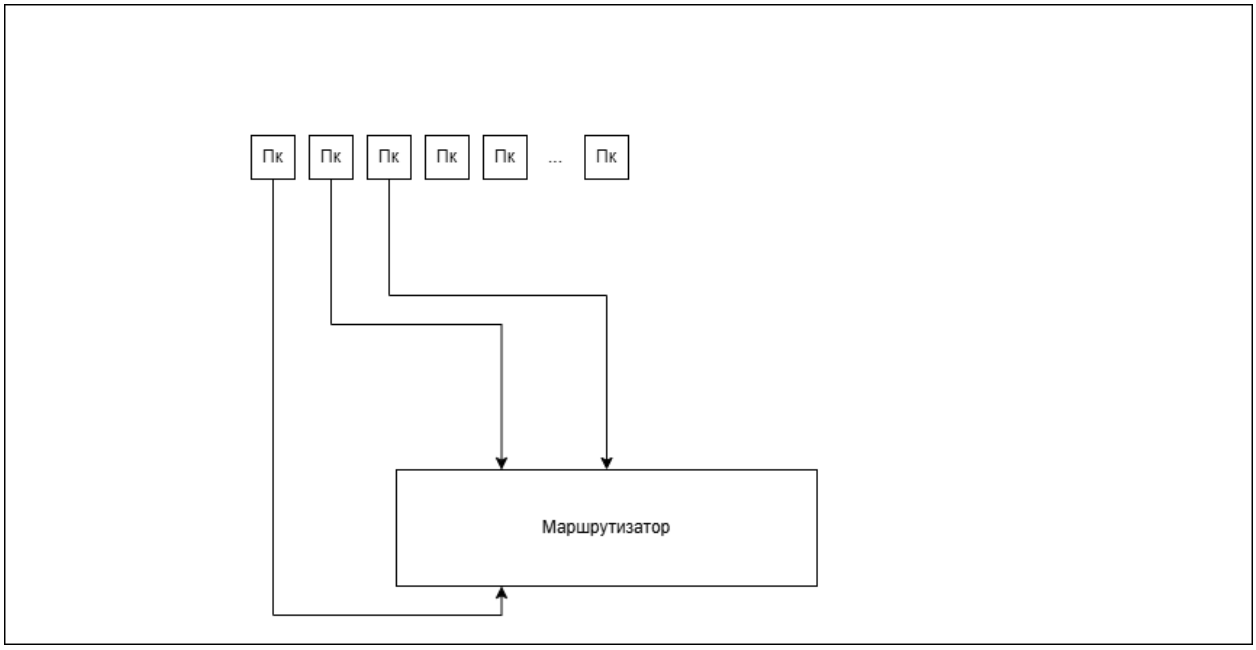


Рисунок Г.3 – Макет рівня гри комп’ютерної гри «Tech Campus»

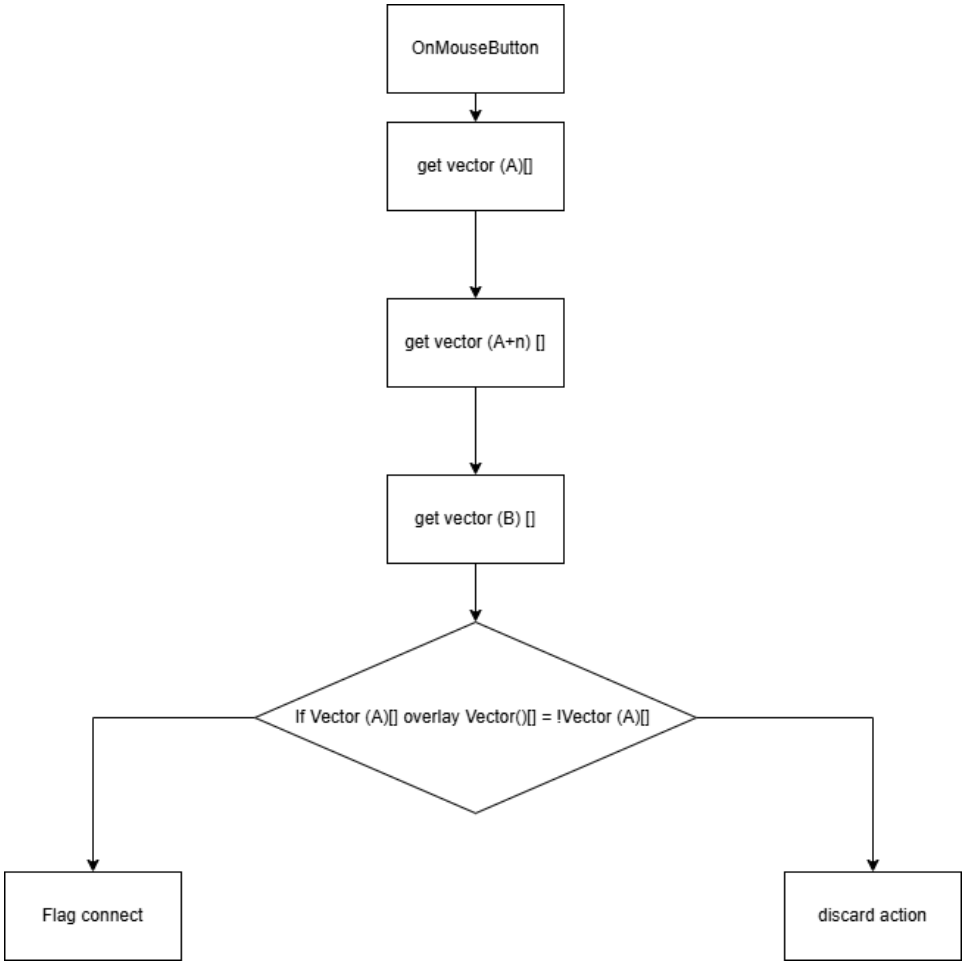


Рисунок Г.4 – Логіка перевірки з’єднань для комп’ютерної гри «Tech Campus»

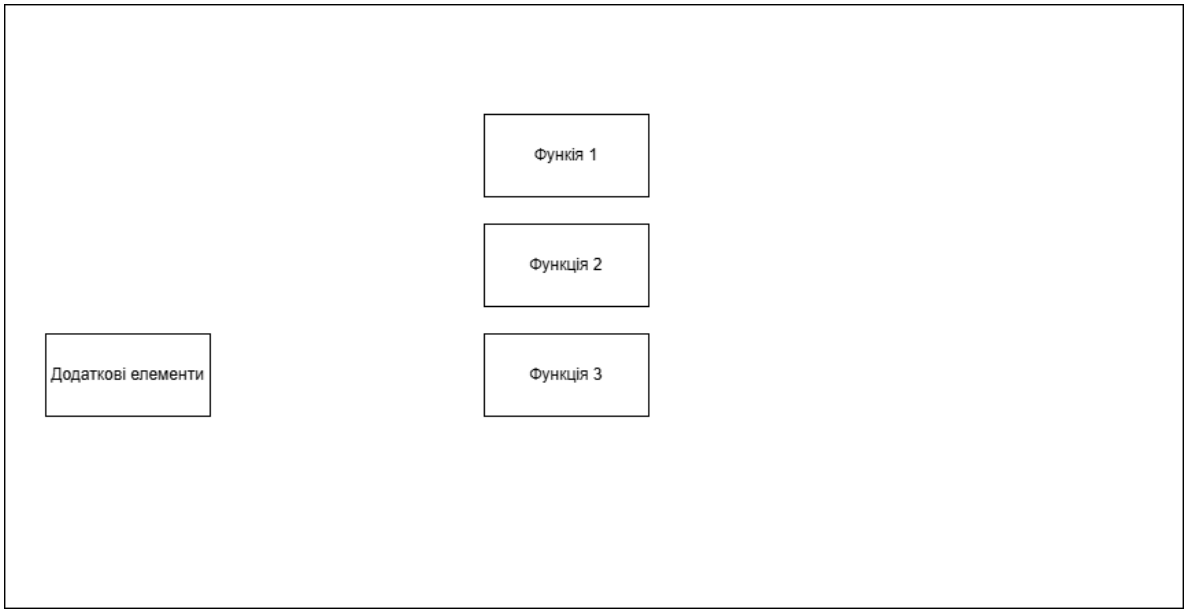


Рисунок Г.5 – Макет меню комп’ютерної гри «Tech Campus»

Рисунок Г.6 – Логіка комп’ютерної гри «Tech Campus»

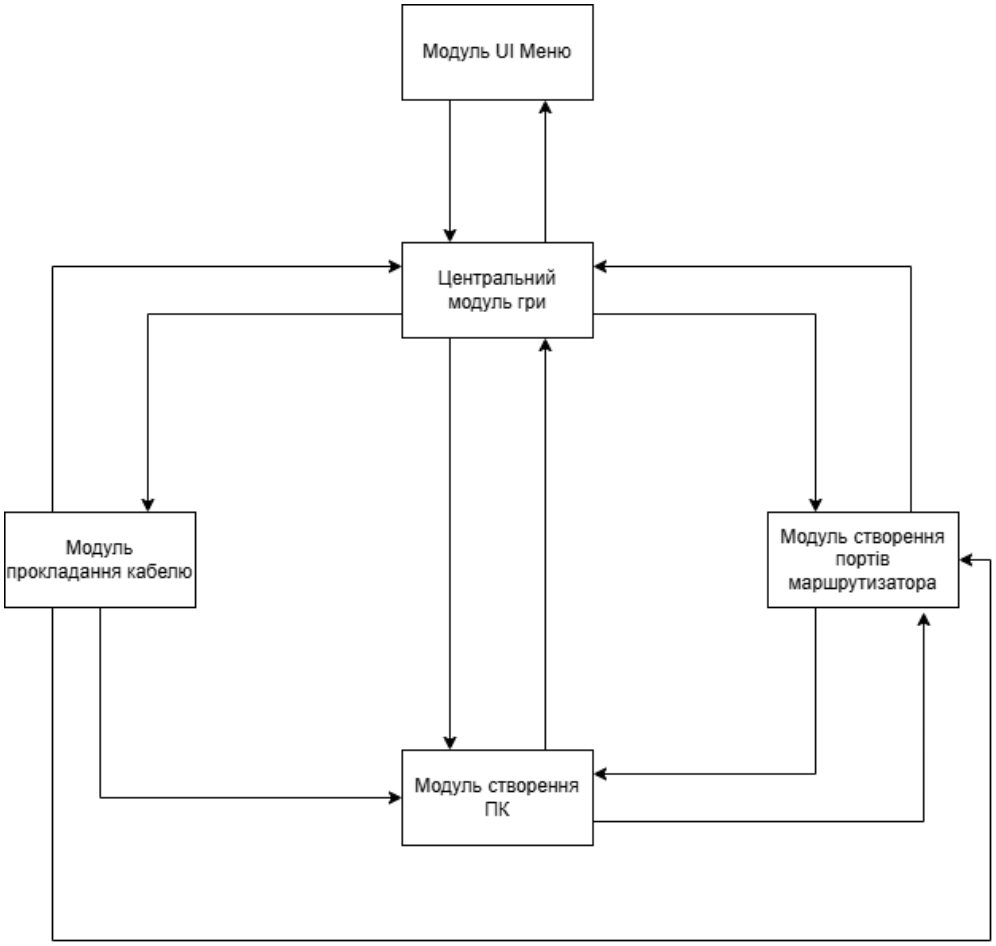


Рисунок Г.7 – Взаємодія всіх модулів комп’ютерної гри «Tech Campus»

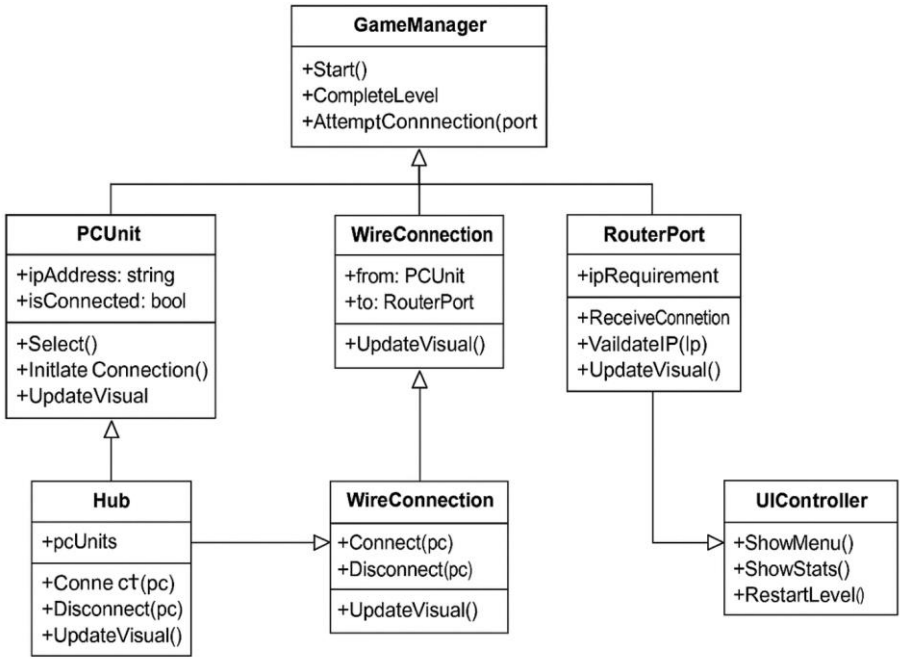
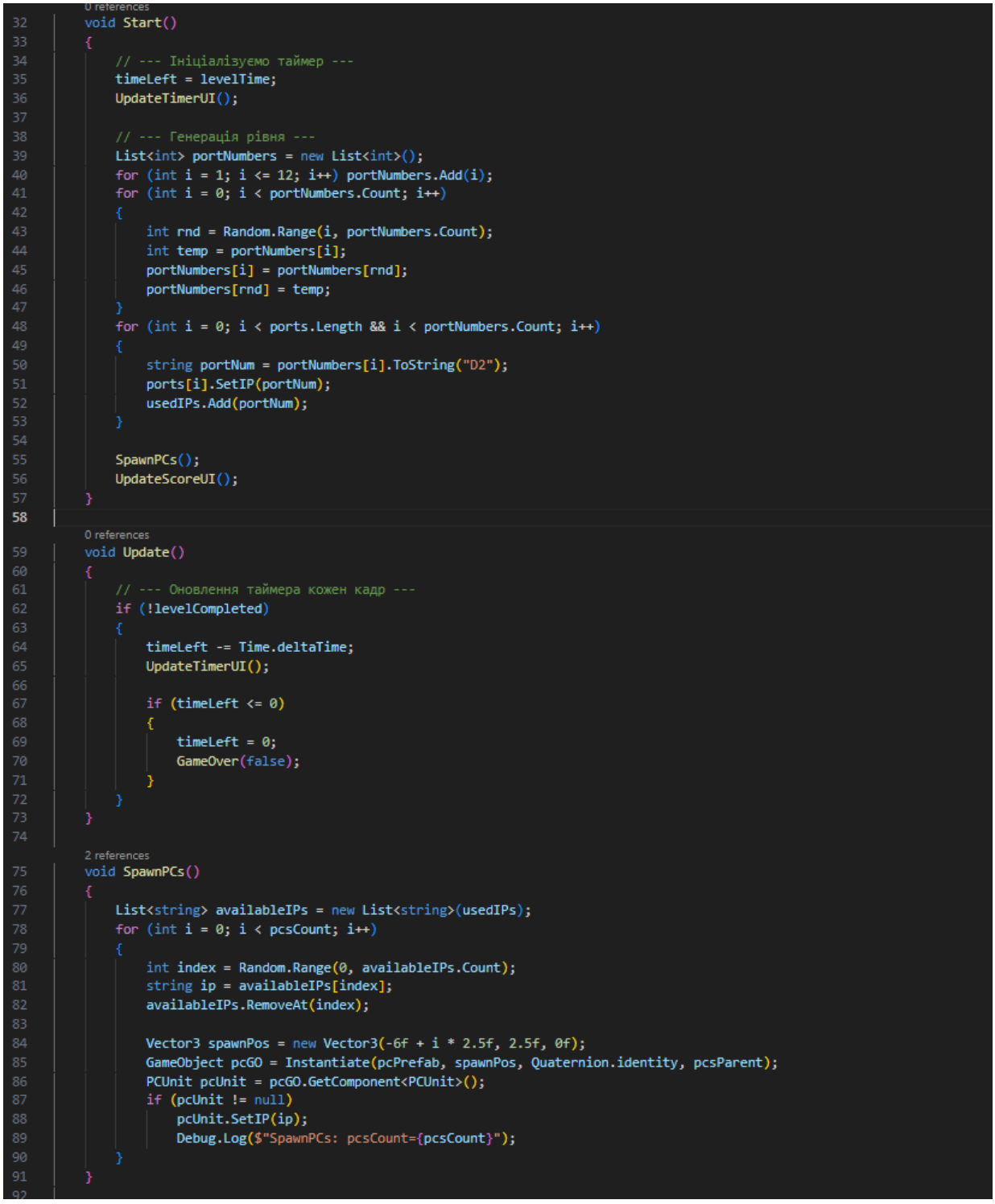


Рисунок Г.8 – UML-діаграма класів програмного модулю комп’ютерної гри «Tech Campus»



```
0 references
void Start()
{
    // --- Ініціалізуємо таймер ---
    timeLeft = levelTime;
    UpdateTimerUI();

    // --- Генерація рівня ---
    List<int> portNumbers = new List<int>();
    for (int i = 1; i <= 12; i++) portNumbers.Add(i);
    for (int i = 0; i < portNumbers.Count; i++)
    {
        int rnd = Random.Range(i, portNumbers.Count);
        int temp = portNumbers[i];
        portNumbers[i] = portNumbers[rnd];
        portNumbers[rnd] = temp;
    }
    for (int i = 0; i < ports.Length && i < portNumbers.Count; i++)
    {
        string portNum = portNumbers[i].ToString("D2");
        ports[i].SetIP(portNum);
        usedIPs.Add(portNum);
    }

    SpawnPCs();
    UpdateScoreUI();
}

0 references
void Update()
{
    // --- Оновлення таймера кожен кадр ---
    if (!levelCompleted)
    {
        timeLeft -= Time.deltaTime;
        UpdateTimerUI();

        if (timeLeft <= 0)
        {
            timeLeft = 0;
            GameOver(false);
        }
    }
}

2 references
void SpawnPCs()
{
    List<string> availableIPs = new List<string>(usedIPs);
    for (int i = 0; i < pcsCount; i++)
    {
        int index = Random.Range(0, availableIPs.Count);
        string ip = availableIPs[index];
        availableIPs.RemoveAt(index);

        Vector3 spawnPos = new Vector3(-6f + i * 2.5f, 2.5f, 0f);
        GameObject pcGO = Instantiate(pcPrefab, spawnPos, Quaternion.identity, pcsParent);
        PCUnit pcUnit = pcGO.GetComponent<PCUnit>();
        if (pcUnit != null)
            pcUnit.SetIP(ip);
            Debug.Log($"SpawnPCs: pcsCount={pcsCount}");
    }
}
```

Рисунок Г.9 – Фрагмент коду компонента «GameManager»

```
0 references
void Update()
{
    // ESC відкриває/закриває меню під час гри
    if (Input.GetKeyDown(KeyCode.Escape) && !mainMenuPanel.activeSelf && !difficultyPanel.activeSelf)
    {
        PauseGame();
    }
}

// ГОЛОВНЕ МЕНЮ
1 reference
public void ShowMainMenu()
{
    if (mainMenuPanel != null) mainMenuPanel.SetActive(true);
    if (difficultyPanel != null) difficultyPanel.SetActive(false);
    Time.timeScale = 0f;
    gameIsPaused = true;
}

5 references
public void HideAllMenus()
{
    if (mainMenuPanel != null) mainMenuPanel.SetActive(false);
    if (difficultyPanel != null) difficultyPanel.SetActive(false);
    Time.timeScale = 1f;
    gameIsPaused = false;
}
```

Рисунок Г.10 – Фрагмент коду компонента «MenuManager»

```
    // Допоміжна функція для перетину відрізків
    1 reference
    public static bool LinesIntersect(Vector2 A, Vector2 B, Vector2 C, Vector2 D)
    {
        float denominator = (B.x - A.x) * (D.y - C.y) - (B.y - A.y) * (D.x - C.x);
        if (denominator == 0) return false;

        float numerator1 = (A.y - C.y) * (D.x - C.x) - (A.x - C.x) * (D.y - C.y);
        float numerator2 = (A.y - C.y) * (B.x - A.x) - (A.x - C.x) * (B.y - A.y);

        float r = numerator1 / denominator;
        float s = numerator2 / denominator;

        return (r >= 0 && r <= 1) && (s >= 0 && s <= 1);
    }
}
```

Рисунок Г.11 – Фрагмент коду компонента «CableDrawer»

```
0 references
public class PCUnit : MonoBehaviour
{
    2 references
    public TextMeshPro ipLabel;  // Прив'язати 3D TextMeshPro (підпис над ПК)
    1 reference
    public string ipAddress;     // IP цієї машини
    1 reference
    public bool isConnected = false; // Чи вже підключено кабелем

    // Встановлення IP-адреси і підпису
    0 references
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }

    // Клік мишею по ПК — запускаємо малювання кабелю
    0 references
    private void OnMouseDown()
    {
        if (!isConnected)
        {
            // Передаємо посилання на цей ПК скрипту CableDrawer
            FindObjectOfType<CableDrawer>().StartDrawingCable(this);
        }
    }
}
```

Рисунок Г.12 – Фрагмент коду компонента «PCUnit»

```
0 references
public class RouterPort : MonoBehaviour
{
    2 references
    public TextMeshPro ipLabel;   // Прив'язати 3D TextMeshPro (підпис над портом)
    1 reference
    public string ipAddress;      // IP для цього порта
    0 references
    public bool isConnected = false; // Чи вже зайнятий порт

    // Встановлення IP-адреси і підпису
    0 references
    public void SetIP(string ip)
    {
        ipAddress = ip;
        if (ipLabel != null)
            ipLabel.text = ip;
    }
}
```

Рисунок Г.13– Фрагмент коду компонента «RouterPort»

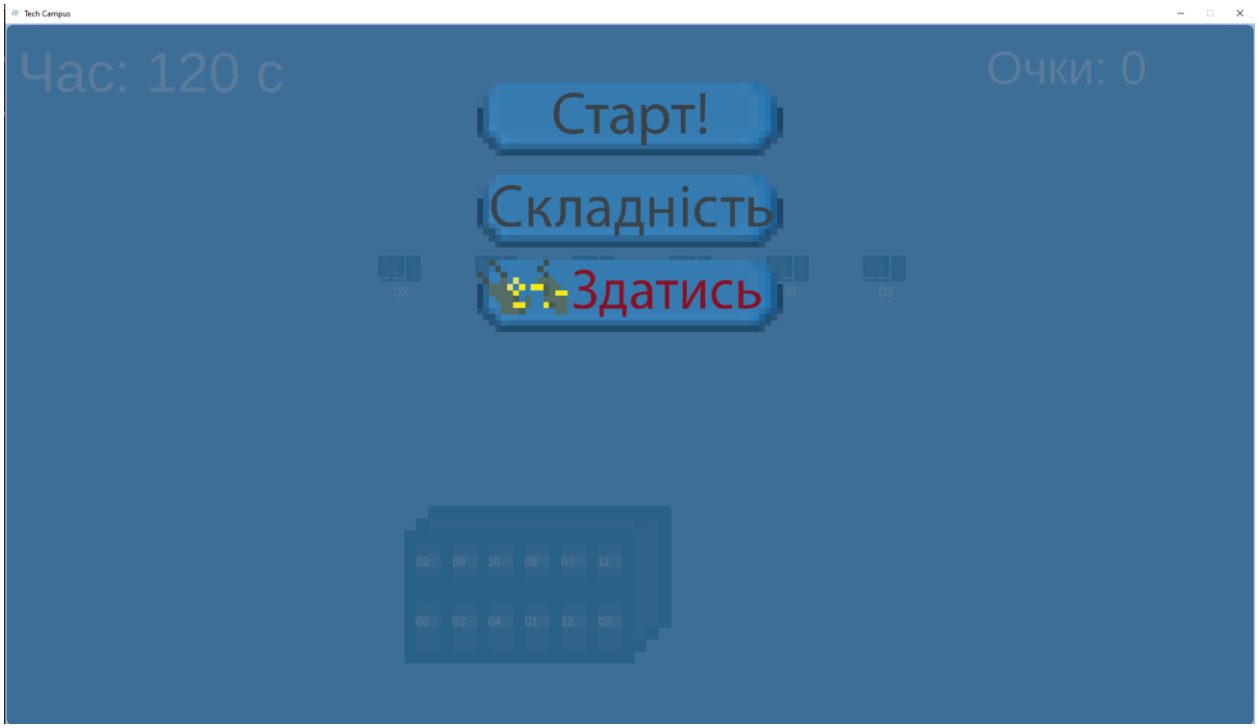


Рисунок Г.14 – Загальний вигляд головного меню меню гри Tech Campus

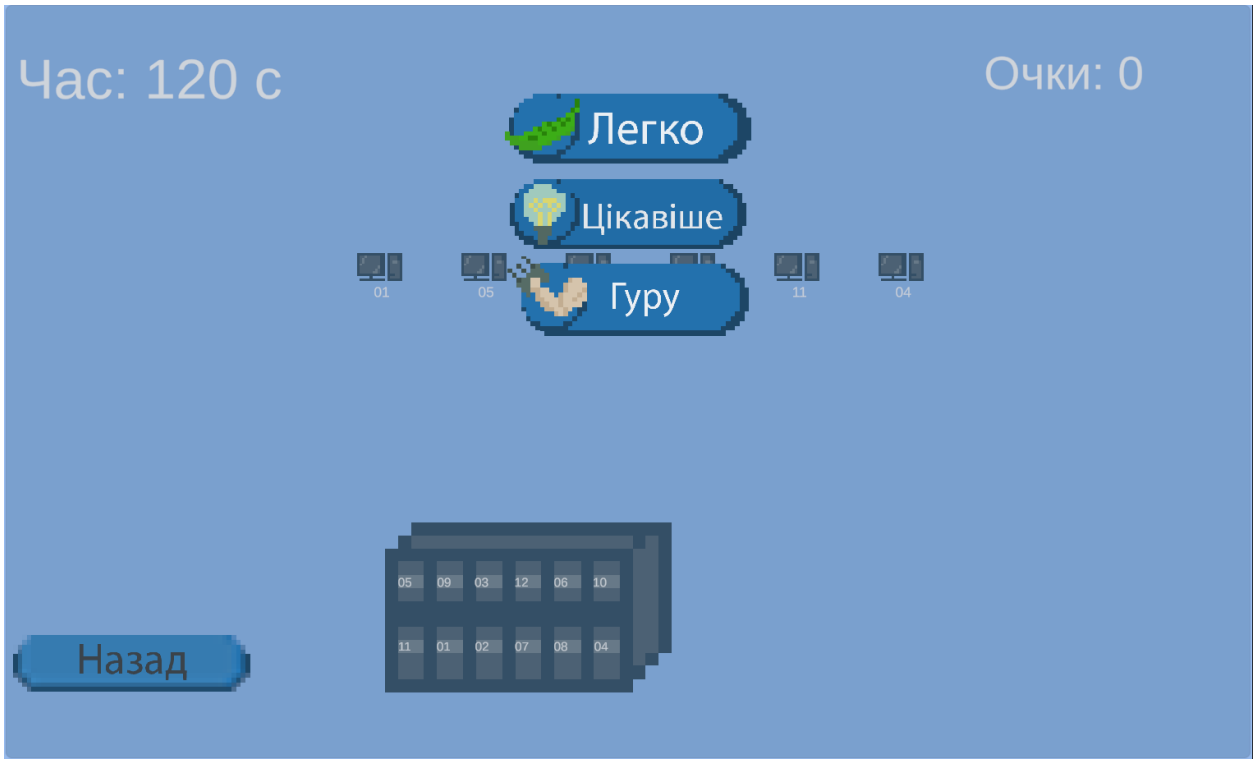


Рисунок Г.15 – Загальний вигляд меню вибору складності

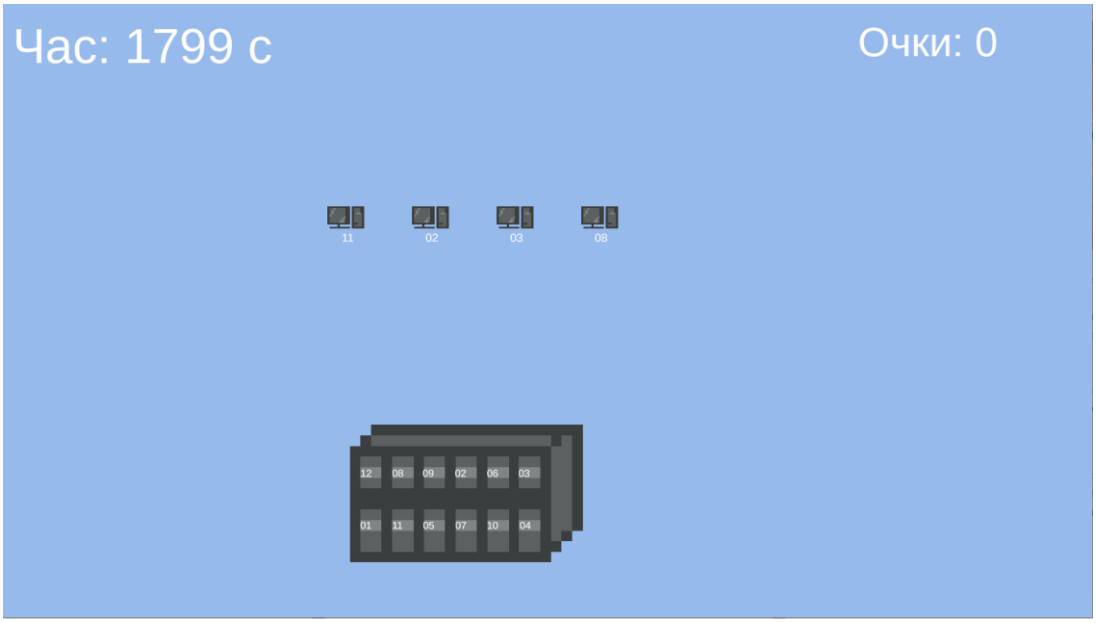


Рисунок Г.16 – Загальний вигляд ігрового рівня

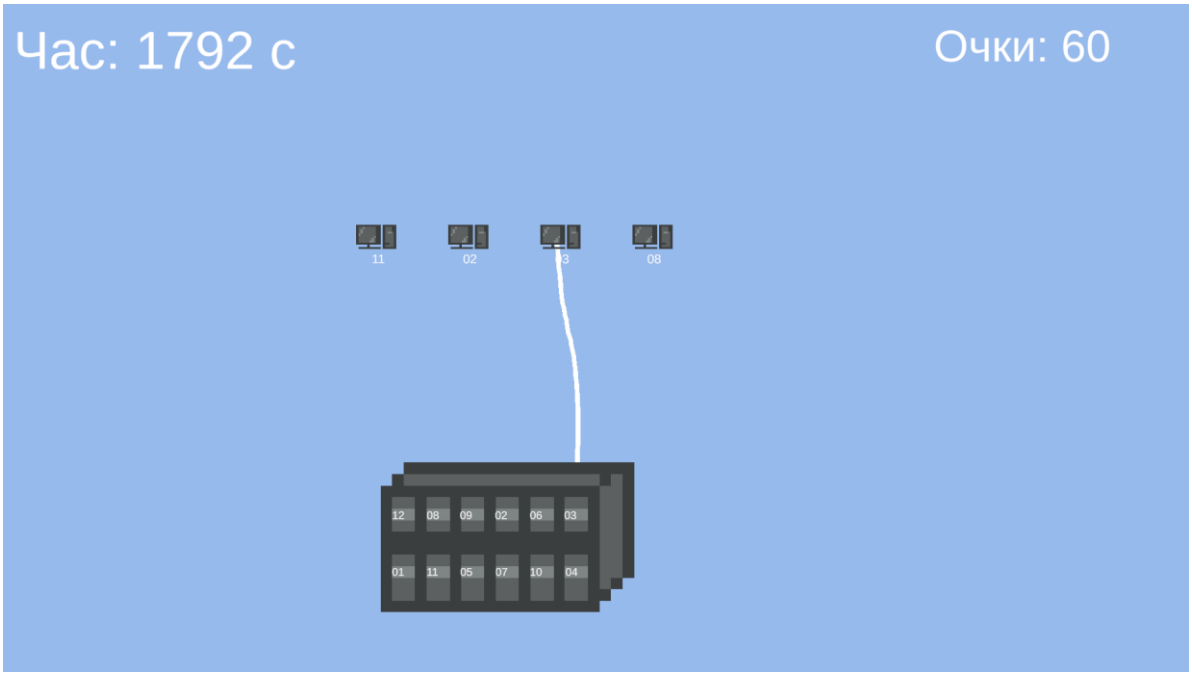


Рисунок Г.17 – Перевірка правильності підключення

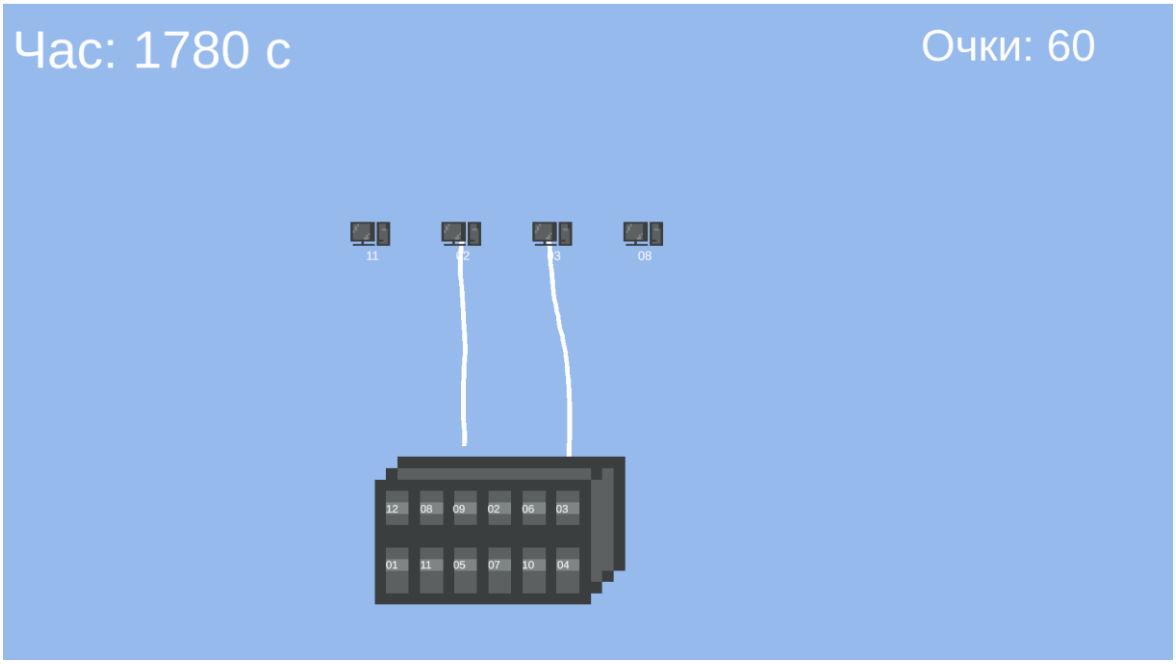


Рисунок Г.18 – Спроба провести неправильний вибір

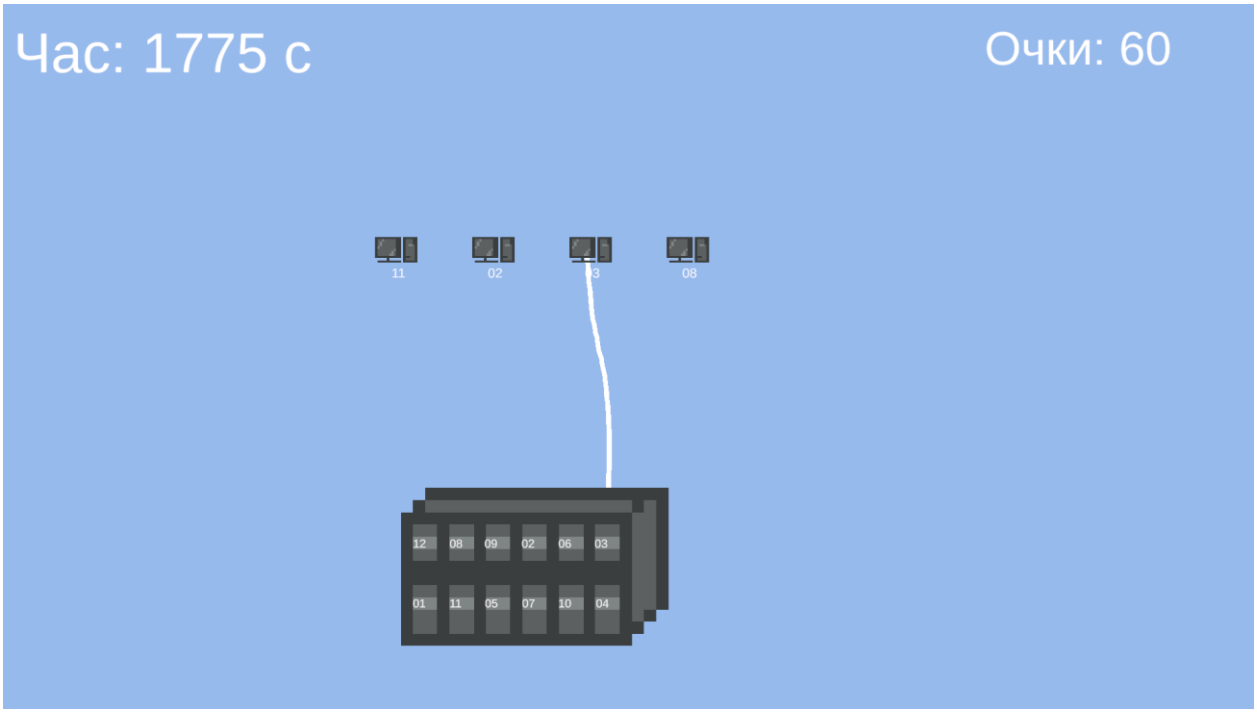


Рисунок Г.19 – Результат неправильного вибору

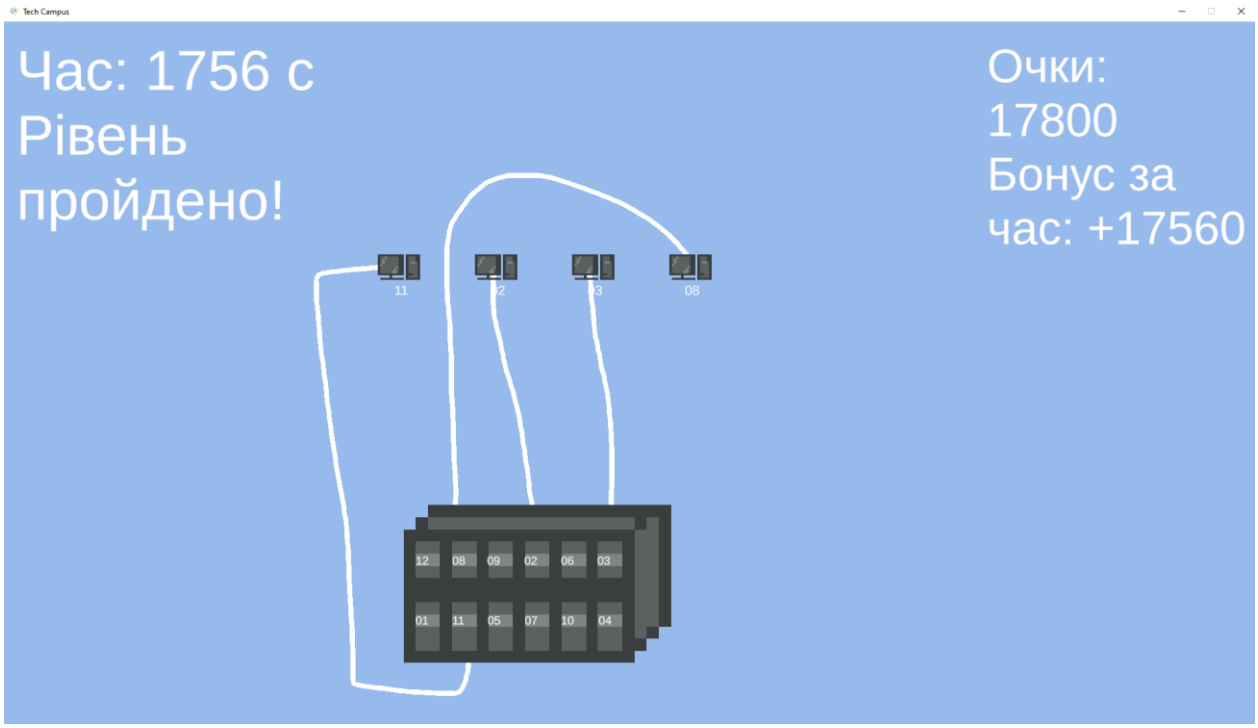


Рисунок Г.20 – Перемога гравця

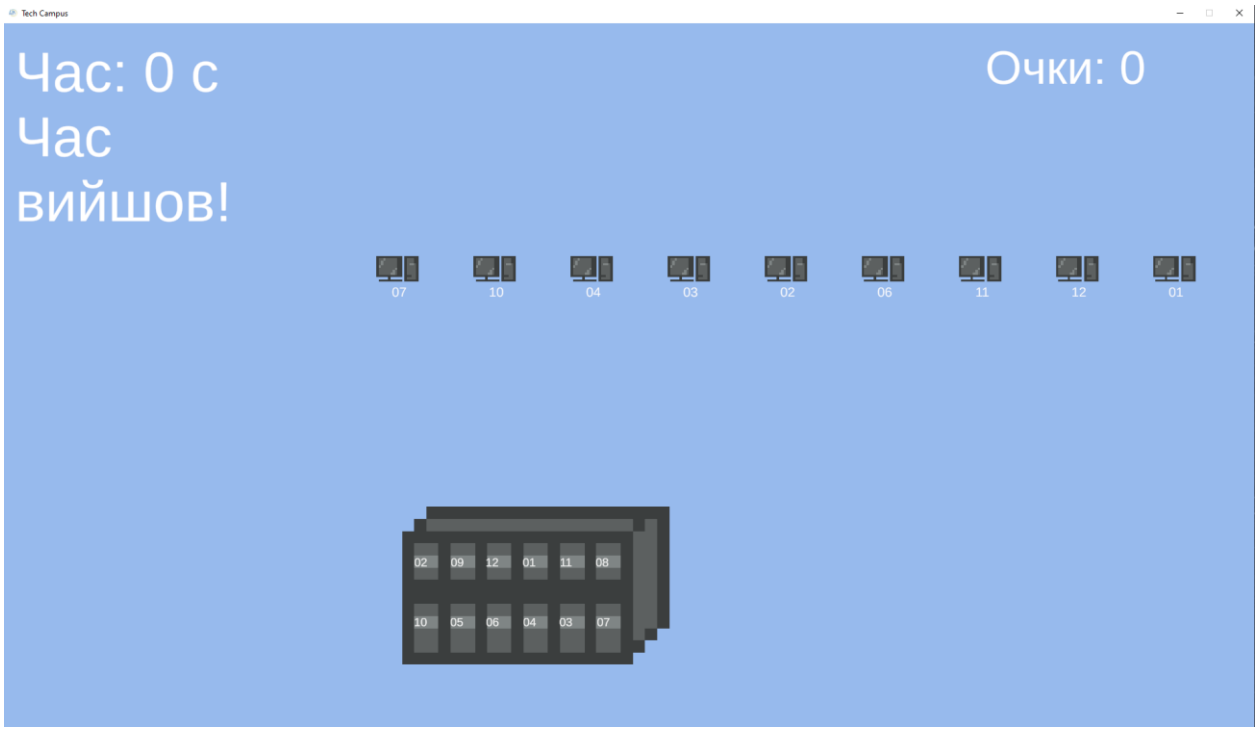


Рисунок Г.21 – Поразка гравця