Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Ігровий застосунок у жанрі детективу на рушії Godot

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 2201118.01.23.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Дмитро ЮЗВАК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІПЗ

Л. П. Бедратюк

02 01 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Юзваку Дмитру Олександровичу

Прізвище, ім'я, по батькові студента

1. Тема роботи Ігровий застосунок у жанрі детективу на рушії Godot

Керівник роботи Яшина Оксана Миколаївна, канд. техн. наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Строк подання студентом роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проєктування програмного забезпечення, програмна реалізація та тестування застосунку

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Три креслення

1. Функціональна діаграма
2. Модульна декомпозиція
3. Діаграма сутність-зв'язок

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., доцент	05.05.26	25.05.26
Антиплагіат	Форкун Ю. В., доцент	05.05.26	25.05.26

7. Дата видачі завдання «02» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2025	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3 Проектування програмного забезпечення	21.02 – 20.03 2026	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.06.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Дмитро ЮЗВАК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи «Ігровий застосунок у жанрі детективу на рушії Godot».

Автор роботи: Юзвак Дмитро Олександрович.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 69 с., 34 рис., 2 табл., 5 дод., 43 джерела.

Графічна частина: 3 креслення ф. А3.

GODOT ENGINE, GDSCRIPT, JSON, ДЕТЕКТИВНИЙ КВЕСТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІГРОВИЙ ЗАСТОСУНОК, ГРАФІЧНИЙ ІНТЕРФЕЙС, ІГРОВІ МЕХАНІКИ, ЖИТТЄВИЙ ЦИКЛ ПЗ, ТЕСТУВАННЯ ПЗ.

Метою кваліфікаційної роботи є проєктування та програмна реалізація ігрового застосунку в жанрі детективного квесту, що базується на сучасних принципах інтерактивного наративу та механіках розслідування.

У межах дослідження проведено аналіз специфіки жанру та існуючих технологічних рішень, визначено комплекс функціональних вимог до системи, а також розроблено детальну архітектурну модель застосунку на основі вузлової структури та подієво-орієнтованого підходу.

Для технічного втілення проєкту використано ігровий рушій Godot та мову програмування GDScript. Це дозволило впровадити ключові модулі системи: механіку пошуку доказів, динамічний інвентар та розгалужену систему діалогів із логічною перевіркою фактів.

Результатом роботи є стабільний ігровий застосунок, що пропонує цілісний досвід розслідування. На завершальному етапі було проведено комплексне тестування, за результатами якого підтверджено технічну справність та повну функціональну готовність розробленого продукту до використання.

26.05.2026
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2201118.01.23.ПЗ	Пояснювальна записка	69		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ. 2201118.01.23.Е8	Функціональна діаграма	1		
5	А3	КвРІПЗ. 2201118.01.23.Е8	Модульна декомпозиція	1		
6	А3	КвРІПЗ. 2201118.01.23.Е8	Діаграма сутність-	1		
			зв'язок			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2201118.01.23.ПЗ		
Виконав		Юзвак Д. О.		25.05	Ігровий застосунок у жанрі детективу на рушії Godot	Літ.	Арк. 3 / Аркушів 69
Керівник		Яшина О. М.		25.05			
Н. контр.		Форкун Ю. В.		25.05	Відомість документів	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		26.05			

ЗМІСТ

ВСТУП

На сьогоднішній день індустрія відеоігор є найбільш швидкозростаючою індустрією розваг у світі, з доходом близько 200 мільярдів доларів США [1]. Жанр детективу є одним з найбільш старіших та найбільш популярніших жанрів відеоігор. Популярність детективів зумовлена глибоким зануренням у процес розслідування та тим, що випробовує спостережливість та логічні навички гравця. Створення якісного продукту у цьому жанрі вимагає не лише цікавої та заплутаної історії, але й хорошого розуміння ігрових механік та здатності їх вміло поєднати. В Україні цей жанр ігор достатньо розвинутий, від простих ігор «point-and-click» до складних продуктів, по типу серії ігор про Шерлока Холмса.

Сучасні тенденції розвитку детективних відеоігор демонструють чіткий вектор на відмову від жорстко лінійних сюжетів на користь глибокого нелінійного наративу та високої варіативності. Сучасний користувач очікує інтерактивності нового рівня, де його аналітичні рішення, повнота зібраних доказів та послідовність ведення допитів безпосередньо формують унікальний шлях розслідування і впливають на кінцевий фінал історії. Крім того, в індустрії спостерігається активне ускладнення внутрішньої логіки таких проєктів: для управління розгалуженими діалоговими деревами та великими масивами текстової інформації розробники все частіше переходять до гнучких інженерних рішень, зокрема до використання систем, керованих даними.

Дана кваліфікаційна робота присвячена розробці відеогри у жанрі детектив на базі ігрового рушія Godot. У грі реалізовано ключові механіки, притаманні жанру: пошук та збір доказів, система діалогів з персонажами, інвентар для зберігання знайдених предметів, а також логічні перевірки для розкриття справи. Використання рушія Godot дозволяє забезпечити високу продуктивність та кросплатформеність проєкту завдяки його вузловій архітектурі та гнучкій системі сценаріїв. Головною метою розробки є створення цілісного та захопливого ігрового досвіду, який дозволить користувачеві повною мірою зануритися у процес розслідування.

Метою кваліфікаційної роботи є проєктування та програмна реалізація ігрового застосунку в жанрі детективного квесту, що базується на сучасних принципах інтерактивного наративу та механіках розслідування.

Згідно з метою, основними задачами кваліфікаційної роботи є:

– проведення змістовного аналізу предметної області детективних ігор, дослідження їх структурних та функціональних особливостей;

– аналіз наявного програмно-технічного забезпечення для розробки ігор та обґрунтування вибору рушія Godot;

– визначення функціональних та нефункціональних вимог до програмного забезпечення;

– проєктування та розробка архітектури ігрового застосунку;

– програмна реалізація ігрових механік (система пошуку доказів, діалоги, інвентар);

– тестування та верифікація розробленого продукту на відповідність поставленим вимогам.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1.Змістовий аналіз предметної області, її структурних та функціональних особливостей

У сучасному світі відеоігри перетворилися на одну з найвпливовіших форм інтерактивного мистецтва та цифрових розваг. Завдяки розвитку технологій та ігрових рушіїв, розробники отримали можливість створювати глибокі наративні проєкти, де гравець є безпосереднім учасником подій. Особливе місце серед ігрових жанрів посідає детектив, який поєднує в собі елементи логічного аналізу, дослідження навколишнього середовища та складну систему взаємодії з персонажами. У таких іграх головним викликом для гравця стає не швидкість реакції, а здатність збирати докази, зіставляти факти та робити правильні висновки для розкриття справи.

Детективний жанр приваблює аудиторію своєю інтелектуальною глибиною, атмосферою та можливістю занурення у розслідування. Популярність цього напрямку зумовлена запитом гравців на якісний сюжет та нелінійність, де кожен знайдений предмет або обрана репліка в діалозі може змінити хід історії. Саме тому розробка ігрових проєктів у цьому жанрі є актуальним та перспективним завданням у сфері сучасної ігрової індустрії [2].

Популярність детективних ігор зумовлена їхньою здатністю забезпечити інтелектуальну стимуляцію та високий рівень занурення в ігровий процес. Створення проєкту в цьому жанрі є перспективним напрямом, оскільки попит на якісні наративні ігри з продуманими механіками розслідування постійно зростає. Використання сучасних інструментів, таких як ігровий рушій Godot, дозволяє ефективно реалізувати складні ігрові сценарії, забезпечуючи при цьому високу продуктивність та кросплатформеність.

Однак успішна розробка відеогри – це не лише створення візуального контенту чи написання сценарію. Це комплексний процес програмної інженерії, який починається з глибокого аналізу предметної області та завершується технічною реалізацією взаємопов’язаних систем [3]. Для детективної гри

критично важливою є стабільна робота логічних тригерів, систем діалогів та обробки стану ігрового світу. Саме цілісний підхід до розробки програмного забезпечення дозволяє створити продукт, який буде стабільним, зручним для користувача та придатним до подальшого масштабування чи розширення сюжетного контенту.

Дана кваліфікаційна робота присвячена розробці ігрового застосунку у жанрі детектив. У межах проєкту реалізуються основні механіки розслідування: система пошуку та аналізу доказів, багаторівневі діалоги з ігровими персонажами, інвентар предметів та механізми перевірки логічних гіпотез гравця. Програмна реалізація здійснюється на базі рушія Godot, що дозволяє гнучко використовувати вузлову структуру та мову GDScript.

Головний акцент роботи зосереджений на повному циклі створення застосунку: від детального вивчення особливостей детективного жанру та формування технічних вимог до безпосереднього написання програмного коду та тестування функціоналу.

Процес розробки включає аналіз функціональних та нефункціональних вимог, обґрунтування вибору технологічного стека, побудову моделей взаємодії компонентів та програмну реалізацію ключових підсистем гри. Це дозволяє отримати завершений ігровий продукт, де технічна стабільність поєднується з цікавим детективним сюжетом.

Важливою особливістю моделювання цієї предметної області є необхідність формалізації складних ментальних процесів розслідування у вигляді чітких алгоритмічних структур. На відміну від динамічних ігор, де домінує фізична взаємодія об'єктів, архітектура детективного квесту базується на управлінні інформаційними станами, причинно-наслідковими зв'язками та логічними обмеженнями. Це вимагає від розробника створення гнучкої системи збереження даних, яка б безпомилково фіксувала поточний прогрес гравця, статус виявлених зачіпок та етапи проходження діалогових гілок. Таким чином, детективна гра розглядається не просто як розважальний додаток, а як комплексна інформаційна система, що оперує великими масивами взаємопов’язаного контенту.

1.2.Аналіз наявного програмно-технічного забезпечення предметної області

Жанр детективу є одним із найбільш інтелектуальних та наративно насичених напрямів у відеоігровій індустрії. З моменту появи перших текстових квестів цей жанр пройшов еволюцію від простих механіків «point-and-click» до складних психологічних симуляторів розслідування. Сьогодні детективні ігри пропонують гравцям широкий спектр підходів до збору доказів, ведення допитів та побудови логічних висновків, що робить цей жанр ідеальним для дослідження методів інтерактивного сторітелінгу.

Аналізуючи сучасні детективні проєкти, було виокремлено ключові аспекти, які необхідно врахувати під час розробки: глибина наративу, система управління доказами, розгалуженість діалогів та візуальна атмосфера. Особливу увагу приділено тому, як ігрові механіки допомагають гравцеві відчути себе справжнім дослідником, а не просто спостерігачем.

У результаті аналізу було сформовано перелік функцій та дизайнерських рішень, які дозволять створити якісну 2D-гру на рушії Godot, що поєднує класичні традиції жанру з сучасними підходами до геймдизайну. Для дослідження було обрано чотири знакових проєкти, які мають унікальні системи розслідування:

- Sherlock Holmes: Crimes and Punishments;
- L.A. Noire;
- The Wolf Among Us;
- Disco Elysium.

Детальний огляд цих проєктів проведено з метою вивчення досвіду провідних розробників та подальшого використання їхніх ідей при проектуванні власного програмного забезпечення. Порівняльний аналіз функціональних особливостей, переваг та недоліків зазначених ігор дозволив логічно обґрунтувати вибір механік для реалізації, що сприятиме відповідності розроблюваного продукту сучасним запитам ринку ігрового ПЗ. На основі отриманих результатів порівняння було чітко визначено оптимальний

функціональний мінімум, необхідний для створення конкурентоспроможного та збалансованого ігрового процесу.

Варто зауважити, що хоча обрані аналоги здебільшого є масштабними високобюджетними розробками зі складною тривимірною графікою, їхні фундаментальні геймдизайнерські патерни ідеально піддаються абстрагуванню та масштабуванню. Головний фокус цього порівняння зосереджений не на візуальній чи технічній складовій цих проєктів, а на їхньому логічному ядрі: способах організації систем збору доказів, принципах побудови діалогових дерев та методах залучення користувача до розслідування. Адаптація та трансформація цих комплексних механік під формат двовимірного середовища із застосуванням керованої даними архітектури дозволить успішно інтегрувати найкращі світові практики жанру у власний оптимізований застосунок.

Першою грою для дослідження буде Sherlock Holmes: Crimes and Punishments (Рисунок 1.1).

Рисунок 1.1 – Обкладинка гри «Sherlock Holmes: Crimes and Punishments»

Гра від компанії Frogwares демонструє одну з найкращих систем дедукції (Рисунок 1.2). Головною особливістю є дедуктивна мапа, де гравець власноруч з’єднує знайдені факти, що може призвести як до правильного, так і до помилкового звинувачення.

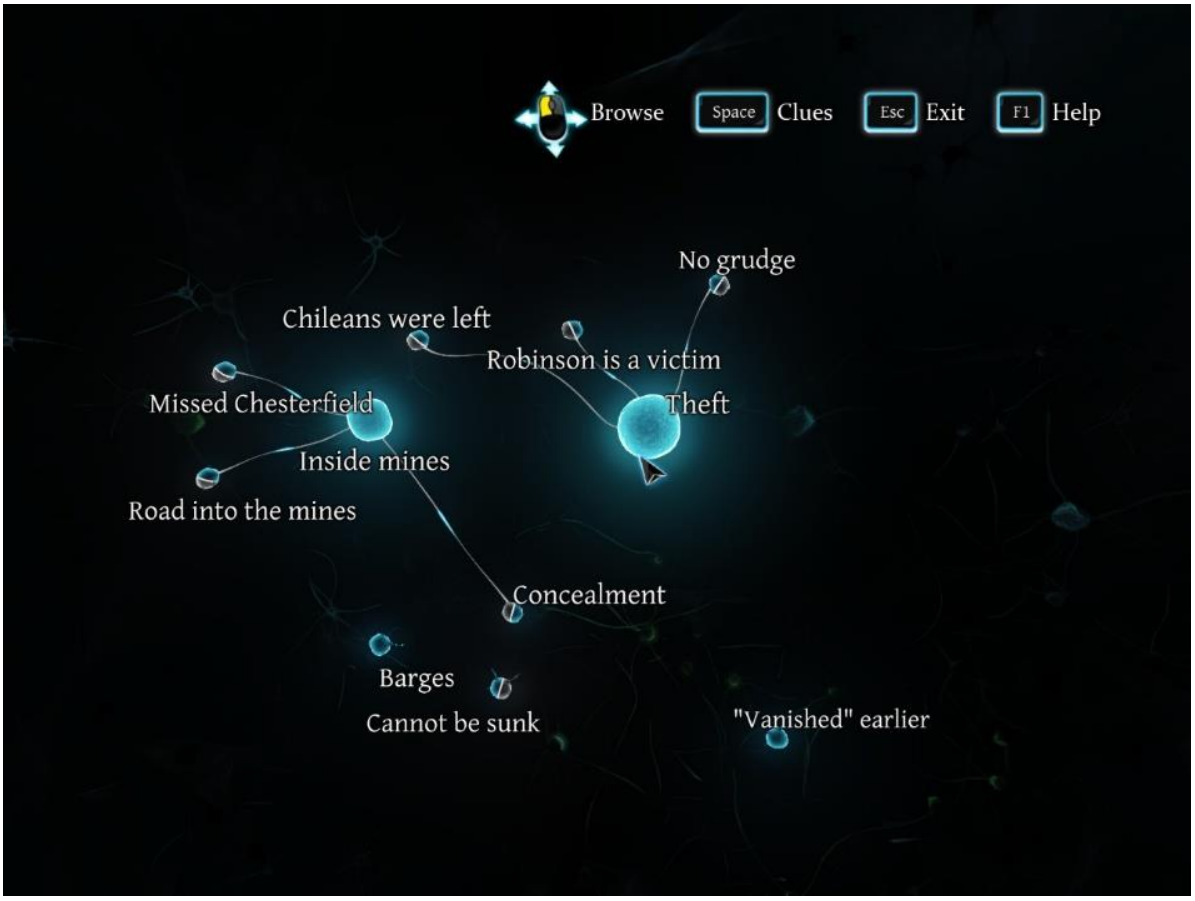


Рисунок 1.2 – Мапа дедукції

Особливості і переваги проєкту:

– гра пропонує унікальну схему дедукції, коли різні докази потрібно з’єднувати між собою, щоб отримати нові висновки, забезпечуючи високу інтелектуальну активність, оскільки гра не пропонує готових рішень, змушуючи розробляти власні версії подій на основі зібраних доказів;

– процес розслідування супроводжується створенням візуальних та психологічних портретів персонажів, підсилюючи атмосферність вікторіанської епохи та дозволяючи отримувати приховану інформацію через безпосереднє візуальне спостереження;

– в кінці розслідування гравець має декілька підозрюваних, серед яких потрібно вибрати правильного, додаючи грі моральний вибір між суворим дотриманням закону та виправданням винного, що доповнює сюжет психологічною глибинею;

– гра інтегрує численні міні-ігри (хімічні досліди, злом замків, реконструкція подій) безпосередньо у структуру розслідування і кожен такий елемент має чітке місце у структурі справи, що забезпечує логічну цілісність усього програмного продукту.

Недоліки:

– локації у грі лінійні, представлені лише невеликими зонами у яких можна проводити дослідження;

– тривалі завантаження локацій порушують темп гри.

Наступною грою йде L.A. Noire (Рисунок 1.3)

Рисунок 1.3 – Обкладинка гри «L.A. Noire»

Проєкт від Rockstar Games та Team Bondi, що став революційним завдяки технології MotionScan для передачі міміки (Рисунок 1.4). Основний акцент

зроблено на психології – гравець повинен розпізнати брехню за виразом обличчя та жестами підозрюваних.

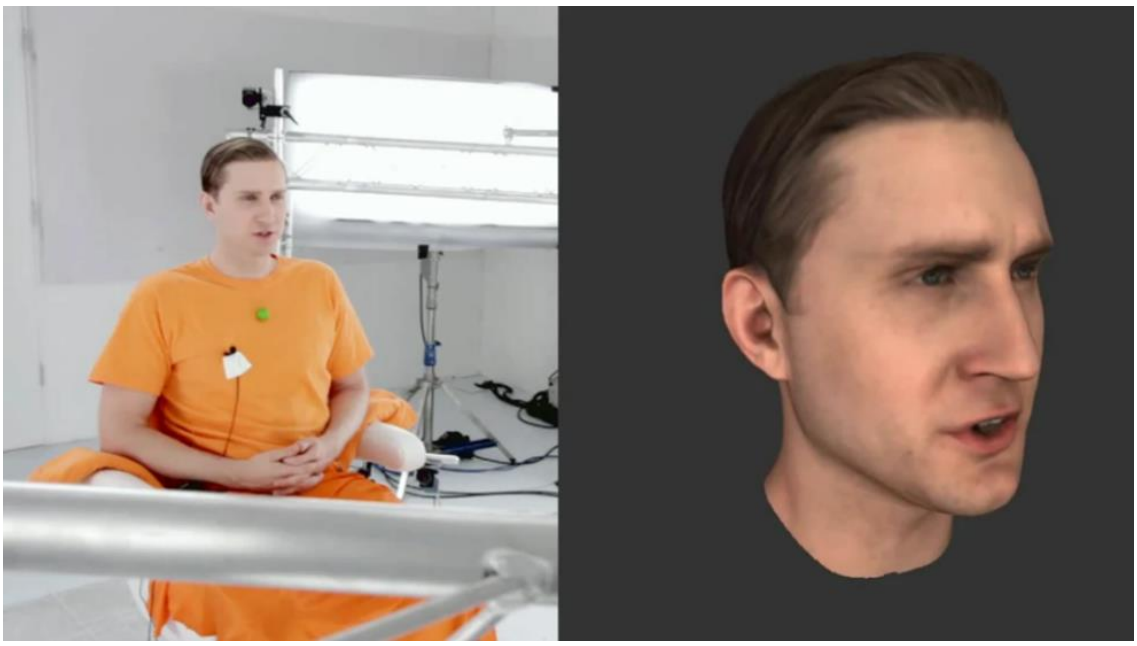


Рисунок 1.4 – Технологія MotionScan для міміки лиць

Особливості:

– технологія MotionScan забезпечує надзвичайно детальну передачу емоцій, що зміщує акцент з простого пошуку предметів на спостережливість та розпізнавання брехні підозрюваних;

– динамічна система допитів, у якій гравець може відігравати роль поганого або хорошого поліцейського, що змушує гравця гнучко обирати лінію поведінки залежно від реакцій співрозмовника;

– автоматизований інтерактивний блокнот забезпечує ефективне впорядкування доказів, локацій та досьє;

– гра реалістично відтворює Лос-Анджелес 1940-х років та поєднує це нуарною стилістикою створюючи відчуття повноцінного інтерактивного фільму.

Недоліки:

– світ у грі великий, але порожній;

– іноді у діалогах присутня неоднозначність поведінки, коли при деяких виборах персонаж реагує занадто агресивно;

Третьою у списку є The Wolf Among Us (Рисунок 1.5).

Рисунок 1.5 – Обкладинка гри «The Wolf Among Us»

Гра від Telltale Games, яка базується на коміксах «Fables» (Рисунок 1.6). Вона демонструє, як детективна історія може бути реалізована через епізодичну структуру та акцент на моральному виборі в умовах обмеженого часу.

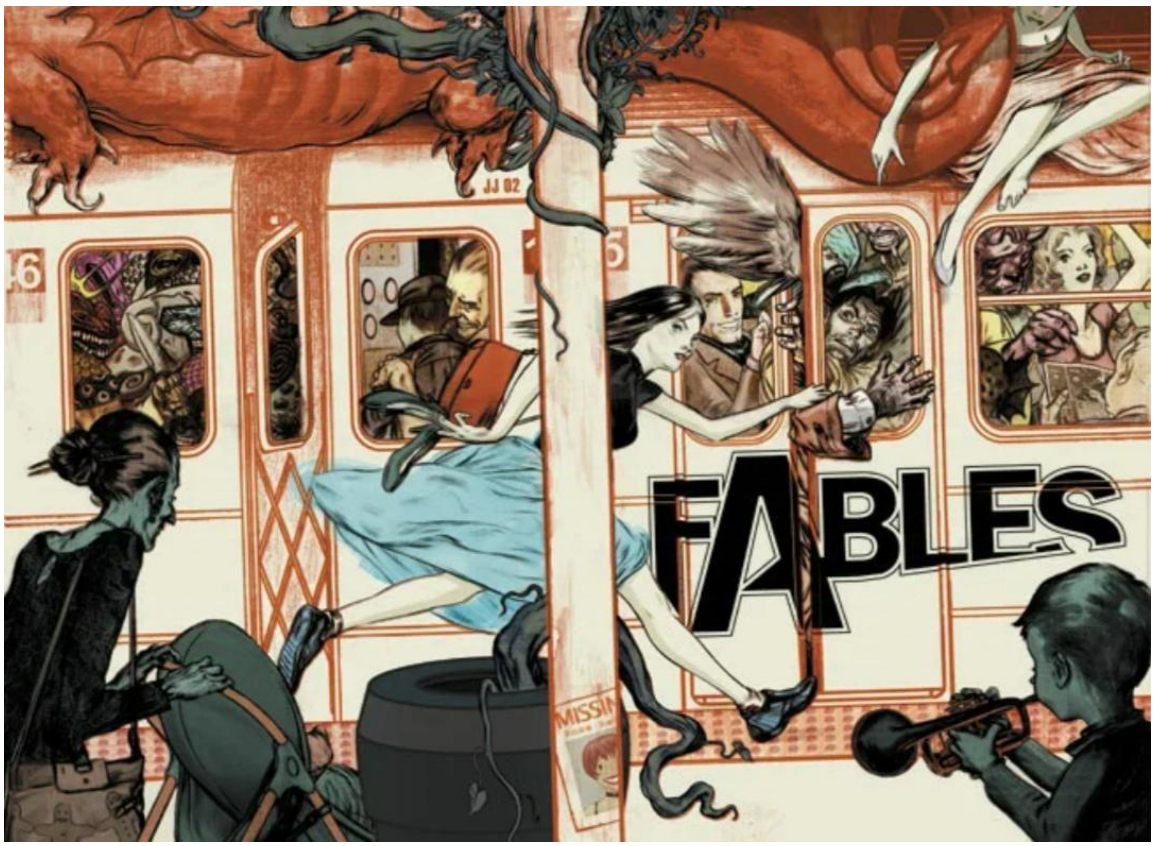


Рисунок 1.6 – Комікс «Fables»

Особливості:

– епізодична структура сюжету, при якій сюжет ділиться на глави з вираженими кульмінаційними моментами, що забезпечує постійну динаміку та високу зацікавленість користувача протягом усього процесу розслідування;

– ключовою механікою є система наслідків, де ігрові персонажі «запам’ятовують» дії та слова гравця, що безпосередньо впливає на їхнє подальше ставлення та розвиток сюжетних ліній.

Недоліки:

– хоча у грі і присутня система вибору, в глобальному плані ці вибори ні до чого не призводять і кінець гри залишається тим же самим;

– гра представлена як детектив, але головоломок тут небагато, а сама гра більше схожа на інтерактивне кіно;

Останньою грою для дослідження буде Disco Elysium (Рисунок 1.7).

Рисунок 1.7 – Обкладинка гри «Disco Elysium»

Найбільш інноваційний представник жанру за останні роки (Рисунок 1.8). Гра замінює традиційні бої на «внутрішні діалоги» з навичками героя (інтуїція, логіка, емпатія тощо), що робить процес розслідування глибоко психологічним.

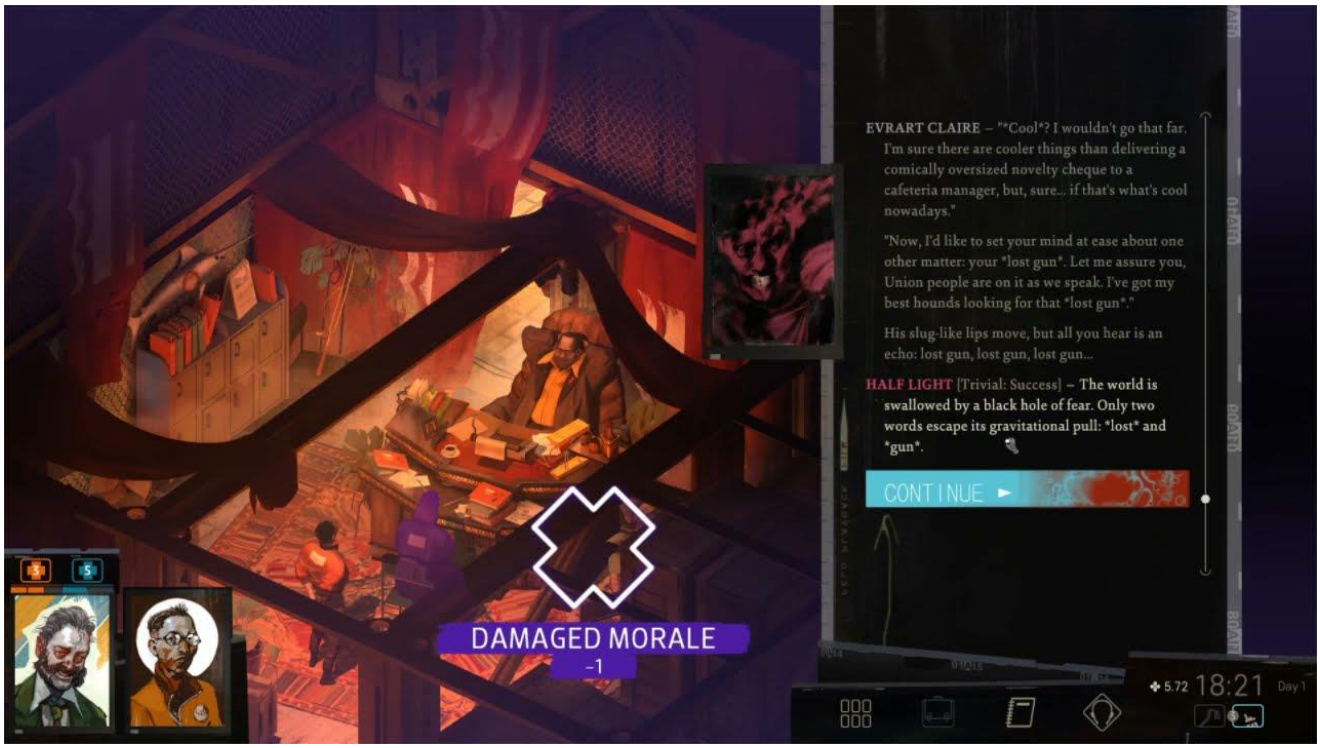


Рисунок 1.8 – Діалоги у грі

Особливості:

– замість стандартних характеристик у грі реалізовано 24 навички, що функціонують як окремі «голоси» у свідомості героя, надаючи поради та формуючи унікальний внутрішній діалог;

– система дозволяє персонажу «обдумувати» ідеї протягом ігрового часу, що призводить до отримання постійних бонусів або докорінної зміни світогляду героя;

– усі ігрові суперечності розв’язуються виключно через розгалужені діалоги та перевірку навичок за допомогою імовірнісних механік (кидків кубиків).

– використання авторської акварельної техніки створює неповторну атмосферу, а насиченість кожної локації прихованими деталями та описами стимулює гравця до ретельного дослідження ігрового світу.

Недоліки:

– насиченість ігрового процесу текстовими матеріалами може суттєво знизити привабливість продукту для казуальної аудиторії, яка надає перевагу динамічним механікам

– може бути складною для новачків через заплутані механіки навичок та ймовірність «провалити» розмову через невдалий кидок кубиків;

– багато часу витрачається на пересування між локаціями.

Провівши аналіз обраних ігор детективного жанру, можна зробити висновок, що попри різноманіття підходів до реалізації, вони мають спільні фундаментальні риси у своїй структурі: акцент на зборі інформації, систему взаємодії з оточенням та наративну залежність ігрового процесу. Незважаючи на високу якість виконання та культовий статус розглянутих проєктів, кожен із них має певні обмеження – від надмірної лінійності та ілюзорності вибору до технічної складності чи перевантаженості текстом, що може бути враховано та удосконалено при створенні нового продукту.

Розглянуті ігри мають подібні вимоги до базової структури: наявність системи доказів, механіки допитів та журналу розслідувань. Проте методи реалізації цих елементів значно варіюються – від кінематографічних 3D-допитів у L.A. Noire до глибоко психологічних внутрішніх монологів у Disco Elysium. Це відкриває можливість для синтезу найкращих практик: поєднання дедуктивної глибини класичних детективів із візуальною виразністю 2D-стилістики та гнучкістю сучасних скриптових систем.

На основі проведеного аналізу було сформовано перелік ключових функціональних вимог до розробки власного ігрового проєкту на рушії Godot. У ньому враховано сильні сторони відомих ігор – зокрема, логічну структуру розслідування та атмосферний сторітелінг – і визначено шляхи для оптимізації ігрових механік під 2D-площину. Це дозволить створити продукт, який забезпечить високу залученість гравця, інтелектуальну складність та реіграбельність, відповідаючи запитам сучасної аудиторії, що цінує якісні детективні історії.

1.3.Аналіз вимог до програмного забезпечення

Основним завданням даного кваліфікаційного проєкту є комплексна інженерна розробка двовимірного ігрового застосунку у жанрі детектив на базі ігрового рушія Godot. По завершенні роботи результатом має стати не лише візуально цілісний, але й технічно оптимізований функціональний програмний продукт. Він включає в себе програмну реалізацію ключових ігрових механік, гнучку систему взаємодії користувача з віртуальним оточенням, а також надійну логічну структуру управління станами розслідування. Важливою складовою проєкту є побудова архітектури зі слабкою зв'язністю компонентів та підготовка вичерпної супровідної технічної документації (зокрема функціональних моделей, UML-діаграм класів та прецедентів, детальних описів алгоритмів обробки даних тощо) [4].

Специфікація вимог до ігрового застосунку була сформована на основі глибокого системного аналізу детективного жанру та ретельного дослідження функціональних особливостей успішних ринкових аналогів (таких як Sherlock Holmes: Crimes and Punishments та Disco Elysium). Цей аналіз дозволив виокремити критично важливі патерни проєктування та інтегрувати сучасні підходи до розробки нелінійного інтерактивного сторітелінгу [2]. У процесі формування технічного завдання особливу увагу було приділено балансу між наративною глибиною та програмною оптимізацією. Відповідно до класичних стандартів інженерії програмного забезпечення, сформовані вимоги чітко поділяються на функціональні та нефункціональні [5].

Функціональні вимоги описують вичерпний перелік ключових можливостей застосунку, які він повинен безперебійно надавати гравцеві під час ігрової сесії. Вони також суворо регламентують внутрішню логіку роботи автономних ігрових підсистем, алгоритми обробки користувацького вводу, правила зміни станів та механізми обміну даними між модулями. Детальне документування цих функцій дозволяє мінімізувати логічні конфлікти на етапі розробки та гарантує безперебійний інтерактивний досвід користувача.

Функціональні вимоги до ігрового застосунку:

– модульна структура ігрових сцен передбачає, що застосунок повинен підтримувати розділення на окремі локації (кімнати), систему керування інвентарем, модуль діалогів;

– дослідження локацій за допомогою переміщення гравця у 2D-просторі та взаємодії з інтерактивними об'єктами у межах невеликих приміщень;

– система доказів та інвентарю дає можливість знаходити, збирати та переглядати предмети, що мають значення для справи, із відображенням їхніх характеристик або описів;

– система діалогів з реалізацією розгалуженого процесу спілкування з підозрюваними, де вибір репліки впливає на отримання нової інформації або ставлення персонажа до гравця;

– варіативність фіналів з програмною підтримкою системи «звинувачення», яка дозволяє гравцеві обрати винного на основі зібраних даних, що призводить до різних сценаріїв завершення гри;

– інтерфейс користувача (UI) для забезпечення взаємодії через журнал розслідувань, де відображаються поточні завдання, знайдені докази та досьє на підозрюваних.

Нефункціональні вимоги визначають якісні характеристики системи та технічні обмеження:

– продуктивність, яка вимагає, щоб гра працювала стабільно із частотою кадрів не менше 60 FPS на цільових платформах;

– надійність, що гарантує коректну роботу системи збереження та завантаження ігрового прогресу без ризику втрати даних про знайдені докази;

– зручність використання, яка передбачає наявність інтуїтивно зрозумілого 2D-інтерфейсу, що не перевантажує гравця зайвою візуальною чи текстовою інформацією;

– масштабованість, за рахунок якої модульна структура проєкту дає змогу розробнику безперешкодно додавати нові ігрові сцени та розгалужені діалогові гілки без радикальної зміни основного програмного коду.

1.4.Висновки. Постановка задачі

Підсумовуючи результати першого розділу, можна стверджувати, що розробка сучасного ігрового застосунку у жанрі детектив є складним та багатогранним завданням, яке потребує інтегрованого підходу. Проведений змістовий аналіз предметної області підтвердив високу актуальність жанру та виявив ключову роль інтерактивного сторітелінгу в успіху таких проєктів.

Аналіз існуючих рішень (Sherlock Holmes: Crimes and Punishments, L.A. Noire, The Wolf Among Us, Disco Elysium) дозволив виділити найбільш ефективні механіки розслідування: відкриті мапи, розгалужені діалоги та системи збору доказів. Водночас було виявлено недоліки аналогів, такі як надмірна лінійність або технічна перевантаженість, що відкриває простір для вдосконалення цих елементів у власному 2D-проєкті.

Метою кваліфікаційної роботи є проєктування та програмна реалізація ігрового застосунку в жанрі детективного квесту, що базується на сучасних принципах інтерактивного наративу та механіках розслідування. Для досягнення поставленої мети необхідно вирішити такі завдання:

– проєктування структури застосунку: розробити логічну модель взаємодії ігрових сцен, враховуючи переходи між локаціями (кімнатами);

– реалізація системи дослідження: розробити механіку переміщення персонажа у 2D-просторі та систему взаємодії з активними об'єктами (точками інтересу);

– створення модуля доказів та інвентарю: програмно реалізувати збір предметів, їх зберігання та відображення в інтерфейсі користувача;

– розробка діалогової системи: створити гнучкий механізм розгалужених діалогів з NPC, що враховує попередні дії гравця та стан розслідування;

– побудова логічного ядра «здогадок»: реалізувати інтерфейс дедукції, де гравець зможе зіставляти знайдені факти для формування висновків;

– забезпечення варіативності: впровадити систему тригерів, що відповідають за вибір підозрюваного та перехід до одного з декількох ігрових фіналів;

– тестування та налагодження: провести верифікацію програмного продукту на відповідність сформованим вимогам та забезпечити стабільність ігрового процесу.

Виконання цих завдань дозволить отримати цілісний програмний продукт, що демонструє повний цикл розробки детективної гри – від концептуального аналізу до програмної реалізації фінального функціоналу.

Технічне завдання подано у Додатку А.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1.Архітектура та функціональна структура додатка

Архітектура програмного забезпечення є фундаментом розробки ігрового продукту, оскільки вона визначає структуру системи, взаємодію її компонентів, а також забезпечує масштабованість, гнучкість та стабільність програмного засобу. Для проєкту розробки ігрового застосунку у жанрі детектив було проаналізовано різні категорії архітектурних рішень, кожна з яких виконує свою специфічну функцію у системі. Існують чотири категорії архітектурних рішень:

- зв’язок;
- розгортання;
- предметна область;
- структура.

Зв’язок визначає спосіб обміну даними між компонентами системи, а саме:

- процедурна архітектура, де взаємодія між модулями відбувається через прямі синхронні виклики функцій та методів, що потребує жорсткої прив'язки компонентів до інтерфейсів один одного;
- конвеєрна архітектура, за якої дані передаються послідовно через ланцюжок обробників (фільтрів), де вихідний потік одного компонента стає вхідним для наступного без прямого зв'язку між ними;
- подієво-орієнтована архітектура у межах якої компоненти системи реагують на події (сигнали) замість прямого виклику функцій, що дозволяє будувати асинхронні, слабо пов’язані системи.

У результаті аналізу було обрано подієво-орієнтовану архітектуру (Рисунок 2.1), оскільки цей стиль є найбільш ефективним для реалізації ігрової логіки в середовищі Godot. Використання механізму сигналів дозволяє ігровим об'єктам (наприклад, інтерактивним доказам) сповіщати про зміну свого стану, на що автоматично реагують незалежні модулі інтерфейсу або журналу розслідувань. Це забезпечує високу гнучкість розробки та дозволяє легко додавати нові ігрові

механіки без переписування існуючих зв'язків між класами [6]. Прикладом може бути коли при знаходженні доказу надходить сигнал, що оновлює журнал розслідування та інтерфейс гравця без прямого виклику методів цих модулів.

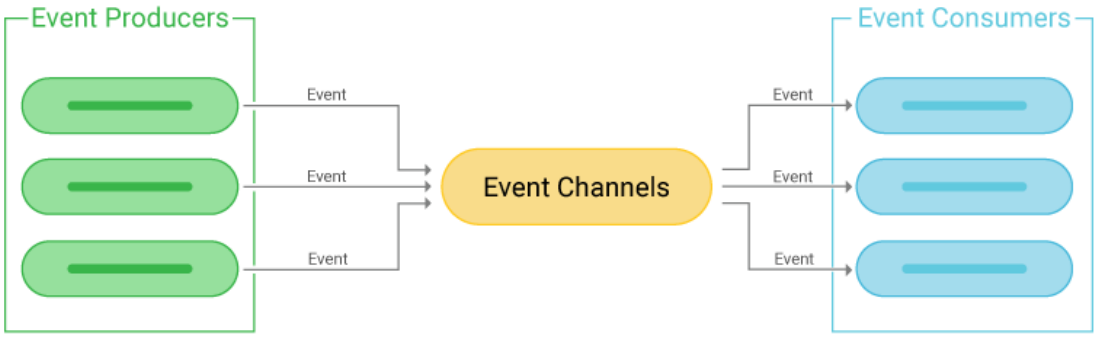


Рисунок 2.1 – Подієво-орієнтована архітектура

Розгортання визначає логічне або фізичне розташування частин системи, а саме:

– однорівнева архітектура, у якій вся система працює як єдиний додаток без чіткої логічної або фізичної сегментації модулів;

– 3-рівнева архітектура, яка розділяє додаток на три логічні рівні (презентація, логіка, дані) для спрощення розробки та підтримки;

– клієнт-серверна архітектура, що передбачає поділ системи на клієнтську частину (інтерфейс) та серверну частину, яка виконує складні обчислення або зберігає дані.

За результатами аналізу було обрано 3-рівневу архітектуру (Рисунок 2.2), завдяки тому, що вона дозволяє розбити застосунок на три незалежні рівні та забезпечити зручне тестування кожного з них [7]. Сама ж архітектура складається з таких рівнів:

– презентаційний рівень, що відображає ігрові сцени, інтерфейс користувача (HUD) та візуальні ефекти;

– логічний рівень, який обробляє всі ігрові механіки, включаючи переміщення, логіку розслідування та дерева діалогів;

– рівень даних, що відповідає за зберігання прогресу гравця, знайдені докази та параметри ігрового світу у файлах ресурсів.

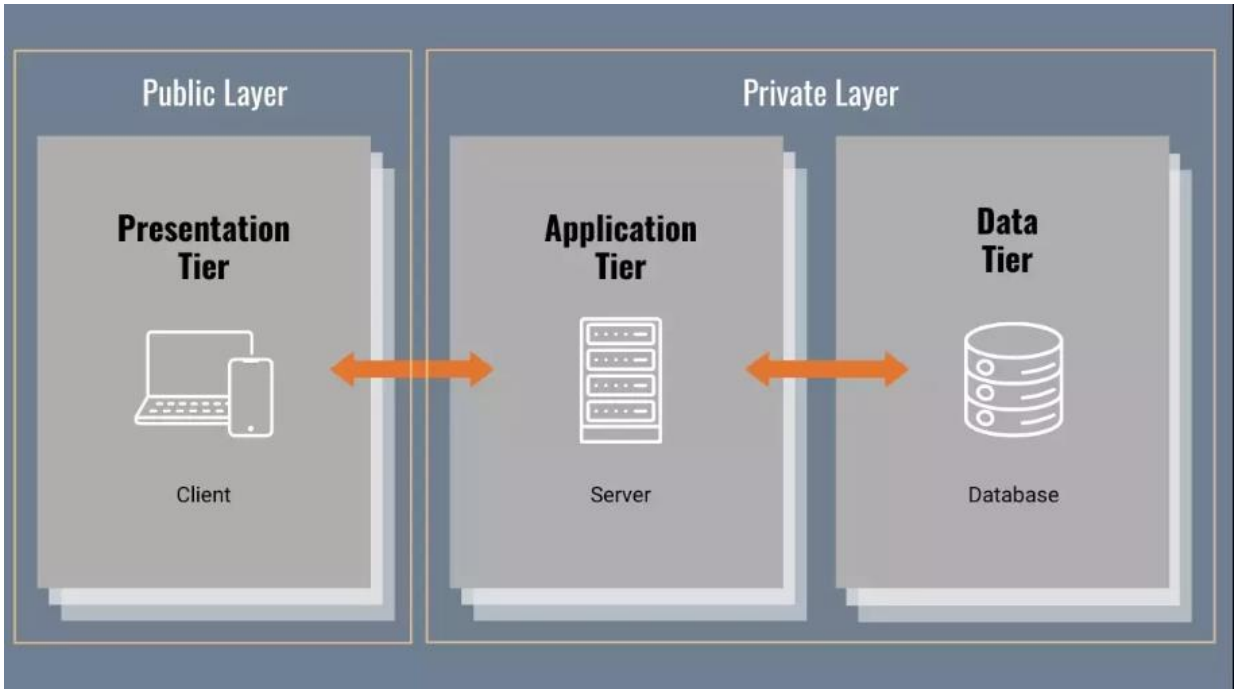


Рисунок 2.2 – 3-рівнева архітектура

Предметна область моделює реальні об’єкти та процеси розслідування у структурі ПЗ, а конкретно:

– архітектура, керована даними, яка фокусується на структурах даних та їх обробці, де поведінка системи визначається вхідними даними, а не жорстко прописаною логікою;

– сервісно-орієнтоване моделювання, що організовує предметну область навколо окремих сервісів, кожен з яких представляє конкретну ігрову можливість;

– компонентно-орієнтована архітектура, що базується на декомпозиції ігрових сутностей на незалежні компоненти, що інкапсулюють специфічну поведінку та дані, дозволяючи гнучко комбінувати їх для створення об'єктів.

Було обрано архітектуру, керовану даними (Рисунок 2.3). Такий вибір зумовлений можливістю ефективно розділити складні ігрові системи на технічне ядро (програмну логіку) та контентну складову (наративну і детективну). Це

суттєво спрощує підтримку коду та дозволяє розробнику фокусуватися на створенні нових сюжетних ліній без втручання в архітектуру рушія [8].

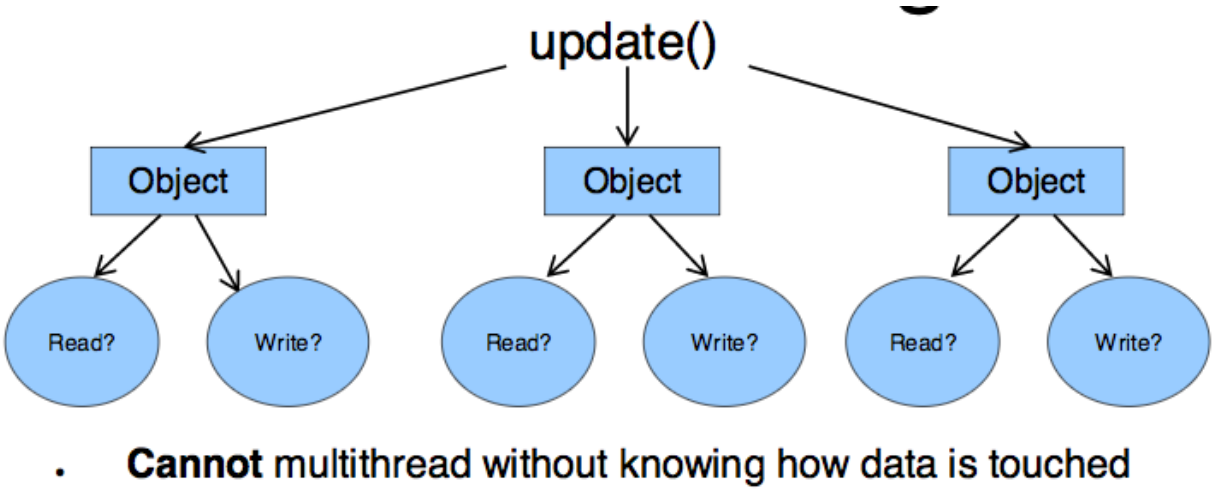


Рисунок 2.3 – Архітектура керована даними

Структура описує внутрішню організацію та взаємодію модулів системи, а саме:

– компонентна архітектура, яка організовує систему у вигляді набору самостійних об'єктів (компонентів) з чітко визначеними інтерфейсами;

– шарова архітектура, у якій система розділена на ієрархічні шари, де кожен шар надає сервіси лише для шару, розташованого безпосередньо над ним;

– чиста архітектура, де структура, де бізнес-логіка повністю ізольована від зовнішніх фреймворків, баз даних та деталей реалізації інтерфейсу;

– об'єктно-орієнтована архітектура, що організовує систему як сукупність автономних об'єктів, що містять власні дані та методи для взаємодії з іншими елементами системи.

У підсумку було обрано компонентну архітектуру (Рисунок 2.4) через її ідеальну відповідність вузловій системі рушія Godot [9, 10], що забезпечує:

– гнучкість, тобто можливість легко додавати нові функції до ігрових об'єктів через прикріплення нових скриптів-компонентів;

– повторне використання, завдяки якому один і той самий функціональний вузол можна застосувати до різних доказів та NPC;

– масштабованість, яка дозволяє створення нових типів загадок шляхом комбінування існуючих програмних компонентів.

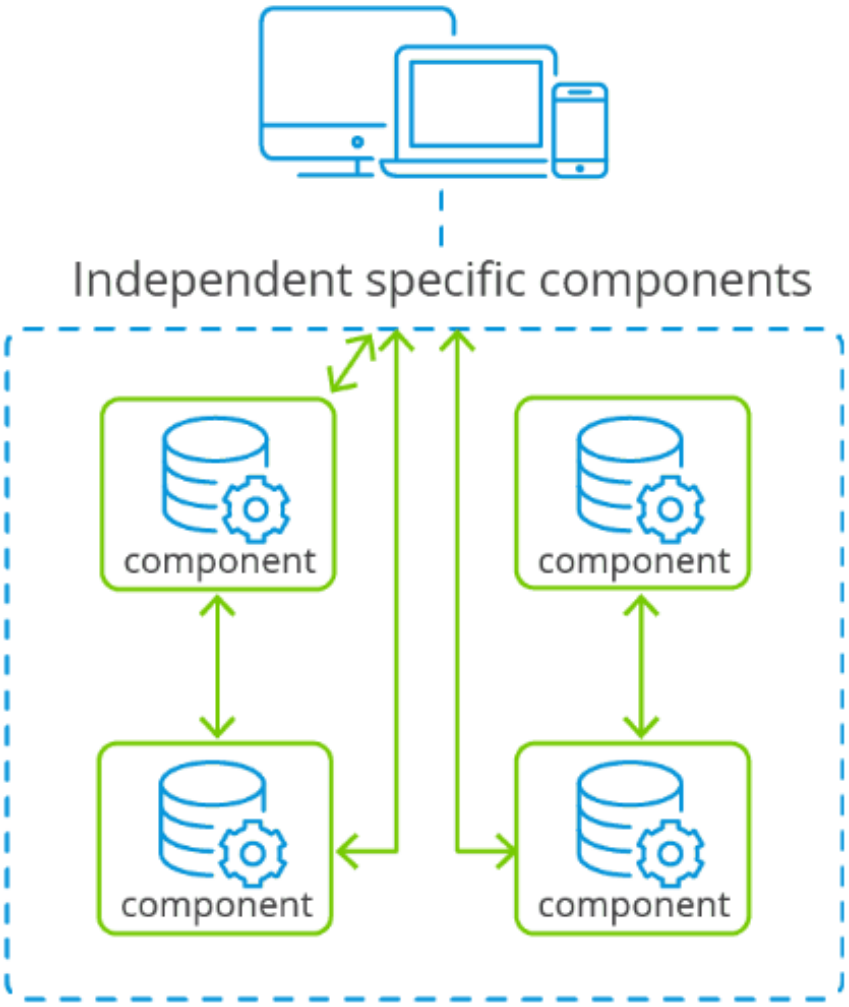


Рисунок 2.4 – Компонентна архітектура

Таким чином, у даному розділі було обґрунтовано комплексний підхід до побудови архітектури ігрового застосунку. Поєднання подієво-орієнтованого зв’язку, трирівневої структури розгортання, дизайну на основі предметної області та компонентної організації модулів забезпечує необхідну гнучкість для реалізації складного детективного сюжету [7, 9].

Обрана архітектурна модель дозволяє ефективно ізолювати ігрову логіку від технічної реалізації інтерфейсу, що значно полегшує процес тестування, підтримки та масштабування програмного продукту. Сформована структура є

оптимальною для розробки на рушії Godot і створює надійний фундамент для подальшої програмної реалізації проєкту.

2.2.Опис декомпозиції

Декомпозиція програмного забезпечення є фундаментальним етапом проєктування, що полягає у розподілі складної системи на сукупність окремих, більш простих та керованих елементів або підсистем. Основна мета цього процесу – перетворення загальної задачі розробки ігрового застосунку у набір взаємопов'язаних модулів та функцій, що значно полегшує процес реалізації, налагодження та подальшої підтримки коду [3, 5] .

Застосування методів декомпозиції дозволяє вирішити ключові завдання:

– зменшення складності завдяки розбиттю великої програми на автономні блоки (скрипти, сцени, класи), кожен з яких виконує конкретну роль;

– модульність за допомогою забезпечення незалежності окремих частин системи, що дозволяє вносити зміни в механіки розслідування без ризику порушити роботу графічного інтерфейсу;

– можливість незалежного проєктування логічних рівнів, таких як система діалогів та база даних доказів;

– локалізація помилок у межах окремих компонентів;

– забезпечення високого рівня повторного використання коду.

У межах даного розділу декомпозиція програмної системи «2D-детектив» розглядається через низку графічних моделей, кожна з яких описує специфічний аспект архітектури та поведінки застосунку:

– функціональна діаграма (загальна декомпозиція);

– модульна декомпозиція;

– діаграма варіантів використання;

– діаграма «сутність-зв’язок»;

– діаграма діяльності;

– діаграма послідовності.

Комплексне використання цих діаграм дозволяє створити повне та несуперечливе уявлення про архітектуру розроблюваного програмного забезпечення, що є необхідною умовою для успішної програмної реалізації проєкту.

Функціональна діаграма є стартовою точкою проєктування. Вона дозволяє представити систему не як набір програмного коду, а як сукупність взаємопов'язаних функцій [11]. Це допомагає зрозуміти, які саме операції має виконувати застосунок для досягнення мети користувача. У даній роботі функціональна діаграма представлена основним блоком A0 (Рисунок 2.5), декомпозицією першого рівня та декомпозицією другого рівня на Рисунках Б.1 – Б.5 в Додатку Б.

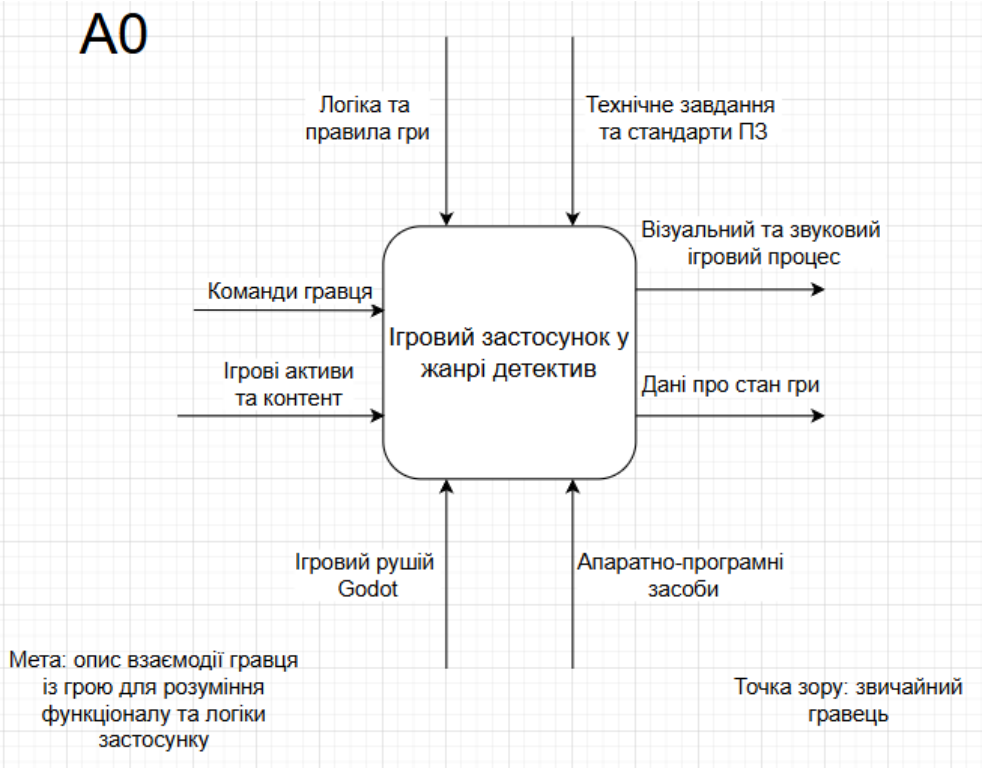


Рисунок 2.5 – Основний блок функціональної діаграми

Основний блок рівня A0 описує весь ігровий цикл гравця – від дослідження першої локації до фінального висунення звинувачення. У моделі враховано такі дані:

– вхід, що включає команди гравця, ігрові активи та контент;

– вихід, який формує візуальний та звуковий ігровий процес, дані про стан гри;

– управління, що визначається логікою та правилами гри, технічним завданням та стандартами ПЗ;

– механізм, представлений ігровим рушієм Godot, апаратно-програмними засобами.

Ця декомпозиція дозволяє описати загальний процес функціонування гри на верхньому рівні абстракції та виділити основні підсистеми, які відповідають за:

– візуалізацію ігрового світу та обробку вводу гравця;

– керування сценарієм та розгалуженими діалогами з NPC;

– збір доказів та систематизацію інформації в журналі розслідування;

– адміністрування станів гри та збереження прогресу на диск;

Модульна декомпозиція дозволяє розподілити складну систему на незалежні блоки із чітко визначеною функціональністю. У даному проєкті це ізолює логіку збору доказів від інтерфейсу блокнота та системи діалогів, що робить структуру гнучкою. Такий підхід значно полегшує налагодження гри, дозволяючи вільно змінювати механіки без ризику порушити стабільність усієї системи. На Рисунку Б.6 в Додатку Б зображена модульна декомпозиція з окремими рівнями модулів та їх класами.

Модульна декомпозиція проєкту реалізована за принципами архітектури Godot, де функціональні блоки розділені на незалежні сценарії та вузли, що взаємодіють через систему сигналів. Відповідно до вибору 3-рівневої архітектури система була поділена на такі частини :

– глобальні системи;

– основні ігрові модулі;

– виконавчі скрипти та ресурси.

Діаграма варіантів використання визначає основні сценарії взаємодії гравця (актора) з ігровою системою: дослідження локацій, збір фізичних доказів,

проведення допитів та управління журналом. Таке моделювання дозволяє зафіксувати функціональні вимоги та спроєктувати логіку переходу між механіками [12, 13]. Діаграма представлена на Рисунку Б.7 в Додатку Б.

У грі передбачений один актор, що може:

– дослідження ігрового світу та інтерактивних об’єктів із подальшим збереженням інформації;

– взаємодію з персонажами через систему розгалужених діалогів;

– аналіз записів у блокноті для формування логічних висновків;

– управління базовими налаштуваннями застосунку.

Діаграма «сутність-зв’язок» описує архітектуру даних проєкту, визначаючи взаємодію локацій, доказів, персонажів та записів блокнота (Рисунок 2.6). Вона фіксує залежність між знайденими предметами та підозрюваними, дозволяючи системі автоматично підраховувати зачіпки для розблокування фіналу гри [14]. Ця модель забезпечує цілісне управління інформацією, де кожен ігровий ресурс має чіткі технічні атрибути (шляхи до текстур, вузлів та сценаріїв), необхідні для коректної роботи рушія.

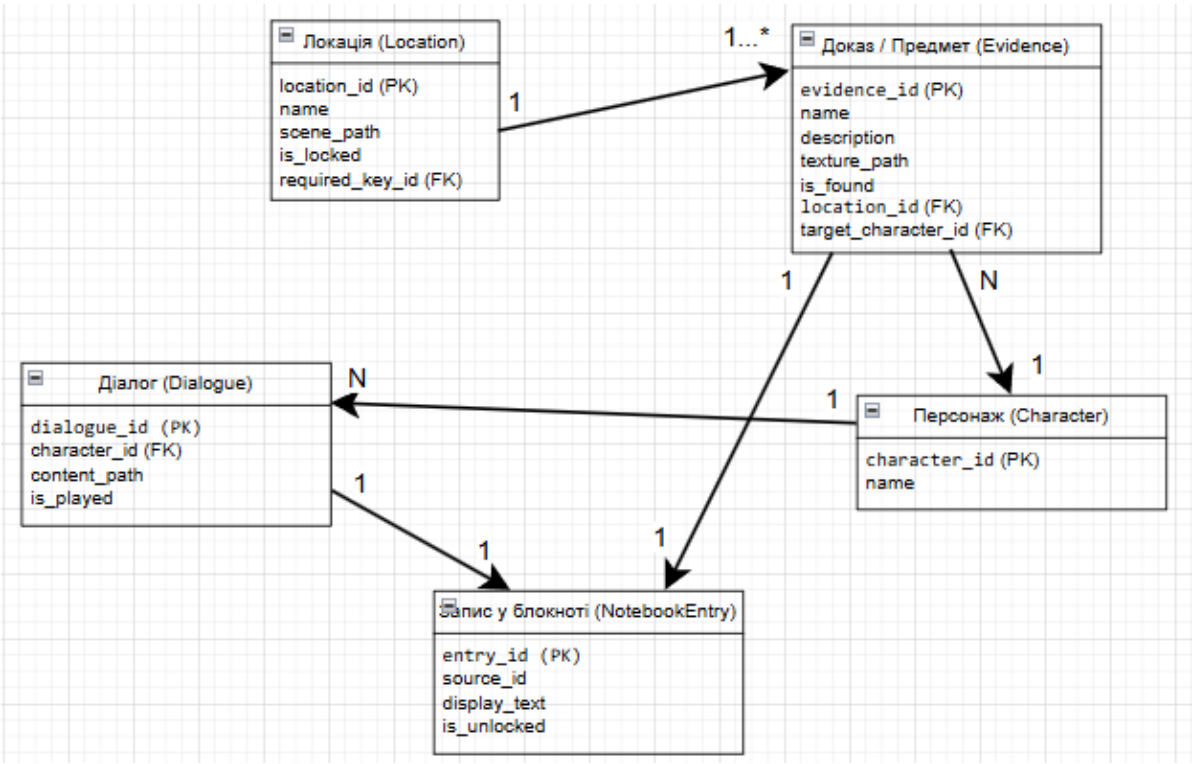


Рисунок 2.6 – Діаграма сутність-зв’язок

На основі цієї структури були виділені такі зв’язки між сутностями:

– локація – доказ (Один до багатьох), що передбачає ситуацію, коли кожна ігрова локація може містити декілька різних доказів або інтерактивних предметів, які гравець повинен знайти для просування розслідуванням;

– доказ – персонаж (Багато до одного), за якого декілька різних зачіпок або знайдених предметів можуть бути асоційовані з одним конкретним підозрюваним, формуючи необхідну кількість аргументів для висунення звинувачення та завершення гри;

– персонаж – діалог (Один до багатьох), де кожен персонаж у грі має набір унікальних діалогових гілок (запитань та відповідей), які розкриваються гравцеві залежно від прогресу розслідування;

– доказ/діалог – запис у блокноті (Один до одного), що означає автоматичне створення унікального запису в журналі за фактом знаходження предмета або завершення ключової розмови, візуалізуючи зібрану інформацію для користувача;

– доказ (Ключ) – локація (Один до одного), де конкретний предмет, що виконує роль ключа, має прямий зв’язок із заблокованою локацією, дозволяючи змінити її статус на доступний після отримання цього предмета.

Діаграма діяльності деталізує алгоритмічну логіку ігрового процесу, візуалізуючи послідовність кроків від початку розслідування до його завершення. Вона описує основний ігровий цикл пошуку зачіпок та специфічні логічні розгалуження, зокрема перевірку наявності необхідної кількості доказів перед активацією можливості звинувачення. Завдяки цій моделі фіксуються умови переходів між діями, що дозволяє забезпечити чітку роботу механік вибору та гарантувати коректне досягнення фіналу гри. Діаграма представлена у вигляді двох діяльностей: пошуку доказів та допиту підозрюваного та висунення звинувачення на Рисунку Б.8 в Додатку Б.

У грі реалізовано асинхронну поведінку кількох систем:

– навколишнє середовище та анімації, де персонажі (NPC) мають власні анімації спокою або випадкові рухи, а графічні ефекти локацій працюють

незалежно від дій гравця, що створює ефект живого світу, який не «замерзає», поки гравець роздумує над доказом;

– моніторинг інтерактивних об'єктів, під час якого система у фоновому режимі постійно сканує зону навколо гравця на предмет колізій із зачіпками або персонажами, дозволяючи миттєво підсвічувати активні предмети або змінювати курсор без затримок у русі головного героя;

– система збору доказів та оновлення UI, яка забезпечує паралельний процес запису даних у блокнот, відтворення звукового ефекту знахідки та виведення візуального сповіщення в момент підбору предмета, гарантуючи миттєвий зворотний зв’язок для гравця без переривання ігрового процесу;

– динамічний аналіз умов діалогу, за якого система асинхронно перевіряє статус зібраних доказів у базі даних, поки гравець читає текст розмови, дозволяючи «на льоту» розблоковувати нові варіанти відповідей або кнопку звинувачення, щойно внутрішня умова буде виконана;

– фонове управління аудіосистемою, завдяки якому зміна фонової музики залежно від локації або напруженості моменту відбувається плавно та незалежно від логічних обчислень рушія, що гарантує відсутність «стрибків» або пауз у звуці.

Діаграма послідовності відображає динамічну взаємодію між основними акторами системи – Гравцем, Системою гри та Ігровим рушієм – у часовому вимірі [12, 15]. Вона детально описує потік повідомлень від моменту фізичного кліку користувача по об'єкту до виконання внутрішніх логічних перевірок і подальшого оновлення візуального стану світу. Зокрема, модель фіксує критичний сценарій перевірки кількості зібраних доказів у реальному часі, що є необхідною умовою для активації фінального етапу розслідування. Такий опис дозволяє спроєктувати надійну систему комунікації між програмними модулями, забезпечуючи чітку синхронізацію детективної логіки та графічного відтворення подій. Діаграма послідовності представлена на Рисунку Б.9 в Додатку Б.

Діаграма послідовності відображає динамічну взаємодію між трьома ключовими компонентами системи: Гравцем, Системою гри (логічний менеджер)

та Ігровим рушієм (Godot). Вона візуалізує повний життєвий цикл розслідування – від ініціалізації локації до збереження результатів вибору.

У ході виконання цього розділу було проведено глибокий системний аналіз та декомпозицію архітектури детективного квесту. Процес моделювання дозволив перетворити творчу концепцію гри на чітко структурований технічний план, готовий до реалізації в ігровому рушії.

Розроблені діаграми формують надійну базу для етапу програмної реалізації. Визначення атрибутів сутностей та чітка фіксація ігрових станів запобігають архітектурним помилками на ранніх стадіях і гарантують створення стабільної, масштабованої системи. Застосований підхід до моделювання забезпечив прозорість ігрових механік та обґрунтованість обраної архітектури. Сформована документація мінімізує ризики виникнення логічних конфліктів у коді, що дозволяє розробнику повністю зосередитися на якості ігрового досвіду та атмосфері детективного розслідування [4, 15].

2.3.Опис залежностей

Опис залежностей є критично важливим етапом проєктування, оскільки він визначає логічну та технічну «зв'язність» усіх компонентів системи. У даному проєкті цей опис виконує роль регулятора ігрового процесу: він гарантує, що гравець не зможе порушити послідовність розслідування (наприклад, висунути звинувачення без доказів) та забезпечує стабільну взаємодію між модулями в середовищі Godot. Фактично, це опис правил, за якими одна частина програми реагує на зміни в іншій. Опис залежностей визначає технічний взаємозв'язок компонентів системи, забезпечуючи логічну цілісність детективного сюжету та стабільність роботи ігрової логіки [16, 17]. Перелік залежностей присутніх у грі:

– міжмодульна залежність виражається у зверненні системи діалогів до менеджера доказів для динамічного розблокування нових реплік на основі знайдених фактів;

– міжпроцесна залежність жорстко блокує активацію етапу фінального звинувачення до моменту завершення перевірки лічильника зібраних доказів;

– залежність всередині даних полягає у строгій прив’язці записів у блокноті до конкретних сутностей доказів, без успішної взаємодії з якими створення нового об'єкта даних є неможливим;

– залежність між станами регулює пріоритетність керування, автоматично призупиняючи режим вільного дослідження локації під час переходу до стану діалогу або перегляду журналу;

– міжрівнева залежність базується на передачі сигналів між скриптовою логікою детективних механік та технічною архітектурою вузлів рушія Godot для синхронізації ігрових подій.

Детальний опис цих залежностей допомагає уникнути логічних конфліктів при розробці, значно спрощує процес налагодження та дозволяє легко масштабувати гру, додаючи нові сценарії без ризику порушити стабільність наявних систем. Розуміння зв'язків між модулями також сприяє створенню чистого та гнучкого коду, де зміна візуального оформлення не впливає на внутрішню математику розслідування [15, 17].

2.4.Опис інтерфейсу модулів

Опис інтерфейсу модулів – це технічна специфікація, що визначає способи взаємодії між окремими компонентами програмної системи без розкриття їхньої внутрішньої реалізації. Основною особливістю такого опису є забезпечення інкапсуляції та декомпозиції: кожен модуль надає набір публічних методів та сигналів, які дозволяють іншим частинам програми отримувати дані або ініціювати дії [18]. У даному проєкті на базі Godot інтерфейси базуються на механізмі «сигналів та слотів», що забезпечує низьку зв'язність – це дозволяє змінювати код одного скрипта без ризику порушення цілісності всієї системи [19]. На основі раніше представленої схеми декомпозиції, основні модулі та їхні

інтерфейси розділені на три основні категорії: Глобальні системи, Основні ігрові модулі, Виконавчі скрипти та ресурси. Ось інтерфейси даних модулів:

– GameManager.gd є центральним інтерфейсом керування життєвим циклом гри. Надає методи для запуску нових рівнів та зміни глобальних станів;

– SaveManager.gd є інтерфейсом для роботи з пам'яттю. Забезпечує методи для серіалізації та збереження поточного прогресу розслідування;

– EventBus.gd є глобальною шиною подій, інтерфейс якої складається із сигналів для реалізації асинхронної взаємодії між модулями без їх прямого звернення один до одного;

– LevelManager.gd відповідає за технічну підготовку сцени та надає методи для ініціалізації об’єктів на локації;

– DialogueManager.gd є логічним інтерфейсом взаємодії з персонажами, що приймає запити на запуск діалогів та взаємодіє з базою даних;

– NotebookController.gd є контролером логіки доказів, що надає інтерфейс для реєстрації знахідок та перевірки умов для розблокування фінального етапу гри;

– PlayerController.gd є інтерфейсом керування головним героєм, що містить методи блокування або активації руху під час сюжетних подій;

– EvidenceResource.gd представляє структурний інтерфейс даних, що визначає властивості доказів (назва, опис, ресурси), доступні для зчитування іншими модулями;

– NotebookUI.gd / DialogueUI.gd є інтерфейсами візуального відображення, що надають методи для оновлення графічних елементів блокнота та рендерингу тексту діалогів.

Використання чітко визначених інтерфейсів дозволяє розмежувати логічну частину проєкту від візуальної, що спрощує налагодження та розширення контенту в даному проєкті. Такий підхід забезпечує стабільність системи при впровадженні складних сюжетних розгалужень, оскільки кожен компонент залишається автономним у межах своєї зони відповідальності [7, 18].

2.5.Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача (UI) у даному проєкті спрямоване на створення інтуїтивно зрозумілого та мінімалістичного середовища, яке виступає основним інструментом дедукції та аналізу інформації. Інтерфейс розроблений таким чином, щоб забезпечити швидкий доступ до зібраних доказів, не відволікаючи гравця від дослідження локацій та занурення в атмосферу розслідування [20, 21].

З технічної точки зору, побудова інтерфейсу реалізована за допомогою базових вузлів класу Control. Використання динамічних контейнерів, таких як VBoxContainer та HBoxContainer, забезпечує автоматичне вирівнювання та сортування елементів списку доказів без необхідності ручного позиціонування. Для гарантування того, що графічний інтерфейс завжди залишатиметься поверх ігрового світу незалежно від рухів камери, всі UI-компоненти розміщено на окремому незалежному шарі обробки CanvasLayer.

Головне меню гри забезпечує базову навігацію, пропонуючи користувачеві функції продовження поточної сесії, налаштування параметрів звуку та графіки, а також можливість виходу з програми. Безпосередньо в ігровому процесі ключові елементи управління зосереджені у верхній правій частині екрана, де розташований інтерактивний значок блокнота. Цей значок функціонує як динамічний індикатор: при знаходженні нових зачіпок система ініціює спливаюче сповіщення, а сам значок змінює колір або отримує додаткову анімацію, коли кількість зібраних доказів стає достатньою для переходу до фінального звинувачення. Для полегшення орієнтації в ігровому світі впроваджено систему візуального відгуку, що підсвічує активні предмети та докази при наведенні на них курсора [21].

Внутрішня структура блокнота розділена на тематичні блоки, що містять текстові описи зачіпок, які для кращого сприйняття супроводжуються візуальними мініатюрами знайдених предметів. Окремий розділ присвячений досьє персонажів, де акумулюється вся інформація про їхні алібі, характер та

стосунки з іншими учасниками сюжету. Під час активації діалогової системи інтерфейс автоматично переходить у режим фокусування, приховуючи другорядні іконки та значки, що дозволяє гравцеві зосередитися на виборі реплік та аналізі портретів підозрюваних.

У висновку можна сказати, що спроєктований інтерфейс гармонійно поєднує функціональну насиченість із простотою використання. Інтеграція візуальних підказок та динамічних оновлень блокнота дозволяє створити безперервний ігровий цикл, де кожна знайдена деталь миттєво відображається в структурі даних. Такий підхід до проєктування UI забезпечує комфортну взаємодію користувача з ігровими механіками та сприяє якісному виконанню завдань розслідування в межах даного проєкту.

2.6.Детальне проєктування модулів

Детальне проєктування модулів у даному проєкті є фінальним етапом переходу від високорівневої архітектури до безпосередньої програмної реалізації. Цей процес передбачає опис внутрішньої логіки, алгоритмів роботи та структур даних кожного окремого скрипта, що забезпечує чітке розуміння того, як саме функціонують ігрові механіки на рівні коду GDScript. Основна увага приділяється реалізації принципів подієво-орієнтованого програмування, де взаємодія між модулями мінімізує їхню взаємозалежність та підвищує стабільність системи.

GameManager.gd реалізований як Singleton, що містить перелік основних станів гри через конструкцію enum. Основний алгоритм модуля відповідає за перемикання між сценами та обробку глобальних сигналів [19, 22]. Модуль містить у собі такі методи:

– change_state(new_state) змінює поточний стан гри, активуючи відповідні логічні обмеження;

– load_scene(path) забезпечує коректне вивантаження поточної сцени та завантаження нової локації або екрана епілогу;

– pause_game(is_paused) призупиняє обробку ігрових процесів під час виклику системних меню.

SaveManager.gd відповідає за збереження та завантаження ігрового прогресу. Модуль збирає дані з інших систем, формує структуру для запису та взаємодіє з файловою системою пристрою та має такі методи:

– save_game() ініціює збір даних про знайдені докази, координати гравця та стан діалогів для запису у файл;

– load_game() зчитує збережений файл та розподіляє отриману інформацію між відповідними модулями для відновлення ігрового процесу;

– serialize_data() перетворює поточні ігрові ресурси та змінні у формат JSON або словник для зручного зберігання.

EventBus.gd центральний вузол для реалізації патерну Observer. Цей модуль не містить складної логіки обробки, а слугує лише посередником для передачі сигналів між об’єктами, що дозволяє уникнути прямих залежностей між скриптами та містить такі методи:

– emit_signal_custom(signal_name, args) є уніфікованим методом для розсилки повідомлень усім підписаним модулям;

– register_listener(signal_name, object, method) дозволяє іншим модулям підписатися на конкретні ігрові події.

LevelManager.gd керує внутрішньою структурою кожної ігрової локації. Він відповідає за розстановку інтерактивних об’єктів, налаштування точок появи гравця та контроль переходів між кімнатами і має дані методи:

– setup_level() ініціалізує всі динамічні елементи на сцені (докази, NPC) згідно з поточним станом проходження;

– spawn_player(spawn_point_id) встановлює вузол гравця у конкретну координату при переході між кімнатами.

DialogueManager.gd обробляє логіку взаємодії з персонажами. Модуль зчитує сценарії розмов, перевіряє умови доступності реплік та передає текст для відображення в інтерфейсі. Модуль має такі методи:

– start_dialogue(dialogue_id) ініціалізує початок розмови, завантажуючи відповідний текстовий ресурс;

– check_clue_requirements() запитує дані у контролера доказів для перевірки можливості відкриття гілки звинувачення;

NotebookController.gd є логічним центром управління всією зібраною інформацією. Він зберігає список отриманих доказів та описів персонажів, забезпечуючи їхню актуальність протягом усієї гри:

– register_evidence(evidence_resource) додає новий об’єкт доказу до списку знайдених та оновлює стан прогресу;

– get_clue_list() повертає повний набір зібраної інформації для відображення в інтерфейсі блокнота;

PlayerController.gd реалізує механіку переміщення та взаємодії головного героя. Модуль обробляє ввід з клавіатури або миші та керує фізичним станом персонажа в ігровому світі та має наступні методи:

– set_movement_enabled(enabled) вмикає або вимикає можливість ходьби (наприклад, під час діалогу);

– _physics_process(delta) обчислює вектори швидкості, обробляє фізику переміщення та колізії головного героя з об'єктами навколишнього середовища у кожному ігровому кадрі.

Interactable.gd є універсальним компонентним скриптом, який призначається вузлам типу Area2D для створення інтерактивних зон. Цей модуль забезпечує слабку зв'язність, дозволяючи налаштовувати реакцію об'єктів без написання унікального коду для кожного з них:

– _on_body_entered(body) та _on_body_exited(body) відслідковують входження об'єкта гравця в зону взаємодії та активують візуальну підказку інтерфейсу;

– interact() викликається контролером гравця та ініціює специфічну подію (наприклад, підбір доказу або старт діалогу) через емісію сигналу в EventBus.

EvidenceResource.gd спеціалізований тип ресурсу, що описує структуру даних кожного окремого доказу. Це дозволяє створювати нові предмети через інспектор рушія без написання нового коду:

– get_info() повертає словник із повною інформацією про доказ (назву, опис та шлях до іконки) для подальшого відображення в інтерфейсі блокнота.

Окрім методів, даний ресурс інкапсулює критично важливі експортовані змінні (атрибути), які налаштовуються безпосередньо в редакторі: evidence_id (унікальний ідентифікатор), title (текстова назва для UI), description (детальний опис зачіпки) та item_icon (посилання на графічну текстуру).

Визначення конкретних методів завершує процес детального проєктування, створюючи вичерпну карту функціональності для розробника. Чітка типізація вхідних і вихідних параметрів кожної функції дозволяє мінімізувати помилки при написанні коду та забезпечує високу швидкість розробки детективних механік [22]. Такий підхід гарантує, що кожна дія гравця обробляється заздалегідь визначеним алгоритмом, що є запорукою стабільної роботи та передбачуваної поведінки даного проєкту в будь-яких ігрових ситуаціях.

2.7.Аналіз та вибір технологій і методів реалізації застосунку

Аналіз та вибір технологій є стратегічним етапом проєктування, який визначає технічну життєздатність, швидкість розробки та ефективність програмного продукту. Проведення такого аналізу дозволяє обґрунтувати вибір інструментарію, що найкраще підходить для реалізації детективних механік, та мінімізувати технічні ризики на етапі кодування. У межах даного проєкту було розглянуто кілька провідних ігрових рушіїв, зокрема Unreal Engine, Unity та Godot. Unreal Engine, попри свою потужність, орієнтований на високовартісні 3D-проєкти та є надлишковим для 2D-квесту через складність архітектури та високі вимоги до апаратного забезпечення. Unity пропонує універсальні інструменти для 2D, проте має закритий вихідний код та складнішу систему ліцензування. Для

реалізації даного проєкту було обрано Godot Engine, оскільки він є відкритим (Open Source) легковаговим рушієм із вбудованим спеціалізованим 2D-ядром (Рисунок 2.7). Його унікальна вузлова архітектура ідеально відповідає модульній декомпозиції системи, дозволяючи створювати гнучкі та продуктивні рішення з мінімальними витратами ресурсів комп'ютера.

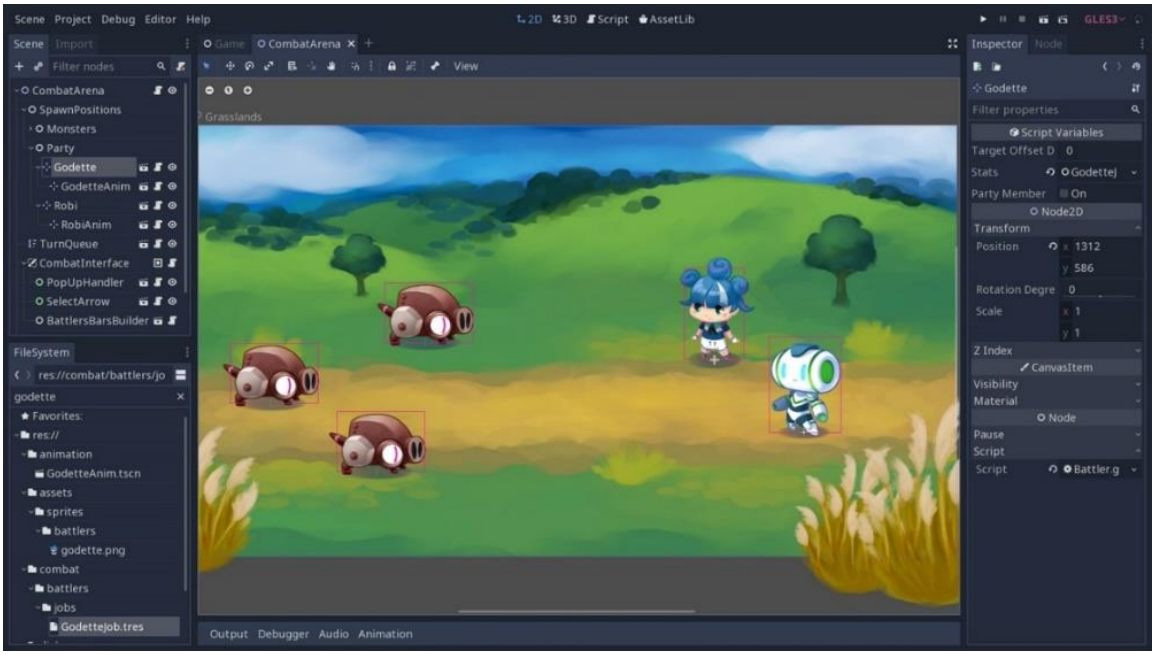


Рисунок 2.7 – Приклад 2D-сцени у Godot

Щодо вибору мови програмування, Godot пропонує використання C# та GDScript, кожна з яких має свої переваги залежно від завдань проєкту. Мова C# забезпечує високу продуктивність для складних математичних обчислень, проте потребує додаткових середовищ виконання (.NET) та має складніший синтаксис. Натомість мова GDScript, яка була обрана для розробки даного проєкту, є високорівневою мовою з Python-подібним синтаксисом, що інтегрована безпосередньо в ядро рушія. Це забезпечує безшовну взаємодію з вузлами Godot, дозволяє швидко прототипувати ігрові механіки та використовувати вбудовану систему сигналів без зайвих програмних нашарувань, що критично важливо для реалізації логіки розслідування та управління станами гри.

Методи реалізації даного проєкту базуються на принципах об'єктно-орієнтованого програмування та подієво-орієнтованої архітектури. Використання

класів та ресурсів дозволяє структурувати дані про докази та персонажів як окремі об'єкти з власними властивостями [9, 22]. Основним методом взаємодії між модулями є система сигналів, яка забезпечує низьку зв'язність компонентів: наприклад, система діалогів не знає про внутрішній устрій блокнота, а лише отримує сигнал про наявність необхідної кількості зачіпок. Для керування глобальними станами та збереженням прогресу застосовується патерн Singleton, що гарантує доступність центральних менеджерів (GameManager, SaveManager) з будь-якої точки програми. Крім того, для організації контенту використовується формат JSON, який дозволяє винести сценарії діалогів за межі коду, спрощуючи їх редагування та локалізацію.

Висновок до даного розділу підтверджує, що обраний технологічний стек у складі Godot Engine, мови GDScript та поєднання ООП із сигнальною архітектурою є найбільш раціональним для створення 2D-детективу. Ці технології забезпечують високу швидкість ітерацій, гнучкість у налаштуванні сюжетних умов та стабільну роботу застосунку на широкому спектрі пристроїв. Сформований набір методів реалізації створює фундамент для написання чистого, структурованого коду, який легко масштабується та підтримується протягом усього циклу розробки.

2.8.Висновки

У ході виконання другого розділу було проведено комплексне проєктування програмного забезпечення для детективного квесту, що дозволило трансформувати концептуальну ідею гри у детальну технічну документацію. Результатом цього етапу стала цілісна архітектурна модель, яка охоплює всі рівні функціонування застосунку – від високорівневої логіки процесів до внутрішньої структури окремих програмних модулів.

Завдяки проведеній декомпозиції (функціональній, структурній та динамічній) було чітко визначено межі системи та взаємодію основних акторів.

Розроблені діаграми варіантів використання, послідовності та станів дозволили заздалегідь виявити та розв'язати потенційні логічні конфлікти, особливо в частині перевірки умов розслідування та переходів між сюжетними етапами. Спроєктована структура бази даних та опис сутностей заклали надійний фундамент для зберігання та обробки ігрової інформації про докази та персонажів.

Важливим результатом стало детальне проєктування модулів та їхніх інтерфейсів. Визначення конкретних методів для таких систем, як GameManager, DialogueManager та NotebookController, дозволило сформувати чітке технічне завдання для етапу розробки. Використання подієво-орієнтованого підходу та системи сигналів у середовищі Godot забезпечило низьку зв'язність модулів, що гарантує стабільність та гнучкість архітектури даного проєкту.

Аналіз технологічного стека підтвердив доцільність використання рушія Godot та мови GDScript, які у поєднанні з форматом JSON для даних створюють оптимальне середовище для реалізації 2D-гри. Проєктні рішення, прийняті щодо інтерфейсу користувача, враховують специфіку детективного жанру та забезпечують комфортну взаємодію гравця з механіками розслідування [23]. Таким чином, отримані результати проєктування є вичерпною базою для переходу до стадії безпосередньої програмної реалізації, гарантуючи відповідність фінального продукту встановленим функціональним вимогам.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1.Особливості програмної реалізації на Godot

Godot Engine вирізняється серед інших ігрових рушіїв своєю унікальною архітектурою, яка базується на концепції «вузлів» (Nodes) та «сцен» (Scenes). На відміну від Unity або Unreal Engine, де ігрові об'єкти зазвичай є контейнерами для компонентів, у Godot кожен елемент гри (від окремого спрайта до цілого рівня) є вузлом, а будь-яка група вузлів може бути збережена як сцена. Це створює ієрархічну структуру, яка ідеально підходить для розробки 2D-ігор, оскільки дозволяє легко створювати екземпляри складних об'єктів, як-от докази чи інтерактивні персонажі, зберігаючи їхню повну автономність [24].

Порівнюючи Godot з конкурентами, варто виділити його легковажність та відкритість. Unreal Engine вимагає значних ресурсів для роботи з 2D та має складний процес компіляції C++, тоді як Unity має розгалужену, але часто надлишкову систему плагінів. Godot пропонує спеціалізоване 2D-ядро з підтримкою піксельної ідеальності та вбудовану мову GDScript, яка максимально інтегрована в API рушія. Це дозволяє реалізувати логіку взаємодії без зайвих накладних витрат на пам'ять, що забезпечує високу швидкість роботи застосунку навіть на слабких апаратних конфігураціях [25].

Особливістю програмної реалізації даного проєкту є активне використання механізму сигналів для забезпечення низької зв'язності модулів. У Godot сигнали працюють за принципом «видав-підписався», що дозволяє, наприклад, вузлу доказу на сцені сповіщати NotebookController про своє знаходження, не маючи прямого посилання на нього. Це спрощує налагодження та розширення гри, оскільки додавання нових об'єктів не потребує переписування глобальної логіки кожного разу при додаванні нових модулів [26, 27].

Для керування даними в даному проєкті впроваджено систему Resources (.tres). Замість того, щоб жорстко прописувати параметри кожного доказу в коді, створено спеціалізований клас EvidenceResource. Це дозволяє створювати нові зачіпки безпосередньо в інспекторі Godot, просто заповнюючи поля тексту та

вибираючи іконку. Такий підхід відокремлює контент від логіки, що є важливою особливістю професійної розробки ігор [28, 29].

Іншою важливою рисою реалізації є використання Singletons для глобальних менеджерів. Скрипти GameManager.gd та SaveManager.gd завантажуються автоматично при старті гри і залишаються доступними з будь-якої сцени. Це дозволяє реалізувати безшовні переходи між локаціями та гарантує, що стан розслідування (зібрані зачіпки, пройдені діалоги) зберігається стабільно протягом усієї ігрової сесії [30, 31].

Однією з фундаментальних відмінностей Godot від інших популярних рушіїв є підхід до рендерингу двовимірної графіки. Більшість універсальних середовищ розробки, зокрема Unity, обробляють 2D-простір як надбудову над своїм 3D-ядром, використовуючи ортогональні камери та симулюючи глибину через Z-координату. Натомість Godot володіє повністю незалежним, математично точним 2D-рушієм, де всі координати простору вимірюються виключно в пікселях. Ця архітектурна різниця усуває типові проблеми з неточним відображенням текстур та артефактами згладжування, а також дозволяє обчислювати двовимірну фізику значно ефективніше, ніж оптимізовані 3D-колізії, що застосовуються в аналогах.

Суттєвою є також різниця у парадигмі програмування та організації робочого процесу. Конкурентні рішення переважно покладаються на компільовані мови (C# у Unity, C++ в Unreal Engine), що вимагає використання зовнішніх середовищ розробки (IDE) та витрат системних ресурсів і часу на перекомпіляцію коду після кожної зміни. Godot, завдяки використанню інтерпретованої мови GDScript та її глибокій нативній інтеграції з редактором, забезпечує практично миттєву компіляцію. Це формує швидший ітеративний цикл розробки: тестування нових гілок діалогів чи логіки механік розслідування можна проводити в режимі реального часу, що суттєво підвищує загальну продуктивність розробника та знижує поріг входження для модифікації проєкту.

У висновку, програмна реалізація на базі Godot Engine дозволила ефективно втілити всі архітектурні рішення, закладені на етапі проєктування.

Використання вузлової системи, вбудованих сигналів та об'єктно-орієнтованих ресурсів забезпечило створення стабільної та гнучкої системи. Обрані методи реалізації гарантують високу продуктивність даного проєкту та закладають основу для його подальшого масштабування без втрати якості коду.

3.2.Програмна реалізація модулів

Програмна реалізація модулів полягає у переведенні розробленої архітектури в код на мові GDScript у середовищі Godot Engine [32]. Акцент зроблено на модульності та незалежності компонентів завдяки використанню сигналів і патерну Singleton, де кожен окремий скрипт ізолює конкретний аспект ігрової логіки, полегшуючи підтримку коду.

Для управління зібраними зачіпками застосовано підхід, керований даними (на базі власних ресурсів рушія), що відокремлює статичні параметри сутностей від логіки контролерів. У поєднанні з глобальною шиною подій це забезпечує слабку зв'язність: підсистеми інтерфейсу та блокнота оновлюються асинхронно без прямої програмної залежності від об'єктів на сцені.

Центральне місце в архітектурі посідає модуль EventBus.gd, реалізований як глобальний вузол (AutoLoad) (Рисунок 3.1). Його основна функція – слугувати шиною подій для асинхронної взаємодії. Це дозволяє, наприклад, модулю доказів сповіщати систему про нову знахідку без прямого звернення до інтерфейсу.

```gdscript
# EventBus.gd
extends Node

signal evidence_collected(evidence_data: EvidenceResource)
signal accusation_ready
signal notebook_updated()

signal dialogue_state_changed(is_active: bool)

signal dialogue_line_showed(speaker_name: String, text_line: String)
```

Рисунок 3.1 – Приклад коду EventBus.gd

Застосування сигналів через EventBus є класичною реалізацією патерну Observer, втіленою на нативному рівні Godot для уникнення жорстких залежностей. Замість постійного відстеження іншими модулями, об'єкт-суб'єкт самостійно транслює подію при зміні стану, а підписники реагують на неї асинхронно через функції зворотного виклику. Це повністю скасовує необхідність ресурсоємного опитування стану об'єктів у головному циклі _process(), суттєво оптимізуючи продуктивність гри.

Модуль GameManager.gd керує глобальним станом гри, використовуючи перерахування для визначення поточного режиму роботи (Рисунок 3.2). Це забезпечує чіткий контроль над тим, які дії доступні гравцеві в конкретний момент. Реалізація перемикання станів дозволяє централізовано керувати поведінкою інших систем.

```gdscript
# GameManager.gd
extends Node

enum GameState { EXPLORATION, DIALOGUE, NOTEBOOK, FINAL }
var current_state: GameState = GameState.EXPLORATION

func change_state(new_state: GameState) -> void:
    current_state = new_state

    match current_state:
        GameState.DIALOGUE:
            EventBus.dialogue_state_changed.emit(true)
        GameState.EXPLORATION:
            EventBus.dialogue_state_changed.emit(false)
```

Рисунок 3.2 – Приклад коду GameManager.gd

Логіка накопичення інформації зосереджена в модулі NotebookController.gd. Цей контролер виступає єдиним центром управління зібраними фактами, зберігаючи їх у внутрішній структурі даних (переважно у вигляді хеш-таблиць або словників) для забезпечення миттєвого доступу та перевірки за унікальними

ідентифікаторами (ID). Він обробляє дані, що надходять від об'єктів на сцені, валідує їх на відсутність дублікатів та перевіряє умови завершення розслідування. Після додавання кожної нової зачіпки контролер ініціює алгоритм зіставлення наявного пулу доказів із масивом критичних фактів, необхідних для розблокування фінальної стадії звинувачення.

Фундаментом для передачі та обробки цієї інформації є використання типізованих ресурсів типу EvidenceResource.gd (Рисунок 3.3). Створення власного класу, успадкованого від базового вузла Resource, дозволяє структурувати дані про докази як об'єкти з розширеним набором метаданих: унікальний текстовий ключ, назва, розгорнутий опис для інтерфейсу блокнота та посилання на графічну текстуру (Texture2D). Завдяки підтримці суворої типізації у сучасних версіях GDScript, такий підхід мінімізує ймовірність виникнення помилок під час виконання програми та полегшує налагодження коду. Крім того, архітектура ресурсів Godot забезпечує їхнє автоматичне кешування: незалежно від кількості звернень модулів до певного доказу, його параметри та графіка завантажуються в оперативну пам'ять лише один раз, що суттєво оптимізує загальну продуктивність ігрової системи.

```
extends Resource
class_name EvidenceResource

@export var evidence_id: String
@export var title: String
@export var description: String
@export var icon: Texture2D
```

Рисунок 3.3 – Приклад коду EvidenceResource.gd

Модуль DialogueManager.gd реалізує алгоритм розгалуження реплік на основі перевірки стану блокнота (Рисунок 3.4). Перед виведенням варіантів

відповіді система звертається до методу, який аналізує кількість зібраних зачіпок. Якщо умова «мінімум 3 докази» виконана, до інтерфейсу додається можливість висунення звинувачення. Процес збереження прогресу через SaveManager.gd базується на серіалізації цих даних у словник, який потім записується у файл формату JSON для забезпечення енергонезалежності результатів гравця [33, 34].

```
1   extends Node
2
3   var dialogue_data: Dictionary = {}
4   var current_lines: Array = []
5   var current_index: int = 0
6   var is_dialogue_active: bool = false
7
8   func _ready() -> void:
9       _load_dialogues("res://dialogues.json")
10
11  func _load_dialogues(file_path: String) -> void:
12      if FileAccess.file_exists(file_path):
13          var file = FileAccess.open(file_path, FileAccess.READ)
14          var parsed_result = JSON.parse_string(file.get_as_text())
15          if parsed_result is Dictionary:
16              dialogue_data = parsed_result
17
18  > func start_dialogue(dialogue_id: String) -> void: ···
26
27  > func show_next_line() -> void: ···
34
35  > func end_dialogue() -> void: ···
39
```

Рисунок 3.4 – Приклад коду DialogueManager.gd

Окрема увага при програмній реалізації приділялася архітектурі обробки даних у системі діалогів. Оскільки нелінійні сценарії допитів зберігаються у зовнішніх файлах формату JSON, DialogueManager.gd містить спеціалізований алгоритм для їх десеріалізації. Цей парсер перетворює текстовий масив даних у вкладені словники GDScript, де кожна репліка репрезентована як окремий графовий вузол із власним ідентифікатором, масивом можливих відповідей гравця та логічними умовами переходу. Така структура дозволяє динамічно

формувати дерево діалогу, блокуючи або відкриваючи певні гілки розмови в режимі реального часу залежно від наявних фактів у NotebookController.

Виконавчі скрипти, зокрема PlayerController.gd, є центральним елементом управління головним героєм, відповідаючи за обробку користувацького вводу, фізичну модель переміщення та зміну станів персонажа (Рисунок 3.5). Модуль циклічно зчитує команди гравця, трансформуючи їх у вектори руху з урахуванням просторових колізій та меж ігрового світу. Крім базової навігації, контролер синхронізується з глобальним менеджером станів, що дозволяє коректно блокувати можливість переміщення під час активних діалогів або перегляду журналу розслідування. Для реалізації механіки взаємодії з оточенням застосовується технологія трасування променів через вузол RayCast2D, який динамічно фіксує активні об'єкти (докази, NPC чи двері) у напрямку погляду гравця. У разі підтвердження дії контролер не обробляє логіку знайденого предмета самостійно, а ініціює подію, надсилаючи сигнал до глобальної шини обміну даними. Такий підхід інкапсулює логіку фізики та вводу в одному скрипті, зберігаючи при цьому архітектуру системи слабкозв'язною.

```
1   extends CharacterBody2D
2   class_name PlayerController
3
4   @export var walk_speed: float = 200.0
5   @export var run_speed: float = 350.0
6   @export var interact_distance: float = 50.0
7
8   var is_movement_enabled: bool = true
9   var last_direction: Vector2 = Vector2.DOWN
10
11  @onready var interact_raycast: RayCast2D = $InteractRayCast
12
13 > func _ready() -> void:···
17
18 > func _on_dialogue_state_changed(is_active: bool) -> void:···
20
21 > func _physics_process(_delta: float) -> void:···
35
36 > func _unhandled_input(event: InputEvent) -> void:···
41
42 > func interact_with_object() -> void:···
50
51 > func set_movement_enabled(enabled: bool) -> void:···
```

Рисунок 3.5 – Приклад коду PlayerController.gd

Своєю чергою, модулі візуалізації NotebookUI.gd та DialogueUI.gd спроєктовані за принципами реактивного програмування і відповідають виключно за графічне представлення ігрового процесу на екрані користувача. Вони функціонують цілком незалежно від базової ігрової логіки: ці скрипти підписані на глобальні події від контролерів і перебувають у режимі пасивного очікування. Як тільки внутрішній стан системи змінюється (наприклад, гравець успішно підбирає нову зачіпку або контролер діалогів перемикає репліку), UI-модулі автоматично перехоплюють цей сигнал і синхронно оновлюють візуальні компоненти. Вони динамічно перемальовують текстові поля, замінюють графічні іконки в інтерфейсі журналу та алгоритмічно перебудовують списки доступних варіантів відповідей. Таке суворе розділення обов'язків гарантує збереження архітектури зі слабкою зв'язністю, де інтерфейс лише реагує на дані, не втручаючись у їхню обробку.

```
1   extends CanvasLayer
2
3   @onready var notebook_panel: Control = $NotebookPanel
4   @onready var evidence_list: VBoxContainer = $NotebookPanel/ScrollContainer/EvidenceList
5
6   func _ready() -> void:
7       notebook_panel.hide()
8
9       process_mode = Node.PROCESS_MODE_ALWAYS
10
11  > func _unhandled_input(event: InputEvent) -> void: ···
17
18  > func _open_notebook() -> void: ···
23
24  > func _close_notebook() -> void: ···
29
30  > func _update_notebook_content() -> void: ···
62
```

Рисунок 3.6 – Приклад коду NotebookUI.gd

Підсумовуючи результати, викладені у даному розділі, можна стверджувати, що етап програмної реалізації модулів був виконаний у повній відповідності до попередньо визначених вимог декомпозиції та затверджених архітектурних рішень. Трансляція теоретичних моделей у виконуваний код на

мові GDScript дозволила створити повноцінний функціональний каркас ігрового застосунку. Завдяки впровадженню патерну Observer (через глобальну шину подій) та використанню підходу, керованого даними, вдалося досягти необхідного рівня слабкої зв'язності компонентів. Кожен розроблений модуль — від контролера фізики гравця до систем управління діалогами та інтерфейсом — виконує строго відведену йому функцію, не порушуючи логічної цілісності всієї системи. Такий інженерний підхід не лише гарантує стабільну і безперебійну роботу ключових детективних механік, але й формує надійну, масштабовану базу для переходу до наступного етапу розробки ПЗ — комплексного тестування, пошуку дефектів та фінальної експлуатації продукту.

3.3.Реалізація інтерфейсу користувача

Реалізація інтерфейсу користувача у даному проєкті спрямована на створення інтуїтивно зрозумілого середовища, яке забезпечує комфортну взаємодію гравця з детективними механіками. Інтерфейс виступає не лише як інструмент навігації, а й як засіб візуалізації прогресу розслідування. Основним принципом побудови UI є мінімалізм та інформативність, що дозволяє гравцеві концентрувати увагу на пошуку зачіпок та аналізі сюжету.

Програмна реалізація інтерфейсу базується на використанні спеціалізованих вузлів типу Control [35]. На відміну від ігрових об'єктів, ці вузли мають специфічні властивості для керування розміщенням, масштабуванням та ієрархією візуальних елементів.

Ключовими технічними особливостями реалізації в даному проєкті є:

– система якорів та зміщень, що дозволяє елементам інтерфейсу автоматично підлаштовуватися під різні роздільні здатності екрана та співвідношення сторін, зберігаючи задані пропорції;

– контейнерна верстка, у якій використання вузлів VBoxContainer, HBoxContainer та MarginContainer забезпечує автоматичне вирівнювання кнопок

та текстових блоків, що спрощує додавання нових елементів меню без ручного налаштування координат.

– шар CanvasLayer, за допомогою якого всі елементи інтерфейсу винесені на окремий графічний шар, що дозволяє HUD та меню залишатися нерухомими на екрані незалежно від переміщення камери гравця по ігровій локації.

Даний проєкт використовує ієрархічну структуру вузлів Control для побудови головного екрана (Рисунок 3.7). Використання MarginContainer дозволяє задати безпечні відступи від країв екрана, а VBoxContainer автоматизує вертикальне розміщення кнопок керування [36, 37]. Кожна кнопка з'єднана через систему сигналів із глобальним менеджером сцен, що забезпечує коректний перехід до ігрового процесу або вихід із застосунку.

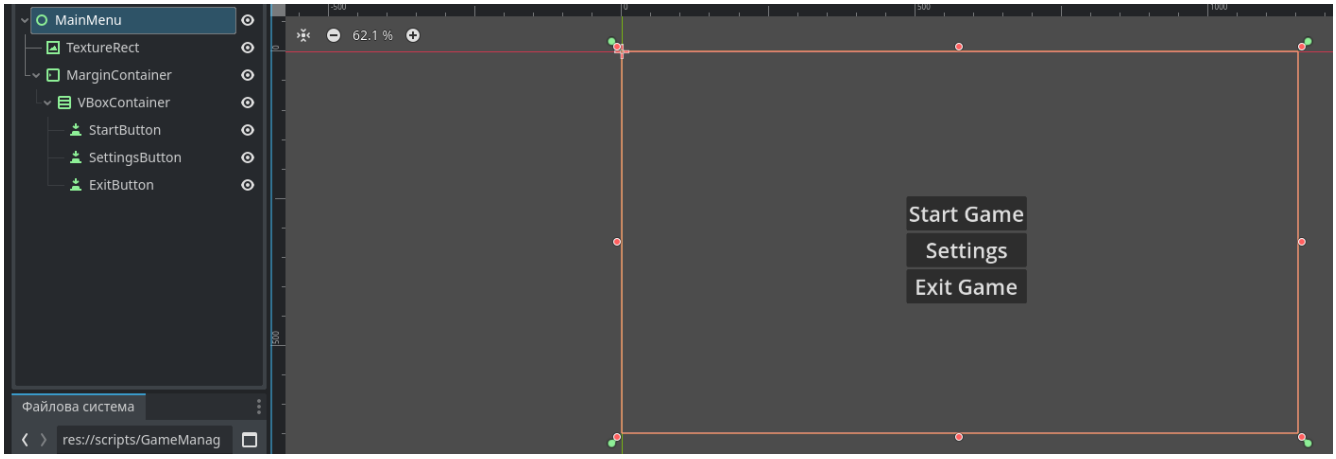


Рисунок 3.7 – Прототип головного меню

Особливістю реалізації меню паузи в даному проєкті є використання вузла CanvasLayer, що забезпечує стабільне відображення інтерфейсу поверх активної ігрової сцени. Для коректного функціонування меню в стані зупинки гри, режим процесу встановлено у значення Always. Це дозволяє скрипту обробляти ввід користувача навіть тоді, коли фізичний рушій та анімації світу призупинені. Візуальне відокремлення меню досягається шляхом використання напівпрозорого шару ColorRect, що створює акцент на елементах керування. На Рисунку 3.8 представлений прототип меню паузи.

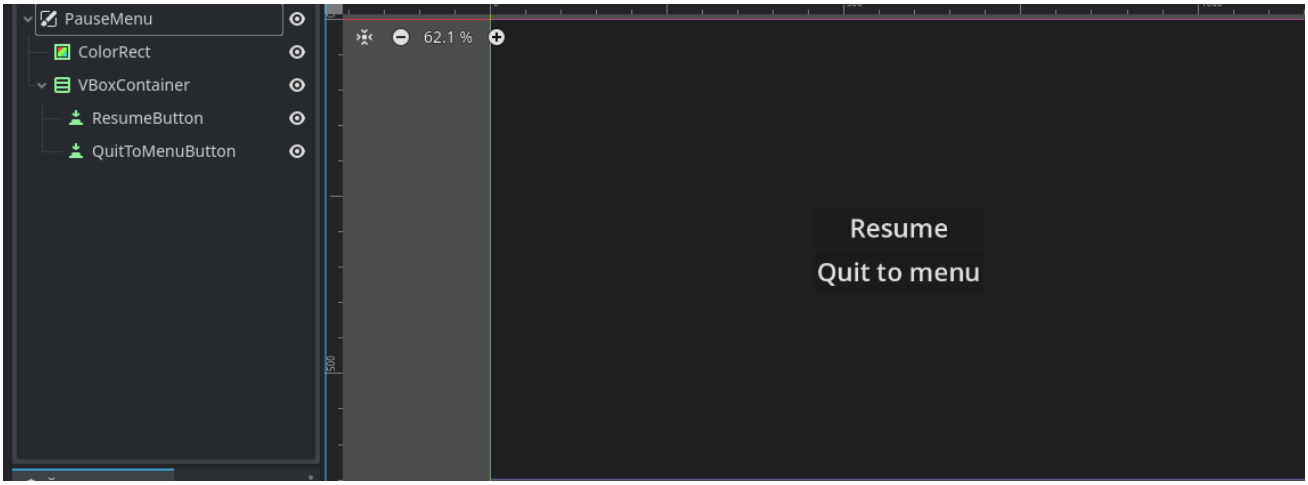


Рисунок 3.8 – Прототип меню паузи

Окремої уваги заслуговує програмна реалізація специфічних для детективного жанру елементів інтерфейсу, зокрема блокнота доказів (Notebook UI). На відміну від статичних головних меню, цей компонент є динамічним і формується під час виконання програми. При отриманні через шину подій сигналу про знаходження нової зачіпки, скрипт інтерфейсу інстанціює заздалегідь підготовлену сцену-префаб елемента списку та додає її до батьківського контейнера. Такий підхід дозволяє відображати актуальний стан зібраних фактів у вигляді структурованого переліку, автоматично масштабуючи область прокрутки (через вузол ScrollContainer) у разі перевищення розмірів екрана.

Інтерфейс діалогової системи (Dialogue UI) також побудований на принципах динамічної генерації контенту. Для відображення реплік персонажів застосовується вузол RichTextLabel, який підтримує використання розмітки BBCode для візуального виділення ключових імен чи фактів та дозволяє програмно реалізувати ефект поступового друкування тексту. Варіанти відповідей гравця генеруються у вигляді кнопок, які динамічно додаються до контейнера і підписуються на сигнали вибору. Крім того, для забезпечення єдиного стилю всього застосунку активно використовується система тем Godot (Theme та StyleBox). Вона дозволяє централізовано налаштувати шрифти, кольори та текстури для всіх UI-елементів, що спрощує процес кастомізації графічної оболонки гри.

Програмна реалізація інтерфейсу користувача в даному проєкті забезпечує гнучкість та стабільність візуальної складової застосунку. Використання спеціалізованих вузлів Control та системи контейнерної верстки в середовищі Godot дозволило створити адаптивні меню, які зберігають свою структуру при різних роздільних здатностях екрана.

Інтеграція інтерфейсу з ігровою логікою через механізм сигналів забезпечує низьку зв’язність компонентів, що дозволяє відокремити графічне представлення від програмних алгоритмів. Розроблена ієрархія головного меню та меню паузи створює надійний фундамент для комфортної взаємодії гравця з механіками розслідування, забезпечуючи інтуїтивно зрозумілу навігацію та швидкий доступ до ключових функцій даного проєкту.

3.4.Вимоги до технічних та програмних засобів

Визначення вимог до технічних та програмних засобів є критично важливим етапом розробки ігрового ПЗ, оскільки ігри є одними з найбільш ресурсомістких видів програмних продуктів. Чітке окреслення апаратних меж дозволяє розробнику оптимізувати ігровий процес, забезпечити стабільну частоту кадрів та гарантувати коректну роботу всіх графічних і логічних модулів на цільових пристроях. У контексті ігрової індустрії синергія між апаратною частиною (CPU, GPU, RAM) та системним програмним забезпеченням (драйвери, API) безпосередньо впливає на занурення користувача в ігровий світ.

Для повноцінного та безперебійного функціонування даного проєкту, розробленого на базі сучасного рушія Godot Engine, необхідно забезпечити відповідність середовища виконання певним критеріям. Оскільки гра використовує 2D-графіку та складні скриптові системи на GDScript, особлива увага приділяється підтримці сучасних графічних API та швидкості обробки фізичних процесів [38]. У таблиці 3.1 наведено вимоги до технічних та програмних засобів ігрового застосунку.

Таблиця 3.1 – Технічні та програмні вимоги

Характеристика	Мінімальні вимоги	Рекомендовані
Операційна система	Windows 10 (64-bit)	
Процесор	Dual-core Intel/AMD з частотою 2.0 GHz	Quad-core Intel Core i5 / AMD Ryzen 5
Оперативна пам’ять	4 GB	8 GB
Відеокарта	Інтегрована графіка з підтримкою Vulkan 1.0 / OpenGL 3.3	Дискретна відеокарта (NVIDIA GTX 1050 / AMD RX 560 або вище)
Вільне місце на диску	500 MB	
Системне ПЗ	DirectX 11, драйвери з підтримкою Vulkan	
Пристрій введення	Клавіатура та миша	

Обґрунтування наведених технічних вимог безпосередньо випливає з архітектурних особливостей рушія Godot 4 та специфіки розробленого двовимірного проєкту. Базового двоядерного процесора з частотою 2.0 ГГц цілком достатньо для паралельної обробки ігрової логіки, десеріалізації масивів даних діалогової системи та обчислення просторових колізій (через вузли Area2D та RayCast2D) без виникнення затримок реакції інтерфейсу. Разом з тим, мінімальний обсяг у 4 ГБ оперативної пам'яті є критичною умовою для забезпечення безперебійної роботи системи кешування. Це дозволяє завантажувати об'єктно-орієнтовані ресурси (файли формату .tres) та графічні спрайти в пам'ять під час ініціалізації сцени, уникаючи постійних звернень до накопичувача та залишаючи необхідний апаратний резерв для фонових процесів операційної системи.

Оскільки застосунок використовує виключно 2D-графіку, навантаження на відеопідсистему є мінімальним. Вимога щодо підтримки Vulkan 1.0 або OpenGL 3.3 продиктована специфікаціями базового рендерера Godot 4, при цьому потужностей сучасних інтегрованих графічних чипів повністю вистачає для стабільного відмальовування сцен зі стабільною частотою кадрів. Обсяг

дискового простору у 500 МБ є оптимальним для розміщення експортованого виконуваного файлу, архіву з ігровими асетами та файлів збереження прогресу. Водночас орієнтація на 64-бітну операційну систему Windows 10 гарантує нативну підтримку необхідних бібліотек, а використання класичних пристроїв введення (клавіатури та миші) відповідає стандартам жанру «point-and-click», забезпечуючи гравцеві необхідну точність позиціювання курсора під час пошуку доказів та навігації по меню.

3.5.Тестування програмного забезпечення

3.5.1. Вибір та обґрунтування методів тестування застосунку

Тестування є одним із найбільш критичних етапів життєвого циклу розробки ігрового програмного забезпечення. На відміну від стандартних прикладних застосунків, де основна увага приділяється коректності обробки даних, у відеоіграх тестування охоплює складну взаємодію між технічною стабільністю, логікою сценарію та користувацьким досвідом. Якісна перевірка ПЗ безпосередньо впливає на занурення гравця в ігровий світ: будь-яка технічна помилка або логічна суперечність у детективному розслідуванні може призвести до неможливості подальшого проходження та руйнування атмосфери твору.

У межах даного проєкту тестування виконує роль інструменту верифікації та валідації прийнятих архітектурних рішень [39]. Воно дозволяє переконатися, що програмні модулі не лише працюють без збоїв, а й коректно взаємодіють у межах єдиної ігрової екосистеми. Своєчасне виявлення «вузьких місць» у коді та оптимізація споживання ресурсів гарантують стабільну частоту кадрів та відсутність витоків пам'яті, що є запорукою технічної досконалості продукту [40]. Крім того, тестування допомагає підтвердити, що всі функціональні вимоги, закладені на етапі проєктування, були успішно реалізовані в середовищі Godot Engine.

Систематичний підхід до випробувань дозволяє мінімізувати ризики виникнення регресійних помилок, коли впровадження нового функціоналу може порушити роботу вже налагоджених компонентів. Зрештою, ретельне тестування виступає гарантом того, що кінцевий продукт буде не лише функціонально повним, а й максимально відшліфованим та стабільним для кінцевого користувача [41].

Для комплексного аналізу якості ПЗ у даному проєкті було обрано три ключові методи тестування, які охоплюють усі аспекти роботи гри:

– функціональне тестування;
– тестування коду;
– тестування продуктивності.

Функціональне тестування передбачає детальну перевірку відповідності всіх реалізованих ігрових механік початковим вимогам технічного завдання. Цей метод обрано як базовий, оскільки в детективному жанрі критично важливою є безпомилкова робота логіки розслідування та управління станами. У межах цього підходу тестується коректність колізій та взаємодії гравця з віртуальним середовищем, правильність розгалужень у системі діалогів, а також точність відображення даних у користувацькому інтерфейсі.

Тестування програмного коду зосереджено на верифікації внутрішньої логіки окремих скриптів GDScript та надійності їхньої взаємодії. Оскільки архітектура проєкту побудована на принципах слабкої зв'язності з інтенсивним використанням глобальної шини подій та патерну Singleton, цей вид тестування є обов'язковим для підтвердження цілісності системи. Головна мета цього етапу — переконатися, що всі кастомні сигнали безпомилково емітуються та перехоплюються відповідними підписниками, змінні стану оновлюються синхронно, а динамічне завантаження чи вивантаження ігрових сцен не призводить до втрати даних або виникнення помилок нульового посилання.

Тестування продуктивності проводиться з метою оцінки стабільності роботи ігрового застосунку та ефективності використання апаратних ресурсів. Цей метод обрано для того, щоб підтвердити технічну оптимізованість

розробленої архітектури. За допомогою вбудованих інструментів профілювання Godot аналізується стабільність частоти кадрів під час насичених ігрових сцен, швидкість ініціалізації нових локацій та обсяг споживання оперативної пам'яті. Такий контроль дозволяє своєчасно виявити потенційні витоки пам'яті, наприклад, через некоректне видалення відпрацьованих вузлів з дерева сцен, і гарантувати плавний, безперервний ігровий досвід для кінцевого користувача.

3.5.2. Аналіз результатів тестування

Функціональне тестування розробленого ПЗ проводилося за методом «чорної скриньки», де основна увага приділялася перевірці відповідності ігрових механік вимогам, визначеним на етапі проєктування. Основною метою цього етапу було підтвердження того, що кожен елемент інтерфейсу та ігрової логіки коректно реагує на дії користувача та забезпечує безперервність ігрового циклу. Тестування зображено на таблиці 3.2.

Таблиця 3.2 – Результати функціонально тестування

Тип тесту	Об'єкт тестування	Опис перевірки	Очікуваний результат	Статус
Функціональний	Блокнот (UI)	Відкриття блокнота після знаходження доказу	Нова зачіпка відображається з коректною іконкою та описом	Пройдено
Функціональний	Меню паузи	Натискання Esc та зміна налаштувань	Гра призупиняється, налаштування зберігаються у SaveManager	Пройдено
Коду (Logic)	NotebookController	Метод add_evidence() при додаванні дубліката	Масив не змінюється, сигнал про збирання не надсилається повторно	Пройдено

Продовження таблиці 3.2

Коду (Logic)	GameManager	Перехід між станами EXPLORATION та DIALOGUE	Ввід гравця блокується, UI діалогів активується миттєво	Прой дено
Продуктивності	Ігрова сцена	Вимірювання FPS під час активного руху та анімації	Стабільний показник 60 FPS без різких просадок	Прой дено

Тестування коду являє собою процес верифікації внутрішньої логіки програмних скриптів на рівні окремих методів та їхніх взаємозв'язків. Основною метою цього етапу є підтвердження того, що кожен програмний модуль виконує свої функції згідно з закладеними алгоритмами, а обмін даними між різними частинами системи відбувається без технічних збоїв чи конфліктів у пам'яті.

У ході роботи було проведено детальне тестування модуля NotebookController.gd, який є ключовим вузлом системи збору доказів. Під час перевірки було підтверджено коректність функціонування логіки додавання нових знахідок, надійність механізму запобігання дублюванню інформації, а також стабільність випромінювання сигналів для синхронізації стану блокнота з графічним інтерфейсом користувача (рисунок 3.7).

```
--- Запуск верифікації NotebookController ---
Доказ додано:
[ОК] test_add_new_evidence: Метод коректно додає новий об'єкт.
Цей доказ вже є у блокноті:
[ОК] test_prevent_duplicate_evidence: Система успішно ігнорує дублікати.
--- Debugging process stopped ---
```

Рисунок 3.7 – Тестування системи доказів

Тестування продуктивності проводилося з метою визначення технічної стабільності застосунку та ефективності використання системних ресурсів рушієм Godot. Для отримання об'єктивних даних використовувався вбудований інструмент Godot Profiler, який дозволяє в реальному часі відстежувати

навантаження на центральний та графічний процесори під час виконання ігрових сценаріїв. Основними метриками, що аналізувалися під час тестування базового руху персонажа та роботи системних модулів, стали показники частоти кадрів (FPS) та час обробки логічних вузлів (Process) (Рисунок 3.6)

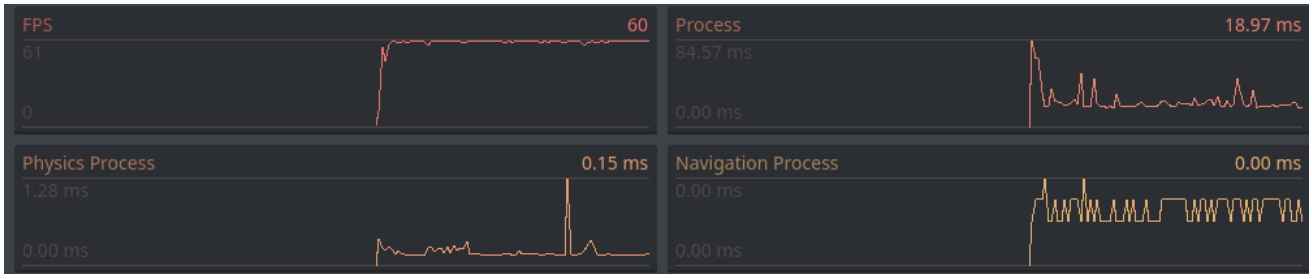


Рисунок 3.6 – Результати тестування продуктивності

3.6.Висновки

У ході виконання третього розділу було здійснено повну програмну реалізацію та всебічне тестування детективного ігрового застосунку, що дозволило перетворити проєктні напрацювання у функціональний програмний продукт. Практичне втілення архітектурних рішень у середовищі Godot Engine підтвердило доцільність обраного технологічного стека, де мова GDScript виступила ефективним інструментом для реалізації складної ігрової логіки. Створена база коду повністю відповідає розробленій раніше структурі, забезпечуючи високу швидкість обробки даних та стабільну взаємодію всіх внутрішніх систем.

Особлива увага була приділена практичній реалізації модульної структури застосунку, де використання синглтонів та глобальної шини подій дозволило досягти низької зв’язності компонентів [42]. Реалізація таких ключових модулів, як GameManager та NotebookController, забезпечила надійне керування станами гри та цілісність процесу збору доказів. Паралельно з цим було втілено систему інтерфейсу користувача, яка завдяки механізмам контейнерної верстки демонструє адаптивність до різних роздільних здатностей екрана, гарантуючи комфортну взаємодію гравця з функціоналом головного меню та вікон паузи.

Завершальним етапом роботи стала комплексна верифікація та валідація розробленого ПЗ, що охоплювала функціональне, кодове та навантажувальне тестування. Проведена перевірка підтвердила повну відповідність ігрових механік встановленим вимогам, а автоматизоване тестування окремих логічних вузлів довело стабільність алгоритмів та відсутність конфліктів у сигнальній мережі. Аналіз продуктивності продемонстрував технічну оптимізацію проєкту, зафіксувавши стабільну частоту кадрів на рівні 60 FPS при мінімальному споживанні ресурсів, що свідчить про високу якість програмного виконання.

Таким чином, результати реалізації та тестування підтвердили технічну зрілість застосунку та його повну готовність до експлуатації. Сформована програмна основа є не лише стабільною та продуктивною, а й масштабованою, що створює надійний фундамент для подальшого наповнення гри сюжетним контентом та її розвитку як цілісного інтерактивного продукту.

ВИСНОВКИ

Отже, у ході виконання кваліфікаційної роботи було проведено повний цикл розробки програмного забезпечення для детективного ігрового застосунку, починаючи від аналізу предметної області та завершуючи всебічним тестуванням готового прототипу. Основна мета роботи, яка полягала у проєктуванні та програмній реалізації ігрового застосунку в жанрі детективного квесту, що базується на сучасних принципах інтерактивного наративу та механіках розслідування, була повністю досягнута. Проєкт продемонстрував ефективність інтеграції теоретичних знань з архітектури програмного забезпечення у практичну реалізацію складного інтерактивного продукту.

Процес проєктування, викладений у другому розділі, став критично важливим етапом, що дозволив сформувати надійний архітектурний фундамент. Використання методів об’єктно-орієнтованого аналізу та моделювання за допомогою UML-діаграм забезпечило чітке розуміння взаємодії ігрових сутностей та логічних процесів. Спроєктована модульна структура з низькою зв’язністю компонентів дозволила заздалегідь розв'язати потенційні конфлікти в системі обробки доказів та діалогів, що значно спростило подальшу програмну реалізацію та забезпечило масштабованість застосунку.

Програмна реалізація проєкту в середовищі Godot Engine підтвердила доцільність обраного технологічного стека. Використання мови GDScript у поєднанні з подієво-орієнтованим підходом дозволило створити продуктивну систему, де ігрова логіка, графічний інтерфейс та обробка даних працюють як єдиний злагоджений механізм. Реалізовані менеджери станів та системи збору інформації продемонстрували високу гнучкість, а адаптивний інтерфейс користувача забезпечив комфортну взаємодію з ігровим світом на різних апаратних конфігураціях.

Особливим технічним здобутком роботи є успішне впровадження підходу керованої даними архітектури. Відокремлення сценарного контенту, зокрема розгалужених діалогових дерев у форматі JSON та параметрів доказів через

об'єктно-орієнтовані ресурси, від основної програмної логіки дозволило створити гнучку систему розробки. Разом із застосуванням технології трасування променів (RayCast2D) для точної просторової ідентифікації інтерактивних об'єктів, це не лише оптимізувало обчислювальні витрати, але й суттєво спростило процес наповнення гри новим контентом без необхідності глибокої модифікації ядра застосунку.

Результати комплексного тестування ПЗ засвідчили високу якість розробленого продукту. Верифікація коду та функціональні випробування підтвердили відсутність логічних помилок у сценаріях розслідування, а аналіз продуктивності продемонстрував технічну оптимізацію застосунку, що виражається у стабільній частоті кадрів та раціональному використанні системних ресурсів. Таким чином, розроблена кваліфікаційна робота, яка була виконана у повній відповідності до чинних методичних вказівок, є завершеним дослідженням, що має практичну цінність і може слугувати базою для подальшого розвитку повноцінного комерційного ігрового проєкту [43].

Практична значимість результатів дослідження також полягає у високому рівні масштабованості створеного продукту. Завдяки кросплатформеності рушія Godot та надійно закладеним архітектурним патернам, розроблена система має потенціал для швидкого портування на мобільні платформи або веб-середовища. Перспективи подальшого розвитку проєкту охоплюють інтеграцію додаткових сюжетних справ, впровадження системи локалізації тексту для охоплення ширшої аудиторії, а також розширення логічних механік, що дозволить успішно трансформувати створений прототип у конкурентоспроможний комерційний реліз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tianyi Gu. Year in review: 2025 to date. December 18, 2025. URL: https://newzoo.com/resources/blog/year-in-review-2025-to-date

(Дата звернення 02.02.2026)

2. Berger R. Dramatic Storytelling and Narrative Design: A Writer's Guide to Video Games and Transmedia. 2nd ed. Boca Raton: CRC Press, 2024. 260 p.

3. Laplante P. A., Kassab M. What Every Engineer Should Know about Software Engineering. 2nd ed. Boca Raton: CRC Press, 2023. 240 p.

4. Wazlawick R. S. Object-Oriented Analysis and Design for Information Systems: Agile Modeling with BPMN, OCL, IFML, and Python. 2nd ed. Cambridge: Morgan Kaufmann, 2024. 391 p.

5. Foster E. C. Software Engineering: A Methodical Approach. 2nd ed. Boca Raton: CRC Press, 2022. 578 p.

6. Event-driven architecture style : Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven (дата звернення: 25.02.2026).

7. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 450 p.

8. Ford N., Richards M., Sadalage P. J., Dehghani Z. Software Architecture: The Hard Parts. Sebastopol : O'Reilly Media, 2021. 496 p.

9. Godot Engine documentation : official site. URL: https://docs.godotengine.org/en/stable/ (Дата звернення 25.02.2026)

10. Bond J. G. Introduction to Game Design, Prototyping, and Development. 3rd ed. New York : Addison-Wesley, 2022. 1008 p.

11. Dennis A., Wixom B. H., Roth R. M. Systems Analysis and Design. 8th ed. Hoboken : Wiley, 2021. 464 p.

12. Gomaa H. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures. Cambridge : Cambridge University Press, 2021. 582 p.

13. Unified Modeling Language (UML) specifications : official site. URL: https://www.omg.org/spec/UML/ (Дата звернення: 25.02.2026)

14. Coronel C., Morris S. Database Systems: Design, Implementation, & Management. 14th ed. Boston : Cengage Learning, 2022. 816 p.

15. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. 20th Anniversary Edition. Boston : Addison-Wesley, 2020. 352 p.

16. Rabin S. Game AI Pro 360: Guide to Architecture. Boca Raton : CRC Press, 2020. 320 p.

17. Ford N., Parsons R., Kua P., Sadalage P. J. Building Evolutionary Architectures: Support Constant Change. 2nd ed. Boston : O'Reilly Media, 2022. 248 p.

18. Zimmermann O., Stocker M., Lübke D., Cesare P., Kapferer S. Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Boston : Addison-Wesley, 2022. 544 p.

19. Bradfield G. Godot 4 Game Development Projects: Build five cross-platform 2D and 3D games with the Godot engine. 2nd ed. Birmingham : Packt Publishing, 2023. 412 p.

20. Hodent C. The Gamer's Brain: How Neuroscience and UX Can Impact Video Game Design. 2nd ed. Boca Raton : CRC Press, 2020. 280 p.

21. Thorsteinsson G. User Interface Design for Video Games. London : Routledge, 2021. 256 p.

22. Dickson L. Godot 4 Game Development Cookbook. Birmingham : Packt Publishing, 2023. 350 p.

23. Solarski C. Interactive Design for Screen: Architecture, Visual Communication, and Game Design. New York : Routledge, 2021. 240 p.

24. Godot Engine. Scene Tree : official site. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/scene_tree.html (Дата звернення 20.04.2026)

25. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2020. 652 p.

26. Godot Engine. Signals : official site. URL: https://docs.godotengine.org/en/stable/getting_started/step_by_step/signals.htm (Дата звернення 10.04.2026).

27. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2021. 672 p.

28. Godot Engine. Resources : official site. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/resources.html (Дата звернення 21.04.2026)

29. Gregory J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018 (reprinted 2024). 1240 p.

30. Godot Engine. Singletons (AutoLoad) : official site. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/singletons_autoload.html (Дата звернення 21.04.2026)

31. McShaffry M., Graham D. Game Coding Complete. 4th ed. Boston : Cengage Learning, 2012 (reprinted 2023). 912 p.

32. Godot Engine. GDScript reference : official site. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html (Дата звернення 22.04.2026)

33. Standard ECMA-404: The JSON Data Interchange Syntax. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (Дата звернення 22.04.2026)

34. Moore M. E. Basics of Game Design. 2nd ed. Boca Raton : CRC Press, 2022. 412 p.

35. Godot Engine. Control nodes (UI) : official site. URL: https://docs.godotengine.org/en/stable/tutorials/ui/index.html (Дата звернення 22.04.2026)

36. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web and Mobile Usability. 4th ed. [S. l.] : Peachpit Press, 2023. 216 p.

37. Game UI Database : official site. URL: https://www.gameuidatabase.com/ (Дата звернення 22.04.2026)

38. Godot Engine. Optimization and Performance : official site. URL: https://docs.godotengine.org/en/stable/tutorials/optimization/index.html (Дата звернення 23.04.2026)

39. Levy L., Novak J. Game Development Essentials: Game QA & Testing. 3rd ed. Boston : Cengage Learning, 2023. 384 p.

40. Kaner C. Testing Computer Software. 3rd ed. [S. l.] : Kaner Fiedler & Associates, 2021. 480 p.

41. Fullerton T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 4th ed. Boca Raton : CRC Press, 2022. 544 p.

42. Highsmith J., Cockburn A., Lines M. Agile Software Development Ecosystems. 2nd ed. Boston : Addison-Wesley, 2024. 416 p.

43. Бедратюк Л. П., Радельчук Г. І., Форкун Ю. В., Яшина О. М. Дипломний проект: методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення». Хмельницький : ХНУ, 2020. 77 с

Додаток А – ТЕХНІЧНЕ ЗАВДАННЯ
(обов’язковий)

1. ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Термін	Значення
Гра	Програмне забезпечення для реалізації ігрового застосунку в жанрі детективу
Гравець	Користувач, що керує протагоністом та проводить розслідування
Доказ	Ігровий об'єкт (EvidenceResource), що містить інформацію, необхідну для розв’язання справи
Блокнот	Інтерфейс та система збереження зібраних фактів, доказів та досьє персонажів
NPC	Неігровий персонаж, з яким Гравець може вступати в діалог
Сигнал	Механізм Godot для асинхронної взаємодії між модулями (EventBus)
Звинувачення	Фінальна ігрова механіка, що потребує наявності певної кількості доказів для активації
Локація	Ігрова сцена, де Гравець здійснює пошук доказів та взаємодію з об'єктами

2. ЗАГАЛЬНІ ВІДОМОСТІ

2.1. Призначення документа

Цей документ містить опис функціональних та технічних вимог для розробки ігрового застосунку в жанрі детективу (далі – Гра).

2.2. Загальні положення

Гра повинна відповідати таким основним вимогам:

– базуватися на вузловій архітектурі рушія Godot Engine 4.x;

– використовувати подієво-орієнтований підхід для мінімізації зв'язності модулів;

– мати інтуїтивний інтерфейс, адаптований під детективний жанр;

– забезпечувати збереження прогресу та цілісність логічних зв'язків розслідування.

2.3. Повне найменування гри та її умовне позначення

Повне найменування: Програмне забезпечення для реалізації ігрового застосунку у детективному жанрі на рушії Godot. Умовне позначення: Гра.

2.4. Найменування замовника та реципієнта

Замовник: Хмельницький Національний Університет.

Виконавець: Студент групи ІПЗ-22-1 Юзвак Д. О.

Керівник проєкту: Яшина О. М.

2.5. Планові терміни початку та закінчення робіт

Початок: 02.01.2026 року.

Завершення: 01.06.2026 року.

2.6. Нормативно правові документи, використані під час створення гри

Нормативно-правові акти, що були використані при аналізі та описі бізнес-процесів:

– Закон України «Про авторське право і суміжні права»;

– ДСТУ ISO/IEC/IEEE 29148:2025.

3. ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ ГРИ

3.1. Мета створення Гри

Мета роботи полягає у проєктуванні та програмній реалізації ігрового застосунку в жанрі детективного квесту, що базується на сучасних принципах інтерактивного наративу та механіках розслідування. Проєкт передбачає втілення повного ігрового циклу, що охоплює активне дослідження локацій, динамічну реєстрацію знайдених доказів у системі «Блокнот», реалізацію розгалужених діалогів та механіку фінального звинувачення, яка базується на логічній перевірці повноти зібраної гравцем інформації для забезпечення цілісного та якісного ігрового досвіду.

3.2. Призначення гри

Гра призначена для розважального та інтелектуального використання, виступаючи інструментом для стимулювання когнітивних навичок користувача, таких як критичне мислення, уважність до деталей та побудова складних логічних ланцюжків у межах детективного сюжету. Окрім розважального призначення, застосунок слугує платформою для практичної демонстрації сучасних патернів проєктування програмного забезпечення, зокрема Singleton, Observer та Resource-based design, ілюструючи ефективність відокремлення ігрової логіки від даних та візуального представлення для створення масштабованих та гнучких ігрових систем.

3.3. Характеристики об'єкта автоматизації

Об’єктом є система інтерактивного наративу, що дозволяє:

- досліджувати 2D-локації та взаємодіяти з активними об'єктами;
- автоматично реєструвати знайдені докази в системі NotebookController;
- проводити діалоги з динамічним розблокуванням реплік на основі знайдених фактів;
- здійснювати логічну перевірку умов для активації фінального етапу.

4. ФУНКЦІОНАЛЬНІ ВИМОГИ

– модульна структура ігрових сцен застосунок повинен підтримувати розділення на окремі локації (кімнати), систему керування інвентарем, модуль діалогів;

– дослідження локацій за допомогою переміщення гравця у 2D-просторі та взаємодії з інтерактивними об'єктами у межах невеликих замкнених приміщень;

– система доказів та інвентарю дає можливість знаходити, збирати та переглядати предмети, що мають значення для справи, із відображенням їхніх характеристик або описів;

– система діалогів з реалізацією розгалуженого процесу спілкування з підозрюваними, де вибір репліки впливає на отримання нової інформації або ставлення персонажа до гравця;

– варіативність фіналів з програмною підтримкою системи «звинувачення», яка дозволяє гравцеві обрати винного на основі зібраних даних, що призводить до різних сценаріїв завершення гри;

– інтерфейс користувача (UI) для забезпечення взаємодії через журнал розслідувань, де відображаються поточні завдання, знайдені докази та досьє на підозрюваних.

5. НЕФУНКЦІОНАЛЬНІ ВИМОГИ

– продуктивність, яка вимагає, щоб гра працювала стабільно із частотою кадрів не менше 60 FPS на цільових платформах;

– надійність, що гарантує коректну роботу системи збереження та завантаження ігрового прогресу без ризику втрати даних про знайдені докази;

– зручність використання, яка передбачає наявність інтуїтивно зрозумілого 2D-інтерфейсу, що не перевантажує гравця зайвою візуальною чи текстовою інформацією;

– кросплатформеність, завдяки якій гнучка архітектура коду на Godot дозволяє проводити легку збірку проєкту під різні операційні системи;

– масштабованість, за рахунок якої модульна структура проєкту дає змогу розробнику безперешкодно додавати нові ігрові сцени та розгалужені діалогові гілки без радикальної зміни основного програмного коду.

6. ВИМОГИ ДО ГРИ

6.1. Мінімальні вимоги:

– ОС Windows 10/11, Linux.

– Процесор Dual-core Intel/AMD 2.0 GHz.

– RAM 4 GB.

– Відеокарта підтримка Vulkan 1.0 або OpenGL 3.3.

– Вільне місце 500 MB.

7. ТЕХНОЛОГІЧНИЙ СТЕК

– Рушій Godot Engine 4.x.

– Мова GDScript.

– Формат даних: JSON, .tres (Godot Resources).

8. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ ГРИ

1. Unit-тестування: Перевірка логіки add_evidence та check_requirements.
2. Інтеграційне тестування: Взаємодія між діалогами та блокнотом через сигнали.
3. Профілювання: Аналіз навантаження через Godot Profiler.
4. Приймальне тестування: Перевірка проходження повного ігрового циклу від запуску до фінального звинувачення.

Додаток Б – ГРАФІЧНІ МАТЕРІАЛИ
(обов’язковий)

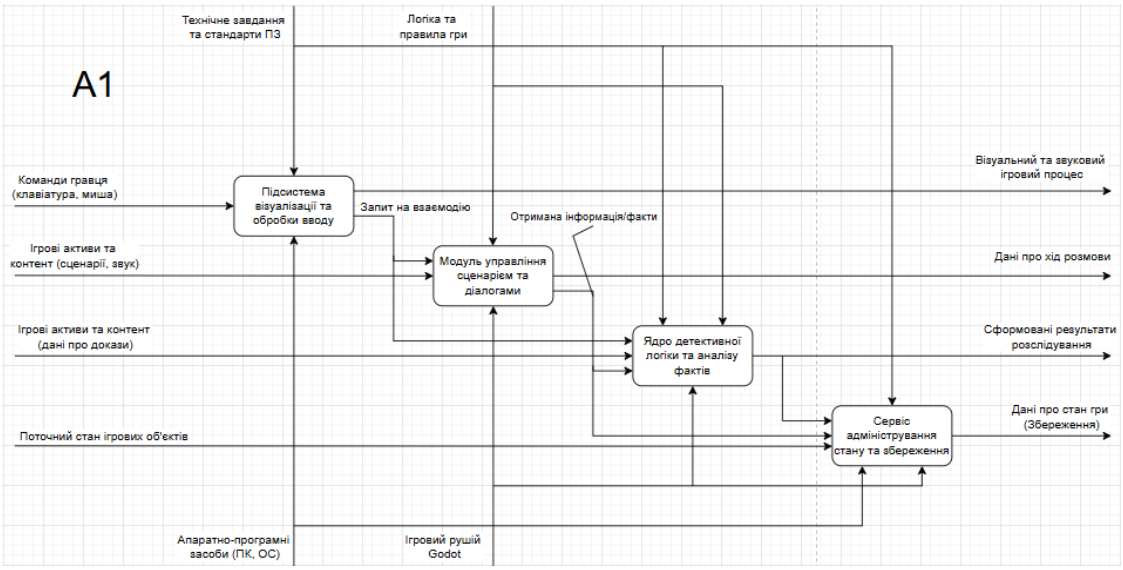


Рисунок Б.1 – Декомпозиція першого рівня

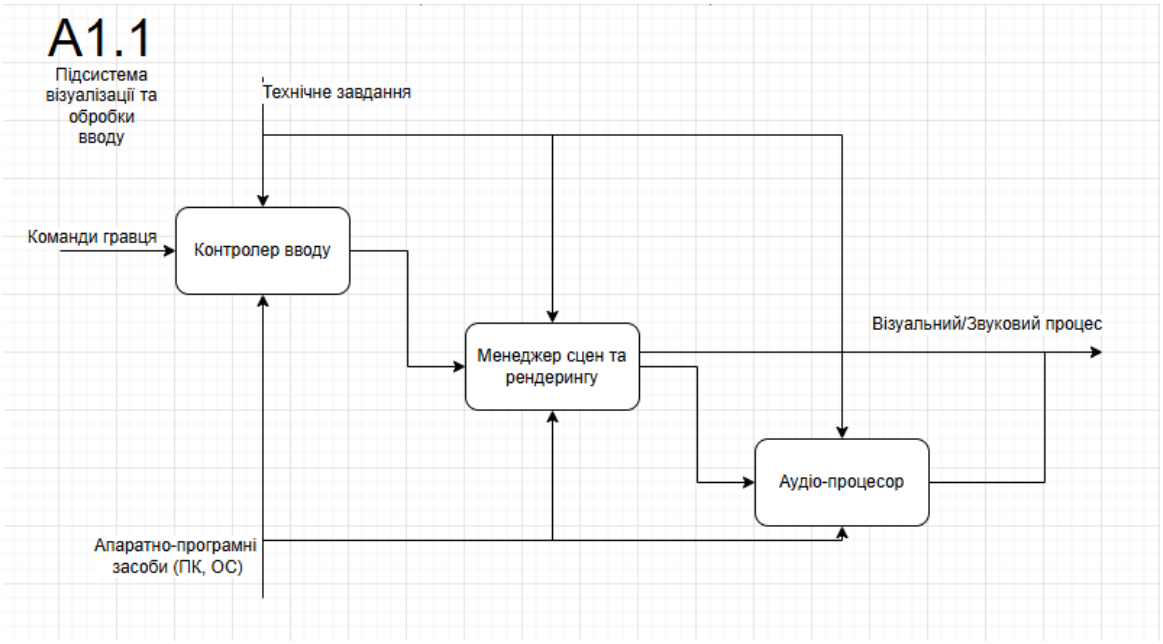


Рисунок Б.2 – Декомпозиція другого рівня блоку Підсистема візуалізації та обробки вводу

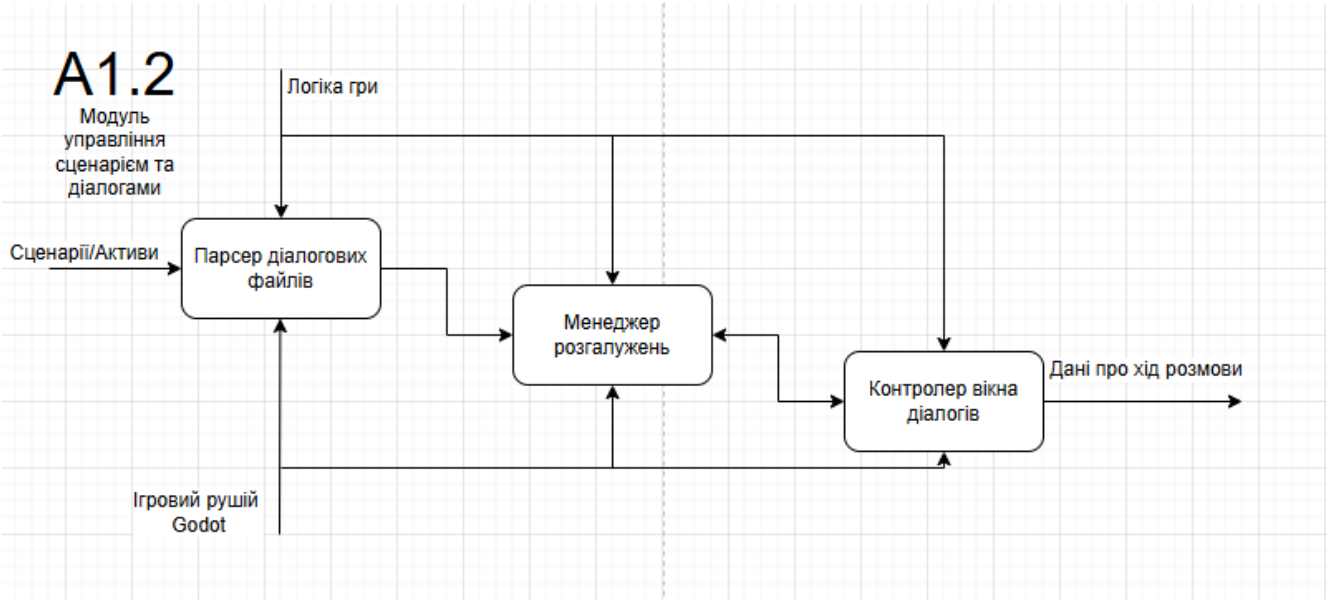


Рисунок Б.3 – Декомпозиція другого рівня блоку Модуль управління сценарієм та діалогами

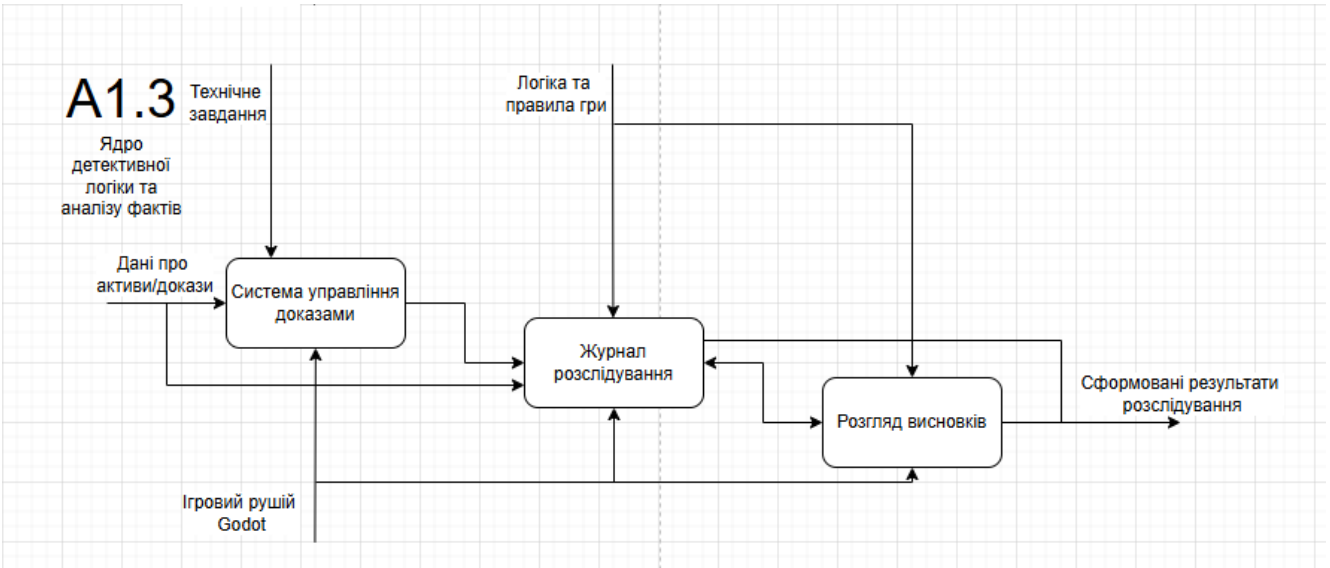


Рисунок Б.4 – Декомпозиція другого рівня блоку Ядро детективної логіки та аналізу фактів

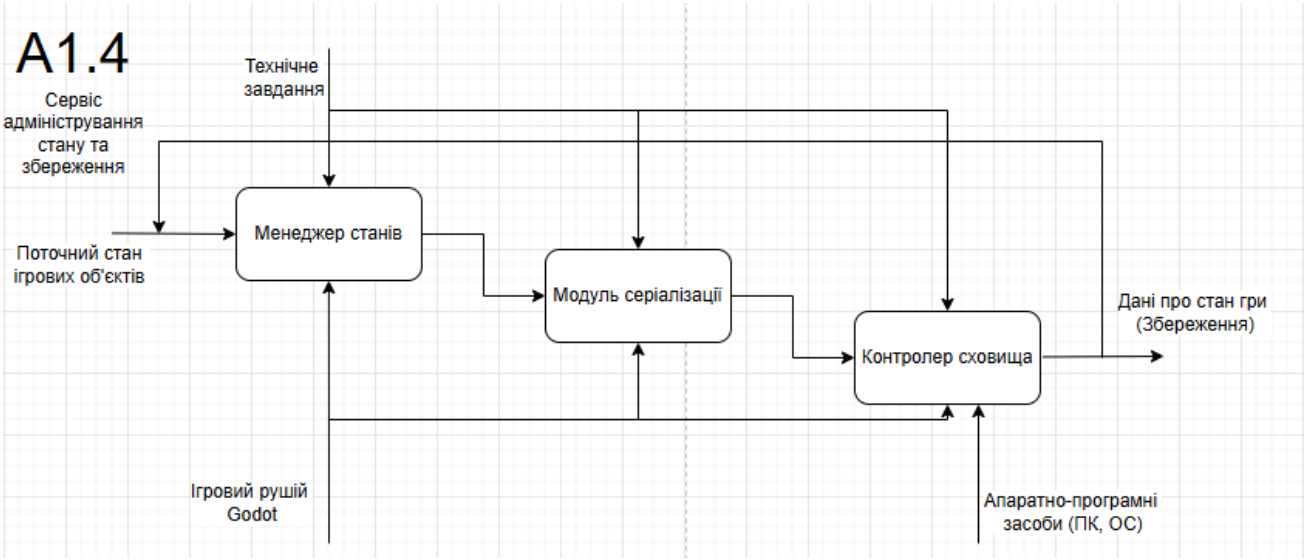


Рисунок Б.5 – Декомпозиція другого рівня блоку Сервіс адміністрування та збереження

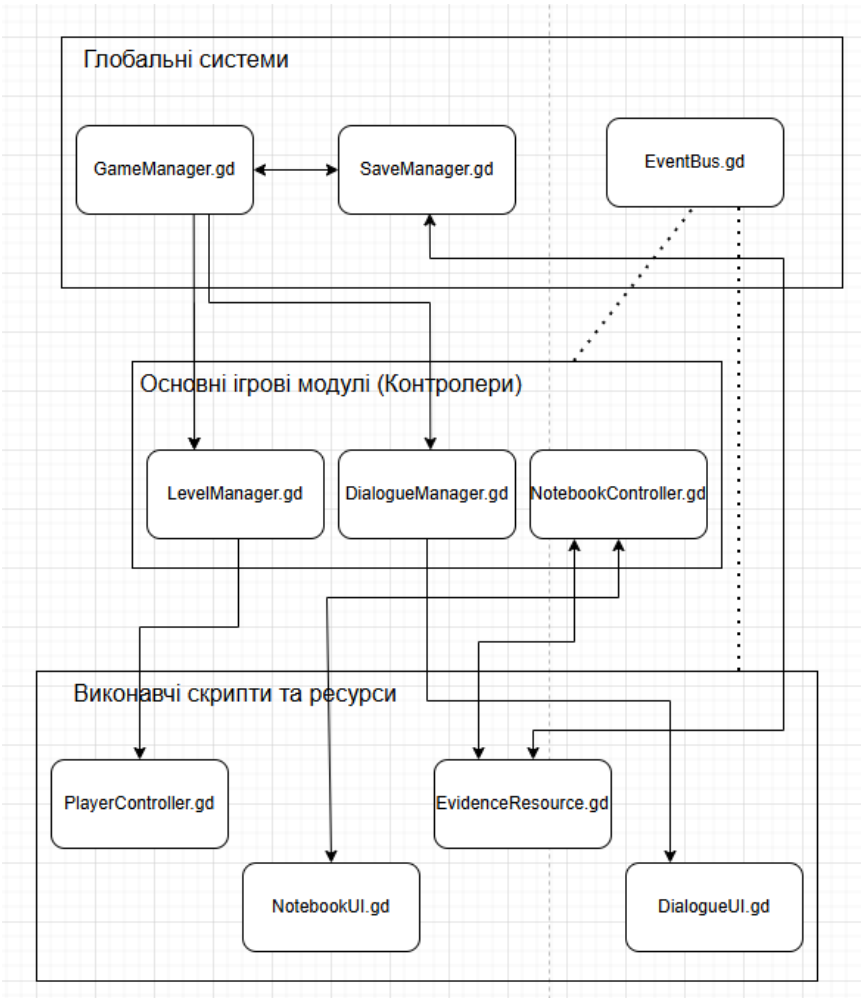


Рисунок Б.6 – Модульна декомпозиція

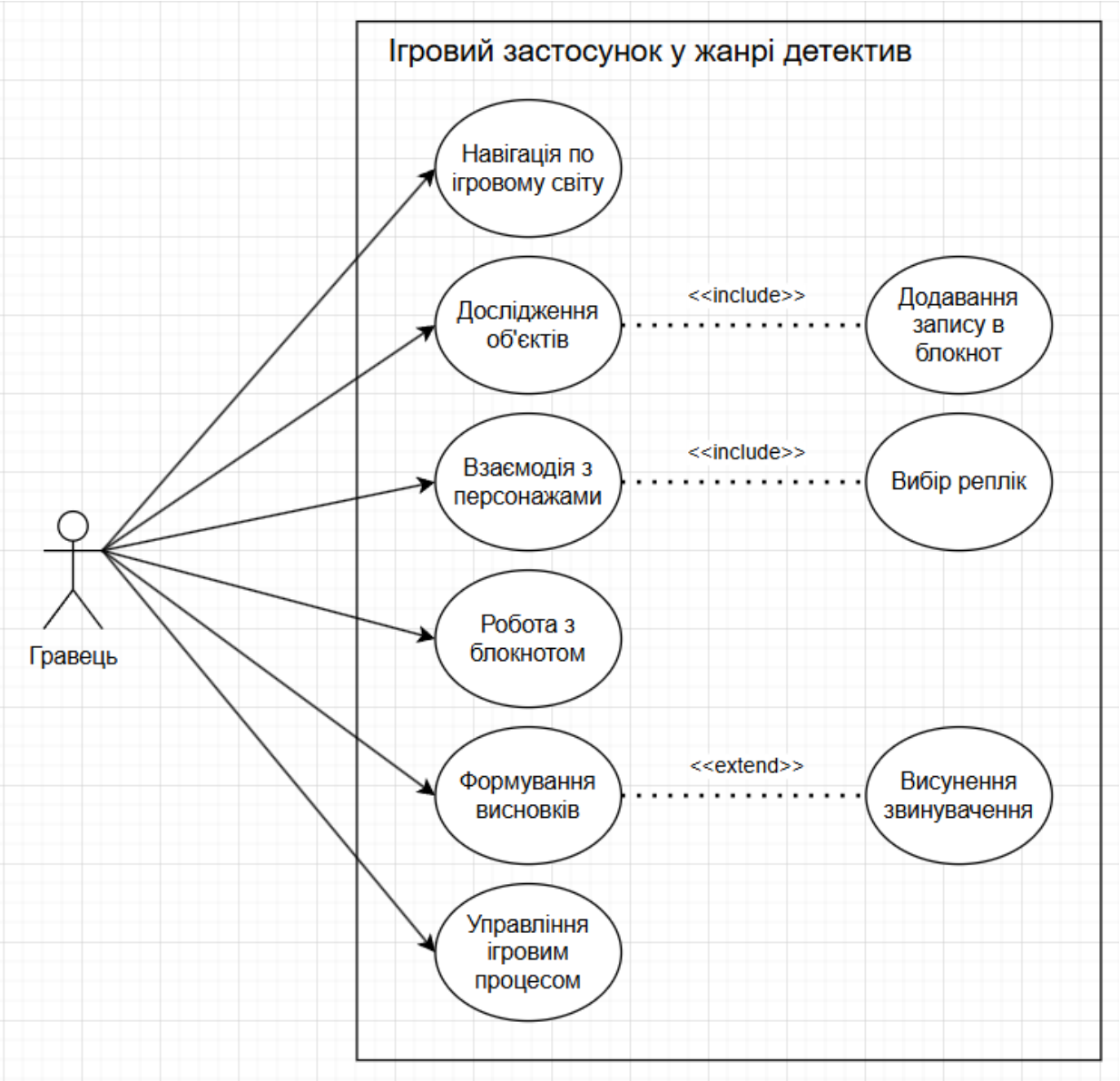


Рисунок Б.7 – Діаграма варіантів використання

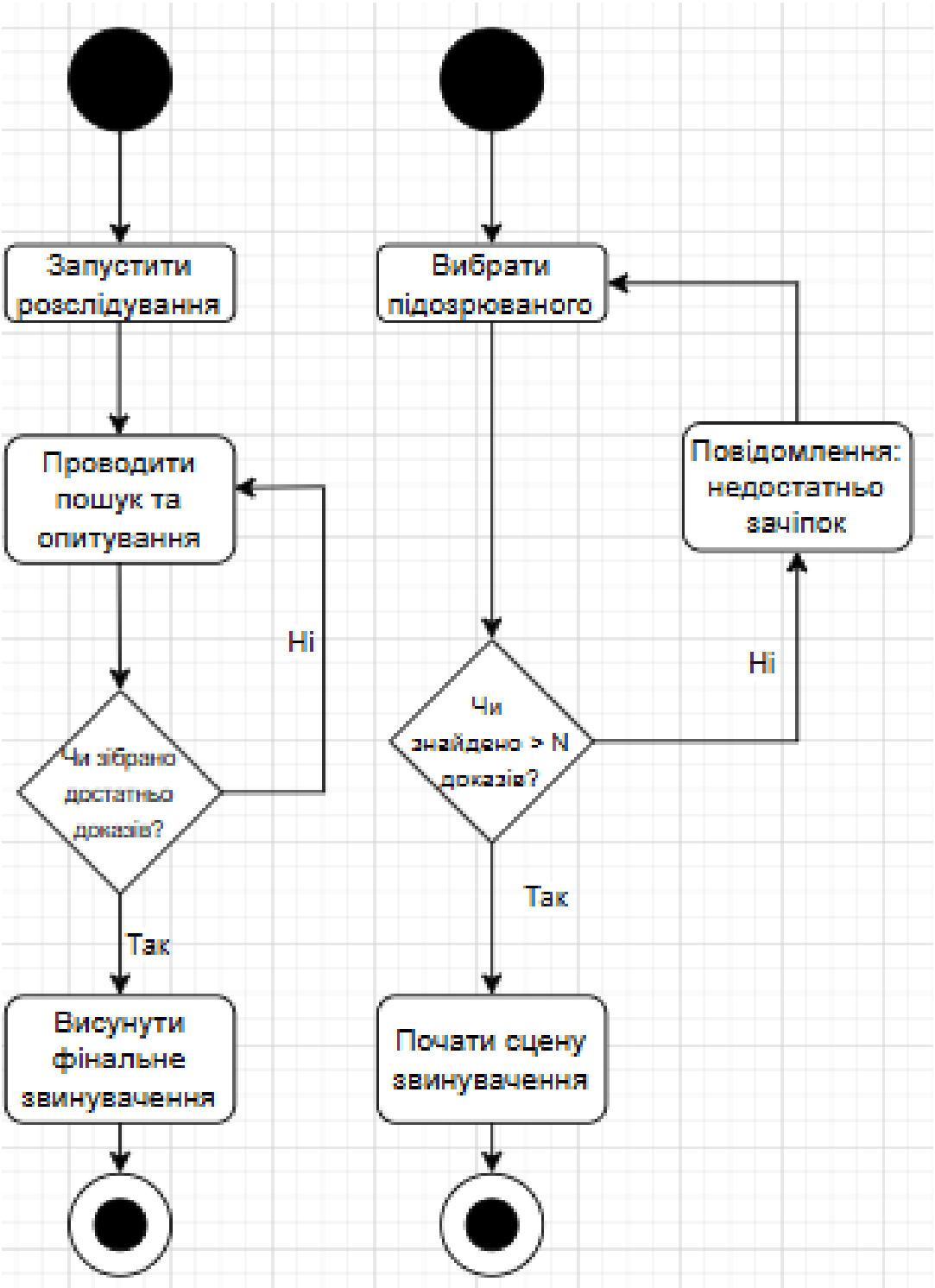


Рисунок Б.8 – Діаграма діяльності

Рисунок Б.9 – Діаграма послідовності

Додаток В – КЕРІВНИЦТВО КОРИСТУВАЧА
(обов’язковий)

У цій грі ви виступаєте в ролі професійного детектива, основним завданням якого є розкриття складної справи шляхом ретельного дослідження локацій та аналізу зібраних фактів. Ви можете вільно переміщуватися ігровим світом, вивчаючи кожен підозрілий об'єкт та вступаючи в діалоги з персонажами, які можуть володіти цінною інформацією. Кожен знайдений доказ автоматично реєструється у вашому блокноті, де ви можете систематизувати отримані дані, переглядати мініатюри знахідок та відстежувати прогрес розслідування. Головна мета – зібрати критичну масу доказів, що дозволить розблокувати унікальні гілки діалогів та активувати етап фінального звинувачення для успішного завершення справи. Будьте уважними до деталей, адже деякі зачіпки можуть стати доступними лише після взаємодії з певними свідками або знаходження ключових предметів.

Елементи керування:

W, A, S, D – переміщення персонажа по ігровій локації;

F – взаємодія з активними об'єктами, збір доказів або початок діалогу з персонажами;

TAB – відкрити або закрити блокнот розслідування для аналізу зібраної інформації;

Esc – виклик меню паузи для призупинення гри, зміни налаштувань або виходу до головного меню.

Додаток Г – ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ
(обов’язковий)

Г.1 Код контролера NotebookController.gd

```
# NotebookController.gd

extends Node

var collected_evidence: Array[EvidenceResource] = []

const EVIDENCE_THRESHOLD: int = 3

func _ready():

	collected_evidence.clear()

func add_evidence(new_evidence: EvidenceResource) -> void:

	if new_evidence == null:

		printerr("Помилка: Спроба додати порожній ресурс доказу.")

		return

	if not _is_already_collected(new_evidence.evidence_id):

		collected_evidence.append(new_evidence)

		EventBus.evidence_collected.emit(new_evidence)

		print("Доказ додано: ", new_evidence.title)

		_check_story_progress()

	else:

		print("Цей доказ вже є у блокноті: ", new_evidence.title)
```

```
func _is_already_collected(id: String) -> bool:
    for item in collected_evidence:
        if item.evidence_id == id:
            return true
    return false
func _check_story_progress() -> void:
    if collected_evidence.size() >= EVIDENCE_THRESHOLD:
        print("Достатньо доказів зібрано. Можна переходити до фіналу.")
        EventBus.accusation_ready.emit()
func get_all_clues() -> Array[EvidenceResource]:
    return collected_evidence.duplicate()
func reset_data() -> void:
    collected_evidence.clear()
```

Г.2 Код контролера PlayerController.gd

```
extends CharacterBody2D
class_name PlayerController
@export var walk_speed: float = 200.0
@export var run_speed: float = 350.0
@export var interact_distance: float = 50.0
var is_movement_enabled: bool = true
var last_direction: Vector2 = Vector2.DOWN
```

```
@onready var interact_raycast: RayCast2D = $InteractRayCast

func _ready() -> void:
    if Engine.has_singleton("EventBus"):
        var event_bus = Engine.get_singleton("EventBus")
    event_bus.dialogue_state_changed.connect(_on_dialogue_state_changed)

func _on_dialogue_state_changed(is_active: bool) -> void:
    set_movement_enabled(not is_active)

func _physics_process(_delta: float) -> void:
    if not is_movement_enabled:
        velocity = Vector2.ZERO
        move_and_slide()
        return
    var direction := Input.get_vector("move_left", "move_right", "move_up",
"move_down")
    if direction != Vector2.ZERO:
        last_direction = direction.normalized()
        interact_raycast.target_position = last_direction * interact_distance
        var current_speed = run_speed if Input.is_action_pressed("run") else
walk_speed
        velocity = direction * current_speed
    else:
        velocity = Vector2.ZERO
```

```
        move_and_slide()
func _unhandled_input(event: InputEvent) -> void:
        if not is_movement_enabled:
                return
        if event.is_action_pressed("interact"):
                interact_with_object()
func interact_with_object() -> void:
        interact_raycast.force_raycast_update()
        if interact_raycast.is_colliding():
                var target = interact_raycast.get_collider()
                if target.has_method("interact"):
                        target.interact()
                elif target.has_method("collect_evidence"):
                        target.collect_evidence()
func set_movement_enabled(enabled: bool) -> void:
        is_movement_enabled = enabled
        if not enabled:
                velocity = Vector2.ZERO
```

Г.3 Код InteractiveEvidence.gd

```
extends Area2D
class_name InteractiveEvidence
```

```
@export var evidence_data: EvidenceResource
@onready var sprite: Sprite2D = $Sprite2D
@onready var collision: CollisionShape2D = $CollisionShape2D
func _ready() -> void:
	if evidence_data != null and sprite != null:
		if evidence_data.icon != null:
			sprite.texture = evidence_data.icon
			sprite.scale = Vector2(0.5, 0.5)
			if collision != null and collision.shape is RectangleShape2D:
				collision.shape = collision.shape.duplicate()
				var texture_size = sprite.texture.get_size() * sprite.scale
				collision.shape.size = texture_size
func interact() -> void:
	if evidence_data == null:
		printerr("Помилка: На об'єкті %s не призначено файл EvidenceResource!" % name)
		return
	if not NotebookController._is_already_collected(evidence_data.evidence_id):
		NotebookController.add_evidence(evidence_data)
		print("Гравець підібрав доказ: ", evidence_data.title)
		queue_free()
	else:
```

```
            print("Цей доказ вже є у блокноті!")
```

Г.4 Код LevelManager.gd

```
extends Node2D

class_name LevelManager

@export var default_spawn_point_id: String = "Entrance"

@export var player_scene: PackedScene

@onready var spawn_points_container: Node2D = $SpawnPoints

@onready var interactive_objects_container: Node2D = $InteractiveObjects

var current_player: PlayerController

func _ready() -> void:

    setup_level()

func setup_level() -> void:

    _setup_player()

    _setup_interactive_objects()

    if Engine.has_singleton("EventBus"):

        var event_bus = Engine.get_singleton("EventBus")

        if event_bus.has_signal("dialogue_state_changed"):

            event_bus.dialogue_state_changed.emit(false)

func _setup_player() -> void:

    current_player = get_node_or_null("Player") as PlayerController

    if current_player == null and player_scene != null:
```

```
            current_player = player_scene.instantiate() as PlayerController
            current_player.name = "Player"
            add_child(current_player)
        if current_player != null:
            spawn_player(default_spawn_point_id)
func spawn_player(spawn_point_id: String) -> void:
        if spawn_points_container == null:
            printerr("LevelManager: Не знайдено контейнер SpawnPoints!")
            return
        var spawn_found = false
        for point in spawn_points_container.get_children():
            if point.name == spawn_point_id and point is Node2D:
                current_player.global_position = point.global_position
                spawn_found = true
                break
        if not spawn_found:
            printerr("LevelManager: Точку спавну '%s' не знайдено!" %
spawn_point_id)
func _setup_interactive_objects() -> void:
        if interactive_objects_container == null:
            return
            if Engine.has_singleton("NotebookController"):
```

```
		var notebook = Engine.get_singleton("NotebookController")
		for obj in interactive_objects_container.get_children():
			if "evidence_resource" in obj and obj.evidence_resource != null:
				var ev_id = obj.evidence_resource.evidence_id
				if notebook.has_method("_is_already_collected") and
notebook._is_already_collected(ev_id):
					obj.queue_free()
```

Додаток Д – ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ
(обов’язковий)

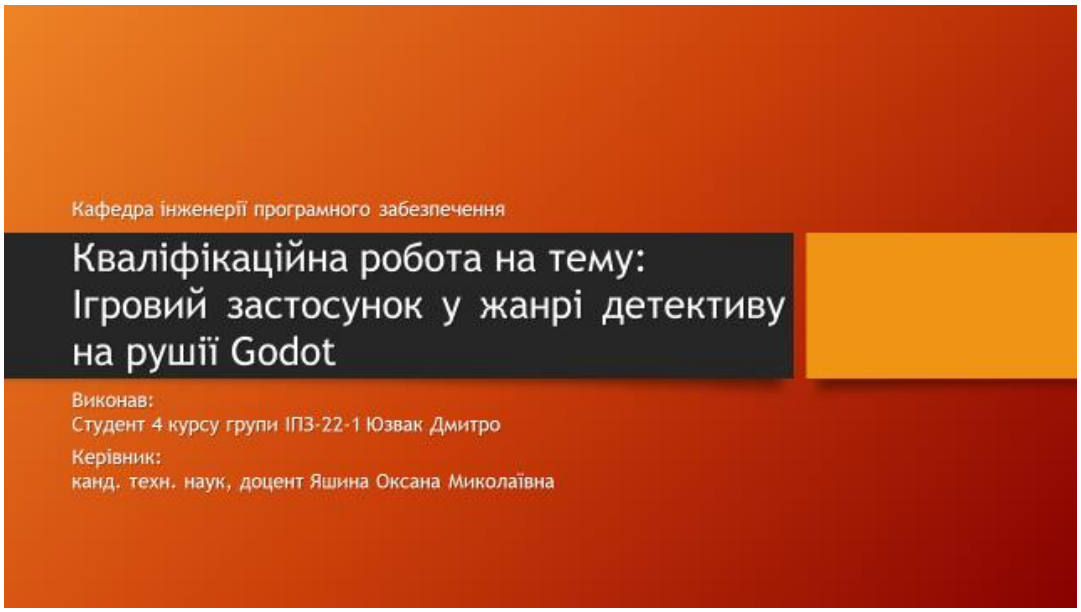


Рисунок Д.1 – Слайд 1 «Титульна сторінка»

Рисунок Д.2 – Слайд 2 «Актуальність теми»

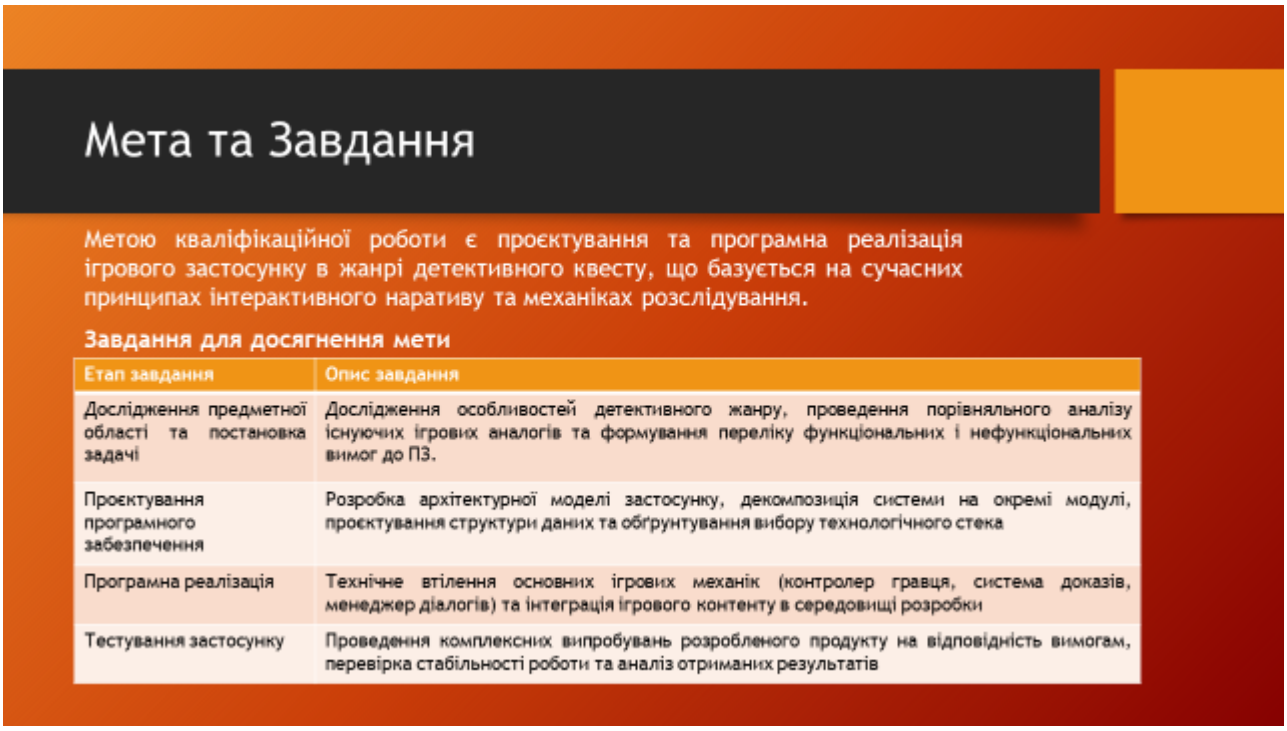


Рисунок Д.3 – Слайд 3 «Мета та завдання»

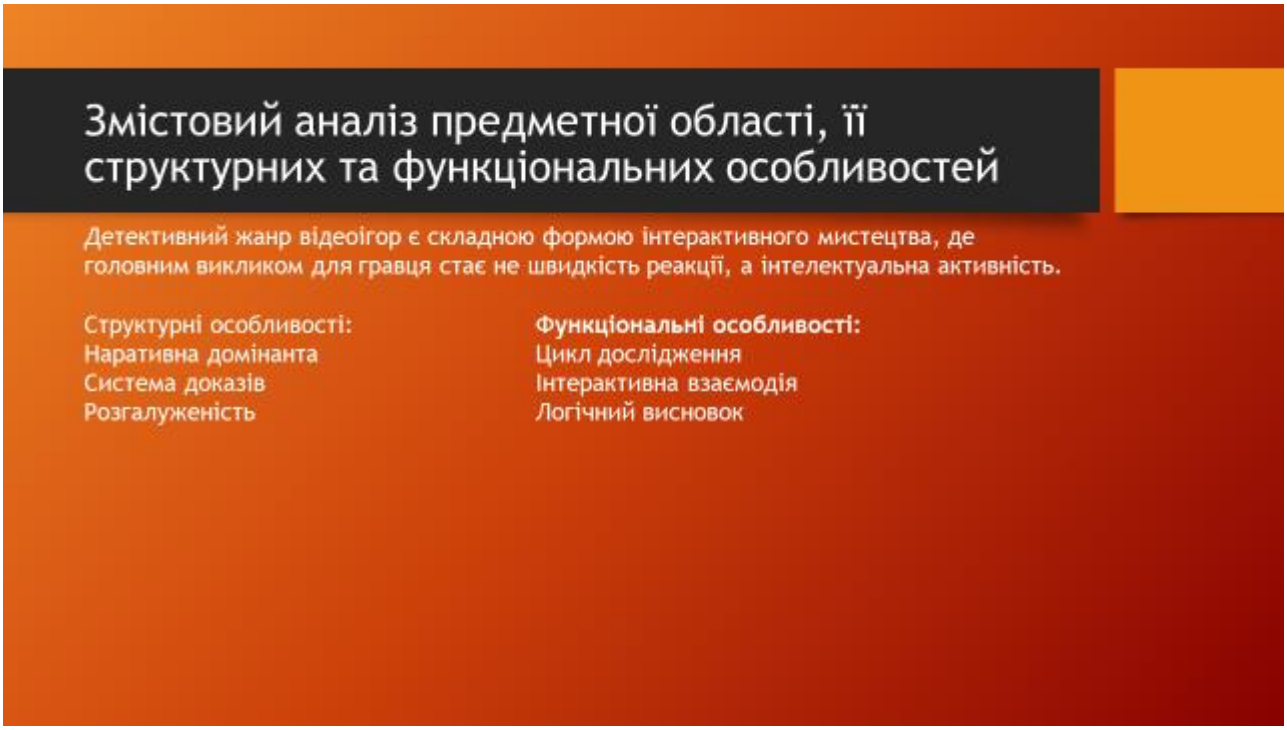


Рисунок Д.4 – Слайд 4 «Змістовий аналіз предметної області, її структурних та функціональних особливостей»

Аналіз наявного програмно-технічного забезпечення

Параметр	S.H. Crimes & Punishments	L.A. Noire	The Wolf Among Us	Disco Elysium
Основна механіка	Мапа дедукції	Розпізнавання міміки	Моральний вибір	Перевірка навичок
Тип розслідування	Логічні зв'язки	Психологічний допит	Наслідки рішень	Внутрішній діалог
Ступінь свободи	Лінійні локації	Відкритий світ	Користувацьке «кіно»	Висока (RPG)
Візуальний стиль	3D Реалізм	3D Фотореалізм	Cell-shading (2D)	Акварельне 2D

Рисунок Д.5 – Слайд 5 «Аналіз наявного програмно-технічного забезпечення»

Визначення функціональних та нефункціональних вимог до ПЗ

Функціональні вимоги	Нефункціональні вимоги
модульна структура ігрових сцен	продуктивність
дослідження локацій	надійність
система доказів та інвентарю	зручність використання
система діалогів	кросплатформеність
варіативність фіналів	масштабованість

Рисунок Д.6 – Слайд 6 «Визначення функціональних та нефункціональних вимог»

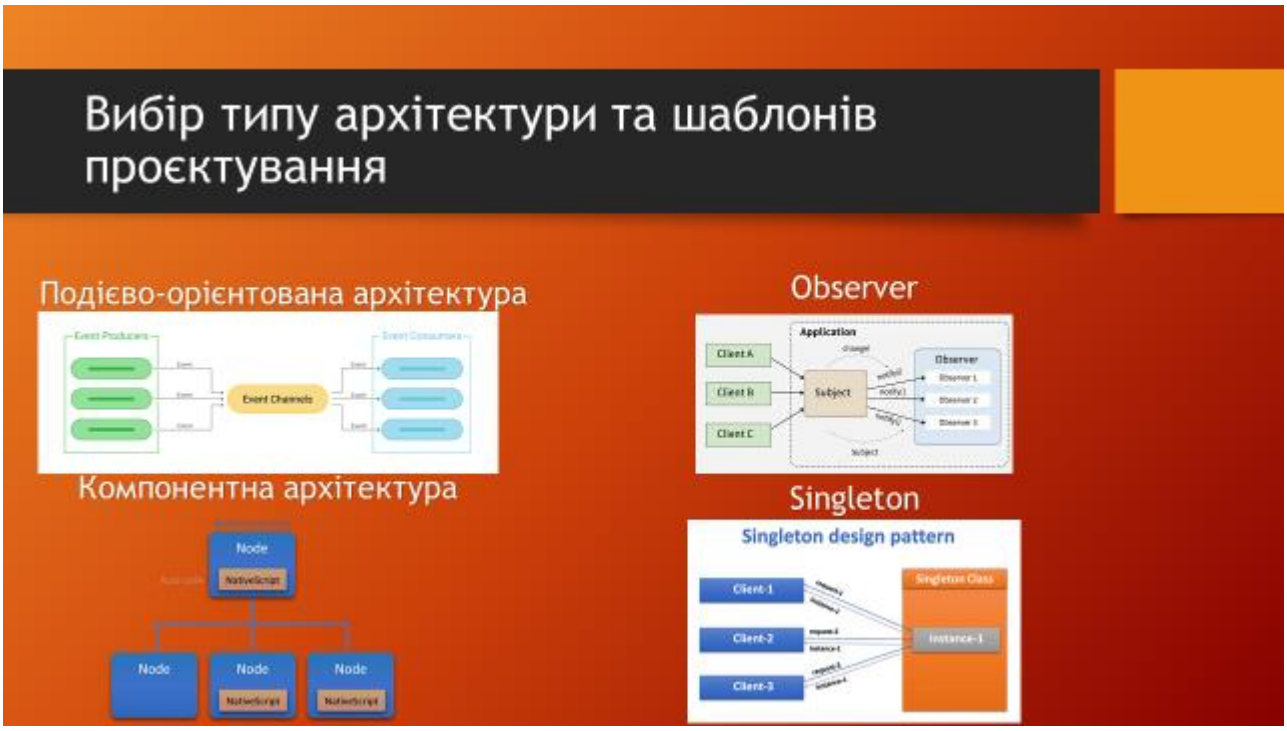


Рисунок Д.7 – Слайд 7 «Вибір типу архітектури та шаблонів проєктування»

Рисунок Д.8 – Слайд 8 «Опис декомпозиції, залежностей, інтерфейсів»

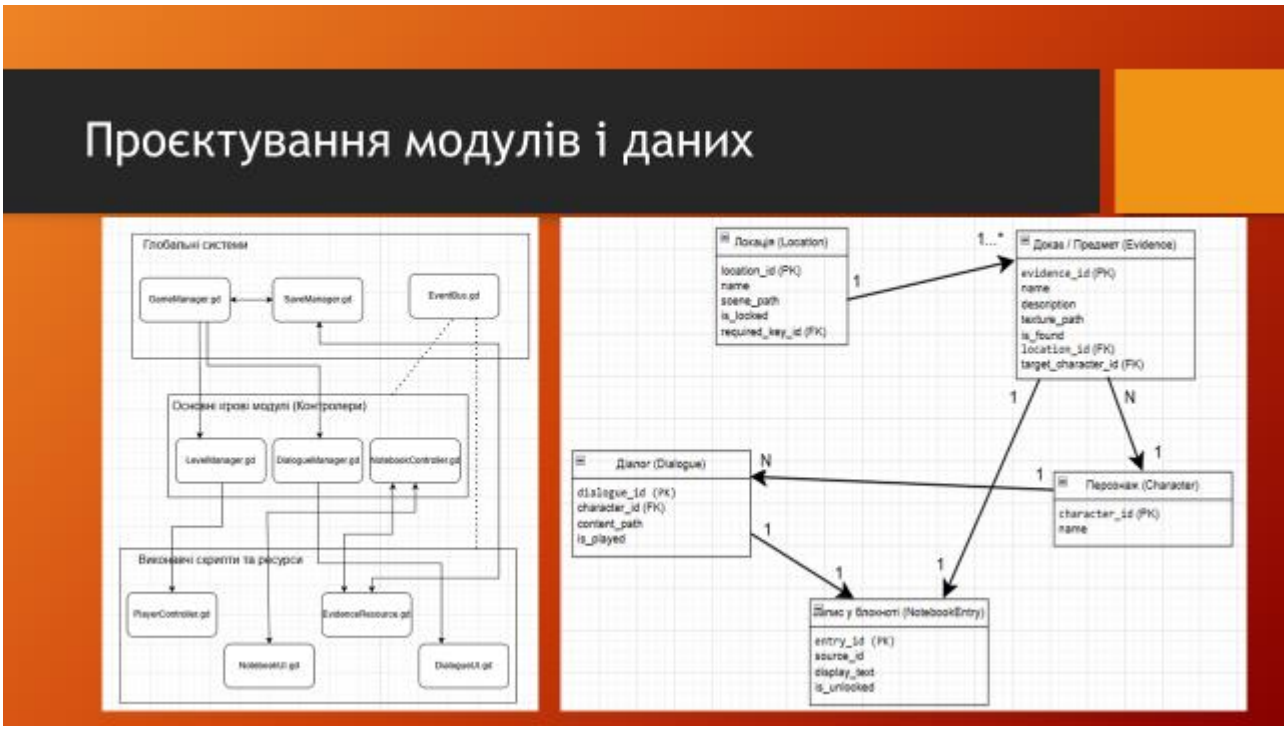


Рисунок Д.9 – Слайд 9 «Проєктування модулів і даних»

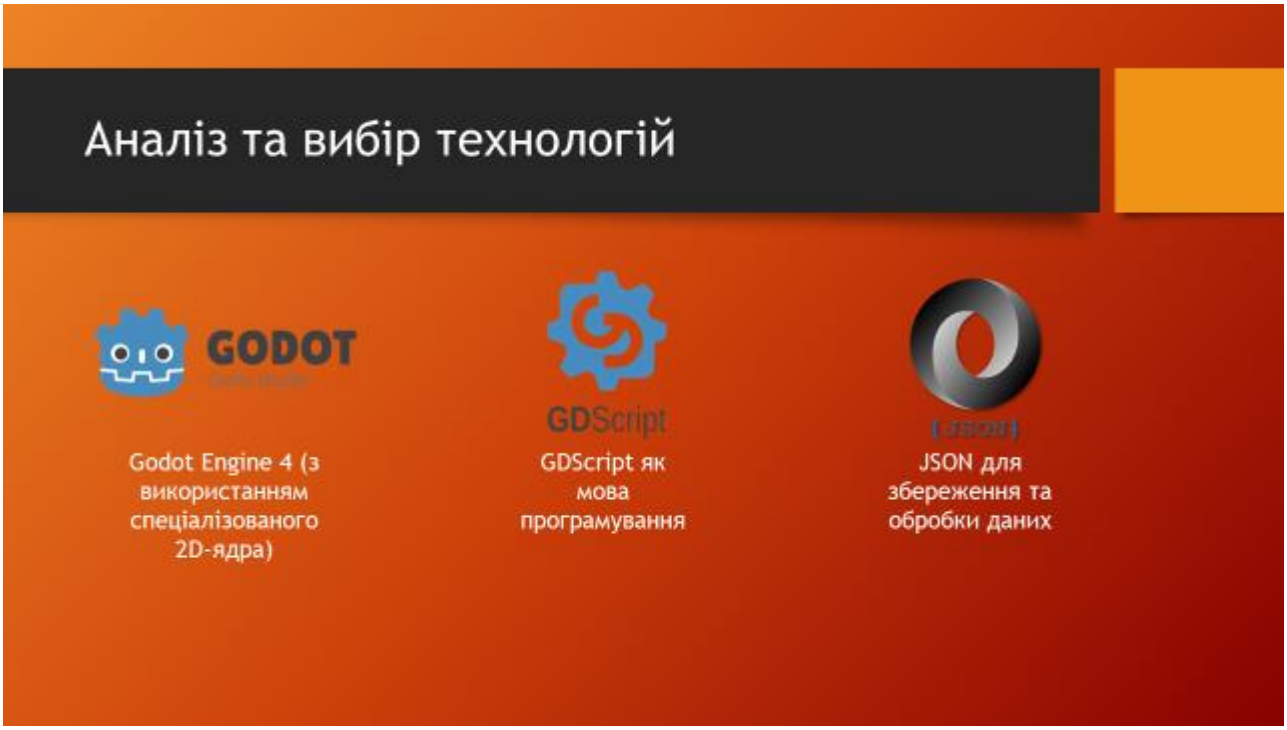


Рисунок Д.10 – Слайд 10 «Аналіз та вибір технологій»

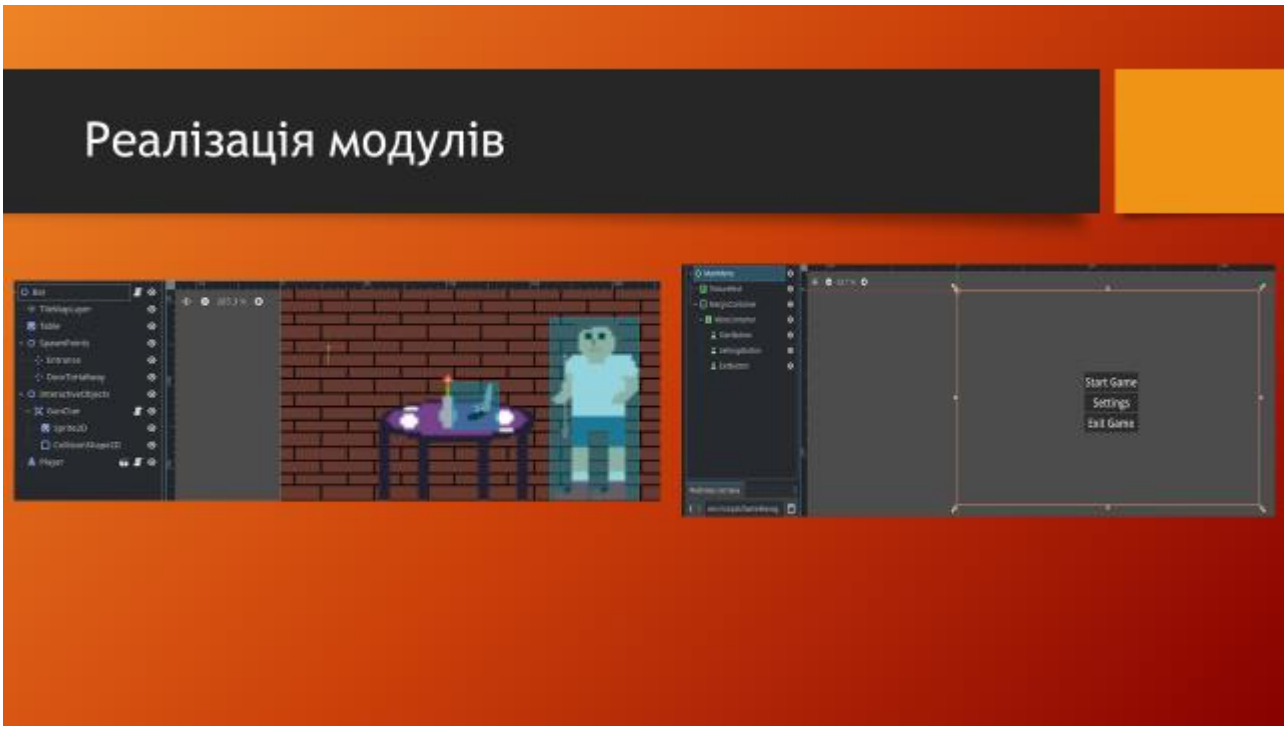


Рисунок Д.11 – Слайд 11 «Реалізація модулів»

Рисунок Д.12 – Слайд 12 «Вимоги до технічного та програмного забезпечення»

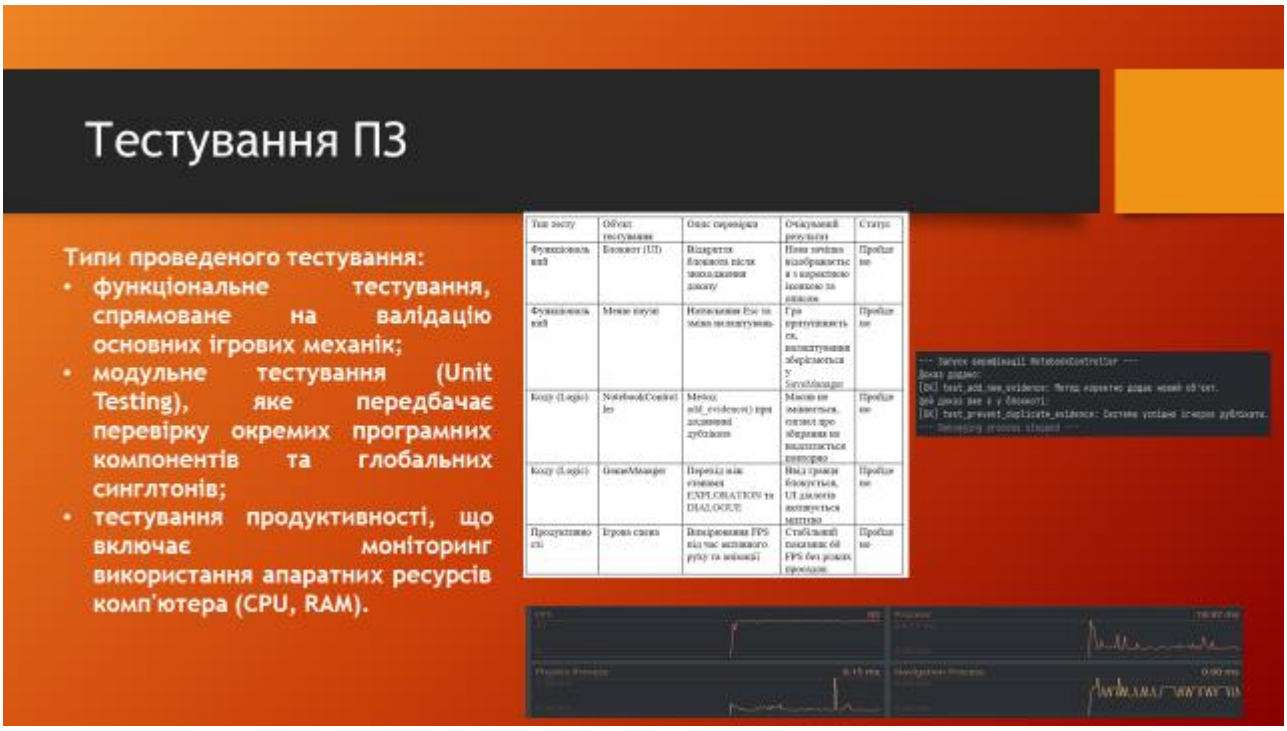


Рисунок Д.13 – Слайд 13 «Тестування ПЗ»

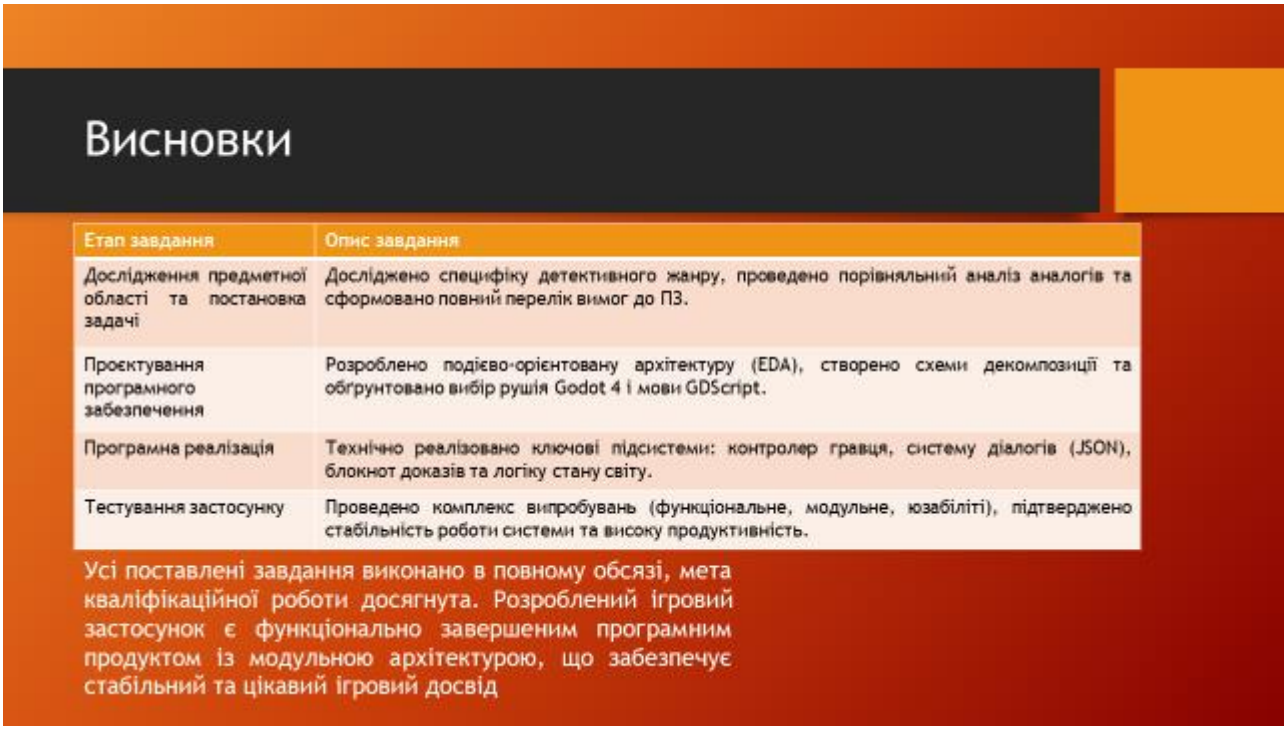


Рисунок Д.14 – Слайд 14 «Висновки»

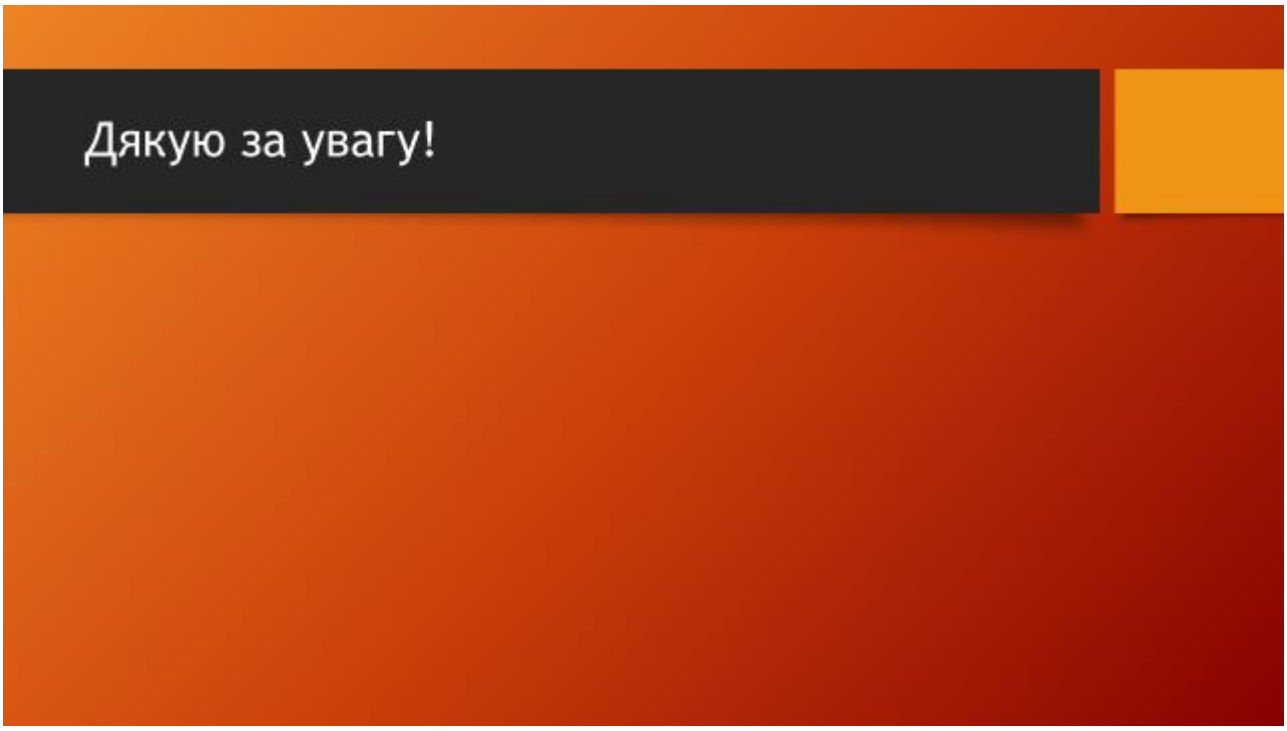
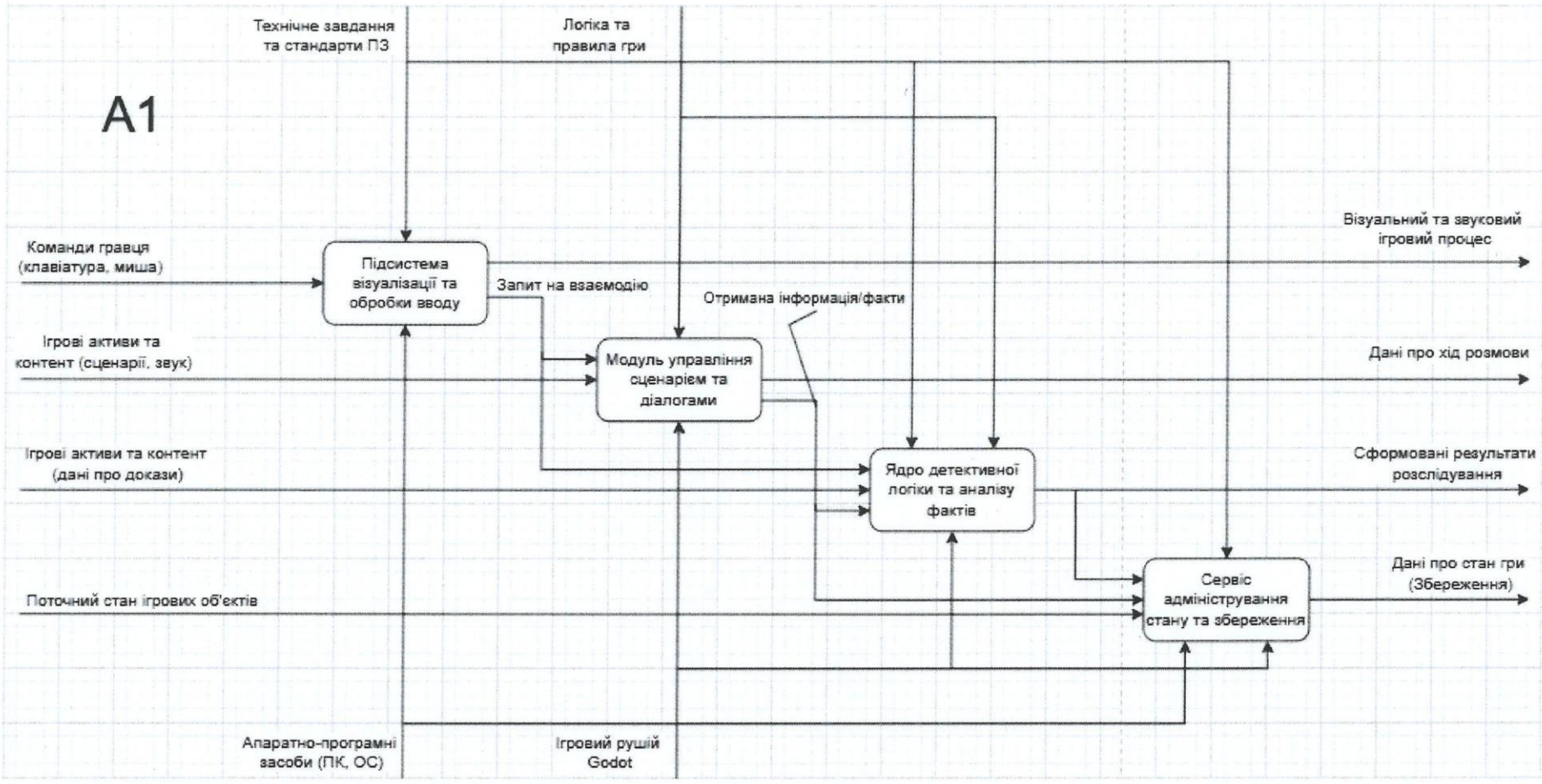


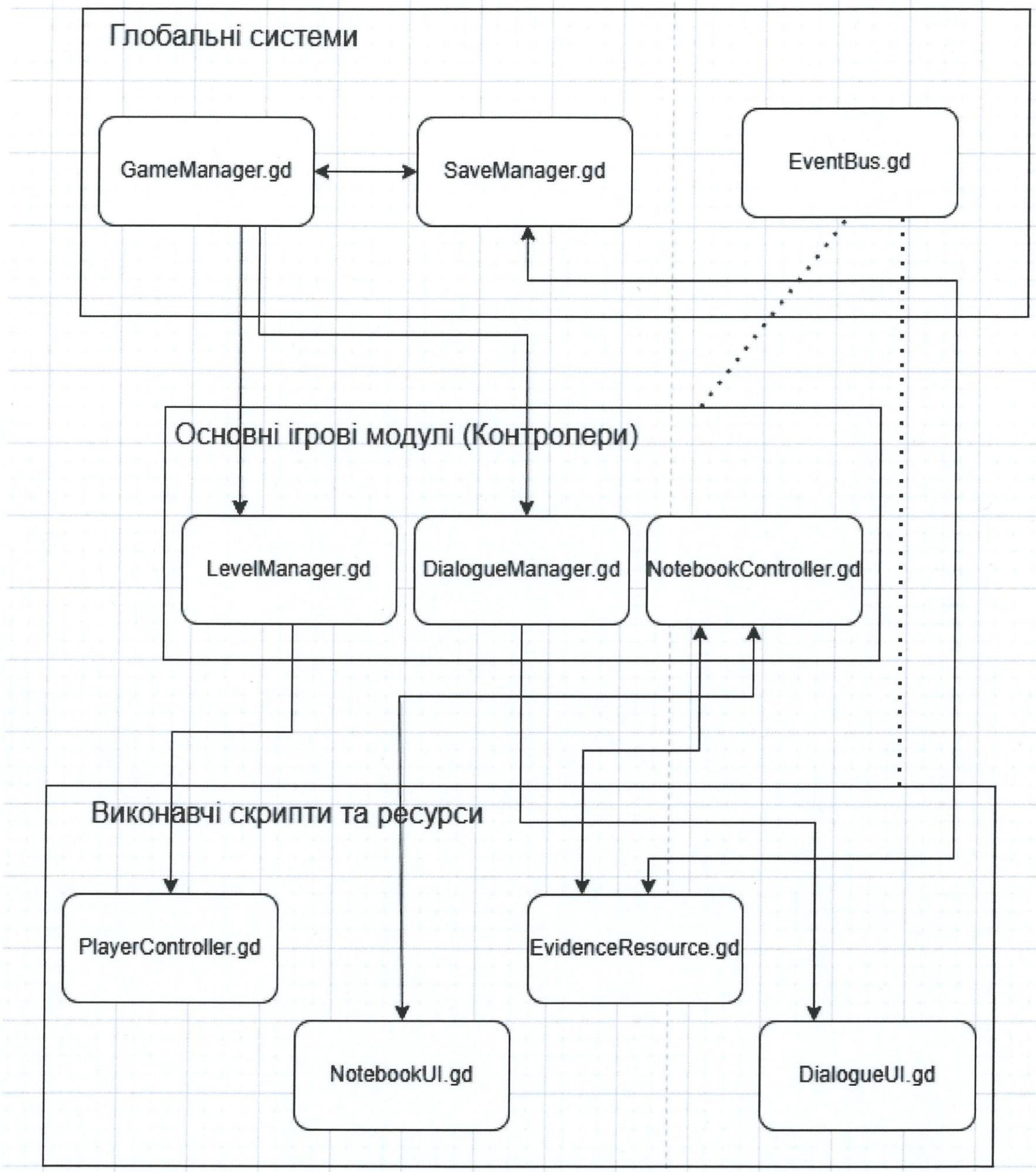
Рисунок Д.15 – Слайд 15 «Дякую за увагу»

ГРАФІЧНА ЧАСТИНА

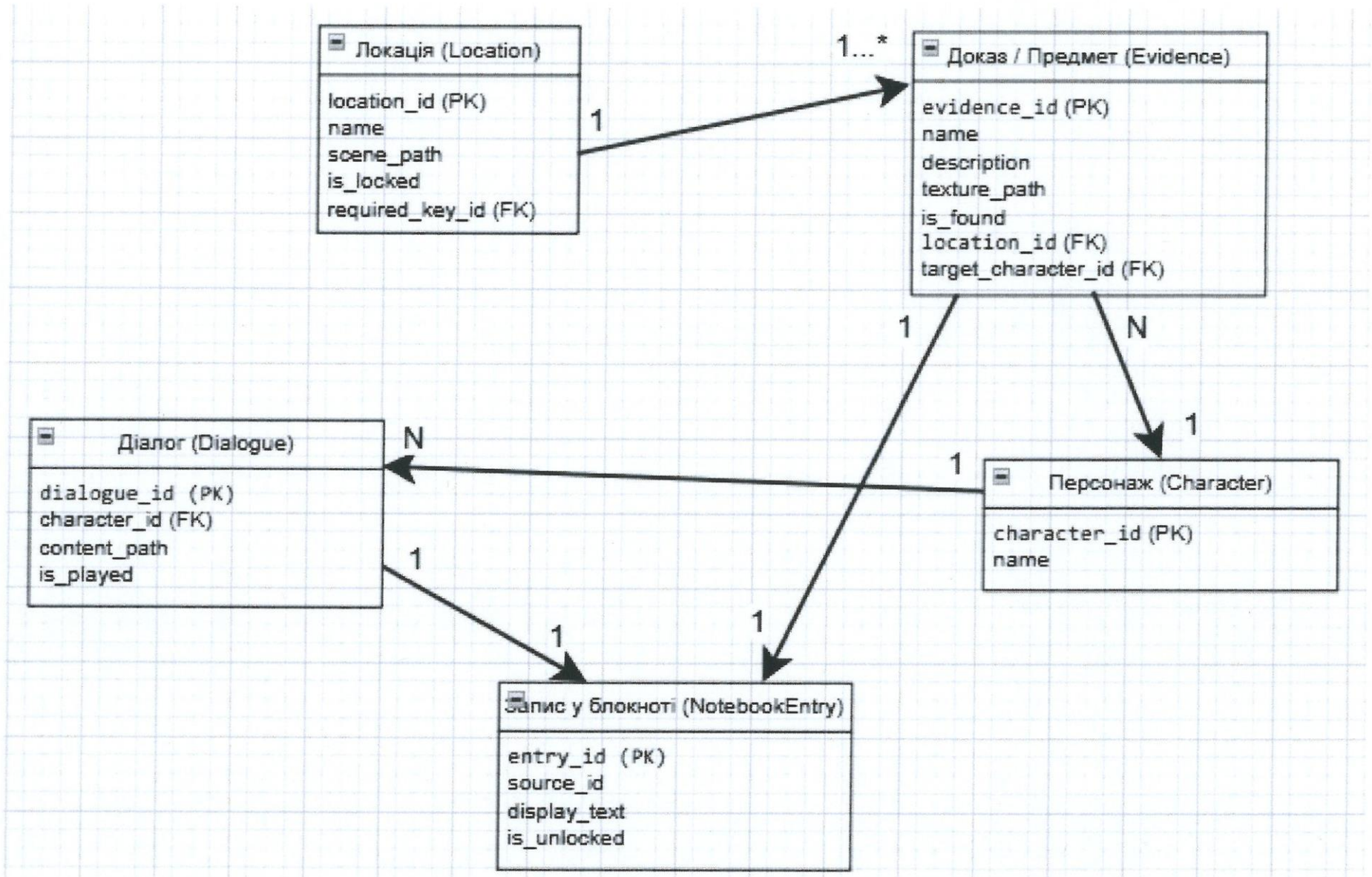
A1



Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2201118.01.23.ВД		
Виконав		Юзвак Д. О.		25.05	Ігровий застосунок у жанрі детективу на рушії Godot	Літ.	Арк. 1 / Аркушів 3
Керівник		Яшина О. М.		25.05			
Н. контр.		Форкун Ю. В.		25.05	Функціональна діаграма	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		26.05			



Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2201118.01.23.ВД		
Виконав		Юзвак Д. О.		25.05	Ігровий застосунок у жанрі детективу на рушії Godot	Літ.	Арк. 2 / Аркушів 3
Керівник		Яшина О. М.		25.05			
Н. контр.		Форкун Ю. В.		25.05	Модульна декомпозиція	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		26.05			



					КвРІПЗ. 2201118.01.23.ВД	
Змн.	Арк.	№ докум.	Підпис	Дата		
Виконав		Юзвак Д. О.		25.05	Ігровий застосунок у жанрі детективу на рушії Godot Діаграма сутність-зв'язок	Літ. / Арк. 3 / Аркушів 3
Керівник		Яшина О. М.		25.05		
Н. контр.		Форкун Ю. В.		25.05		ХНУ, ІПЗ-22-1
Зав. каф.		Бедратюк Л.П.		26.05		

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Юзвака Дмитра Олександровича
факультет ІТ, IV курс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

08.05.2026
дата

підпис

Wed May 20 00:50:31 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 11.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 10%**

ID: 271748 Назва: БКР_Ігровий застосунок у жанрі детективу на рушії Godot Додано в БД: 2026-05-20 Автора: Дмитро ЮЗВАК Керівники: канд. техн. наук, доцент Оксана ЯШИНА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	84372	553	11217 (13%)	100 (18%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269642	Назва: Переддипломна практика Додано в БД: 2026-03-02 Автора: Дмитро ЮЗВАК Керівники: Яшина О. М., канд. техн. наук, доцент Консультанти: Опоненти:	9141 (11.0%)	66 (12.0%)

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Дмитро ЮЗВАК

Співавтор:

Назва: Ігровий застосунок у жанрі детективу на рушії Godot

Науковий керівник: канд. техн. наук, доцент Оксана ЯШИНА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:0.75%

Коефіцієнт подібності 2:0%

Мікропробіли: 19

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-20 01:06:37.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 25.05.26 експерт Ю.В. Бакун

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Юзвак Дмитро Олександрович

Тема Ігровий застосунок у жанрі детективу на рушії Godot

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 69

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглядається процес створення двовимірного детективного квесту з використанням сучасних підходів до геймдизайну та програмування. Проведено аналіз предметної області та існуючих аналогів, на основі якого сформовано функціональні й нефункціональні вимоги до продукту. Обґрунтовано вибір рушія Godot, мови GDScript та формату JSON для обміну даними. Спроєктовано керовану даними архітектуру та реалізовано подієво-орієнтовану взаємодію компонентів. Розроблено ключові ігрові механіки: систему переміщення, пошуку доказів, розгалужених діалогів та електронного блокнота. Проведене багаторівневе тестування підтвердило стабільну та коректну роботу програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У кваліфікаційній роботі розглядається створення детективної гри на базі Godot Engine. У вступі обґрунтовано актуальність теми та визначено завдання проєктування. У першому розділі проаналізовано предметну область, досліджено наявні аналоги та сформульовано вимоги до програмного забезпечення. У другому розділі спроєктовано керовану даними архітектуру, логічну модель взаємодії ігрових сутностей та деталізовано програмні модулі. У третьому розділі реалізовано ключові компоненти системи (механіки пошуку доказів, діалоги, інтерфейс) і проведено комплексне тестування, яке підтвердило працездатність системи та її відповідність вимогам. Результатом роботи стало створення сучасного та надійного ігрового продукту, що забезпечує користувачу повноцінний інтерактивний досвід розслідування.

4. Позитивні сторони роботи Кваліфікаційна робота є актуальною, оскільки інтерактивний сторітелінг та детективні відеоігри користуються стабільним попитом на ринку, а розробка власного ігрового продукту на базі сучасного рушія Godot Engine пропонує ефективне та потужне рішення для створення такого програмного забезпечення. Робота вирізняється грамотним застосуванням принципів об'єктно-орієнтованого програмування та модульного проєктування. Заслуговує на високу оцінку продумане архітектурне проєктування, що включає вибір оптимальної керованої даними архітектури, розробку логічної моделі взаємодії ігрових сутностей, а також якісну програмну реалізацію з використанням сучасних інструментів та патернів.

5. Негативні сторони роботи У розробленій діалоговій системі було б доцільно додатково реалізувати функцію збереження історії повідомлень (log) для можливості повторного перегляду реплік NPC гравцем.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи. Представлені діаграми моделювання та схеми архітектурної взаємодії модулів. Пояснювальна записка структурована логічно і оформлена згідно з вимогами чинних державних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку. Матеріал пояснювальної записки структурований, послідовний та логічно вибудований, що дозволяє легко зрозуміти прийняті рішення у рамках тематики розробки ігрового застосунку. Наведений графічний матеріал дає можливість наочно оцінити архітектуру та функціонал створеного програмного забезпечення.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана на високому технічному рівні, повністю відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Савенко Олег Станіславович, доктор технічних наук, професор, професор кафедри комп'ютерної інженерії та інформаційних систем ЖНУ

"27" травня 2026 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Ігровий застосунок у жанрі детективу на рушії Godot»
Автор: Юзвак Дмитро Олександрович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Ящина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 11.0% з одного джерела. Загальна сумарна подібність у базі даних складає 13% за символами та 18% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 0.75%, коефіцієнт подібності 2 – 0%. Виявлено 19 мікропробілів, не виявлено зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 07.06.2026

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ЯШИНА