

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА
на тему: «Розробка ігрового симулятора прикордонного контролю»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артьом ДЕРКАЧ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
_____ Артьом ДЕРКАЧ

Керівник: _____ Оксана ЗОЛОТУХІНА
канд. техн. наук, доцент

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Деркачу Артьому Ігоровичу

1. Тема кваліфікаційної роботи: «Розробка ігрового симулятора прикордонного контролю»

керівник кваліфікаційної роботи канд. техн. наук, доцент Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних
технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: матеріали предметної галузі прикордонного контролю, аналіз ігрових симуляторів відповідної тематики, документація Unity та C#, вихідний код і ресурси Unity-проєкту «На Кордоні».

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі ігрових симуляторів професійної діяльності та існуючих програмних рішень.

2. Огляд засобів реалізації ігрового симулятора прикордонного контролю.

3. Проєктування архітектури, ігрового циклу, системи генерації кейсів, перевірки документів, оцінювання та прогресії.

4. Програмна реалізація та тестування ігрового симулятора прикордонного контролю.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма ігрового циклу.
6. Діаграма класів і компонентів.
7. Екранні форми симулятора.
8. Результати тестування.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Формування вимог до ігрового симулятора прикордонного контролю	08.03-14.03.2026	
4	Проектування архітектури та UML-діаграм симулятора	15.03-28.03.2026	
5	Програмна реалізація основних модулів гри	29.03-19.04.2026	
6	Тестування ігрового симулятора	20.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-10.05.2026	
8	Розробка демонстраційних матеріалів	11.05-17.05.2026	
9	Попередній захист роботи	18.05-22.05.2026	
10	Подання кваліфікаційної роботи	23.05-30.05.2026	

Здобувач вищої освіти

(підпис)

Артьом ДЕРКАЧ

Керівник

кваліфікаційної роботи

(підпис)

Оксана ЗОЛОТУХІНА

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 58 стор., 5 табл., 15 рис., 21 джерело.

Мета роботи - спрощення процесу підготовки та відпрацювання навичок перевірки документів та прийняття рішень інспектором прикордонного контролю з використанням ігрової симуляції.

Об'єкт дослідження – процеси перевірки документів та прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

Предмет дослідження – методи та програмні засоби ігрової симуляції процесів перевірки документів і підтримки прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

Короткий зміст роботи: У роботі проаналізовано предметну галузь ігрових симуляторів професійної діяльності та розглянуто програмні рішення, пов'язані з перевіркою документів і прикордонним контролем. Обґрунтовано вибір Unity, C#, Unity UI, TextMeshPro, Universal Render Pipeline, Git та Blender. Спроектовано архітектуру симулятора, ігровий цикл, систему генерації кейсів, перевірки документів, оцінювання та прогресії. Реалізовано основні програмні модулі: генерацію мандрівників, відображення паспорта й картки в'їзду, службовий монітор, панель рішень, рух NPC і транспорту, роботу шлагбаума та підсумок зміни. Проведено тестування основних функціональних можливостей гри.

Сферою використання симулятора є ознайомлення користувачів з логікою прикордонного контролю, тренування уважності, аналізу документів і прийняття рішень у типових та нестандартних ситуаціях.

КЛЮЧОВІ СЛОВА: ІГРОВИЙ СИМУЛЯТОР, ПРИКОРДОННИЙ КОНТРОЛЬ, UNITY, C#, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, NPC, 3D-ГРА

ЗМІСТ

ВСТУП.....	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ.....	11
1.1 Особливості ігрових симуляторів професійної діяльності.....	12
1.2 Аналіз існуючих програмних рішень	13
1.3 Визначення вимог до ігрового симулятора прикордонного контролю	18
1.4 Опис ігрових механік та сценаріїв перевірки документів	20
2 ЗАСОБИ РЕАЛІЗАЦІЇ ІГРОВОГО СИМУЛЯТОРА	23
2.1 Середовище розробки	23
2.2 Ігровий рушій Unity.....	24
2.3 Мова програмування C#.....	26
2.4 Засоби створення інтерфейсу та 3D-графіки	28
2.5 Система контролю версій та допоміжні інструменти	29
3 ПРОЄКТУВАННЯ ІГРОВОГО СИМУЛЯТОРА	32
3.1 Загальна архітектура системи.....	32
3.2 Проєктування ігрового циклу.....	34
3.3 Проєктування системи генерації кейсів мандрівників.....	37
3.4 Проєктування системи перевірки документів	40
3.5 Проєктування системи оцінювання та прогресії	43
3.6 UML-діаграми системи	47
4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ.....	51
4.1 Реалізація основного ігрового циклу	51
4.2 Реалізація генерації мандрівників і службових ситуацій	53
4.3 Реалізація UI, документів і службового монітора	55
4.4 Реалізація NPC, транспорту та шлагбаума.....	57
4.5 Тестування ігрового симулятора.....	60
ПЕРЕЛІК ПОСИЛАНЬ	66
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	68
ДОДАТОК В. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ	75

ВСТУП

Сучасні ігрові технології все частіше використовуються не лише для розваг, а й для моделювання різних професійних процесів. За допомогою симуляторів можна відтворити роботу певної системи або спеціаліста у віртуальному середовищі, де користувач має змогу взаємодіяти з об'єктами, аналізувати інформацію та приймати рішення. Такий підхід є зручним, оскільки дозволяє багаторазово проходити різні ситуації без ризику для реального середовища.

Прикордонний контроль є процесом, у якому важливе значення мають уважність, швидкість аналізу та правильність прийнятого рішення. Інспектор повинен перевіряти документи, зіставляти дані, звертати увагу на мету поїздки, інформацію про транспортний засіб та можливі ознаки порушень. Помилка у такому процесі може призвести до неправильного рішення: пропуску особи з порушенням або необґрунтованої відмови. Саме тому тема створення ігрового симулятора прикордонного контролю є актуальною.

Існуючі ігрові аналоги, пов'язані з перевіркою документів або роботою прикордонних служб, реалізують цю тему по-різному. Одні з них зосереджені переважно на перевірці документів, інші - на огляді транспорту або загальній атмосфері службової роботи. Водночас не всі рішення поєднують у собі процедурну генерацію ситуацій, службовий монітор, перевірку документів і даних автомобіля, систему оцінювання, механіку хабаря та повноцінний 3D-процес роботи контрольно-пропускного пункту.

У межах кваліфікаційної роботи розробляється ігровий симулятор прикордонного контролю «На Кордоні». Програмний продукт реалізовано на ігровому рушії Unity з використанням мови програмування C#. Гравець виконує роль інспектора прикордонного контролю, який під час зміни приймає мандрівників, перевіряє паспорт, картку мети в'їзду, службовий монітор, дані автомобіля та поведінку особи. Після перевірки користувач приймає рішення: дозволити в'їзд, відмовити у перетині кордону або прийняти хабар.

Особливістю розробленого симулятора є процедурна генерація службових кейсів. У грі можуть виникати різні ситуації: дійсні документи, прострочений паспорт, невідповідність персональних даних, підозріла мета поїздки, неповна інформація, спроба хабаря, невідповідність транспортного засобу або службове орієнтування. Завдяки цьому ігровий процес не є одноманітним, а користувач щоразу має аналізувати нові умови.

Об'єктом дослідження є процеси перевірки документів та прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

Предметом дослідження є методи та програмні засоби ігрової симуляції процесів перевірки документів і підтримки прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

Метою роботи є спрощення процесу підготовки та відпрацювання навичок перевірки документів та прийняття рішень інспектором прикордонного контролю з використанням ігрової симуляції.

Для досягнення поставленої мети необхідно виконати такі завдання:

- 1) проаналізувати предметну галузь ігрових симуляторів професійної діяльності;
- 2) розглянути існуючі програмні рішення, пов'язані з прикордонним або документальним контролем;
- 3) визначити функціональні та нефункціональні вимоги до симулятора «На Кордоні»;
- 4) обґрунтувати вибір Unity, C# та інших засобів реалізації;
- 5) спроектувати архітектуру симулятора, ігровий цикл і основні програмні модулі;
- 6) реалізувати генерацію мандрівників, перевірку документів, службовий монітор, систему рішень, оцінювання, NPC, транспорт і шлягбаум;
- 7) провести тестування основних функцій гри.

У роботі використано методи аналізу предметної галузі, порівняльного аналізу аналогів, об'єктно-орієнтованого проектування, UML-моделювання, процедурної генерації ігрових ситуацій та функціонального тестування.

Практична новизна роботи полягає у поєднанні в одному 3D-симуляторі процедурної генерації службових кейсів, зв'язки даних між паспортом, карткою в'їзду, службовим монітором і автомобілем, механіки хабаря, показника службової цілісності, системи прогресії інспектора та анімованого руху мандрівників і транспорту на КПП. Розроблений симулятор може використовуватися як інтерактивний програмний продукт для ознайомлення з логікою прикордонного контролю, тренування уважності, аналізу документів і прийняття рішень у типових та нестандартних ситуаціях.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань та додатків. У першому розділі розглянуто предметну галузь та аналоги. У другому розділі обґрунтовано вибір засобів реалізації. У третьому розділі описано проектування системи. У четвертому розділі наведено реалізацію та тестування ігрового симулятора.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Ігровий симулятор прикордонного контролю належить до програмних продуктів, у яких ігрова взаємодія поєднується з моделюванням окремих елементів професійної діяльності. У такому застосунку важливо не тільки створити візуальне середовище, а й правильно передати логіку процесу: послідовність перевірки особи, роботу з документами, аналіз службової інформації, оцінювання ризикових ситуацій та прийняття рішення користувачем.

Для проєкту «На Кордоні» предметна галузь охоплює змодельований процес роботи інспектора на контрольно-пропускному пункті. Гравець взаємодіє з мандрівниками, перевіряє паспортні дані, картку мети в'їзду, інформацію службового монітора, дані транспортного засобу та поведінкові ознаки персонажа. На основі цих даних користувач повинен визначити, чи можна дозволити перетин кордону, чи необхідно відмовити у в'їзді. Додатковим ускладненням є наявність нестандартних ситуацій, зокрема спроби хабаря, невідповідності документів або службового орієнтування.

Особливість цієї предметної галузі полягає в тому, що вона потребує балансу між реалістичністю та зручністю ігрового процесу. Надмірне ускладнення правил може зробити гру незручною для користувача, а занадто спрощена перевірка документів знижує цінність симуляції. Тому під час аналізу важливо визначити, які елементи прикордонного контролю доцільно перенести в гру, а які потрібно подати у спрощеній, але логічно зрозумілій формі.

У цьому розділі розглядаються загальні особливості симуляторів професійної діяльності, аналізуються існуючі ігрові рішення, пов'язані з перевіркою документів та прикордонним контролем, а також визначаються вимоги до власного програмного продукту. Результати аналізу використовуються для подальшого вибору засобів реалізації, проєктування архітектури симулятора та опису основних функціональних модулів гри.

1.1 Особливості ігрових симуляторів професійної діяльності

Ігрові симулятори професійної діяльності відрізняються від більшості розважальних ігор тим, що їх основою є не лише сюжет або візуальна складова, а відтворення певного робочого процесу. У таких програмних продуктах користувач виконує роль спеціаліста, який діє в умовах, наближених до конкретної професійної ситуації. При цьому гра не повинна повністю копіювати реальну діяльність, оскільки надмірна деталізація може зробити процес складним і незручним. Важливішим є правильний вибір тих елементів, які формують суть професійної задачі.

Для симулятора прикордонного контролю такими елементами є перевірка документів, порівняння даних, оцінювання мети поїздки, аналіз службової інформації та прийняття рішення щодо пропуску особи. Користувач має не просто натискати на кнопки, а працювати з набором даних, у якому можуть бути приховані помилки або суперечності. Саме тому основою ігрового процесу стає уважність, логічне зіставлення інформації та здатність швидко визначити, чи відповідає ситуація встановленим правилам.

У професійно-орієнтованих симуляторах важливе місце займає повторюваний цикл дій. Користувач отримує нову ситуацію, вивчає доступну інформацію, виконує перевірку, приймає рішення та отримує результат. Такий принцип дозволяє поступово ускладнювати гру без різкої зміни її механіки. У випадку прикордонного контролю це може проявлятися у появі різних типів мандрівників, документів, транспортних засобів, службових повідомлень і нестандартних подій.

Окремою вимогою до такого типу ігор є зрозумілий інтерфейс. У симуляторі користувач постійно працює з інформацією, тому документи, службовий монітор, кнопки рішень і показники результатів мають бути розміщені так, щоб не заважати аналізу. Інтерфейс повинен підтримувати ігровий процес, а не відволікати від

нього. Для проєкту «На Кордоні» це особливо важливо, оскільки основна дія відбувається навколо перевірки паспорта, картки мети в'їзду, даних автомобіля та службової бази.

Ефективність прямо залежить від варіативності сценаріїв. Одноманітні завдання швидко перетворюються на механічне повторення, через що симулятор втрачає свою навчальну цінність. Натомість процедурна генерація ситуацій дозволяє щоразу створювати унікальні комбінації умов та порушень. Це змушує користувача критично оцінювати кожну задачу, а не діяти за зазубреним шаблоном.

У проєкті «На Кордоні» професійна діяльність інспектора подається у спрощеній, але логічно зрозумілій ігровій формі. Гравець перевіряє мандрівників, аналізує документи та службову інформацію, після чого приймає рішення про дозвіл або заборону в'їзду. Додатково гра враховує нестандартні ситуації, зокрема невідповідність даних, службове орієнтування, проблеми з транспортним засобом або спробу хабаря. Це дозволяє поєднати елементи симуляції, логічної гри та системи оцінювання рішень.

Отже, ігровий симулятор професійної діяльності має поєднувати доступність гри з логікою реального процесу. Для теми прикордонного контролю ключовими є робота з документами, аналіз даних, варіативність службових ситуацій, зрозумілий інтерфейс і система оцінювання дій користувача. Саме ці особливості враховуються під час подальшого аналізу аналогів, формування вимог і проєктування симулятора «На Кордоні».

1.2 Аналіз існуючих програмних рішень

Перед розробкою симулятора «На Кордоні» було розглянуто кілька ігор, у яких уже реалізовано близькі за змістом механіки: перевірку документів, контроль відвідувачів, роботу службовця, огляд транспорту або прийняття рішень у межах заданих правил. Ці проєкти не повторюють повністю задум власної розробки, але

дають змогу зрозуміти, які підходи вже використовуються в жанрі та які обмеження варто врахувати під час створення нового програмного продукту.

Одним із найбільш відомих прикладів є Papers, Please (див. рис. 1.1). У цій грі користувач виконує роль інспектора, який перевіряє документи людей на пункті пропуску. Основна увага приділяється пошуку помилок у датах, іменах, дозволах, фотографіях та інших даних. Сильною стороною гри є те, що вона добре передає напруження роботи з великою кількістю інформації та змушує уважно порівнювати документи. Водночас гра виконана у 2D-стилі й не показує повноцінну роботу контрольно-пропускного пункту як простору з транспортом, NPC та анімованим середовищем.

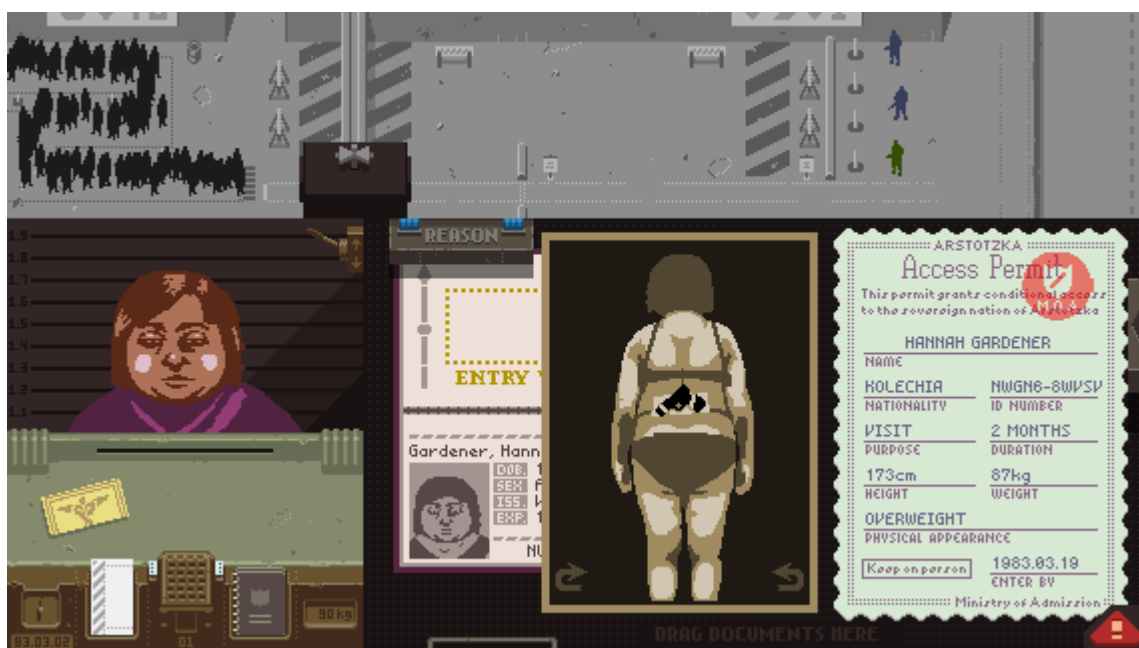


Рис.1.1 Інтерфейс гри Papers, please

Гра Border Officer (див. рис. 1.2) ближча до 3D-симулятора, оскільки в ній є переміщення по локації та ширше відчуття присутності. Проект поєднує службову роботу з додатковими побутовими елементами, тому прикордонний контроль не завжди залишається єдиним центром ігрового процесу. Це рішення є корисним для аналізу 3D-подачі, але перевірка документів у ньому не має такої деталізації, яка потрібна для симулятора, орієнтованого саме на логіку контролю даних.



Рис.1.2 Інтерфейс гри Border Officer

У Contraband Police (див. рис. 1.3) основний акцент зроблено на огляді транспортних засобів, пошуку заборонених предметів і роботі у тривимірному середовищі. Цей проєкт має високий рівень інтерактивності, оскільки гравець може взаємодіяти з автомобілями та об'єктами в ігровому світі. Проте така деталізація вимагає значних апаратних ресурсів, а сама механіка більше спрямована на пошук контрабанди, ніж на порівняння документів, службових записів і персональних даних.



Рис.1.3 Ігровий процес Contraband Police

Ще одним прикладом є Not Tonight (див. рис. 1.4). У цій грі користувач також перевіряє документи та приймає рішення щодо допуску персонажів. Перевагою проєкту є швидкий темп і зрозумілий цикл дій. Водночас перевірка документів подана досить спрощено, а сама гра більше орієнтована на стилізовану подачу та сатиричний контекст, ніж на моделювання роботи прикордонного пункту.

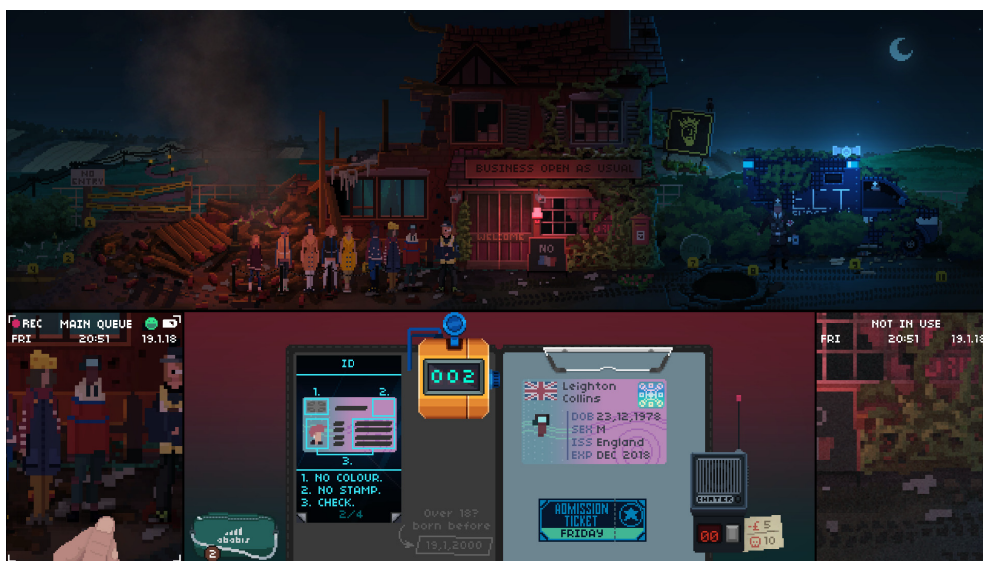


Рис.1.4 Інтерфейс гри Not Tonight

Для порівняння розглянутих ігор із власною розробкою було сформовано таблицю 1.1. У ній «На Кордоні» подано як розроблюваний проєкт, а не як сторонній аналог

Специфіка ігрового моделювання професійної діяльності вимагає чіткого балансу: гра має бути достатньо реалістичною, щоб відображати суть роботи інспектора, але водночас залишатися динамічним програмним продуктом. Щоб зрозуміти, які рішення у цьому напрямі вже пройшли перевірку практикою та аудиторією, необхідно звернутися до досвіду розробників схожих ігор. Дослідження існуючих аналогів допоможе запозичити вдалі інтерфейсні знахідки, уникнути типових геймдизайнерських помилок та чітко окреслити конкурентні переваги майбутнього симулятора «На Кордоні».

Таблиця 1.1

Порівняльна характеристика існуючих ігрових рішень з «На Кордоні»

Критерій	Papers, Please	Border Officer	Contraband Police	Not Tonight	«На Кордоні»
Формат гри	2D	3D	3D	2D	3D
Основна дія	Перевірка документів	Робота прикордонника і побутові задачі	Огляд авто та пошук порушень	Контроль відвідувачів	Перевірка мандрівників на КПП
Робота з документами	Детальна	Середня	Допоміжна	Спрощена	Паспорт, картка в'їзду, службові дані
Транспорт	Немає	Обмежено	Детальний огляд	Немає	Дані авто, рух транспорту, шлагбаум
Службова база / монітор	Частково	Обмежено	Частково	Обмежено	Реалізовано службовий монітор
Варіативність ситуацій	Наявна	Обмежена	Наявна	Обмежена	Процедурна генерація службових кейсів
Хабар або етичний вибір	Моральний тиск через сюжет	Окремі ситуації	Не основна механіка	Другорядно	Окрема дія з впливом на цілісність служби
Прогресія гравця	Обмежена	Часткова	Наявна	Обмежена	Досвід, рівень і покращення інспектора

Порівняння показує, що кожен із розглянутих проєктів має окремі сильні сторони. Papers, Please добре реалізує логіку перевірки документів, Border Officer використовує 3D-простір, Contraband Police демонструє глибшу взаємодію з транспортом, а Not Tonight має простий і швидкий цикл прийняття рішень. Разом з тим ці рішення не поєднують у межах одного продукту перевірку паспорта, картки в'їзду, службового монітора, даних автомобіля, механіку хабаря, оцінювання рішень, прогресію інспектора та анімований 3D-процес роботи КПП.

1.3 Визначення вимог до ігрового симулятора прикордонного контролю

Після аналізу предметної галузі та існуючих ігрових рішень можна сформулювати вимоги до власного програмного продукту. Для симулятора «На Кордоні» важливо не обмежуватися лише візуальним відтворенням контрольно-пропускного пункту. Основна цінність гри полягає в тому, що користувач має послідовно працювати з інформацією, знаходити невідповідності та приймати рішення, яке впливає на результат зміни.

Під час формування вимог враховано, що симулятор повинен бути зрозумілим для користувача без спеціальної підготовки, але водночас містити достатньо ситуацій для аналізу. Тому перевірка мандрівника у грі поділяється на кілька простих, але пов'язаних між собою етапів: ознайомлення з документами, перегляд службової інформації, аналіз даних автомобіля, оцінювання поведінки персонажа та вибір рішення. Такий підхід дозволяє зробити ігровий процес логічним і не перевантажувати користувача зайвими службовими деталями.

Функціональні вимоги до симулятора наведено в таблиці 1.2.

Таблиця 1.2

Функціональні вимоги до ігрового симулятора «На Кордоні»

Позначення	Вимога	Короткий опис
ФВ-1	Головне меню	Користувач повинен мати можливість розпочати нову гру, перейти до налаштувань або завершити роботу з програмою.
ФВ-2	Запуск ігрової зміни	Система повинна формувати зміну, у межах якої гравець послідовно перевіряє мандрівників.
ФВ-3	Генерація мандрівників	Гра повинна створювати різні службові кейси з коректними або проблемними даними.
ФВ-4	Відображення паспорта	Користувач повинен бачити основні паспортні дані: ПІБ, дату народження, громадянство, номер паспорта та строк дії.
ФВ-5	Відображення картки в'їзду	Система повинна показувати мету поїздки, адресу перебування, приймаючу сторону, строк візиту та пов'язані дані.
ФВ-6	Службовий монітор	У грі має бути екран службової інформації для перевірки особи, маршруту, автомобіля, ризику та рекомендацій.
ФВ-7	Перевірка транспортного засобу	Симулятор повинен враховувати номер, модель або інші дані автомобіля, які можуть збігатися або не збігатися з документами.

Продовження таблиці 1.2

Функціональні вимоги до ігрового симулятора «На Кордоні»

Позначення	Вимога	Короткий опис
ФВ-8	Прийняття рішення	Гравець повинен мати змогу дозволити в'їзд, відмовити у перетині кордону або прийняти хабар у відповідних ситуаціях.
ФВ-9	Оцінювання дій користувача	Система повинна нараховувати бали за правильні рішення та застосовувати штрафи за помилки.
ФВ-10	Показник службової цілісності	У разі прийняття хабаря має знижуватися показник цілісності служби.
ФВ-11	Прогресія інспектора	Гра повинна підтримувати накопичення досвіду, підвищення рівня та покращення окремих навичок.
ФВ-12	Рух NPC і транспорту	Симулятор повинен відтворювати під'їзд автомобіля, вихід мандрівника, передачу документів і виїзд після рішення.
ФВ-13	Пауза та налаштування	Користувач повинен мати можливість призупинити гру, змінити гучність і чутливість керування.
ФВ-14	Підсумок зміни	Після завершення перевірки мандрівників система повинна показувати результат зміни, кількість правильних рішень, помилки та отриманий досвід.

Основою ігрового розмаїття симулятора прикордонного контролю є функція генерації службових кейсів. Циклічність або ідентичність помилок у документах мандрівників призведе до швидкого звикання користувача та механічного виконання завдань. Щоб цього уникнути, алгоритм має динамічно моделювати різні типи ситуацій: від повністю легального перетину до виявлення прострочених паспортів, розбіжностей у персональних даних чи непевної мети візиту. Також система повинна комбінувати критичні випадки, як-от спроби підкупу, технічні несправності транспорту, брак документів або роботу за службовими орієнтуваннями.

Нефункціональні вимоги визначають не окремі дії користувача, а загальні характеристики роботи програмного продукту. Для гри важливими є стабільність, швидкість реакції інтерфейсу, зручність керування та можливість запуску на звичайному персональному комп'ютері. Такі вимоги наведено в таблиці 1.3.

Таблиця 1.3

Нефункціональні вимоги до ігрового симулятора «На Кордоні»

Позначення	Вимога	Критерій виконання
НФВ-1	Платформа запуску	Робота на Windows 10/11 64-bit.
НФВ-2	Продуктивність	Підтримка стабільного ігрового процесу з цільовим показником до 60 кадрів/с на рекомендованій конфігурації.
НФВ-3	Час реакції інтерфейсу	Натискання кнопок рішення, паузи та меню має оброблятися без помітної затримки для користувача(до 0.4 с).
НФВ-4	Зручність керування	Основні дії повинні виконуватися через зрозумілі клавіші або кнопки інтерфейсу(W,A,S,D,F,G,H, ЛКМ, ПКМ).
НФВ-5	Читабельність документів	Текст у паспорті, картці в'їзду та службовому моніторі має бути достатньо великим для читання під час гри(1/5 від екрану гравця).
НФВ-6	Збереження прогресу	Дані про рівень, досвід і покращення користувача повинні зберігатися між ігровими сесіями.
НФВ-7	Локалізація	Основні елементи інтерфейсу, документи та повідомлення мають бути українською мовою.
НФВ-8	Стабільність	Під час проходження зміни гра не повинна аварійно завершувати роботу при стандартному сценарії використання.

Сформовані вимоги показують, що симулятор «На Кордоні» повинен поєднувати кілька рівнів взаємодії: роботу з документами, аналіз службових даних, оцінювання транспорту, прийняття рішення та отримання результату. Це відрізняє проєкт від простого візуального прототипу, оскільки в основі гри лежить не лише 3D-сцена, а система правил, перевірок і наслідків для користувача.

Визначені функціональні та нефункціональні вимоги є основою для подальшого вибору інструментів розробки, побудови архітектури симулятора та реалізації його основних модулів. У наступних розділах ці вимоги використовуються для обґрунтування вибору технологій, проєктування структури програмного продукту та перевірки роботи реалізованої гри.

1.4 Опис ігрових механік та сценаріїв перевірки документів

У симуляторі «На Кордоні» перевірка мандрівника побудована навколо одного службового випадку. Кожен такий випадок містить інформацію про людину, її документи, автомобіль, службові примітки та рішення, яке має прийняти

гравець. Саме ці дані формують основу ігрової ситуації та визначають, чи повинен інспектор дозволити перетин кордону, чи відмовити у в'їзді.

Першим елементом інформаційної моделі є мандрівник. Для нього задаються персональні відомості: ім'я, громадянство, дата народження, професія, поведінка під час перевірки та інші дані, які можуть допомогти або, навпаки, ускладнити прийняття рішення. У грі ця інформація не подається одним списком, а розподіляється між різними джерелами. Частину даних гравець бачить у паспорті, частину - у картці мети в'їзду, а додаткові відомості отримує через службовий монітор.

Паспорт використовується як головний документ для встановлення особи. У ньому містяться основні дані мандрівника, номер документа, громадянство, дата видачі та строк дії. Картка мети в'їзду доповнює перевірку інформацією про причину поїздки, адресу перебування, приймаючу сторону, транспортний засіб та інші деталі. Завдання гравця полягає в тому, щоб порівняти ці відомості між собою і помітити можливі розбіжності.

Службовий монітор у моделі виконує роль внутрішнього джерела перевірки. Він показує інформацію, якої немає безпосередньо в документах, зокрема службові попередження, дані про маршрут, ризик, транспорт або рекомендацію щодо подальших дій. Завдяки цьому гравець не обмежується переглядом паспорта й картки, а має працювати з кількома джерелами одночасно.

Окремо в симуляторі враховується автомобіль, на якому прибув мандрівник. Його номер, модель або інші ознаки можуть відповідати зазначеним даним або містити невідповідність. Це робить перевірку ширшою, оскільки рішення залежить не тільки від документів особи, а й від транспортної інформації. Наприклад, якщо в картці зазначено один номер авто, а фактичні дані не збігаються, такий випадок має розглядатися як проблемний.

Усі ці елементи об'єднуються в службовий кейс. Він визначає тип ситуації: звичайна перевірка без порушень, прострочений паспорт, різні дані в документах, підозріла мета поїздки, неповна картка, службове орієнтування, невідповідність

автомобіля або спроба хабаря. Для кожного випадку система має правильний варіант рішення. Якщо дані коректні, мандрівника можна пропустити. Якщо виявлено порушення або службове попередження, гравець повинен відмовити у перетині кордону.

Після вибору дії система порівнює рішення користувача з очікуваним результатом. Правильне рішення збільшує рахунок і досвід, а помилка погіршує результат зміни. Механіка хабаря працює окремо: вона може дати гравцю гроші, але знижує показник службової цілісності. Таким чином, інформаційна модель враховує не лише саму перевірку, а й наслідки поведінки інспектора.

Ігровий процес симулятора базується на взаємодії гравця (інспектора) з мандрівником та його атрибутами (див. рис. 1.5). Користувач виконує роль інспектора, завдання якого — комплексна перевірка особи. Мандрівник надає для аналізу паспорт та картку в'їзду, керує автомобілем і комунікує з гравцем, моделюючи різні ситуації, зокрема спроби підкупу. Інспектор, своєю чергою, перевіряє документи, оглядає транспорт та звіряє дані зі службовим монітором, де відображені оперативні орієнтування. Фіналом кейсу є вердикт інспектора (дозвіл на в'їзд або відмова), що визначає результат перевірки та впливає на подальший ігровий прогрес.

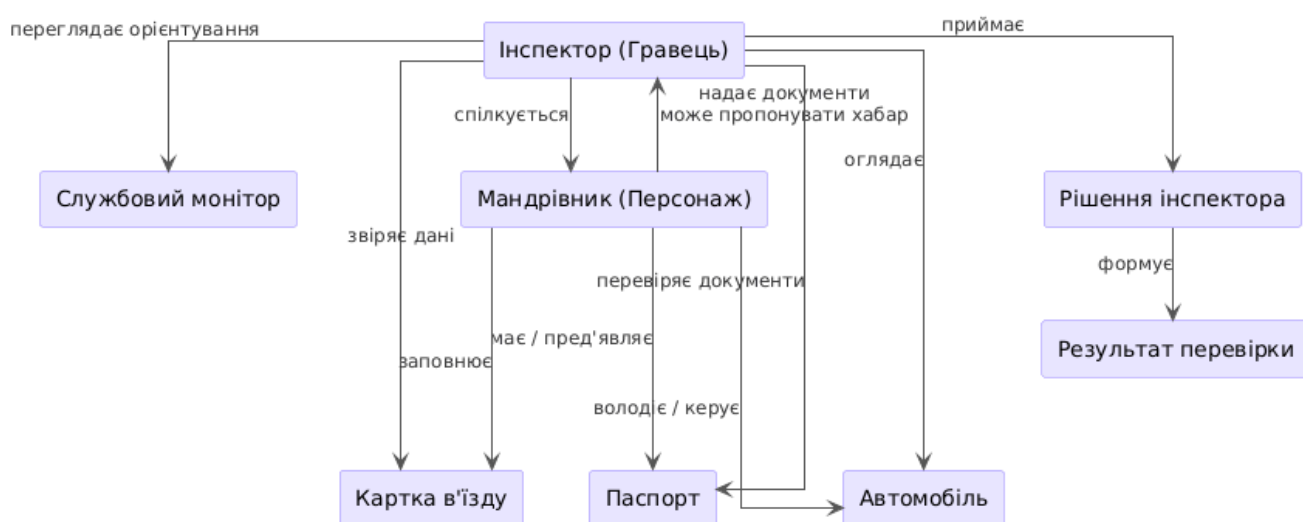


Рис 1.5 Логічна структура взаємодії гравця з ігровими об'єктами та персонажами

2 ЗАСОБИ РЕАЛІЗАЦІЇ ІГРОВОГО СИМУЛЯТОРА

2.1 Середовище розробки

Для реалізації ігрового симулятора «На Кордоні» було обрано середовище розробки, яке дозволяє поєднати програмування ігрової логіки з візуальним налаштуванням сцени. Оскільки проєкт є тривимірною грою, важливим було не лише написання коду, а й робота з об'єктами ігрового світу: персонажами, транспортом, камерою, інтерфейсом, документами, шлагбаумом та елементами контрольно-пропускного пункту.

Основна частина розробки виконувалася в Unity Editor. У цьому середовищі створювалися сцени, налаштовувалося розташування об'єктів, підключалися компоненти, перевірялася робота інтерфейсу та виконувалося тестування ігрового процесу безпосередньо під час розробки. Для проєкту використано дві основні сцени: сцену головного меню та сцену прикордонного пункту, у якій відбувається основна взаємодія гравця з мандрівниками.

Структура проєкту побудована за типовою для Unity логікою. У папці Assets розміщуються сцени, скрипти, префаби, матеріали, елементи інтерфейсу та інші ресурси. Така організація є зручною для поступового розширення симулятора, оскільки програмні модулі й візуальні елементи не змішуються між собою. Наприклад, окремі скрипти відповідають за генерацію службових кейсів, роботу ігрової зміни, відображення документів, рух NPC, транспорт, систему оцінювання та підсумок проходження.

Для написання C#-коду використовувалися Visual Studio. Цей редактор забезпечує зручну роботу з класами, методами та файлами проєкту, що є важливим для симулятора з великою кількістю взаємопов'язаних скриптів. У розробленій грі один модуль формує дані мандрівника, інший керує послідовністю перевірки, окремі компоненти оновлюють інтерфейс, а система оцінювання визначає

результат дій користувача. Через це середовище розробки має підтримувати швидкий перехід між частинами коду та зручне внесення змін.

Перевагою Unity Editor під час створення цього проєкту стала можливість одразу перевіряти взаємодію програмної логіки з об'єктами сцени. Після внесення змін можна було запускати гру в редакторі та перевіряти під'їзд автомобіля, появу мандрівника, передачу документів, роботу службового монітора, кнопок рішень, паузи й підсумкового екрана. Це дало змогу швидше знаходити помилки не тільки в коді, а й у налаштуванні сцени.

Таким чином, обране середовище розробки відповідає специфіці проєкту, оскільки дозволяє одночасно працювати з програмною логікою, 3D-сценою та інтерфейсом користувача. Це забезпечило можливість послідовно реалізувати основні частини симулятора: меню, ігрову зміну, перевірку документів, NPC, транспорт, шлагбаум і систему оцінювання рішень.

2.2 Ігровий рушій Unity

У розробці проєкту «На Кордоні» ігровий рушій Unity використано як основну платформу для створення тривимірного симулятора. Вибір цього рушія пов'язаний з тим, що в межах одного середовища можна працювати зі сценою, програмними компонентами, інтерфейсом, анімацією та налаштуваннями ігрового процесу. Для симулятора прикордонного контролю це є важливим, оскільки гра складається з багатьох елементів, які мають працювати узгоджено: мандрівники, транспорт, документи, службовий монітор, шлагбаум, камера та система прийняття рішень.

Unity дає змогу будувати гру на основі об'єктів сцени, до яких додаються окремі компоненти. У проєкті такий підхід використовується для розділення відповідальності між різними частинами симулятора. Наприклад, транспортний засіб має власну логіку під'їзду, зупинки та виїзду з пункту перевірки. Мандрівник окремо відповідає за появу, рух до інспектора, очікування рішення та вихід зі

сцени. Документи, службовий монітор і панель рішень також реалізуються як окремі елементи, що спрощує налаштування їхньої поведінки.

У проєкті передбачено дві основні сцени. Перша сцена використовується для головного меню, де користувач може розпочати гру, змінити налаштування або завершити роботу із застосунком. Друга сцена містить основний ігровий простір - контрольно-пропускний пункт. Саме в ній відбувається під'їзд автомобіля, поява мандрівника, передача документів, перевірка інформації та вибір рішення. Такий поділ робить структуру гри зрозумілішою і дозволяє відокремити початковий екран від основного процесу симуляції.

Програмна частина реалізована мовою C# у вигляді скриптів, які підключаються до об'єктів Unity. У межах симулятора ці скрипти відповідають за генерацію службових кейсів, перебіг зміни, роботу інтерфейсу, підрахунок балів, обробку хабаря, збереження прогресу, рух NPC і транспорту. Завдяки такій структурі логіка гри не зводиться до одного великого модуля, а поділена на окремі частини, кожна з яких виконує конкретне завдання.

Для візуальної частини використовується Universal Render Pipeline, що дозволяє налаштовувати освітлення, матеріали та загальне відображення 3D-сцени. У випадку проєкту «На Кордоні» це потрібно для створення зрозумілого ігрового середовища без надмірного навантаження на систему. Оскільки симулятор не орієнтований на фотореалістичну графіку, головним завданням є не максимальна деталізація, а стабільна робота сцени та зручне сприйняття об'єктів користувачем.

У грі значна частина інформації подається через текстові елементи: паспортні дані, картку мети в'їзду, службові записи, рекомендації монітора, показники зміни та підсумковий результат. Для цього використовуються засоби Unity UI та TextMeshPro. Вони дозволяють створити читабельні написи, кнопки та панелі, що є необхідним для гри, де користувач постійно порівнює дані з різних джерел.

Unity відповідає потребам розроблюваного симулятора, оскільки забезпечує роботу з 3D-середовищем, компонентами сцени, C#-скриптами, інтерфейсом та анімацією. Для проєкту «На Кордоні» цей рушій став основою, яка дозволила

поєднати візуальну модель контрольно-пропускного пункту з логікою перевірки мандрівників, системою оцінювання рішень і взаємодією користувача з ігровим середовищем.

2.3 Мова програмування C#

Для реалізації програмної логіки симулятора «На Кордоні» використано мову програмування C#, яка є основною мовою розробки сценаріїв у Unity. Її застосування дозволило описати поведінку об'єктів гри, правила перевірки мандрівників, роботу інтерфейсу, систему оцінювання та взаємодію між окремими частинами симулятора.

У межах проєкту C# використовується не як допоміжний інструмент, а як основа всієї ігрової логіки. Саме через скрипти визначається, як формується службовий кейс, які дані отримує мандрівник, як відображаються документи, як працює службовий монітор і яким чином система перевіряє рішення гравця. Наприклад, окремі класи відповідають за генерацію мандрівників, перебіг зміни, підрахунок балів, прогресію інспектора, рух NPC, роботу автомобіля та шлагбаума.

Перевагою C# для такого проєкту є чітка структура коду. Мова підтримує класи, властивості, методи, переліки та об'єктно-орієнтований підхід, що зручно для опису ігрових сутностей. У симуляторі прикордонного контролю є багато об'єктів із різними ролями: мандрівник, паспорт, картка в'їзду, транспортний засіб, службовий кейс, результат перевірки та прогрес гравця. Завдяки C# ці сутності можна подати у вигляді окремих моделей і класів, між якими встановлюється зрозумілий зв'язок.

Для проєкту «На Кордоні» важливою є також робота з умовами та різними типами ситуацій. У грі можуть з'являтися дійсні документи, прострочений паспорт, невідповідність даних, підозріла мета поїздки, проблема з автомобілем або спроба хабаря. Такі випадки зручно описувати через переліки та логічні перевірки в C#. Це

спрощує визначення правильного рішення для кожного кейсу та дозволяє легко додавати нові типи ситуацій у майбутньому.

C# також використовується для керування подіями гри. Коли користувач натискає кнопку рішення, система повинна отримати поточний кейс, порівняти вибір гравця з правильним варіантом, оновити рахунок, змінити показник службової цілісності за потреби та перейти до наступного мандрівника. Усі ці дії виконуються послідовно через методи відповідних класів, що забезпечує контрольований перебіг ігрової зміни.

Окремо слід зазначити взаємодію C#-скриптів з інтерфейсом Unity. У симуляторі значна частина інформації подається через UI: паспорт, картку в'їзду, службовий монітор, панель рішень, HUD та екран підсумків. Скрипти передають дані у відповідні текстові поля, оновлюють кнопки, показують повідомлення й змінюють стан елементів інтерфейсу залежно від поточного етапу гри. Завдяки цьому користувач бачить не статичну сцену, а змінний процес перевірки.

Мова C# також підходить для реалізації збереження прогресу. У проєкті передбачено накопичення досвіду, підвищення рівня та покращення навичок інспектора. Такі дані мають зберігатися між ігровими сесіями, тому в коді реалізовано роботу з постійними значеннями прогресу. Це дозволяє користувачу не починати розвиток з нуля після кожного запуску гри.

Таким чином, C# є основним засобом реалізації логіки симулятора «На Кордоні». За допомогою цієї мови описано генерацію службових кейсів, взаємодію з документами, роботу ігрової зміни, оцінювання рішень, механіку хабаря, прогресію гравця, NPC, транспорт і службовий інтерфейс. Використання C# у поєднанні з Unity забезпечило можливість побудувати проєкт як набір взаємопов'язаних модулів, кожен із яких відповідає за окрему частину ігрового процесу.

2.4 Засоби створення інтерфейсу та 3D-графіки

У розробленому симуляторі інтерфейс користувача є одним із основних елементів ігрового процесу. Через нього гравець отримує документи, службові відомості, підказки, статистику зміни та доступ до рішень. Тому під час створення інтерфейсу головним завданням було не декоративне оформлення, а зручне подання інформації, яку потрібно швидко прочитати й порівняти.

Для побудови екранних елементів застосовано засоби Unity UI. З їх допомогою створено головне меню, екран паузи, панель прийняття рішень, службовий монітор, блоки паспорта, картки мети в'їзду, ігровий HUD та екран підсумків. Кожен із цих елементів має окреме призначення. Наприклад, документи використовуються для перевірки персональних даних, службовий монітор - для уточнення інформації з бази, а панель рішень - для завершення перевірки конкретного мандрівника.

Оскільки значна частина взаємодії пов'язана з текстом, для його відображення використано TextMeshPro. Цей інструмент дозволяє якісніше працювати з написами, ніж стандартні текстові компоненти Unity. У межах симулятора це важливо для паспорта, картки в'їзду, службових записів, повідомлень про ризики, статистики та назв кнопок. Читабельність таких елементів напряму впливає на ігровий процес, адже користувач має помічати невідповідності в даних, а не витрачати час на розбір погано видимого тексту.

Інтерфейс побудовано відповідно до послідовності перевірки. Спочатку гравець переглядає документи мандрівника, потім звіряє їх зі службовою інформацією, після чого обирає рішення. Така організація допомагає не перевантажувати екран і водночас залишає доступ до всіх даних, які потрібні для аналізу ситуації. Окремі елементи інтерфейсу також використовуються для відображення рахунку, кількості помилок, грошей, цілісності служби та прогресу зміни.

Тривимірна частина гри створює простір контрольно-пропускного пункту. До неї входять дорога, будка інспектора, транспорт, мандрівники, шлагбаум, документи, робочий стіл та інші об'єкти оточення. Їхнє призначення полягає не тільки у візуальному наповненні сцени, а й у підтримці логіки проходження перевірки. Автомобіль під'їжджає до пункту, NPC наближається до зони контролю, документи передаються інспектору, а після прийняття рішення транспорт залишає сцену або отримує відмову.

Для налаштування зовнішнього вигляду 3D-сцени використано матеріали, освітлення, камеру та Universal Render Pipeline. Це дало змогу сформувати зрозуміле візуальне середовище без надмірної деталізації. Для цього типу симулятора важливішою є не фотореалістична графіка, а стабільна робота сцени, правильне розміщення об'єктів і зрозуміла візуальна подача службового процесу.

У проєкті також застосовуються префаби. Вони використовуються для повторюваних об'єктів, зокрема панелей інтерфейсу, документів, мандрівників, транспорту, шлагбаума та елементів оточення. Такий підхід спрощує підтримку проєкту: якщо потрібно змінити властивості типового об'єкта, достатньо відредагувати відповідний префаб, а не кожену копію вручну.

Засоби інтерфейсу та 3D-графіки в симуляторі виконують практичну роль. Unity UI і TextMeshPro забезпечують подання документів, службових даних та елементів керування, а 3D-сцена створює простір, у якому відбувається робота прикордонного пункту. Разом ці компоненти дозволяють користувачу сприймати перевірку як послідовний ігровий процес, а не як набір окремих екранів.

2.5 Система контролю версій та допоміжні інструменти

Специфіка розробки симулятора в середовищі Unity зумовлює необхідність чіткої структуризації не лише програмного коду, а й усього масиву супровідних ресурсів: сцен, префабів, матеріалів, текстур та конфігурацій проєктних пакетів. Складна архітектура ігрового застосунку передбачає глибоку взаємозалежність

активів (assets). Відтак, будь-яка модифікація окремої сцени, скрипта чи префаба безпосередньо впливає на суміжні ігрові механіки, що вимагає впровадження суворого контролю зв'язків між об'єктами ще на етапі проектування.

Для контролю змін було обрано систему Git. Її використання дає змогу фіксувати окремі етапи розробки та зберігати працездатні версії проєкту. Це доцільно для симулятора, у якому поступово додавалися головне меню, ігрова сцена, генерація службових кейсів, інтерфейс документів, службовий монітор, рух NPC, транспорт і система оцінювання рішень. За наявності помилки можна порівняти зміни та швидше визначити, який модуль потребує виправлення.

Контроль версій у розроблюваному проєкті обмежено лише тими файлами, що критичні для збирання застосунку. До репозиторію додаються три базові каталоги: Assets (скрипти, сцени, медіафайли), Packages (залежності середовища) та ProjectSettings (конфігурація проєкту). Інші папки — зокрема Library, Temp, Logs та Obj — є тимчасовими результатами роботи рушія Unity. Оскільки вони автоматично відтворюються при відкритті проєкту на будь-якому робочому місці, їх ігнорування дозволяє уникнути захаращення репозиторію надлишковими даними.

Для віддаленого збереження результатів розробки може застосовуватися GitHub. Репозиторій зручний як резервна копія та як спосіб зберігати проміжні стани роботи. У межах дипломного проєкту це дозволяє окремо фіксувати завершені етапи: налаштування сцени, реалізацію основних класів, підключення інтерфейсу, додавання логіки рішень і тестування готових механік.

Серед допоміжних інструментів також можна виділити Blender. Він використовується для підготовки або редагування 3D-об'єктів перед імпортом у Unity. Для симулятора прикордонного контролю це актуально, оскільки сцена містить службову будку, дорогу, транспорт, шлагбаум, робоче місце інспектора та інші елементи оточення.

Додатково в розробці використовуються готові ресурси: 3D-моделі, текстури, іконки, звукові ефекти та елементи інтерфейсу. Їх застосування дає змогу не витрачати надмірний час на другорядні візуальні матеріали та зосередитися на реалізації логіки симулятора.

Git, GitHub, Blender і допоміжні графічні та звукові ресурси забезпечують організовану розробку ігрового застосунку. Вони не замінюють Unity та C#, але доповнюють основний стек, допомагаючи зберігати версії проєкту, працювати з 3D-об'єктами та підтримувати цілісність матеріалів під час розробки.

Ефективна інтеграція проєкту Unity із системою Git вимагає обов'язкового використання правильно налаштованого файлу `.gitignore`, який автоматично відсікає тимчасові кеш-файли рушія, обмежуючи репозиторій лише згаданими каталогами *Assets*, *Packages* та *ProjectSettings*. Окрім цього, для коректної фіксації змін у текстовому форматі (зокрема для сцен та префабів) у налаштуваннях проєкту (*Editor Settings*) режим серіалізації активів було переведено у статус *Force Text*. Оскільки ігрові проєкти часто містять вагомні бінарні ресурси (текстури високої роздільної здатності, аудіофайли та 3D-моделі), що швидко перевантажують стандартне сховище Git, для оптимізації роботи з великими файлами доцільно впровадити розширення Git Large File Storage (LFS). Це дозволяє замінювати важкі об'єкти текстовими покажчиками всередині репозиторію, зберігаючи самі файли на окремому сервері, що суттєво пришвидшує процеси клонування та синхронізації проєкту на GitHub.

3 ПРОЄКТУВАННЯ ІГРОВОГО СИМУЛЯТОРА

3.1 Загальна архітектура системи

Проєктування ігрового симулятора починається з визначення основних частин системи та зв'язків між ними. Для розробленої гри важливо було побудувати таку структуру, у якій ігровий процес, генерація службових ситуацій, інтерфейс, NPC, транспорт і система оцінювання працюють не окремо, а як єдиний механізм. Саме тому архітектура проєкту поділена на кілька функціональних блоків, кожен із яких відповідає за свою частину симуляції. (загальну архітектуру ігрового симулятора представлено на рис. 3.1).

Центральним елементом архітектури є модуль керування ігровим процесом. Він відповідає за перебіг зміни, підготовку черги мандрівників, запуск перевірки, обробку рішення гравця та перехід до наступного службового кейсу. У програмній реалізації таку роль виконує менеджер ігрової логіки, який координує роботу інших компонентів і визначає поточний стан гри.

Окремий блок відповідає за формування службових ситуацій. Він створює дані мандрівника, документи, інформацію про транспорт, службові попередження та тип можливого порушення. Завдяки цьому кожна перевірка не є повністю однаковою: у грі можуть з'являтися як коректні документи, так і проблемні випадки, пов'язані з простроченим паспортом, розбіжностями в даних, підозрілою метою поїздки, службовим орієнтуванням або невідповідністю автомобіля.

Інтерфейсний блок забезпечує відображення всієї інформації, з якою працює користувач. До нього належать вікна паспорта, картки мети в'їзду, службового монітора, панель прийняття рішень, HUD, пауза та підсумковий екран. Цей блок не приймає рішення замість гравця, але надає йому дані, необхідні для аналізу ситуації. Через інтерфейс користувач бачить документи, службові записи, статистику зміни та наслідки власних дій.

Для відтворення роботи контрольно-пропускного пункту в архітектурі виділено блок NPC і транспорту. Він відповідає за під'їзд автомобіля, появу мандрівника, його рух до інспектора, передачу документів, очікування рішення та завершення перевірки. До цього ж блоку належить робота шлагбаума, який візуально показує результат рішення: дозвіл на проїзд або відмову.

Система оцінювання та прогресії обробляє наслідки дій користувача. Після прийняття рішення система порівнює вибір гравця з правильним результатом для поточного кейсу. У разі правильного рішення збільшується рахунок і досвід, у разі помилки фіксується штраф. Якщо користувач приймає хабар, змінюються фінансові показники та знижується службова цілісність. Такий підхід дозволяє пов'язати ігрову дію з конкретним результатом.

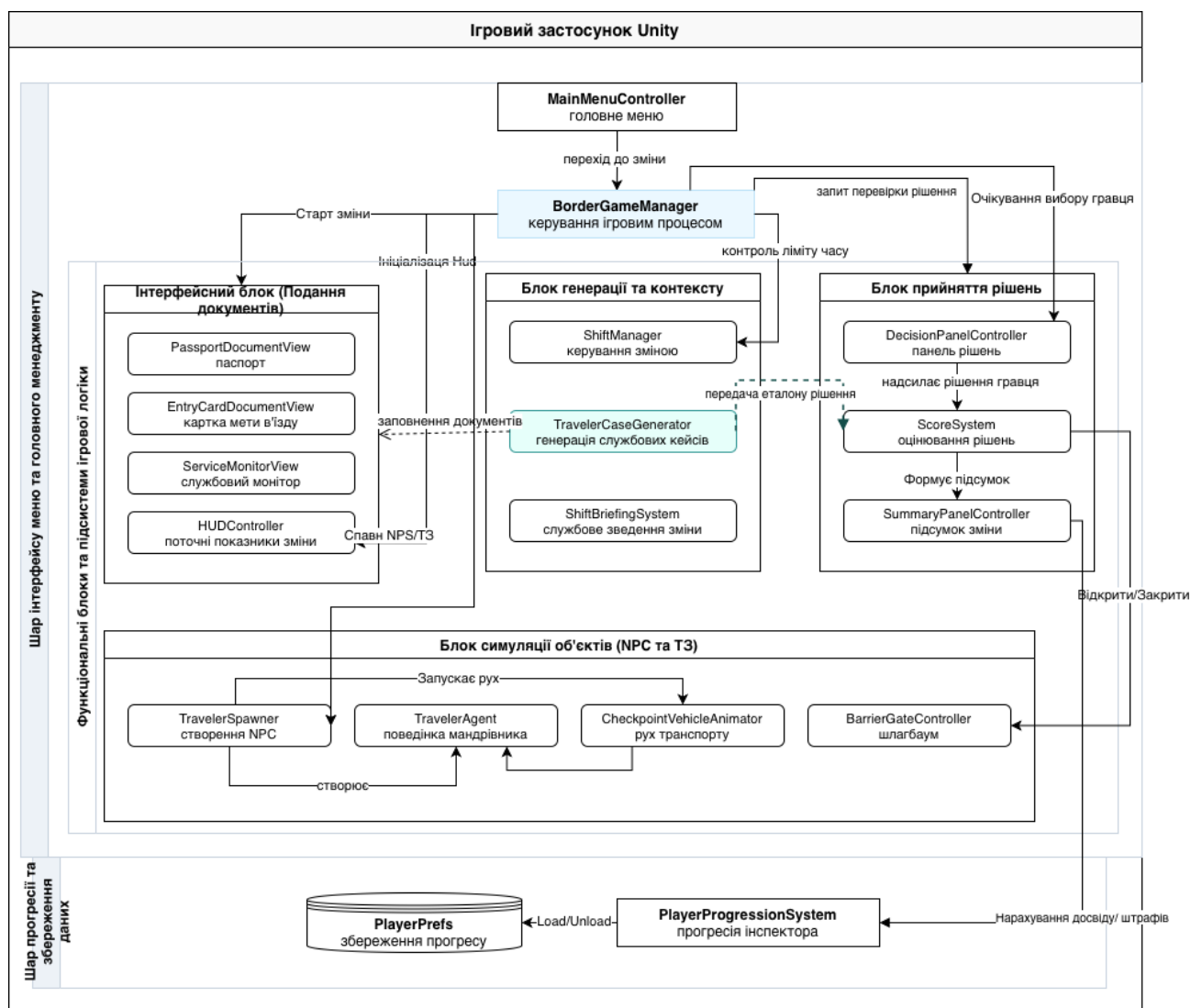


Рис. 3.1 Загальна архітектура ігрового симулятора

У межах цієї архітектури кожен модуль виконує окрему функцію, але результат формується через їхню взаємодію. Генератор створює службовий кейс, менеджер гри запускає його проходження, інтерфейс показує дані, NPC і транспорт відтворюють сцену перевірки, а система оцінювання визначає наслідки рішення користувача. Такий поділ спрощує підтримку проєкту і дозволяє надалі розширювати симулятор новими типами документів, службових ситуацій, сценаріїв поведінки NPC або додатковими режимами гри.

3.2 Проєктування ігрового циклу

Ігровий цикл симулятора побудовано навколо однієї робочої зміни інспектора. Такий підхід дозволяє подати перевірку мандрівників не як окрему дію, а як послідовний процес, у якому кожен новий випадок впливає на загальний результат проходження. У межах зміни користувач отримує службові ситуації, аналізує надані дані, приймає рішення та після завершення всіх перевірок бачить підсумок своєї роботи.

На початку гри користувач переходить із головного меню до основної сцени прикордонного пункту. Після запуску зміни система готує чергу мандрівників і формує для кожного з них окремий службовий кейс. У цьому кейсі містяться дані особи, паспорт, картка мети в'їзду, службова інформація, відомості про автомобіль та можливі ознаки порушення. Завдяки цьому кожна перевірка має власний набір умов, а користувач не виконує повністю однакові дії.

Основний етап ігрового циклу починається з появи автомобіля на пункті контролю. Транспорт під'їжджає до визначеної позиції, після чого мандрівник наближається до робочої зони інспектора та передає документи. На цьому етапі гравець отримує доступ до паспорта, картки в'їзду та службового монітора. Його завдання полягає в тому, щоб уважно порівняти дані між собою та визначити, чи є підстави для відмови у перетині кордону.

Після аналізу інформації користувач обирає одне з доступних рішень. У базовому сценарії він може дозволити в'їзд або відмовити мандрівнику. Якщо службовий кейс передбачає спробу хабаря, у грі також може бути використана дія, пов'язана з прийняттям неправомірної вигоди. Такий вибір впливає не лише на поточний результат, а й на показник службової цілісності, тому гравець має враховувати наслідки своїх дій.

Після прийняття рішення система порівнює вибір користувача з правильним рішенням для поточного кейсу. Якщо дія була правильною, гравець отримує бали та досвід. Якщо рішення помилкове, фіксується помилка і результат зміни погіршується. Далі запускається завершальна частина перевірки: мандрівник залишає зону контролю, транспорт виїжджає або отримує відмову, а система переходить до наступного кейсу.

Зміна завершується після перевірки всієї черги мандрівників. Після цього користувач бачить підсумковий екран, де відображаються результати: кількість правильних рішень, помилки, набрані бали, отриманий досвід, гроші та стан службової цілісності. Якщо гравець накопичив достатньо досвіду або очок покращень, він може розвивати окремі навички інспектора, що впливає на подальше проходження.

Такий ігровий цикл є зручним для симулятора прикордонного контролю, оскільки він має чіткий початок, повторювану основну дію та зрозуміле завершення. Користувач поступово проходить однакову за структурою, але різну за змістом послідовність перевірок. Це дозволяє зберегти стабільну логіку гри й водночас забезпечити варіативність завдяки процедурній генерації службових кейсів.

Графічне представлення описаного ігрового циклу у вигляді діаграми діяльності зображено на рисунку 3.2.

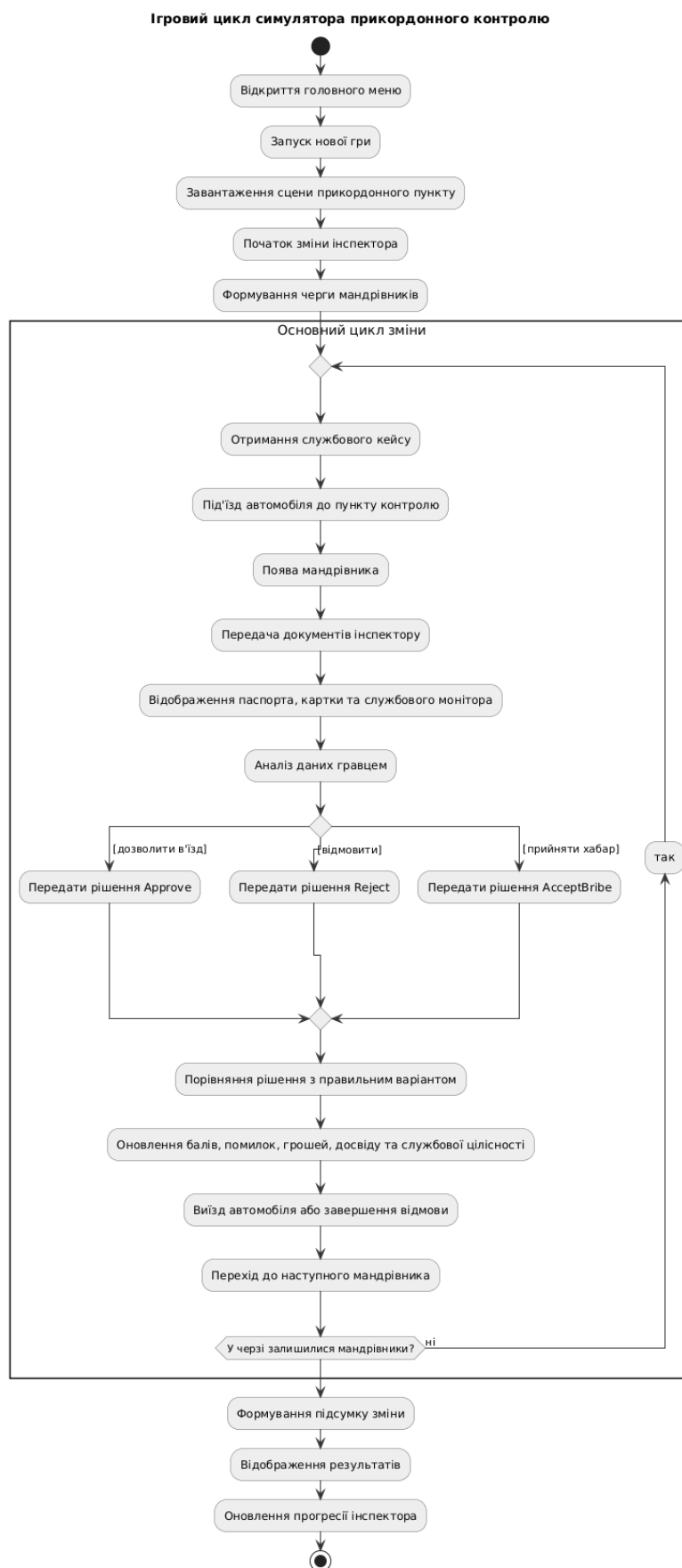


Рис. 3.2 Діаграма ігрового циклу симулятора прикордонного контролю

Проектування ігрового циклу передбачає послідовний перехід між основними станами гри: запуск зміни, формування кейсу, прибуття мандрівника, перевірка даних, прийняття рішення, оцінювання результату, перехід до наступного випадку та відображення підсумків. Така структура забезпечує зрозумілий порядок взаємодії користувача з програмним продуктом і створює основу для реалізації основних модулів симулятора.

3.3 Проектування системи генерації кейсів мандрівників

Система генерації кейсів є одним із ключових елементів симулятора, оскільки саме вона визначає зміст кожної перевірки. Якщо всі мандрівники мали б однакові документи та однакові умови проходження контролю, ігровий процес швидко став би передбачуваним. Тому під час проектування було передбачено формування різних службових ситуацій, у яких користувач має щоразу аналізувати новий набір даних.

У межах симулятора кожен кейс описує окремого мандрівника, який прибуває на контрольно-пропускний пункт. Для нього формуються персональні дані, паспорт, картка мети в'їзду, службова інформація, дані автомобіля та можливі ознаки порушення. Після створення кейсу система одразу визначає, яке рішення має бути правильним для цієї ситуації: дозволити в'їзд або відмовити у перетині кордону.

Загальний алгоритм вибору типу ситуації та покроковий процес створення даних відображено на схемі генерації службового кейсу мандрівника (див. рис. 3.3). Генерація кейсів побудована за принципом вибору типу ситуації. У грі можуть створюватися як звичайні мандрівники без порушень, так і проблемні випадки. До таких випадків належать прострочений паспорт, невідповідність персональних даних, підозріла мета поїздки, неповністю заповнена картка, спроба хабаря, невідповідність транспортного засобу, службове орієнтування або вибіркова

вторинна перевірка. Такий набір ситуацій дозволяє охопити кілька різних причин, через які інспектор може прийняти рішення про відмову.

Під час створення звичайного кейсу дані в документах і службовому моніторі мають узгоджуватися між собою. У такій ситуації паспорт є дійсним, персональні дані збігаються, картка в'їзду заповнена коректно, а транспортний засіб відповідає вказаній інформації. Якщо ж генерується проблемний кейс, система навмисно створює одну або кілька невідповідностей, які гравець повинен помітити під час перевірки.

Важливо, що порушення не подаються користувачу безпосередньо. Гравець бачить лише документи, службовий монітор, дані автомобіля та поведінку мандрівника. Висновок потрібно зробити самостійно на основі порівняння цих відомостей. Наприклад, у випадку простроченого паспорта необхідно звернути увагу на строк дії документа. Якщо згенеровано невідповідність персональних даних, гравець має порівняти ПІБ у паспорті та картці в'їзду. Якщо проблема пов'язана з транспортом, потрібно перевірити дані автомобіля в документах і службовій інформації.

Кейс спроби хабаря не обмежується формальною перевіркою документів, оскільки додає до гри етичний вибір. Користувач може прийняти хабар, однак це знижує показник службової цілісності. Завдяки цьому система генерації не лише створює технічні помилки в документах, а й формує нестандартні ситуації, які впливають на загальний результат зміни.

У проєктуванні також враховано, що різні типи кейсів мають з'являтися з різною частотою. Звичайні випадки повинні траплятися частіше, щоб гра не перетворювалася на послідовність суцільних порушень. Проблемні ситуації, навпаки, мають з'являтися достатньо регулярно, щоб користувач залишався уважним. Такий підхід робить ігровий процес більш природним і не дозволяє гравцю автоматично відмовляти кожному мандрівнику.

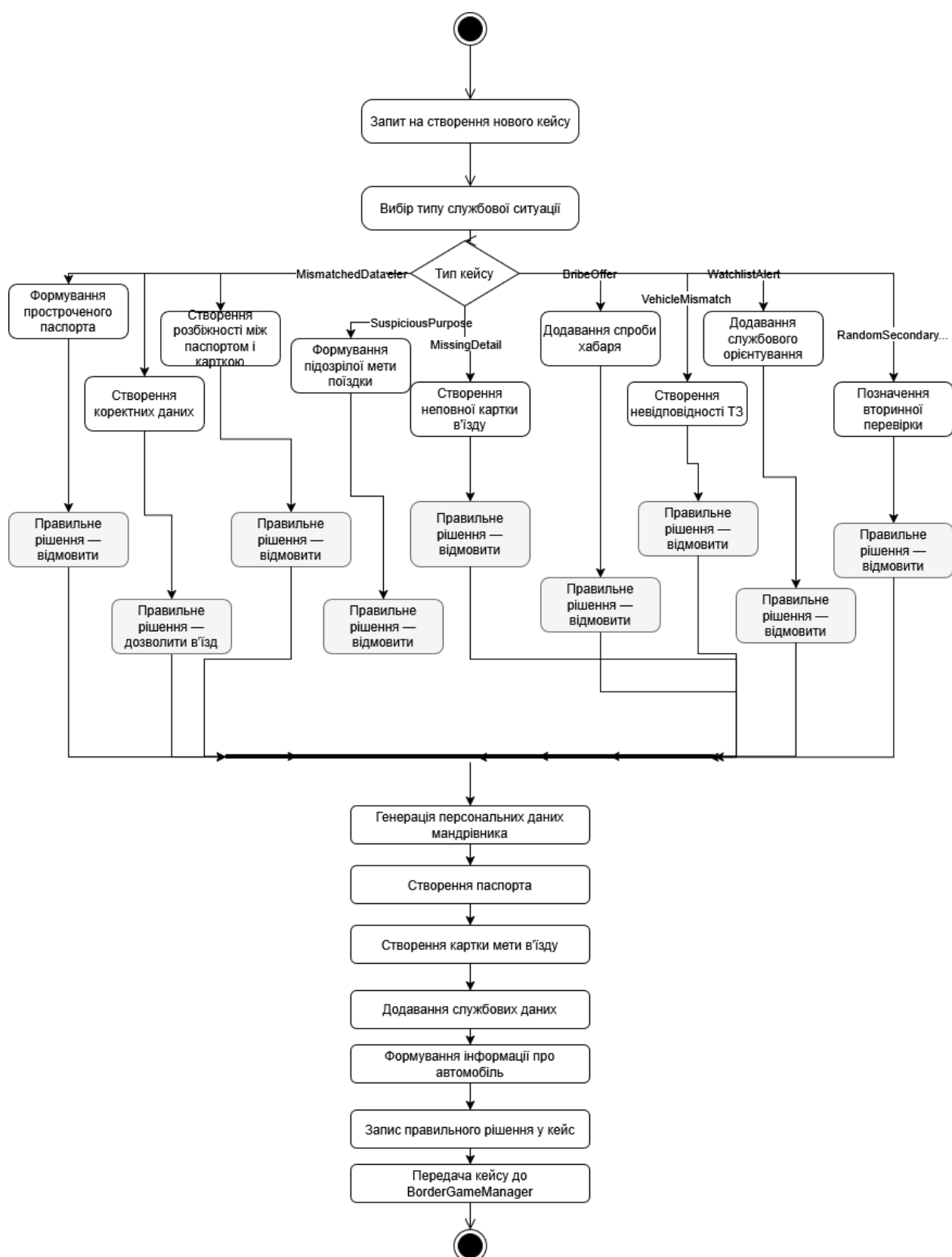


Рис. 3.3 Схема генерації службового кейсу мандрівника

Після створення кейс передається до основного ігрового циклу. Далі його дані використовуються в інтерфейсі документів, службовому моніторі, системі руху мандрівника та модулі оцінювання. Таким чином, генератор не працює ізольовано: він готує основу для всієї подальшої перевірки, а інші компоненти симулятора відображають і обробляють сформовану інформацію.

Система генерації кейсів забезпечує варіативність ігрового процесу та створює умови для логічного аналізу даних. Завдяки їй кожна перевірка має власний зміст, а рішення гравця залежить не від запам'ятовування сценарію, а від уважного порівняння документів, службової інформації та транспортних даних.

3.4 Проєктування системи перевірки документів

Система перевірки документів у симуляторі є основою прийняття рішення гравцем. Саме вона визначає, які дані користувач бачить під час службового кейсу, з яких джерел отримує інформацію та за якою логікою може виявити порушення. Під час проєктування цього модуля було важливо зробити перевірку зрозумілою, але не занадто простою. Гравець не повинен одразу отримувати готову відповідь від системи, оскільки основна ігрова дія полягає в самостійному порівнянні документів і службових даних. Взаємодію користувача з інформаційними джерелами та логіку обробки цих даних відображає схема перевірки документів і службових даних у симуляторі (див. рис. 3.4).

У межах симулятора перевірка побудована навколо трьох основних джерел інформації: паспорта, картки мети в'їзду та службового монітора. Паспорт містить базові персональні дані мандрівника: ім'я, громадянство, дату народження, номер документа, орган видачі та строк дії. Картка мети в'їзду доповнює цю інформацію даними про причину поїздки, адресу перебування, приймаючу сторону, строк візиту, транспорт і додаткові відомості. Службовий монітор використовується як внутрішня база перевірки, де відображаються реєстрові дані, маршрут, службові ризики, інформація про автомобіль та рекомендації.

Проектування цього модуля передбачає, що кожне джерело даних не дублює інше повністю, а містить частину інформації, необхідної для аналізу. Завдяки цьому користувач має зіставляти відомості між собою. Наприклад, ім'я в паспорті повинно відповідати імені в картці в'їзду, строк дії паспорта має бути актуальним, заявлена мета поїздки не повинна суперечити службовим попередженням, а дані автомобіля мають збігатися з інформацією в документах або моніторі.

У системі передбачено кілька типів порушень, які можуть бути виявлені під час перевірки. До них належать прострочений паспорт, невідповідність персональних даних, неповна картка в'їзду, підозріла мета поїздки, службове орієнтування, невідповідність транспортного засобу або необхідність вторинної перевірки. Такі порушення не показуються гравцю як окреме повідомлення про помилку. Вони закладені в самі дані кейсу, тому користувач має знайти їх під час перегляду документів і службової інформації.

Важливою особливістю є те, що система перевірки не приймає рішення замість користувача. Вона лише надає дані, необхідні для аналізу. Остаточний вибір виконує гравець через панель рішень. Після цього система порівнює обрану дію з правильним рішенням, яке було визначене під час генерації кейсу. Якщо всі дані коректні й немає службових попереджень, правильним рішенням є дозвіл на в'їзд. Якщо хоча б один важливий критерій порушено, користувач повинен відмовити у перетині кордону.

Для реалізації такого підходу в архітектурі проєкту документи та службовий монітор подаються як окремі інтерфейсні модулі. Паспортний блок відповідає за відображення основних даних особи, блок картки в'їзду - за дані поїздки, а службовий монітор - за додаткову інформацію з бази. Усі ці елементи отримують дані з поточного службового кейсу, але відображають їх у різному вигляді. Це дозволяє зберегти зв'язок між даними та водночас зробити перевірку схожою на роботу з кількома незалежними джерелами.

Транспортна інформація при перевірці враховується окремо. У симуляторі автомобіль не є лише візуальним об'єктом, оскільки його дані також можуть

впливати на правильність рішення. Якщо номер або модель автомобіля не відповідають інформації з документів чи службового монітора, такий випадок вважається проблемним. Завдяки цьому перевірка не обмежується паспортом, а охоплює ширший службовий контекст.

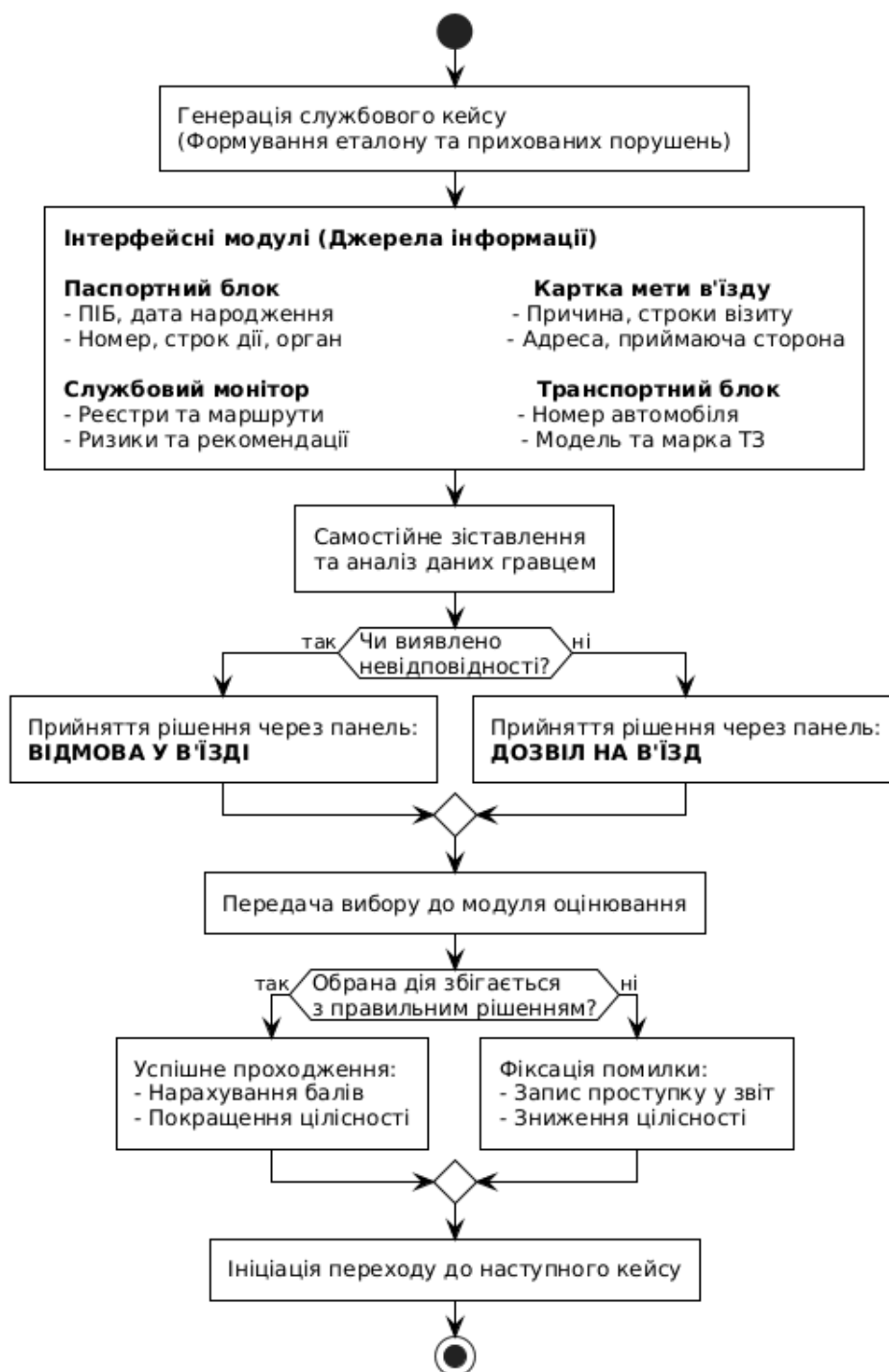


Рис. 3.4 Схема перевірки документів і службових даних у симуляторі

Після завершення аналізу користувач обирає одну з доступних дій. Далі система передає результат до модуля оцінювання, який визначає наслідки рішення: нарахування балів, фіксацію помилки, зміну показника службової цілісності або перехід до наступного кейсу. Таким чином, система перевірки документів є проміжною ланкою між генерацією кейсу та оцінюванням дій користувача.

Під час проектування системи перевірки документів основну увагу приділено взаємозв'язку між різними джерелами даних. Паспорт, картка в'їзду, службовий монітор і транспортна інформація формують набір умов, за якими гравець має самостійно зробити висновок. Такий підхід дозволяє зберегти логіку прикордонного контролю та перетворити перевірку документів на повноцінну ігрову механіку.

3.5 Проектування системи оцінювання та прогресії

Система оцінювання в розробленому симуляторі потрібна для того, щоб після кожної перевірки гравець одразу бачив наслідок свого рішення. У реальному процесі прикордонного контролю помилка інспектора має значення, тому в грі також передбачено підрахунок правильних і неправильних дій. Це робить перевірку не формальною, а пов'язаною з результатом усієї зміни. Загальну логіку функціонування цього модуля відображає схема роботи системи оцінювання та прогресії (див. рис. 3.5).

Після завершення службового кейсу система порівнює вибір користувача з правильним рішенням. Якщо мандрівник має коректні документи, відсутні службові попередження та збігаються дані автомобіля, правильним варіантом є дозвіл на в'їзд. Якщо ж у кейсі присутнє порушення, наприклад прострочений паспорт, розбіжність у персональних даних, підозріла мета поїздки, службове орієнтування або невідповідність транспортного засобу, очікуваним рішенням є відмова.

У проєкті передбачено просту й зрозумілу систему балів. За правильне рішення гравець отримує позитивний результат, а за помилку - штраф. Такий підхід дозволяє оцінювати не швидкість натискання кнопок, а правильність аналізу документів і службових даних. Користувач має уважно переглянути паспорт, картку в'їзду, службовий монітор і дані автомобіля, після чого зробити висновок.

Окремо враховується хабар. Ця механіка використовується для того, щоб показати не лише технічну перевірку документів, а й етичний бік роботи інспектора. Якщо гравець приймає хабар, він отримує гроші, але втрачає частину службової цілісності. Тому така дія не є повноцінною заміною правильного рішення: вона може дати короточасну вигоду, але погіршує загальний результат зміни.

Після проходження зміни система підсумовує результати. До підсумку входять набрані бали, кількість правильних рішень, помилки, отримані гроші, досвід і стан службової цілісності. Завдяки цьому користувач бачить не тільки фінальний рахунок, а й причини цього результату. Наприклад, зміна може бути завершена з високою кількістю балів, але зі зниженим показником цілісності через прийняті хабарі.

Взаємозв'язок між поточними результатами та довгостроковим розвитком персонажа деталізовано на схемі прогресії (див. рис. 3.5). Прогресія інспектора використовується для зв'язку між кількома ігровими сесіями. Після зміни гравець отримує досвід, який залежить від якості проходження. Досвід нараховується за правильні перевірки, завершення зміни, відсутність хабарів, збереження високої службової цілісності та серії правильних рішень. Це заохочує користувача проходити гру уважно, а не просто швидко завершувати кожен кейс.

У системі розвитку передбачено кілька напрямів покращень. Процедури впливають на результат за правильні рішення, аналітика покращує корисність службового монітора, етика зменшує втрату цілісності під час корупційних ситуацій, а логістика пов'язана зі швидкістю обробки черги мандрівників. Такі

покращення не прибирають необхідність самостійного аналізу, але роблять подальші зміни більш динамічними.

Для збереження прогресу між запусками гри фіксуються рівень користувача, досвід, очки покращень і поточні рівні навичок. Завдяки цьому гравець може продовжувати розвиток інспектора після завершення попередньої сесії. Такий підхід робить симулятор більш цілісним, оскільки результати попередніх змін мають значення для подальшого проходження.

Система оцінювання та прогресії поєднує поточний результат перевірки з довгостроковим розвитком гравця. Вона враховує правильність рішень, помилки, хабарі, службову цілісність, досвід і покращення інспектора. Завдяки цьому кожна дія користувача має наслідок як для окремого службового кейсу, так і для загального проходження гри.

Програмну архітектуру модуля оцінювання реалізовано на базі патерна Singleton (клас `ProgressionManager`), що забезпечує глобальний доступ до статистики гравця з усіх сцен проєкту. Обчислення балів та рівня цілісності відбувається динамічно, а вивчені навички діють як коефіцієнти-модифікатори для базових значень нагород і штрафів. Для збереження прогресу між сесіями застосовано серіалізацію даних у формат JSON із записом у локальну пам'ять через механізм `PlayerPrefs`. Це захищає досягнення від втрати при раптовому закритті гри та спрощує подальше розширення системи навичок інспектора.

У підсумку, інтегрована система оцінювання та прогресії трансформує рутинний процес перевірки документів у динамічний ігровий досвід, де кожне рішення має вагомі наслідки. Вона забезпечує баланс між реалістичністю симулятора та класичними механіками гейміфікації, мотивуючи користувача до глибшого аналізу службових кейсів та постійного вдосконалення навичок віртуального інспектора.

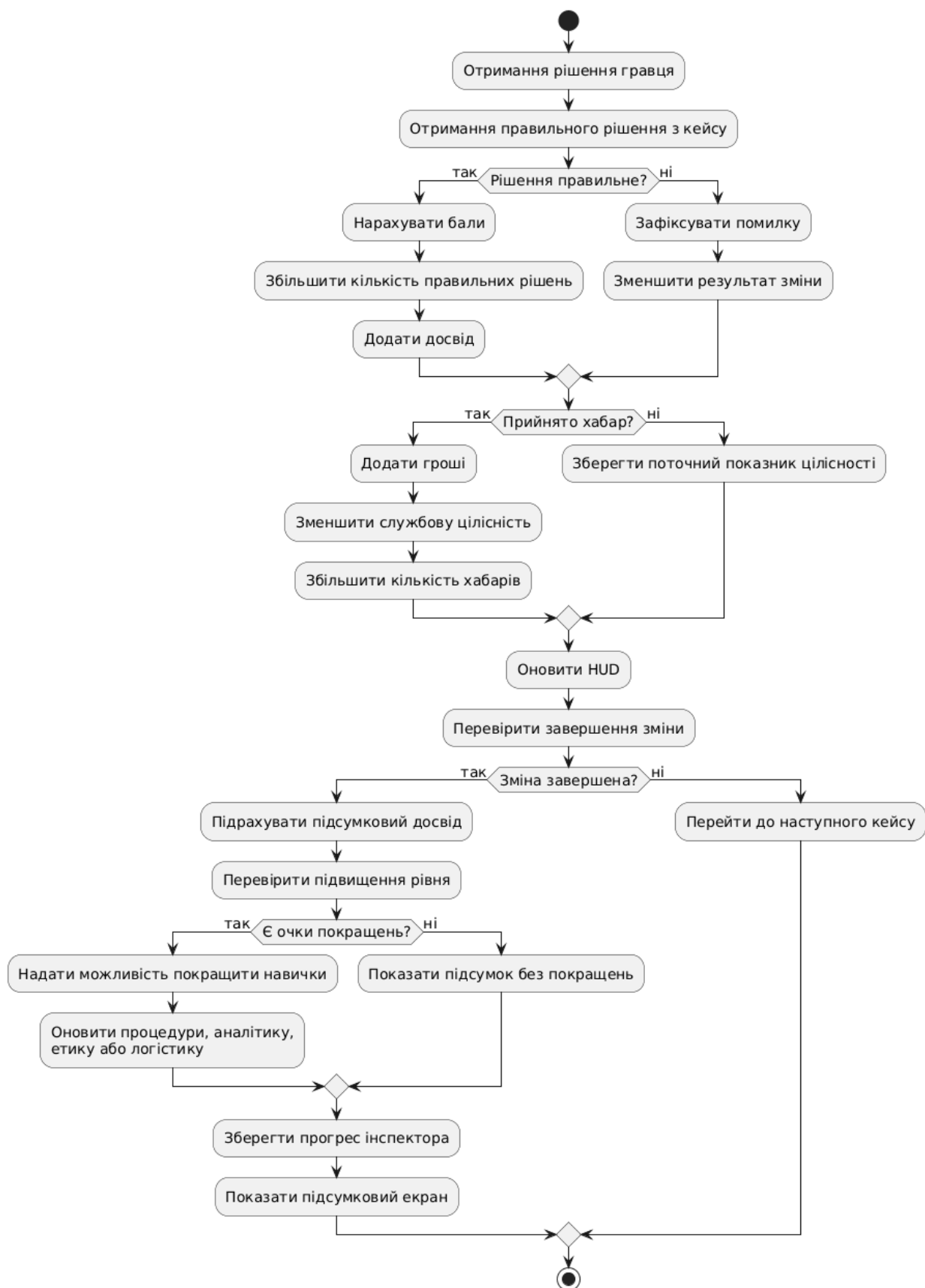


Рис. 3.5 Схема роботи системи оцінювання та прогресії

3.6 UML-діаграми системи

Архітектура та функціональна структура симулятора формалізовані за допомогою інструментарію UML. Для деталізації специфікації вимог, визначення меж системи та моделювання сценаріїв взаємодії користувача з програмними модулями розроблено діаграму прецедентів (див. рис. 3.6). Вона відображає склад ролей, функціональні можливості інтерфейсу інспектора та логіку обробки службового кейсу програмними компонентами симулятора.

На діаграмі варіантів використання (див. рис. 3.6) основним актором є користувач (гравець), який виконує роль інспектора. Окрім базових ігрових дій, таких як запуск зміни чи керування налаштуваннями через меню паузи, його ключова активність зосереджена на верифікації атрибутів поточного кейсу. Гравець взаємодіє з екранними формами документів (паспорта, картки в'їзду), аналізує дані монітора та безпосередньо реалізує прецедент прийняття рішення шляхом авторизації одного зі сценаріїв: затвердження в'їзду, відмови або взаємодії із прихованими геймплейними механіками (прийняття хабаря).

Набір прецедентів визначає межі взаємодії гравця з інтерфейсом контролю та доступний йому функціонал. Статична структура розробленого програмного продукту зафіксована на діаграмі класів яка деталізує логіку взаємодії ядра застосунку з доменними моделями персонажів, транспорту та документів.

Розподіл обов'язків між діями оператора та внутрішніми підсистемами гри зафіксовано на рівні об'єктної моделі застосунку. Обробка бізнес-логіки після вибору гравця відбувається автоматично: за генерацію параметрів сутностей відповідає модуль `TravelerCaseGenerator`, перевірку коректності дій здійснює `ScoreSystem`, а збереження прогресу — `PlayerProgressionSystem`.

Компонентну структуру цих модулів наведено на рисунку 3.7.



Рис. 3.6 Діаграма варіантів використання ігрового симулятора

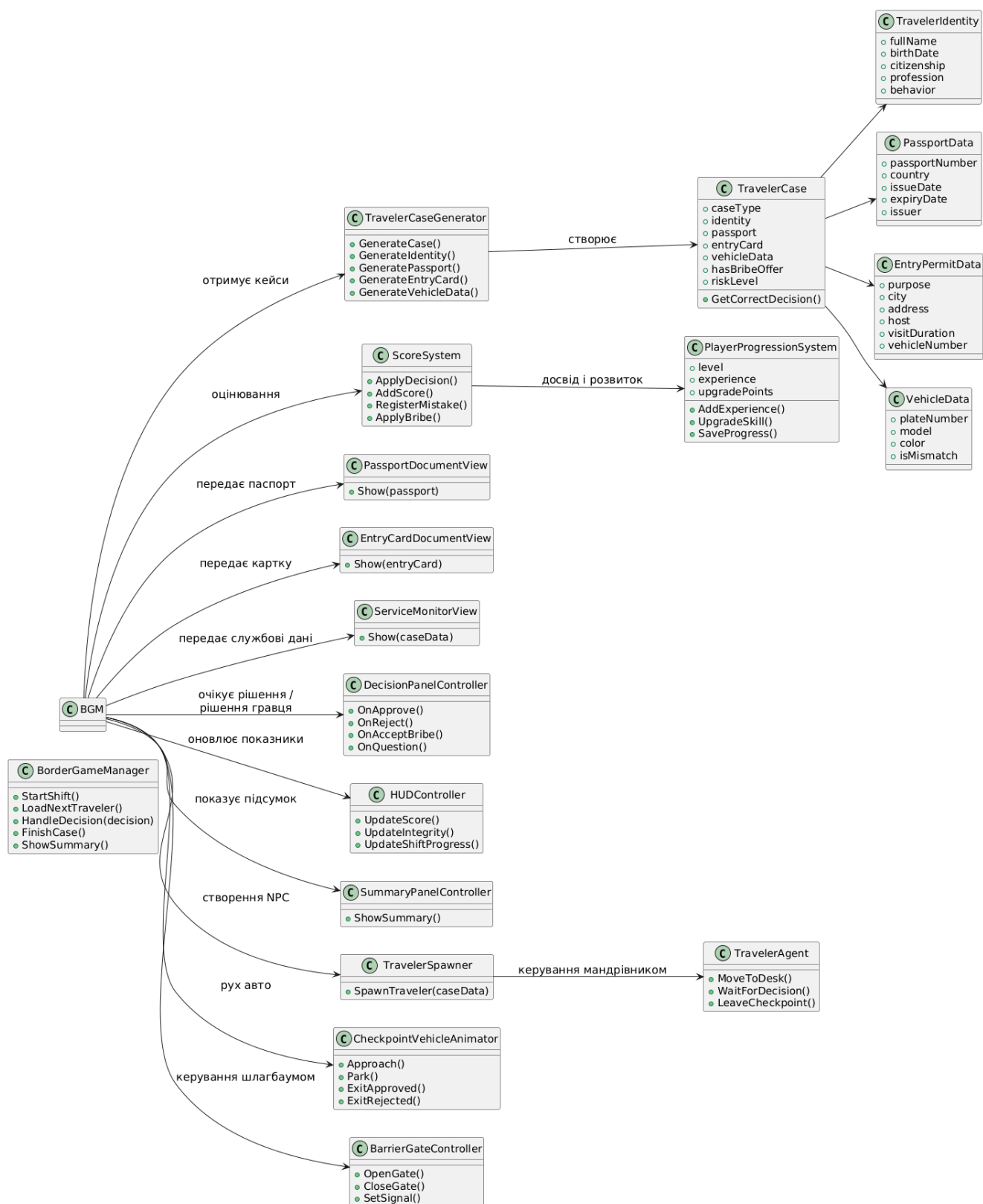


Рис. 3.7 Діаграма основних класів і компонентів симулятора

Діаграма послідовності для одного службового кейсу (див. рис 3.8) показує порядок взаємодії між користувачем, менеджером гри, генератором кейсів, інтерфейсом, системою оцінювання та модулями NPC/транспорту. Діаграма пояснює як відбувається один повний цикл перевірки: від створення мандрівника до оновлення результатів після рішення.

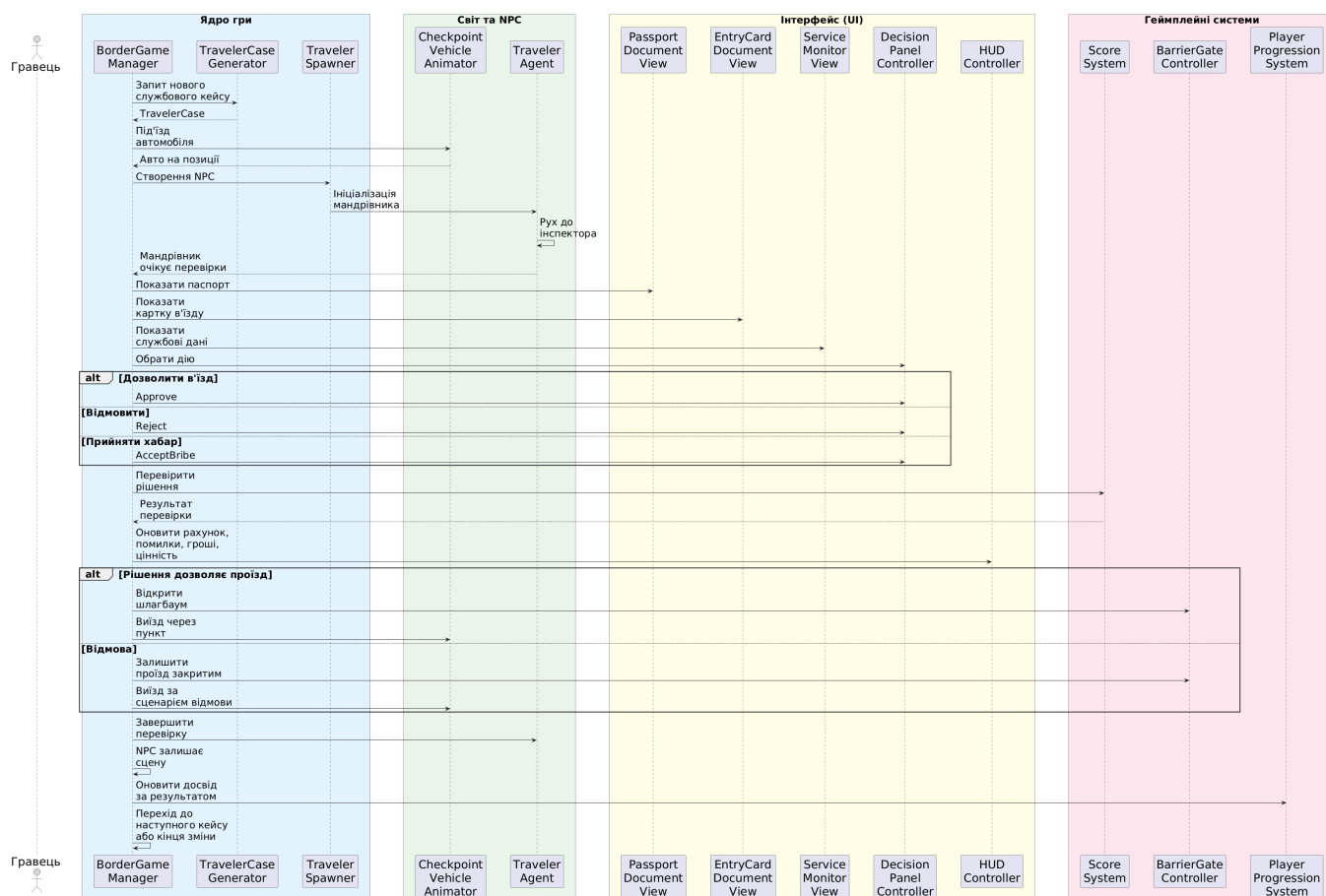


Рис. 3.8 Діаграма послідовності проходження одного службового кейсу

Діаграма варіантів використання описує дії гравця, діаграма класів або компонентів показує будову основних модулів, а діаграма послідовності уточнює порядок взаємодії під час перевірки мандрівника. У подальшому ці схеми використовуються як основа для опису реалізації програмних компонентів у четвертому розділі.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Реалізація основного ігрового циклу

Основний ігровий цикл реалізовано як послідовність дій, що відтворюють одну робочу зміну інспектора прикордонного контролю. У межах цієї зміни гравець приймає мандрівників, переглядає документи, аналізує службові дані, ухвалює рішення та після завершення черги отримує підсумок проходження. Такий підхід дозволяє подати гру не як набір окремих екранів, а як безперервний процес роботи контрольно-пропускного пункту.

Керування основним перебігом гри виконує скрипт `BorderGameManager`. Він координує запуск зміни, отримання нового службового кейсу, появу мандрівника, показ документів, обробку рішення користувача та перехід до наступного відвідувача. Цей компонент пов'язує між собою генератор кейсів, інтерфейсні модулі, систему оцінювання, NPC, транспорт і підсумковий екран.

На початку зміни система формує набір мандрівників, які мають пройти перевірку. Для цього використовується логіка генерації службових випадків, де кожному персонажу призначаються персональні дані (див рис 4.1), документи, транспортна інформація та можливі ознаки порушення. Після підготовки черги гра переходить до обробки першого кейсу. Надалі цей процес повторюється до завершення всієї зміни.

Під час кожної перевірки автомобіль під'їжджає до контрольної зони, після чого мандрівник з'являється біля робочого місця інспектора. У цей момент користувач отримує доступ до основної інформації: паспорта, картки мети в'їзду та службового монітора. Інтерфейс оновлюється відповідно до поточного кейсу, тому кожен мандрівник має власний набір даних для аналізу.

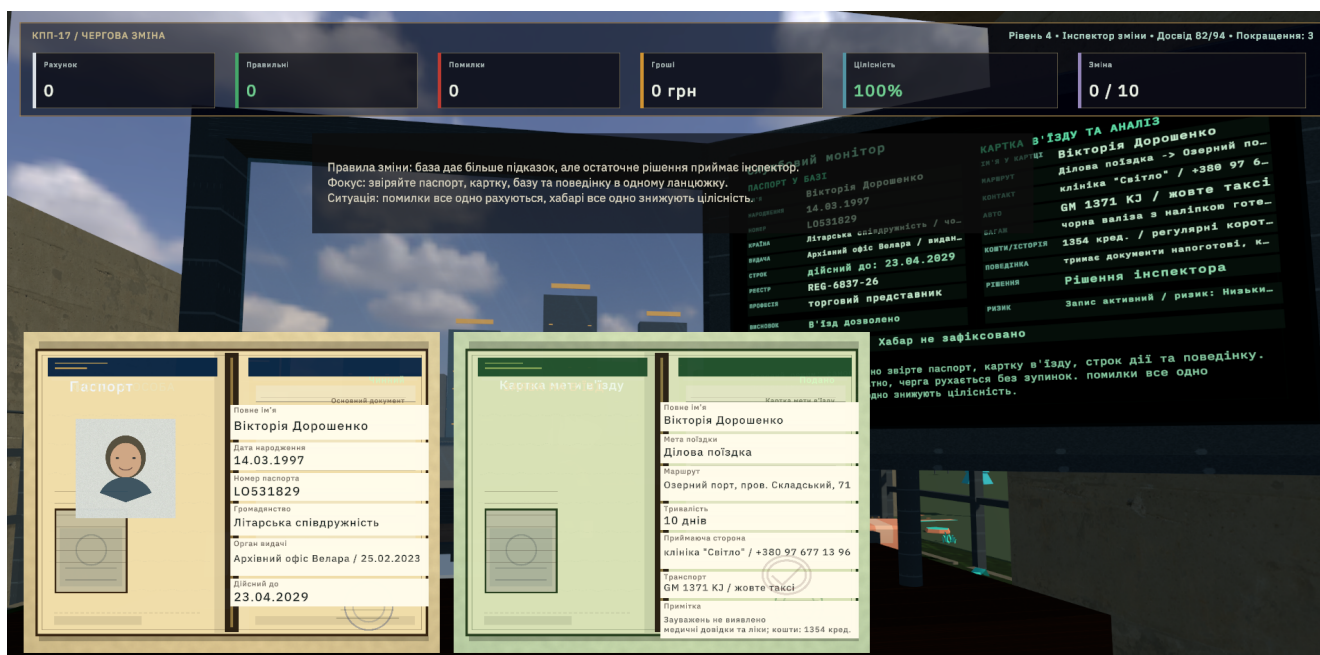


Рис. 4.1 Ігрова сцена перевірки мандрівника на контрольно-пропускному пункті

Після перегляду документів гравець приймає рішення за допомогою панелі керування. Основними діями є дозвіл на в'їзд або відмова у перетині кордону. В окремих випадках може бути доступна дія, пов'язана з хабарем. Після вибору рішення BorderGameManager передає результат до системи оцінювання, де він порівнюється з правильним рішенням для поточного кейсу.

Якщо користувач діє правильно, система нараховує бали та досвід. У разі помилки результат зміни погіршується. Якщо гравець приймає хабар, змінюються фінансові показники, але зменшується службова цілісність. Завдяки цьому кожне рішення впливає не тільки на поточний службовий випадок, а й на загальний результат зміни.

Після обробки рішення запускається завершення кейсу. Мандрівник залишає зону перевірки, транспорт продовжує рух або отримує відмову, а система переходить до наступного відвідувача. Коли всі мандрівники з черги опрацьовані, відкривається підсумковий екран. На ньому користувач бачить кількість правильних рішень, помилки, рахунок, отриманий досвід, гроші та стан службової цілісності.

Основний цикл можна подати як послідовність: запуск зміни, генерація кейсу, прибуття мандрівника, відображення документів, рішення користувача, оцінювання, завершення кейсу та перехід до наступної перевірки.

4.2 Реалізація генерації мандрівників і службових ситуацій

Генерація мандрівників у симуляторі реалізована як окремий програмний механізм, який формує зміст кожної перевірки. Його призначення полягає в тому, щоб гравець не працював з однаковими випадками, а щоразу отримував нову службову ситуацію з власними даними, документами, транспортом і можливими порушеннями.

Основою цього механізму є клас `TravelerCaseGenerator`. Він відповідає за створення службових кейсів, які потім використовуються в основному ігровому циклі. Під час генерації система визначає тип випадку, формує дані мандрівника, створює паспорт, картку мети в'їзду, службові записи, інформацію про автомобіль та додаткові ознаки ризику. Після цього готовий кейс передається до менеджера гри, який відображає його в інтерфейсі та запускає процес перевірки.

Для опису однієї службової ситуації використовується модель `TravelerCase`. Вона об'єднує всю інформацію, необхідну для перевірки: персональні дані мандрівника, паспортні відомості, картку в'їзду, поведінку персонажа, службові примітки, транспортні дані, можливість хабаря та правильне рішення. Такий підхід дозволяє зберігати всі дані одного випадку в єдиній структурі, а інші модулі гри можуть отримувати з неї лише потрібну частину інформації.

У симуляторі передбачено кілька типів кейсів. До них належать звичайний мандрівник без порушень, прострочений паспорт, невідповідність персональних даних, підозріла мета візиту, неповна інформація, спроба хабаря, невідповідність транспортного засобу, службове орієнтування та вибіркова вторинна перевірка. Тип кейсу визначає, які саме дані будуть змінені або позначені як ризикові.

Наприклад, у випадку з простроченим паспортом система створює документ, строк дії якого вже завершився. Якщо генерується невідповідність персональних даних, у паспорті та картці в'їзду можуть бути різні імена або інші розбіжності. У випадку з транспортом перевіряється відповідність номера чи моделі автомобіля тим даним, які вказані у документах або службовому моніторі. Завдяки цьому кожен тип порушення має власну логіку, а гравець повинен знайти його самостійно під час перегляду інформації.

Важливо, що порушення не виводиться користувачу прямо. Гравець бачить лише сформовані документи, службову інформацію та дані автомобіля. Саме тому основна складність полягає не в натисканні потрібної кнопки, а в уважному порівнянні кількох джерел даних. Такий спосіб реалізації краще відповідає жанру симулятора, оскільки рішення залежить від аналізу, а не від готової підказки.

Для урізноманітнення зміни різні типи кейсів мають різну ймовірність появи. Коректні мандрівники трапляються частіше, а проблемні ситуації додаються поступово в межах черги. Це потрібно для того, щоб гра не перетворювалася на просте очікування порушення в кожному випадку. Користувач має перевіряти всіх мандрівників уважно, оскільки серед них можуть бути як безпечні, так і ризикові випадки.

ShiftBriefingSystem формує службове зведення для зміни й може впливати на контекст окремих перевірок. Наприклад, у грі можуть з'являтися службові попередження, інформація про транспорт, підозрілі маршрути або інші умови, які потрібно враховувати під час прийняття рішення (див. рис. 4.2). Завдяки цьому перевірка не обмежується тільки паспортом і карткою в'їзду, а сприймається як частина ширшої службової ситуації.

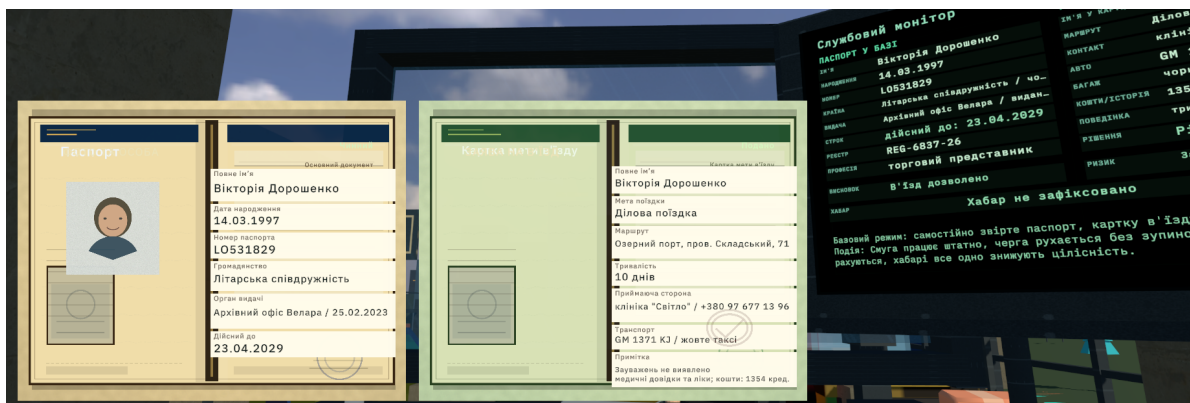


Рис. 4.2 Приклад згенерованого службового кейсу з документами та службовим монітором

Після створення кейсу його дані використовуються одразу кількома частинами гри. Інтерфейс документів відображає паспорт і картку в'їзду, службовий монітор показує додаткові відомості, система NPC і транспорту відтворює прибуття мандрівника, а модуль оцінювання зберігає правильне рішення для подальшої перевірки дії користувача. Таким чином, генерація мандрівників є не окремою допоміжною функцією, а основою всього ігрового циклу.

Реалізована система генерації забезпечує варіативність службових ситуацій і підтримує основну ідею симулятора - перевірку уважності та логічного аналізу даних. Завдяки класам TravelerCaseGenerator, TravelerCase та пов'язаним з ними моделям у грі формуються різні типи мандрівників, документів, службових ризиків і транспортних невідповідностей, що робить кожну зміну менш передбачуваною.

4.3 Реалізація UI, документів і службового монітора

Інтерфейс у симуляторі реалізовано як частину основного ігрового процесу, оскільки саме через нього користувач отримує дані для перевірки мандрівника. На відміну від звичайного допоміжного меню, UI у цьому проєкті виконує роль робочого місця інспектора: показує документи, службову інформацію, доступні рішення, поточні показники зміни та результат перевірки.

Для відображення паспортних даних використовується модуль `PassportDocumentView`. Він отримує інформацію з поточного службового кейсу та виводить її у вигляді документа. У паспорті показуються ПІБ мандрівника, дата народження, громадянство, номер паспорта, орган видачі, дата видачі та строк дії. Ці дані потрібні для перевірки особи та виявлення можливих порушень, зокрема простроченого документа або невідповідності персональних відомостей.

Картка мети в'їзду реалізована через модуль `EntryCardDocumentView`. Вона містить інформацію про мету поїздки, місто перебування, адресу, строк візиту, приймаючу сторону, транспортний засіб, примітки, багаж і кошти. Цей документ доповнює паспорт і дозволяє гравцю перевірити, чи узгоджуються між собою заявлені дані. Наприклад, розбіжність імені, неповна адреса або невідповідність автомобіля можуть стати підставою для відмови.

`ServiceMonitorView` реалізує службовий монітор, який подає інформацію з внутрішньої бази: реєстрові дані особи, маршрут, професію, приймаючу сторону, транспорт, багаж, кошти, поведінкові ознаки, ризик і службову рекомендацію. Його завдання полягає в тому, щоб надати гравцю додаткове джерело перевірки, яке не дублює повністю паспорт або картку в'їзду. Завдяки цьому користувач має звіряти дані з кількох джерел, а не орієнтуватися лише на один документ.

Панель прийняття рішень реалізована через `DecisionPanelController`. Вона забезпечує обробку основних дій користувача: дозволити в'їзд, відмовити у перетині кордону або прийняти хабар у відповідному службовому випадку. Також передбачено можливість поставити додаткове питання мандрівнику. Після натискання кнопки рішення відповідна дія передається до основного менеджера гри, де відбувається оцінювання результату.

Для відображення поточного стану гри використовується `HUDController`. Він показує рахунок, кількість правильних рішень, помилки, гроші, службову цілісність, прогрес зміни та інформацію про кар'єру інспектора. Завдяки цьому користувач бачить не тільки документи, а й загальний стан проходження. Це

важливо, оскільки кожна помилка або прийнятий хабар впливають на підсумок зміни.

Після завершення перевірки всієї черги відкривається підсумковий екран, за який відповідає `SummaryPanelController`. На ньому відображаються результати проходження: кількість перевірених мандрівників, правильні рішення, помилки, отриманий досвід, гроші та стан службової цілісності. Також користувач може перейти до покращення навичок інспектора, якщо для цього накопичено відповідні ресурси.

У грі також реалізовано екран паузи через `GameplayPauseController`. Він дозволяє тимчасово зупинити проходження, змінити гучність, налаштувати чутливість керування або повернутися до головного меню. Це необхідно для зручності користувача, оскільки симулятор передбачає роботу з великою кількістю інформації, і гравець має мати можливість призупинити процес без втрати поточного стану.

Таким чином, реалізований UI об'єднує кілька функціональних частин: документи, службову базу, панель рішень, HUD, паузу та підсумковий екран. Кожен із цих елементів має окреме призначення, але разом вони формують робочий процес інспектора. Завдяки цьому користувач може послідовно отримувати інформацію, аналізувати її, приймати рішення та бачити наслідки своїх дій.

4.4 Реалізація NPC, транспорту та шлагбаума

У розробленому симуляторі тривимірна сцена використовується не лише як фон для перевірки документів. Вона відтворює послідовність дій на контрольно-пропускному пункті: автомобіль під'їжджає до зони контролю, мандрівник виходить, наближається до робочого місця інспектора, передає документи, очікує рішення та після завершення перевірки залишає сцену. Така побудова робить процес більш цілісним, оскільки користувач бачить не тільки інтерфейс документів, а й загальну логіку роботи пункту пропуску.

За появу мандрівника та його взаємодію зі сценою відповідає модуль `TravelerSpawner`. Він використовується для створення NPC у потрібній точці та підключення його до поточного службового кейсу. Завдяки цьому персонаж не існує окремо від логіки гри, а пов'язаний із даними, які гравець бачить у документах і службовому моніторі. Після завершення перевірки цей же процес переходить до наступного мандрівника.

Рух NPC реалізовано за допомогою `TravelerAgent`. Цей компонент керує переміщенням персонажа між основними точками сцени: зоною появи, місцем очікування біля інспектора та виходом після прийняття рішення. У поведінці персонажа передбачено кілька станів, зокрема рух до робочого місця, очікування рішення та завершення перевірки. Це дозволяє не просто миттєво показувати нового мандрівника, а відтворювати коротку послідовність його дій у просторі.

Для візуального оживлення персонажів використовуються `TravelerAnimationController` і `ProceduralHumanoidAnimator`. Вони відповідають за базову анімацію руху, положення корпусу, жест передачі документів та інші прості дії NPC. У межах дипломного проєкту така реалізація є доцільною, оскільки вона дозволяє показати поведінку мандрівника без використання складної системи професійних анімацій. Головним завданням було забезпечити зрозумілу дію персонажа: він підходить до інспектора, передає документи та залишає місце перевірки.

Транспортна частина реалізована через `CheckpointVehicleAnimator`. Цей компонент керує під'їздом автомобіля до пункту контролю, зупинкою біля робочої зони та подальшим рухом після рішення гравця. Автомобіль у симуляторі має не лише візуальне значення, оскільки дані транспортного засобу можуть бути частиною службового кейсу. Якщо номер або інша інформація про авто не відповідає документам чи монітору, це стає підставою для відмови у перетині кордону.

Роботу шлагбаума забезпечує `BarrierGateController`. Після позитивного рішення шлагбаум відкривається, дозволяючи транспортному засобу залишити

пункт контролю. У разі відмови логіка руху відрізняється, що візуально підкреслює наслідок прийнятого рішення. Також у роботі шлагбаума враховується стан сигналів, що допомагає користувачу краще сприймати результат перевірки.

Для передачі документів використовується окрема логіка, пов'язана з DocumentTransferAnimator. Вона відповідає за момент, коли мандрівник передає документи інспектору, після чого користувач отримує можливість переглянути паспорт, картку в'їзду та службову інформацію. Завдяки цьому поява документів в інтерфейсі не виглядає відірваною від подій у сцені, а пов'язується з діями NPC.

Усі ці компоненти працюють разом із основним менеджером гри. Спочатку запускається службовий кейс, потім автомобіль під'їжджає до контрольної зони, NPC підходить до інспектора, документи стають доступними для перегляду, після чого користувач приймає рішення (див. рис 4.3).

Далі транспорт і мандрівник завершують свою частину сценарію, а система переходить до наступного випадку.



Рис. 4.3. Сцена з NPC

Реалізація NPC, транспорту та шлагбаума забезпечує візуальне відтворення роботи контрольно-пропускного пункту. Ці елементи не замінюють логіку перевірки документів, але доповнюють її та роблять ігровий процес більш послідовним. Завдяки взаємодії персонажів, автомобілів, документів і шлагбаума користувач сприймає перевірку як єдиний службовий процес, а не як окремий набір інтерфейсних дій.

4.5 Тестування ігрового симулятора

Тестування розробленого симулятора проводилося для перевірки коректної роботи основних механік гри та взаємодії між її модулями. Оскільки програмний продукт є Unity-грою, перевірка виконувалася не лише на рівні окремих скриптів, а й у межах повного ігрового сценарію: запуск сцени, поява мандрівника, відображення документів, прийняття рішення, оновлення показників і завершення зміни.

Верифікація програмного продукту була зосереджена на функціональному тестуванні критичних компонентів, що формують ядро ігрового процесу. Програма випробувань охоплювала перевірку логіки ініціалізації (запуск головного меню та коректність завантаження ігрової сцени) та базових механік симулятора. Зокрема, підтверджено працездатність алгоритму процедурної генерації службових кейсів, коректність рендерингу інтерфейсних елементів (паспорта, картки в'їзду та службового монітора) і точність калькуляції в системі нарахування балів.

Окремо протестовано логіку обробки нештатних ситуацій (спроби підкупу), поведінкові моделі NPC та транспортних засобів, а також спрацьовування тригерів шлагбаума, меню паузи та підсумкового екрана сесії.

Тестування виконувалося методом проходження типових і проблемних сценаріїв. Для цього запускалася зміна, після чого перевірялися різні типи кейсів: коректний мандрівник, прострочений паспорт, невідповідність персональних даних, підозріла мета візиту, проблема з автомобілем, службове орієнтування та

спроба хабаря. У кожному випадку перевірялося, чи правильно система визначає очікуване рішення та чи коректно реагує на вибір гравця.

Під час перевірки окремо аналізувалася правильність роботи системи оцінювання. Для цього один і той самий тип кейсу проходився з правильним і неправильним рішенням. Наприклад, для мандрівника без порушень очікуваною дією був дозвіл на в'їзд, а відмова вважалася помилкою. Для кейсу з простроченим паспортом або невідповідністю даних правильним рішенням була відмова. Така перевірка підтвердила, що система оцінювання враховує не сам факт натискання кнопки, а логіку конкретної ситуації. Результати основних тестових перевірок наведено в таблиці 4.1.

Також перевірялася робота інтерфейсу. Документи мають оновлюватися після переходу до нового мандрівника, а службовий монітор - показувати інформацію саме поточного кейсу. Якщо попередні дані залишалися б на екрані, це могло б призвести до неправильного рішення гравця. Під час тестування було підтверджено, що паспорт, картка в'їзду, монітор і HUD оновлюються разом із новим службовим випадком.

Протестувано рух NPC і транспорту, оскільки ці елементи мають працювати узгоджено з ігровою логікою. Перевірялося, чи автомобіль під'їжджає до потрібної точки, чи мандрівник переходить до місця перевірки, чи документи передаються інспектору, а також чи транспорт залишає сцену після завершення кейсу. Це дозволило переконатися, що 3D-події не конфліктують із відображенням документів і рішеннями користувача.

У результаті тестування встановлено, що основні функціональні модулі симулятора працюють відповідно до поставлених вимог. Гра коректно запускається, формує службові кейси, відображає документи, обробляє рішення гравця, оновлює показники зміни, відтворює рух NPC і транспорту та показує підсумок проходження. Надалі тестування може бути розширене за рахунок перевірки більшої кількості кейсів, додавання нових типів документів і тестування продуктивності на різних конфігураціях комп'ютерів.

Таблиця 4.1

Результати тестування основних функцій симулятора

№	Об'єкт тестування	Очікуваний результат	Фактичний результат
1	Запуск головного меню	Відображаються кнопки початку гри, налаштувань і виходу	Меню відкривається коректно
2	Перехід до ігрової сцени	Після запуску гри відкривається сцена прикордонного пункту	Перехід виконується без помилок
3	Генерація службового кейсу	Для мандрівника створюються документи, службові дані та правильне рішення	Дані формуються та передаються в інтерфейс
4	Відображення документів	Паспорт і картка в'їзду показують дані поточного мандрівника	Документи оновлюються відповідно до кейсу
5	Робота службового монітора	Монітор показує додаткову інформацію, ризики та рекомендації	Дані монітора відображаються коректно
6	Прийняття правильного рішення	За правильну дію нараховуються бали та досвід	Рахунок і досвід оновлюються
7	Прийняття помилкового рішення	Помилка фіксується, результат зміни погіршується	Система коректно враховує помилку
8	Механіка хабаря	Гроші збільшуються, а службова цілісність зменшується	Показники змінюються відповідно до логіки
9	Рух NPC і транспорту	Мандрівник і автомобіль проходять послідовність прибуття та виходу	Рух об'єктів виконується в правильному порядку
10	Робота шлагбаума	Після позитивного рішення шлагбаум відкривається	Шлагбаум реагує на результат перевірки
11	Пауза	Гра зупиняється, відкривається меню паузи	Пауза працює коректно
12	Підсумок зміни	Після завершення черги показується статистика проходження.	Відповідає очікуваному результату. Підсумковий екран коректно відображає результати.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання з розробки тривимірного ігрового симулятора професійної діяльності «На кордоні». Відповідно до визначених у вступі завдань, отримано такі основні результати:

1. Проаналізовано предметну галузь ігрових симуляторів професійної діяльності. Встановлено, що ключовими факторами ефективності таких програмних продуктів є не лише візуальне відтворення середовища, а й точне моделювання логіки реальних процесів, варіативність ігрових ситуацій, ергономічність інтерфейсу та пряма залежність ігрового прогресу від наслідків рішень користувача.
2. Розглянуто існуючі програмні рішення у сфері документального та прикордонного контролю (зокрема Papers, Please, Border Officer, Contraband Police та Not Tonight). На основі проведеного порівняльного аналізу виявлено їхні сильні сторони та обґрунтовано необхідність створення нового симулятора, який би наскрізно поєднував у єдиному 3D-просторі механіки перевірки документів, роботу зі службовим монітором, аналіз транспортних даних, систему оцінювання, прогресію інспектора та механіку взаємодії з нештатними ситуаціями (спробами підкупу).
3. Визначено функціональні та нефункціональні вимоги до створюваного симулятора. Сформовано специфікацію параметрів генерації персонажів, логіки виявлення порушень у документах, критеріїв оцінювання дій гравця, а також технічні вимоги щодо продуктивності та стабільності ігрового рушія.
4. Обґрунтовано вибір технологічного стеку розробки. Використання комп'ютерного рушія Unity, об'єктно-орієнтованої мови програмування C#, інструментів Unity UI та TextMeshPro, а також конвеєра рендерингу Universal Render Pipeline (URP) дозволило забезпечити ефективне паралельне функціонування 3D-сцени, обчислювальної логіки, поведінкових скриптів NPC, фізики транспорту та інтерфейсних ассетів документів.

5. Спроектовано архітектуру симулятора, ігровий цикл та склад основних програмних модулів. За допомогою інструментарію UML (діаграм прецедентів, компонентів та послідовності) формалізовано наскрізну логіку ігрової зміни. Вона охоплює запуск сесії, процедурне формування кейсу, прибуття мандрівника, аналіз паперів, винесення вердикту інспектором, калькуляцію балів та перехід до наступної перевірки.
6. Реалізовано ключові програмні модулі гри. Центральний керуючий компонент BorderGameManager забезпечує координацію ігрового циклу та синхронізацію суміжних підсистем. Алгоритм процедурного моделювання ситуацій реалізовано через класи TravelerCaseGenerator та TravelerCase, що дозволило автоматизувати створення унікальних службових кейсів: від повністю коректних документів до виявлення прострочених паспортів, розбіжностей у персональних даних, сумнівної мети візиту, неповного пакета інформації, роботи за службовими орієнтуваннями, невідповідностей автомобіля чи спроб дати хабар.

Для роботи з інтерфейсом реалізовано окремі модулі відображення паспорта, картки мети в'їзду, службового монітора, панелі рішень, HUD, паузи та підсумкового екрана. Завдяки цьому користувач отримує інформацію з кількох джерел і має самостійно зіставляти дані перед прийняттям рішення. Реалізація NPC, транспорту та шлагбаума дозволила подати перевірку не лише як роботу з документами, а як послідовний 3D-процес на контрольно-пропускному пункті.

7. Проведено функціональне тестування основних компонентів гри. Під час тестування було перевірено запуск меню, перехід до ігрової сцени, генерацію службових кейсів, відображення документів, роботу службового монітора, прийняття правильних і помилкових рішень, механіку хабаря, рух NPC, транспорт, шлагбаум, паузу та підсумок зміни. Результати тестування показали, що основні функції симулятора працюють відповідно до поставлених вимог, а взаємодія між модулями забезпечує повний цикл проходження ігрової зміни.

Практичне значення роботи полягає в тому, що створений програмний продукт може використовуватися як інтерактивний симулятор для ознайомлення з логікою прикордонного контролю, тренування уважності, аналізу документів і прийняття рішень у типових та нестандартних ситуаціях. Проєкт також має можливість подальшого розвитку: додавання нових типів документів, розширення службових сценаріїв, покращення поведінки NPC, деталізація транспортної перевірки, створення навчального режиму та розширення статистики проходження.

Поставлену мету кваліфікаційної роботи досягнуто. У результаті виконання роботи було спроектовано та реалізовано 3D-ігровий симулятор прикордонного контролю, який поєднує процедурну генерацію службових кейсів, перевірку документів і транспортних даних, службовий монітор, систему оцінювання рішень, механіку хабаря, прогресію інспектора та анімовану роботу контрольно-пропускного пункту.

1. Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

Деркач А.І., Золотухіна О.А. Визначення концепту ігрового симулятора прикордонного контролю «На кордоні». II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 390-391.

2. Деркач А.І., Золотухіна О.А. Проєктування ігрової логіки перевірки документів прикордонного контролю у симуляторі «На кордоні». VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С. 151-153.

ПЕРЕЛІК ПОСИЛАНЬ

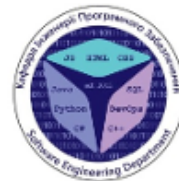
1. Про прикордонний контроль : Закон України від 05.11.2009 № 1710-VI // База даних «Законодавство України» / Верховна Рада України. URL: <https://zakon.rada.gov.ua/go/1710-17> (дата звернення: 13.05.2026).
2. Про Державну прикордонну службу України : Закон України від 03.04.2003 № 661-IV // База даних «Законодавство України» / Верховна Рада України. URL: <https://zakon.rada.gov.ua/go/661-15> (дата звернення: 13.05.2026).
3. Державна прикордонна служба України. Пункти пропуску (пункти контролю). URL: <https://dpsu.gov.ua/uk/punkti-propusku-punkti-kontrolyu> (дата звернення: 13.05.2026).
4. Zyda M. From Visual Simulation to Virtual Reality to Games. Computer. 2005. Vol. 38, № 9. P. 25-32. DOI: <https://doi.org/10.1109/MC.2005.297>.
5. Connolly T. M., Boyle E. A., MacArthur E., Hainey T., Boyle J. M. A systematic literature review of empirical evidence on computer games and serious games. Computers & Education. 2012. Vol. 59, Iss. 2. P. 661-686. DOI: <https://doi.org/10.1016/j.compedu.2012.03.004>.
6. Hunicke R., LeBlanc M., Zubek R. MDA: A Formal Approach to Game Design and Game Research. Proceedings of the AAAI Workshop on Challenges in Game AI. 2004. URL: <https://aaai.org/papers/ws04-04-001-mda-a-formal-approach-to-game-design-and-game-research/> (дата звернення: 13.05.2026).
7. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. URL: <https://www.routledge.com/The-Art-of-Game-Design-A-Book-of-Lenses-Third-Edition/Schell/p/book/9781138632059> (дата звернення: 13.05.2026).
8. Unity Technologies. Unity 6.4 User Manual. URL: <https://docs.unity3d.com/6000.4/Documentation/Manual/UnityManual.html> (дата звернення: 13.05.2026).
9. Unity Technologies. Unity Scripting API. URL: <https://docs.unity3d.com/6000.4/Documentation/ScriptReference/index.html> (дата звернення: 13.05.2026).

10. Unity Technologies. MonoBehaviour. Unity Scripting API. URL: <https://docs.unity3d.com/6000.4/Documentation/ScriptReference/MonoBehaviour.html> (дата звернення: 13.05.2026).
11. Unity Technologies. Unity UI: Unity User Interface. URL: <https://docs.unity3d.com/Packages/com.unity.ugui%401.0/manual/index.html> (дата звернення: 13.05.2026).
12. Unity Technologies. TextMesh Pro Documentation. URL: <https://docs.unity3d.com/Packages/com.unity.textmeshpro%403.2/manual/index.html> (дата звернення: 13.05.2026).
13. Unity Technologies. Introduction to the Universal Render Pipeline. URL: <https://docs.unity3d.com/6000.6/Documentation/Manual/urp/urp-introduction.html> (дата звернення: 13.05.2026).
14. Microsoft. C# language documentation. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/> (дата звернення: 13.05.2026).
15. Chacon S., Straub B. Pro Git. 2nd ed. Apress, 2014. URL: <https://git-scm.com/book/en/v2> (дата звернення: 13.05.2026).
16. GitHub Docs. About repositories. URL: <https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories> (дата звернення: 13.05.2026).
17. Blender Foundation. Blender Documentation. URL: <https://docs.blender.org/> (дата звернення: 13.05.2026).
18. Papers, Please. Official Website / 3909 LLC. URL: <https://papersplea.se/> (дата звернення: 13.05.2026).
19. Border Officer. Steam Store Page / Cheesecake Dev. URL: https://store.steampowered.com/app/1057180/Border_Officer/ (дата звернення: 13.05.2026).
20. Contraband Police. Steam Store Page / Crazy Rocks, PlayWay S.A. URL: https://store.steampowered.com/app/756800/Contraband_Police/ (дата звернення: 13.05.2026).
21. Not Tonight. Steam Store Page / PanicBarn, No More Robots. URL: https://store.steampowered.com/app/733790/Not_Tonight/ (дата звернення: 13.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка ігрового симулятора прикордонного контролю

Виконав студент 4 курсу
групи ПД-42

Деркач Артьом Ігорович
Керівник роботи

канд. техн. наук, доц. Золотухіна Оксана Анатоліївна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу підготовки та відпрацювання навичок перевірки документів та прийняття рішень інспектором прикордонного контролю з використанням ігрової симуляції.

Об'єкт дослідження: процеси перевірки документів та прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

Предмет дослідження: методи та програмні засоби ігрової симуляції процесів перевірки документів і підтримки прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь ігрових симуляторів професійної діяльності.
2. Розглянути існуючі програмні рішення, пов'язані з прикордонним або документальним контролем.
3. Визначити функціональні та нефункціональні вимоги до симулятора процесів перевірки документів та прийняття рішень інспектором прикордонного контролю у типових і нестандартних ситуаціях.
4. Обґрунтувати вибір засобів програмної реалізації.
5. Спроекувати архітектуру, ігровий цикл і основні програмні модулі застосунку.
6. Реалізувати генерацію мандрівників, перевірку документів, службовий монітор, систему рішень, NPC, транспорт і шлагбаум.
7. Провести тестування ігрового застосунку.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Papers, Please	Border Officer	Contraband Police	Not Tonight	На Кордоні
Формат	2D	3D	3D	2D	3D
Документи	детальна перевірка	середня	допоміжна	спрощена	паспорт + картка + монітор
Транспорт	—	обмежено	детальний огляд	—	дані авто, рух, шлагбаум
Службова база	частково	обмежено	частково	обмежено	службовий монітор
Варіативність	наявна	обмежена	наявна	обмежена	процедурна генерація кейсів
Етичний вибір	моральний тиск	окремі ситуації	не головне	другорядно	Хабар і баланс
Прогресія	обмежена	часткова	наявна	обмежена	досвід, рівень, навички

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Взаємодія з відвідувачем (Почати розмову, попросити документи, брати хабар).
2. Можливість перевірки документів (паспорт, віза, дозволи).
3. Прийняття рішення (дозволити або заборонити в'їзд).
4. Виявлення помилок у документах.

Нефункціональні вимоги до гри:

1. Сумісність з операційною системою Windows, MacOS.
2. Забезпечення стабільного фреймрейту (60к/с).
3. Мінімальні системні вимоги: Dual Core, 4GB RAM, GTX 730 Windows 10 (64-bit) / mac OS 10.15 Catalina
4. Рекомендовані системні вимоги: Quad Core, 8GB RAM, GTX 750 Ti Windows 10 (64-bit) / mac OS 11 Big Sur

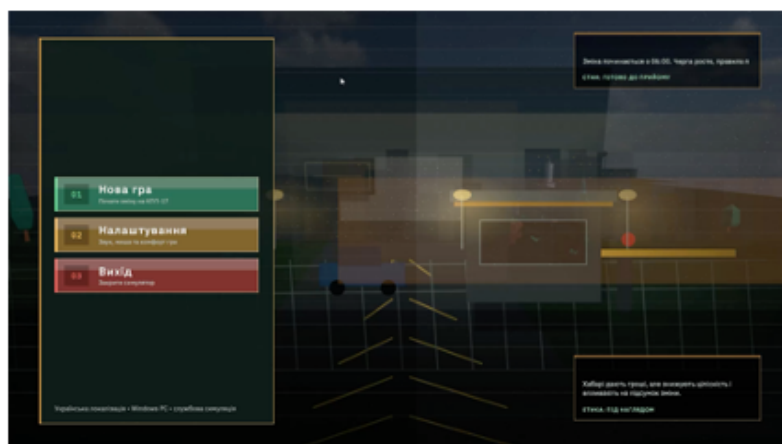
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



6

ЕКРАННІ ФОРМИ



Ігрова сцена з приходом NPC

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Деркач А.І., Золотухіна О.А. Визначення концепту ігрового симулятора прикордонного контролю «На кордоні». II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 390-391.
2. Деркач А.І., Золотухіна О.А. Проектування ігрової логіки перевірки документів прикордонного контролю у симуляторі «На кордоні». VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С. 151-153.

12

ВИСНОВКИ

1. Проведений аналіз предметної галузі професійних ігрових симуляторів дозволив визначити та систематизувати ключові особливості моделювання процесів прикордонного контролю.
2. На основі порівняльного аналізу аналогів обґрунтовано концепцію власної розробки, що поєднує механіки перевірки документів, огляду авто, взаємодії з монітором, NPC-процеси та систему прогресії.
3. Шляхом дослідження потреб цільової аудиторії та технічних обмежень сформульовано й деталізовано функціональні та нефункціональні вимоги до симулятора.
4. З урахуванням сучасних технологій та вимог проєкту обрано доцільний технологічний стек: ігровий рушій Unity, мову програмування C#, конвеєр рендерингу URP, а також системи Unity UI та TextMeshPro.
5. Розроблено модульну архітектуру, базовий ігровий цикл, алгоритми генерації кейсів, логіку перевірки документів та систему оцінювання гравця.
6. Створено та інтегровано у середовищі Unity ключові програмні модулі: BorderGameManager, TravelerCaseGenerator, UI документів, ScoreSystem та системи управління NPC, транспортом і шлагбаумом.
7. Після завершення основних етапів розробки проведено тестування комплексних тест-кейсів, що дозволило верифікувати працездатність ігрових сценаріїв, механіку хабаря та фінал зміни.

ДОДАТОК В. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

Модель службового кейсу мандрівника та визначення правильного рішення

Файл: Assets/Scripts/Data/TravelerCase.cs

```
public enum TravelerCaseType
{
    ValidTraveler,
    ExpiredPassport,
    MismatchedData,
    SuspiciousPurpose,
    MissingDetail,
    BribeOffer,
    VehicleMismatch,
    WatchlistAlert,
    RandomSecondaryInspection
}

[Serializable]
public enum BorderDecision
{
    Approve,
    Reject,
    AcceptBribe
}

[Serializable]
public class TravelerCase
{
    public TravelerCaseType caseType;
    public TravelerIdentity identity = new
TravelerIdentity();
    public PassportData passport = new
PassportData();
    public EntryPermitData entryPermit =
new EntryPermitData();
    public bool hasDataMismatch;
    public bool hasExpiredPassport;
    public bool hasMissingDetail;
    public bool offersBribe;
    public int bribeAmount;
    public string suspiciousNote;
    public string databaseStatus;
    public string riskLevel;
    public string databaseNote;
    public string behaviorNote;
    public string inspectionHint;
    public string travelerStatement;
    public string interviewQuestion;
    public string interviewAnswer;
    public string
secondaryInspectionReason;
    public string vehicleInspectionNote;
    public string liveEventNote;
    public bool
requiresSecondaryInspection;
    public bool hasVehicleMismatch;
    public bool hasWatchlistAlert;
```

```
public bool isRandomComplianceCheck;
public bool isValid;

    public bool CanOfferBribe =>
offersBribe && bribeAmount > 0;

    public BorderDecision
GetCorrectDecision()
    {
        return isValid ?
BorderDecision.Approve :
BorderDecision.Reject;
    }

    public string GetIssueSummary()
    {
        if (hasExpiredPassport)
        {
            return "Паспорт прострочений";
        }

        if (hasDataMismatch)
        {
            return "Виявлено
невідповідність даних";
        }

        if (hasMissingDetail)
        {
            return "Бракує обов'язкових
даних";
        }

        if (hasVehicleMismatch)
        {
            return "Дані авто не
збігаються";
        }

        if (hasWatchlistAlert)
        {
            return "Є службове
орієнтування";
        }

        if (requiresSecondaryInspection)
        {
            return isRandomComplianceCheck
? "Вибіркова вторинна перевірка" :
"Потрібна вторинна перевірка";
        }
    }
```

Генерація службових кейсів мандрівників з урахуванням ваг типів ситуацій

Файл: Assets/Scripts/Gameplay/TravelerCaseGenerator.cs

```

    public List<TravelerCase>
CreateShiftCases(int count)
    {
        List<TravelerCase> cases = new
List<TravelerCase>(count);

        for (int index = 0; index < count;
index++)
        {
            cases.Add(GenerateCase(GetWeig
htedCaseType()));
        }

        return cases;
    }

    private TravelerCaseType
GetWeightedCaseType()
    {
        int totalWeight =
            validWeight +
            expiredPassportWeight +
            mismatchedDataWeight +
            suspiciousPurposeWeight +
            missingDetailWeight +
            bribeOfferWeight +
            vehicleMismatchWeight +
            watchlistAlertWeight +
            randomSecondaryInspectionWeigh
t;

        if (totalWeight <= 0)
        {
            return
TravelerCaseType.ValidTraveler;
        }

        int roll = Random.Range(0,
totalWeight);

        if (roll < validWeight)
        {
            return
TravelerCaseType.ValidTraveler;
        }

        roll -= validWeight;
        if (roll < expiredPassportWeight)
        {
            return
TravelerCaseType.ExpiredPassport;
        }

        roll -= expiredPassportWeight;
        if (roll < mismatchedDataWeight)
        {
            return
TravelerCaseType.MismatchedData;
        }

        roll -= mismatchedDataWeight;
        if (roll <
suspiciousPurposeWeight)
        {
            return
TravelerCaseType.SuspiciousPurpose;
        }

        roll -= suspiciousPurposeWeight;

```

```

        if (roll < missingDetailWeight)
        {
            return
TravelerCaseType.MissingDetail;
        }

        roll -= missingDetailWeight;
        if (roll < bribeOfferWeight)
        {
            return
TravelerCaseType.BribeOffer;
        }

        roll -= bribeOfferWeight;
        if (roll < vehicleMismatchWeight)
        {
            return
TravelerCaseType.VehicleMismatch;
        }

        roll -= vehicleMismatchWeight;
        if (roll < watchlistAlertWeight)
        {
            return
TravelerCaseType.WatchlistAlert;
        }

        return
TravelerCaseType.RandomSecondaryInspection
;
    }

    private TravelerCase
GenerateCase(TravelerCaseType caseType)
    {
        TravelerCase travelerCase =
CreateBaseValidCase();
        travelerCase.caseType = caseType;

        switch (caseType)
        {
            case
TravelerCaseType.ValidTraveler:
                travelerCase.isValid =
true;
                travelerCase.databaseStatu
s = "Запис активний";
                travelerCase.riskLevel =
"Низький";
                travelerCase.databaseNote
= "Збігів із попередженнями немає";
                break;

            case
TravelerCaseType.ExpiredPassport:
                travelerCase.isValid =
false;
                travelerCase.hasExpiredPas
sport = true;
                travelerCase.passport.isEx
pired = true;
                travelerCase.passport.expi
ryDate = GetPastDate();
                travelerCase.databaseStatu
s = "Документ прострочений";
                travelerCase.riskLevel =
"Високий";
                travelerCase.databaseNote
= "Дата дієвості в базі вже минула";

```

```

        travelerCase.inspectionHint = "Паспортний запис прострочений.  
Правильна дія - не пропустити.";
        break;

    case
    TravelerCaseType.MismatchedData:
        travelerCase.isValid =
        false;

        travelerCase.hasDataMismatch = true;

        travelerCase.entryPermit.hasMismatch = true;

        travelerCase.entryPermit.fullName =
        GenerateDifferentFullName(travelerCase.identity.fullName);

```

```

        travelerCase.entryPermit.note = "Ім'я у картці мети в'їзду не  
збігається з паспортом";

        travelerCase.databaseStatus = "Невідповідність";

        travelerCase.riskLevel =
        "Середній";

        travelerCase.databaseNote =
        "Ім'я в картці не збігається з  
паспортним записом";

        travelerCase.inspectionHint = "Звірте повне ім'я в паспорті та  
картці. Невідповідність є підставою для  
відмови.";

        break;

```

Керування зміною, прибуттям мандрівника та обробкою рішення гравця

Файл: Assets/Scripts/Gameplay/BorderGameManager.cs

```

public void StartShift()
{
    Time.timeScale = 1f;
    travelerQueue.Clear();
    currentTraveler = null;
    currentTravelerCase = null;
    isAwaitingDecision = false;
    lastDecisionAllowsVehicleDeparture = false;

    progressionSystem?.ApplyBonusesTo(scoreSystem);
    shiftBriefingSystem?.GenerateForShift();
    scoreSystem?.ResetSession();
    shiftManager?.ResetShift();
    summaryPanelController?.Hide();

    if (decisionPanelController != null)
    {
        decisionPanelController.SetWaitingState();
        decisionPanelController.SetInteractable(false, false);
    }

    barrierGate?.SnapClosed();
    travelerSpawner?.ResetVehicle();
    checkpointVehicle?.ResetVehicle();
    documentTransferAnimator?.HideImmediately();
    ClearDocuments();
    FillTravelerQueue();
    RefreshHud();
    hudController?.UpdateShiftBriefing(shiftBriefingSystem?.CurrentBriefing);

    if
    (shiftBriefingSystem?.CurrentBriefing != null)
    {
        hudController?.ShowFeedback("Зведення зміни",
        shiftBriefingSystem.CurrentBriefing.radioMessage, 3.2f);
    }
}

```

```

ScheduleNextTraveler(firstTravelerDelay);
}

private void FillTravelerQueue()
{
    if (travelerCaseGenerator == null || shiftManager == null)
    {
        return;
    }

    List<TravelerCase> generatedCases =
    travelerCaseGenerator.CreateShiftCases(shiftManager.TravelersPerShift);
    shiftBriefingSystem?.ApplyToCases(generatedCases);

    for (int index = 0; index < generatedCases.Count; index++)
    {
        travelerQueue.Enqueue(generatedCases[index]);
    }

    private IEnumerator
    SpawnNextTravelerRoutine()
    {
        spawnRoutine = null;

        if (shiftManager == null || shiftManager.IsShiftComplete || travelerQueue.Count == 0)
        {
            FinishShift();
            yield break;
        }

        if (decisionPanelController != null)
        {
            decisionPanelController.SetWaitingState(
                "Підготовка до перевірки",
                "Авто мандрівника під'їжджає до обочини." +

```

```

(shiftBriefingSystem != null ?
shiftBriefingSystem.GetArrivalLine() :
"Слідкуйте за монітором.));
    decisionPanelController.SetInteractable(false, false);
}

    ClearDocuments();
    currentTravelerCase =
travelerQueue.Dequeue();
    serviceMonitorView?.ConfigureAnalysisLevel(progressionSystem != null ?
progressionSystem.AnalysisLevel : 0);
    serviceMonitorView?.Display(currentTravelerCase);

    if (travelerSpawner != null)
    {
        yield return
travelerSpawner.SpawnTravelerRoutine(currentTravelerCase, spawnedTraveler =>
currentTraveler = spawnedTraveler);
    }
    else
    {
        currentTraveler = null;
    }

    private void
HandleTravelerArrived(TravelerAgent
traveler)
    {
        if (currentTravelerCase == null)
        {
            return;
        }

        isAwaitingDecision = true;
        passportDocumentView?.Display(currentTravelerCase);
        entryCardDocumentView?.Display(currentTravelerCase);
        serviceMonitorView?.ConfigureAnalysisLevel(progressionSystem != null ?
progressionSystem.AnalysisLevel : 0);
        serviceMonitorView?.Display(currentTravelerCase);
        documentTransferAnimator?.PlayTransfer();
        decisionPanelController?.ShowTraveler(currentTravelerCase);
        decisionPanelController?.SetInteractable(true,
currentTravelerCase.CanOfferBribe);
        hudController?.ShowFeedback("Документи передано", "Перевірте паспорт, картку мети в'їзду та службовий монітор.",
1.35f);
        AudioManager.Instance?.PlayTravelerArrival();
        AudioManager.Instance?.PlayDocumentSlide();
    }

    private void
HandleDecisionSelected(BorderDecision
decision)
    {

```

```

        if (!isAwaitingDecision ||
currentTravelerCase == null ||
currentTraveler == null || scoreSystem ==
null || shiftManager == null)
        {
            return;
        }

        isAwaitingDecision = false;
        decisionPanelController?.SetInteractable(false, false);

        DecisionOutcome outcome =
scoreSystem.EvaluateDecision(currentTravelerCase, decision);
        ProgressionEvent progressionEvent
=
progressionSystem?.RecordDecision(outcome);
        shiftManager.RegisterTravelerProcessed();
        RefreshHud();

        string feedbackBody =
outcome.secondaryMessage;
        if (progressionEvent != null &&
!string.IsNullOrEmpty(progressionEvent.feedbackMessage))
        {
            feedbackBody += "\n" +
progressionEvent.feedbackMessage;
        }

        hudController?.ShowFeedback(outcome.primaryMessage, feedbackBody,
feedbackDuration);

        if (outcome.tookBribe)
        {
            AudioManager.Instance?.PlayBribe();
        }
        else if (!outcome.isCorrect)
        {
            AudioManager.Instance?.PlayError();
        }
        else
        {
            AudioManager.Instance?.PlayDecision();
        }

        AudioManager.Instance?.PlayStamp();
        ;
        lastDecisionAllowsVehicleDeparture
= decision != BorderDecision.Reject;
        if
(lastDecisionAllowsVehicleDeparture)
        {
            barrierGate?.SetGateOpen(true)
;
        }
        else
        {
            barrierGate?.SetGateOpen(false)
;
        }
    }

```

```

        currentTraveler.LeaveCheckpoint();
        currentTravelerCase = null;

    private void FinishShift()
    {
        isAwaitingDecision = false;
        ClearDocuments();
        decisionPanelController?.SetWaitingState("Зміну завершено", "Перегляньте підсумок зміни");
        decisionPanelController?.SetInteractive(false, false);

        if (summaryPanelController != null && scoreSystem != null && shiftManager != null)
        {
            ShiftResult result = scoreSystem.BuildShiftResult(shiftManager.TravelersPerShift, shiftManager.ProcessedTravelers);
            ProgressionEvent shiftProgression = progressionSystem?.RecordShiftCompleted(result);
            RefreshHud();
        }
    }

```

```

        summaryPanelController.Show(result, progressionSystem?.GetSnapshot(), shiftProgression != null ? shiftProgression.feedbackMessage : string.Empty);
    }

    private void RefreshHud()
    {
        if (hudController == null || scoreSystem == null || shiftManager == null)
        {
            return;
        }

        ShiftResult result = scoreSystem.BuildShiftResult(shiftManager.TravelersPerShift, shiftManager.ProcessedTravelers);
        hudController.UpdateStats(result);
        hudController.UpdateProgression(progressionSystem?.GetSnapshot());
    }

```

Оцінювання рішення, хабаря та підсумку зміни

Файл: Assets/Scripts/Systems/ScoreSystem.cs

```

    public DecisionOutcome EvaluateDecision(TravelerCase travelerCase, BorderDecision decision)
    {
        DecisionOutcome outcome = new DecisionOutcome();
        bool tookBribe = decision == BorderDecision.AcceptBribe;
        bool isCorrect = !tookBribe && decision == travelerCase.GetCorrectDecision();

        outcome.tookBribe = tookBribe;
        outcome.isCorrect = isCorrect;

        if (tookBribe)
        {
            int bribeReward = travelerCase.bribeAmount > 0 ? travelerCase.bribeAmount : defaultBribeReward;
            int effectiveIntegrityPenalty = Mathf.Max(0, integrityPenalty - integrityPenaltyReduction);
            money += bribeReward;
            bribesTaken++;
            integrity = Mathf.Clamp(integrity - effectiveIntegrityPenalty, 0, startingIntegrity);

            outcome.scoreDelta = 0;
            outcome.moneyDelta = bribeReward;
            outcome.integrityDelta = -effectiveIntegrityPenalty;
        }
    }

```

```

        outcome.primaryMessage = "Хабар взято";
        outcome.secondaryMessage = "Гроші +" + bribeReward + ". Цілісність служби -" + effectiveIntegrityPenalty + ". " + travelerCase.GetIssueSummary();
        return outcome;
    }

    if (isCorrect)
    {
        int scoreGain = correctDecisionScore + correctDecisionScoreBonus;
        correctDecisions++;
        score += scoreGain;
        outcome.scoreDelta = scoreGain;
        outcome.primaryMessage = "Правильне рішення";
        outcome.secondaryMessage = travelerCase.GetOutcomeDetails(decision);
    }
    else
    {
        mistakes++;
        score += incorrectDecisionScore;
        outcome.scoreDelta = incorrectDecisionScore;
        outcome.primaryMessage = "Неправильне рішення";
        outcome.secondaryMessage = travelerCase.GetOutcomeDetails(decision);
    }

    return outcome;
}

```

```

    public ShiftResult
    BuildShiftResult(int totalTravelers, int
    processedTravelers)
    {
        return new ShiftResult
        {
            totalTravelers =
            totalTravelers,
            processedTravelers =
            processedTravelers,

```

```

            correctDecisions =
            correctDecisions,
            mistakes = mistakes,
            score = score,
            money = money,
            bribesTaken = bribesTaken,
            integrityRemaining = integrity
        };
    }
}

```

Нарахування досвіду, завершення зміни та покращення інспектора

Файл: Assets/Scripts/Systems/PlayerProgressionSystem.cs

```

    ProgressionEvent progressionEvent
    = new ProgressionEvent();

    if (outcome == null)
    {
        return progressionEvent;
    }

    if (outcome.tookBribe)
    {
        currentCorrectStreak = 0;
        progressionEvent.feedbackMessage = "Досвід не нараховано: хабар шкодить службовій репутації.";
        SaveProgression();
        return progressionEvent;
    }

    int xp = outcome.isCorrect ?
    correctDecisionXp : mistakeLearningXp;

    if (outcome.isCorrect)
    {
        currentCorrectStreak++;

        if (streakThreshold > 0 &&
        currentCorrectStreak % streakThreshold ==
        0)
        {
            xp += streakBonusXp;
            progressionEvent.streakBonus = true;
        }
        else
        {
            currentCorrectStreak = 0;
        }

        AddExperience(xp,
        progressionEvent);
        progressionEvent.feedbackMessage =
        BuildDecisionProgressionMessage(progressionEvent);
        SaveProgression();
        return progressionEvent;
    }

    public ProgressionEvent
    RecordShiftCompleted(ShiftResult result)
    {

```

```

        ProgressionEvent progressionEvent
        = new ProgressionEvent();
        int xp = shiftCompletionXp;

        if (result != null)
        {
            if (result.mistakes == 0)
            {
                xp += cleanShiftBonusXp;
            }

            if (result.bribesTaken == 0)
            {
                xp += noBribeBonusXp;
            }

            if (result.integrityRemaining
            >= 80)
            {
                xp +=
                highIntegrityBonusXp;
            }

            AddExperience(xp,
            progressionEvent);
            progressionEvent.feedbackMessage =
            BuildShiftProgressionMessage(progressionEvent, result);
            SaveProgression();
            return progressionEvent;
        }

        public bool
        TryPurchaseUpgrade(ProgressionUpgradeType
        upgradeType, out string message)
        {
            message = string.Empty;

            if (upgradePoints <= 0)
            {
                message = "Немає очок покращень. Завершуйте зміни та набирайте досвід.";
                return false;
            }

            int currentLevel =
            GetUpgradeLevel(upgradeType);

            if (currentLevel >=
            maxUpgradeLevel)
            {

```

```

        message =
GetUpgradeName(upgradeType) + " вже має
максимальний рівень.";
        return false;
    }

    SetUpgradeLevel(upgradeType,
currentLevel + 1);
    upgradePoints--;
    SaveProgression();
    message =
GetUpgradeName(upgradeType) + " покращено
до рівня " + GetUpgradeLevel(upgradeType)
+ ".";
    return true;
}

public ProgressionSnapshot
GetSnapshot()
{
    return new ProgressionSnapshot
    {
        level = level,
        experience = experience,
        experienceToNextLevel =
ExperienceToNextLevel,
        upgradePoints = upgradePoints,
        rankName = GetRankName(level),
        procedureLevel =
procedureLevel,
        analysisLevel = analysisLevel,
        ethicsLevel = ethicsLevel,
        logisticsLevel =
logisticsLevel,
        maxUpgradeLevel =
maxUpgradeLevel
    };
}

public string
GetUpgradeName(ProgressionUpgradeType
upgradeType)
{
    switch (upgradeType)
    {
        case
ProgressionUpgradeType.Procedures:
            return "Процедури";
        case
ProgressionUpgradeType.Analysis:
            return "Аналітика бази";
        case
ProgressionUpgradeType.Ethics:
            return "Етика служби";
        case
ProgressionUpgradeType.Logistics:
            return "Логістика черги";

        private void AddExperience(int amount,
ProgressionEvent progressionEvent)
        {
            int xp = Mathf.Max(0, amount);
            progressionEvent.xpGained += xp;
            experience += xp;

            while (experience >=
ExperienceToNextLevel)
            {
                experience -=
ExperienceToNextLevel;
                level++;
                upgradePoints++;
                progressionEvent.leveledUp =
true;
                progressionEvent.newLevel =
level;
                progressionEvent.upgradePoints
Gained++;
            }
        }
    }
}

```

Програмна реалізація масивів даних для процедурної генерації службових кейсів

```

private readonly string[] firstNames =
{
    "Олена", "Ігор", "Марія", "Тарас", "Леся", "Дмитро",
    "Наталія", "Сергій",
    "Юлія", "Максим", "Софія", "Андрій", "Оксана", "Богдан",
    "Катерина", "Роман",
    "Мирослава", "Павло", "Вікторія", "Арсен", "Дарина",
    "Назар", "Ярина", "Степан"
};

private readonly string[] lastNames =
{
    "Марчук", "Кравець", "Власюк", "Гнатенко", "Мельник",
    "Дорошенко", "Савчук", "Климчук",
    "Середа", "Бондар", "Петренко", "Левченко", "Ткаченко",
    "Яремчук", "Коваль", "Руденко",
    "Гордієнко", "Черненко", "Соловей", "Мороз", "Шевчук",
    "Гаврилюк", "Палій", "Данилюк"
};

private readonly string[] nationalities =
{
    "Республіка Севера",
    "Маренська федерація",
    "Острівна держава Велар",
    "Союз долини Ріон",
    "Літарська співдружність",
    "Князівство Арден",
    "Південна Ліга Норі",
    "Вільні міста Каро"
};

private readonly string[] occupations =
{
    "інженер", "викладач", "водій", "медик", "фотограф",
    "торговий представник",
    "студент", "перекладач", "механік", "фермер", "кур'єр",
    "музикант"
};

private readonly string[] validPurposes =
{
    "Туризм",
    "Ділова поїздка",
    "Транзит",
    "Візит до родини",
}

```

```

"Коротке навчання",
"Медична консультація",
"Участь у ярмарку",
"Сервісний ремонт обладнання"
};

private readonly string[] destinationCities =
{
    "Лугоград", "Вежин", "Старий Перевал", "Мирне",
    "Озерний порт", "Залізний Ріг", "Кам'яна брама", "Діброва"
};

private readonly string[] streetNames =
{
    "вул. Прикордонна", "вул. Соснова", "просп. Митний",
    "вул. Ранкова", "пров. Складський",
    "вул. Лікарняна", "вул. Вокзальна", "просп. Центральний"
};

private readonly string[] hostNames =
{
    "Готель \"Північний\"", "ТОВ \"Ремсервіс\"", "родина
    Климчуків", "Навчальний центр \"Обрій\"",
    "клініка \"Світло\"", "складська база \"Міст\"", "ярмарковий
    комітет", "приватний орендодавець"
};

private readonly string[] vehicleModels =
{
    "синій седан", "сірий універсал", "білий фургон", "зелений
    пікап", "жовте таксі", "чорний хетчбек"
};

private readonly string[] declaredItems =
{
    "особисті речі та одяг",
    "робочий ноутбук і зарядні пристрої",
    "медичні довідки та ліки",
    "зразки тканин для ярмарку",
    "інструменти для сервісного ремонту",
    "подарунки для родини",
    "фотообладнання",
    "невеликий багаж без комерційних товарів"
};

private readonly string[] luggageDescriptions =
{
    "одна дорожня сумка",
    "валіза середнього розміру",
    "рюкзак і папка з документами",
    "дві невеликі сумки",
    "інструментальний кейс",
    "чорна валіза з наліпкою готелю",
    "медичний кофр",
    "сірий тканинний рюкзак"
};

private readonly string[] previousVisits =
{
    "перший в'їзд за рік",
    "2 попередні в'їзди без порушень",
    "транзит минулого місяця",
    "регулярні короткі візити",
    "архівних попереджень немає",
    "останній виїзд підтверджено",
    "запис оновлено цього тижня"
};

```

```

private readonly string[] behaviorNotes =
{
    "говорить спокійно, відповіді послідовні",
    "нервує через довгу чергу, але документи подає охайно",
    "поспішає на зустріч і часто дивиться на годинник",
    "тримає документи наготові, контактний",
    "відповідає коротко, але без суперечностей",
    "перепитує правила в'їзду, виглядає розгублено",
    "уважно стежить за перевіркою документів"
};

private readonly string[] inspectionHints =
{
    "Звірте ім'я, номер паспорта, строк дії та адресу
    перебування.",
    "Перевірте, чи збігається ім'я в паспорті та картці мети
    в'їзду.",
    "Порівняйте мету поїздки з адресою, приймаючою
    стороною та багажем.",
    "Зверніть увагу на строк дії паспорта й повноту контактних
    даних.",
    "Якщо всі поля коректні, дозвольте в'їзд; порушення -
    підстава для відмови."
};

private readonly string[] travelerStatements =
{
    "Іду за вказаною адресою, документи підготував
    заздалегідь.",
    "Поспішаю, бо приймаюча сторона чекає біля вокзалу.",
    "Авто орендоване, номер записаний у картці.",
    "У багажі тільки особисті речі та робочі документи.",
    "Маршрут короткий, повернуся після завершення справ.",
    "Контакт приймаючої сторони може підтвердити мету
    поїздки.",
    "Черга затримала мене, але дані в документах актуальні."
};

private readonly string[] interviewQuestions =
{
    "Назвіть адресу перебування.",
    "Хто приймає вас у країні?",
    "Скільки днів плануєте залишатися?",
    "Що знаходиться у багажі?",
    "Якою є точна мета поїздки?",
    "Чи збігається номер авто з карткою в'їзду?"
};

private readonly string[] complianceNotes =
{
    "Система вибіркового контролю позначила кейс для
    вторинної перевірки.",
    "Черговий супервайзер просить перевірити цей запис у
    другій зоні.",
    "Автоматичний ризик-профіль вимагає додаткового
    опитування.",
    "Кейс потрапив у контрольну вибірку якості перевірок."
};

private readonly string[] issuingAuthorities =
{
    "Консульський відділ 03", "Міграційний стіл 12",
    "Паспортне бюро 07",
    "Центр документів Севера", "Архівний офіс Велара",
    "Служба реєстру Ріон"
};

private readonly string[] suspiciousNotes =

```

```
{
"Пояснення маршруту звучить непереконливо",
"Мандрівник уникає прямої відповіді про місце перебування",
"Мета поїздки сформульована надто розмито",
```

```
"Документи подано з явною поспішністю",
"Адреса перебування виглядає неповною",
"Заявлена мета не збігається з багажем"
};
```

Відображення паспорта, картки в'їзду та службового монітора

Файл: Assets/Scripts/UI/*DocumentView.cs;
Assets/Scripts/UI/ServiceMonitorView.cs

```
public override void
Display(TravelerCase travelerCase)
{
    if (travelerCase == null ||
travelerCase.passport == null)
    {
        Clear();
        return;
    }

    string status =
travelerCase.passport.isExpired ?
"Недійсний" : "Чинний";
    string warning =
travelerCase.hasExpiredPassport
        ? "Паспорт прострочений"
        :
travelerCase.passport.isMissingDetail ?
"Бракує даних у паспорті" : "Основний
документ";

    SetCommonFields("Паспорт", status,
warning);
    SetLabel(fullNameText,
travelerCase.passport.fullName);
    SetLabel(birthDateText,
travelerCase.passport.dateOfBirth);
    SetLabel(passportNumberText,
travelerCase.passport.passportNumber);
    SetLabel(nationalityText,
travelerCase.passport.nationality);
    SetLabel(issuingAuthorityText,
travelerCase.passport.issuingAuthority + "
/ " + travelerCase.passport.issueDate);
    SetLabel(expiryDateText,
travelerCase.passport.expiryDate);
}

public override void Clear()
{
    base.Clear();
    SetLabel(fullNameText,
string.Empty);
    SetLabel(birthDateText,
string.Empty);
    SetLabel(passportNumberText,
string.Empty);
    SetLabel(nationalityText,
string.Empty);
    SetLabel(issuingAuthorityText,
string.Empty);
    SetLabel(expiryDateText,
string.Empty);
}

public override void
Display(TravelerCase travelerCase)
{
```

```
if (travelerCase == null ||
travelerCase.entryPermit == null)
{
    Clear();
    return;
}

string status =
travelerCase.entryPermit.isSuspicious ?
"Потребує перевірки" : "Подано";
string warning =
travelerCase.hasDataMismatch
    ? "Дані не збігаються"
    :
travelerCase.hasMissingDetail
    ? "Неповна картка"
    :
travelerCase.entryPermit.isSuspicious ?
"Підозріла мета" : "Картка мети в'їзду";

SetCommonFields("Картка мети
в'їзду", status, warning);
SetLabel(fullNameText,
travelerCase.entryPermit.fullName);
SetLabel(purposeText,
travelerCase.entryPermit.purposeOfVisit);
SetLabel(destinationText,
travelerCase.entryPermit.destinationCity +
", " +
travelerCase.entryPermit.stayAddress);
SetLabel(durationText,
travelerCase.entryPermit.stayDuration);
SetLabel(hostText,
travelerCase.entryPermit.hostName + " / "
+ travelerCase.entryPermit.hostPhone);
SetLabel(vehicleText,
travelerCase.entryPermit.vehiclePlate + "
/ " +
travelerCase.entryPermit.vehicleModel);
string note =
string.IsNullOrEmpty(travelerCase.sus
piciousNote) ?
travelerCase.entryPermit.note :
travelerCase.suspiciousNote;
string details =
travelerCase.entryPermit.declaredItems +
"; кошти: " +
travelerCase.entryPermit.declaredFunds;
SetLabel(noteText, note + "\n" +
details);
}

public override void Clear()
{
    base.Clear();
    SetLabel(fullNameText,
string.Empty);
    SetLabel(purposeText,
string.Empty);
```

```

        SetLabel(destinationText,
string.Empty);
        SetLabel(durationText,
string.Empty);
        SetLabel(hostText, string.Empty);
        SetLabel(vehicleText,
string.Empty);
        SetLabel(noteText, string.Empty);
    }
}

public void Display(TravelerCase
travelerCase)
{
    if (travelerCase == null)
    {
        Clear();
        return;
    }

    SetVisible(true);

    string expectedDecision =
GetAnalysisRecommendation(travelerCase);

    PassportData passport =
travelerCase.passport;
    TravelerIdentity identity =
travelerCase.identity;
    SetLabel(fullNameText, passport !=
null ? passport.fullName :
identity.fullName);
    SetLabel(birthDateText, passport
!= null ? passport.dateOfBirth :
identity.dateOfBirth);
    SetLabel(passportNumberText,
passport != null ? passport.passportNumber
: identity.passportNumber);
    SetLabel(nationalityText,
(passport != null ? passport.nationality :
identity.nationality) + " / " +
identity.genderLabel);
    SetLabel(issuingAuthorityText,
passport != null ?
passport.issuingAuthority + " / видано " +
passport.issueDate : "-");
    SetLabel(passportExpiryText,
passport != null ? (passport.isExpired ?
"прострочено: " : "дійсний до: ") +
passport.expiryDate : "-");
    SetLabel(registryText,
identity.registryId);
    SetLabel(occupationText,
identity.occupation);
    EntryPermitData entryPermit =
travelerCase.entryPermit;
    SetLabel(entryNameText,
entryPermit != null ? entryPermit.fullName
: "-");
    SetLabel(routeText, entryPermit !=
null ? entryPermit.purposeOfVisit + " -> "
+ entryPermit.destinationCity : "-");
    SetLabel(hostText, entryPermit !=
null ? entryPermit.hostName + " / " +
entryPermit.hostPhone : "-");
    SetLabel(vehicleText, entryPermit
!= null ? entryPermit.vehiclePlate + " / "
+ entryPermit.vehicleModel : "-");
    SetLabel(luggageText, entryPermit
!= null ? entryPermit.luggageDescription +
" / " + entryPermit.declaredItems : "-");
    SetLabel(fundsText, entryPermit !=
null ? entryPermit.declaredFunds + " / " +
entryPermit.previousVisits : "-");
    string behavior =
travelerCase.behaviorNote;
    if
(!string.IsNullOrEmpty(travelerCase.t
ravelerStatement))
    {
        behavior =
AppendLine(behavior, "Заява: " +
travelerCase.travelerStatement);
    }

    SetLabel(behaviorText, behavior);
    SetLabel(expectedDecisionText,
expectedDecision);
    SetLabel(findingText,
travelerCase.GetIssueSummary());
    string risk =
travelerCase.databaseStatus + " / ризик: "
+ travelerCase.riskLevel;
    if
(!string.IsNullOrEmpty(travelerCase.s
econdaryInspectionReason))
    {

```

Створення NPC, рух транспорту та керування шлягбаумом

Файл: Assets/Scripts/NPC/TravelerSpawner.cs;

Assets/Scripts/NPC/TravelerAgent.cs;

Assets/Scripts/Gameplay/CheckpointVehicleAnimator.cs;

Assets/Scripts/Gameplay/BarrierGateController.cs

```

    public IEnumerator
SpawnTravelerRoutine(TravelerCase
travelerCase, Action<TravelerAgent>
travelerSpawned)
    {
        if (!HasRequiredReferences())
        {
            travelerSpawned?.Invoke(null);
            yield break;
        }

        if (checkpointVehicle != null)

```

```

    {
        yield return
checkpointVehicle.ArriveAndPark();
    }

    TravelerAgent traveler =
CreateTravelerAtSpawn(travelerCase);
    travelerSpawned?.Invoke(traveler);
    checkpointVehicle?.CloseDoorAfterP
assengerExit();
}

public void DepartVehicle(bool
approved)
{

```

```

        checkpointVehicle?.PlayDeparture(
approved);
    }

    public void ResetVehicle()
    {
        checkpointVehicle?.ResetVehicle();
    }

    private TravelerAgent
CreateTravelerAtSpawn(TravelerCase
travelerCase)
    {
        Transform selectedSpawnPoint =
checkpointVehicle != null &&
checkpointVehicle.PassengerExitPoint !=
null
        ?
checkpointVehicle.PassengerExitPoint
        : spawnPoint;

        TravelerAgent traveler =
Instantiate(travelerPrefab,
selectedSpawnPoint.position,
selectedSpawnPoint.rotation);
        traveler.Initialize(travelerCase,
inspectionPoint, exitPoint, movementSpeed,
approachWaypoints, exitWaypoints);
        return traveler;

    public void Initialize(TravelerCase
travelerCase, Transform
targetInspectionPoint, Transform
targetExitPoint, float speed, Transform[]
approachPath, Transform[] exitPath)
    {
        AssignedCase = travelerCase;
        inspectionPoint =
targetInspectionPoint;
        exitPoint = targetExitPoint;
        approachWaypoints = approachPath;
        exitWaypoints = exitPath;
        moveSpeed = speed;
        currentState =
TravelerState.MovingToDesk;
        SetActivePath(BuildPath(approachWa
ypoints, inspectionPoint));
        SnapFacingTarget(currentTarget);

        if (carriedDocuments != null)
        {
            carriedDocuments.transform.loc
alPosition =
carriedDocumentsHandLocalPosition;
            carriedDocuments.transform.loc
alRotation = Quaternion.Euler(8f, -12f,
0f);
            carriedDocuments.SetActive(tru
e);
        }

        SetWalking(true);
    }

    private void Update()
    {
        if (currentTarget == null ||
currentState ==

```

```

TravelerState.WaitingForDecision ||
currentState == TravelerState.Finished)
    {
        return;
    }

    Vector3 targetPosition =
currentTarget.position;
    Vector3 moveDirection =
targetPosition - transform.position;
    moveDirection.y = 0f;

    if (moveDirection.sqrMagnitude >
0.0001f)
    {
        Quaternion targetRotation =
Quaternion.LookRotation(moveDirection.norm
alized, Vector3.up);
        transform.rotation =
Quaternion.Slerp(transform.rotation,
targetRotation, rotationSpeed *
Time.deltaTime);
    }

    transform.position =
Vector3.MoveTowards(transform.position,
targetPosition, moveSpeed *
Time.deltaTime);

    float remainingDistance =
Vector3.Distance(
        new
Vector3(transform.position.x, 0f,
transform.position.z),
        new Vector3(targetPosition.x,
0f, targetPosition.z));

    if (remainingDistance <=
arrivalDistance)
    {
        ReachCurrentTarget();
    }

    public void LeaveCheckpoint()
    {
        currentState =
TravelerState.Leaving;
        SetActivePath(BuildPath(exitWaypoi
nts, exitPoint));
        SnapFacingTarget(currentTarget);
        SetWalking(true);
    }

    private void ReachCurrentTarget()
    {
        if (currentState ==
TravelerState.MovingToDesk)
        {
            if (TryAdvancePath())
            {
                return;
            }

            currentState =
TravelerState.WaitingForDecision;
            SetWalking(false);
            StartCoroutine(PresentDocument
sThenWait());

```

```

        return;

    public IEnumerator ArriveAndPark()
    {
        ResetVehicle();
        gameObject.SetActive(true);
        AudioManager.Instance?.PlayVehicleEngine();
        yield return
        MoveTo(approachPosition, moveSpeed);
        yield return MoveTo(parkPosition,
        parkingSpeed);
        currentSpeed = 0f;
        IsParked = true;
        yield return SetDoorOpen(true);
        yield return new
        WaitForSeconds(passengerExitDelay);
    }

    public void
    CloseDoorAfterPassengerExit()
    {
        StopDoorRoutine();
        doorRoutine =
        StartCoroutine(SetDoorOpen(false));
    }

    public void PlayPassThrough()
    {
        PlayDeparture(true);
    }

    public void PlayDeparture(bool
    approved)
    {
        StopMovementRoutine();
        StopDoorRoutine();
        movementRoutine =
        StartCoroutine(DepartureRoutine(approved))
        ;
    }

    private IEnumerator
    DepartureRoutine(bool approved)
    {
        gameObject.SetActive(true);
        yield return SetDoorOpen(false);

        if (approved)
        {
            AudioManager.Instance?.PlayVehicleEngine();
            yield return
            MoveTo(laneMergePosition, parkingSpeed);
            yield return
            MoveTo(approvedExitPosition, moveSpeed);

```

```

        }
        else
        {
            AudioManager.Instance?.PlayVehicleEngine();
            yield return
            MoveTo(approachPosition, parkingSpeed);
            yield return
            MoveTo(rejectedExitPosition, moveSpeed);
        }

        currentSpeed = 0f;
        IsParked = false;
        yield return new
        WaitForSeconds(resetDelay);
        ResetVehicle();

        public void SetGateOpen(bool isOpen)
        {
            EnsureRotations();

            if (this.isOpen == isOpen)
            {
                RefreshSignals(isOpen);
                return;
            }

            this.isOpen = isOpen;
            targetRotation = isOpen ?
            openRotation : closedRotation;
            autoCloseTime = -1f;
            RefreshSignals(isOpen);

            if (isOpen)
            {
                AudioManager.Instance?.PlayGateOpen();
            }
            else
            {
                AudioManager.Instance?.PlayGateClose();
            }
        }

        public void OpenForDuration(float
        duration)
        {
            SetGateOpen(true);
            autoCloseTime = Time.time +
            Mathf.Max(0.1f, duration);
        }

```