КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА СТРАТЕГІЧНОЇ ВІДЕОГРИ

Виконав студент 4-го курсу
Сергій ЯРЕМА _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук _____
Ярослав ЛІНДЕР (підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«___»_____2021р.,
протокол №___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 44 сторінки, 8 використаних джерел, 7 рисунків.

Ключові слова: СТРАТЕГІЧНІ ВІДЕОІГРИ, ІГРОВІ ПРОГРАМИ, МОДУЛЬНІ СИСТЕМИ, АРХІТЕКТУРА ІГРОВИХ ПРОГРАМ, ДІАЛОГОВІ СИСТЕМИ.

Об’єктом роботи є стратегічна відеогра. Предметом роботи є побудова гнучкої архітектури ігрового застосунку з можливістю додавання нових візуальних елементів та механік ігрового процесу на базі обраних підходів розробки.

Метою роботи є опис загальних підходів до створення відеоігор, обрання оптимальних методів розробки та реалізація ігрових механік на прикладі стратегічної відеогри.

Інструменти розробки: мова програмування C#, ігровий рушій Unity, графічні редактори Adobe Photoshop й Adobe Illustrator, пакет для створення тривимірної комп'ютерної графіки Blender, інтегроване середовище розробки програмного забезпечення Microsoft Visual Studio.

Результатом роботи є стратегічна відеогра, аналіз та реалізація оптимальних підходів до розробки запланованої відеогри. Даний ігровий проект є новим з точки зору способу поєднання ігрової механіки менеджменту ресурсів та жанру текстової новели зі стратегічними елементами.

Відеогра була опублікована на Інтернет-платформі для розповсюдження відеоігор itch.io [1]. На даній платформі проведено тестування зацікавленості гравців у відеогрі. Також проведено очне тестування проекту спеціалістами у галузі ігрового дизайну.

Схожими за своєю суттю та наративом є відеоігри Reigns та Lapse. Основними відмінностями є зміна наративу гри та використання тривимірної графіки для збільшення варіативності візуального представлення у грі.

Описаний аналіз підходів до розробки відеоігор може бути використаним при створенні нових проектів і слугувати набором методів для побудови гнучкої архітектури ігрового застосунку. Розроблений редактор відеогри дозволяє створювати власні сюжети і відповідно доповнення або цілком нові версії відеоігор у жанрах, що потребують поетапного сюжетного викладення.

Створена програмна база відеогри дозволяє доповнювати гру сюжетним та графічним змістом. Для розвитку проекту можуть бути залучені графічні дизайнери та сценаристи з метою наповнення ігрового процесу новими подіями.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Зі стрімким розвитком технологій все більше людей долучаються до цифрових розваг, одним із найпопулярніших різновидів цифрових розваг є відеоігри. Ігрова індустрія щороку показує тренд на збільшення обороту та притоку інвестицій в нові ігрові проекти.

Актуальність роботи та підстави для її виконання. Розробка відеоігор є комплексною задачею з безліччю підходів до реалізації запланованих ігрових механік, при цьому мало уваги приділяється опису побудови гнучкої архітектури ігрового застосунку на реальному прикладі. В цій роботі поетапно описується мотивація для використання різних практик створення відеоігор, що можна застосовувати й для інших проектів.

Для створення відеоігор, що будуть цікавими користувачам часто використовують підхід поєднання вже відомо працюючих ігрових елементів та механік. Стратегічні відеоігри у класичному вигляді як жанр останні десять років перебували у стагнації, проте паралельно активно виникали проекти, що містять у собі поєднання елементів ігор-стратегій з іншими ігровими механіками. Розроблена в рамках цієї роботи відеогра є поєднанням стратегії управління ресурсами та візуальної новели.

Мета й завдання роботи. Метою цієї роботи є розробка відеогри з гнучкою архітектурою та поетапний опис мотивації вибору підходів до реалізації ігрового проекту.

Завданнями роботи є аналіз загальновживаних практик розробки відеоігор та обґрунтування вибору конкретних підходів для проектування й реалізації конкретної відеогри.

Об’єкт, методи і засоби дослідження або розроблення. Об’єктом розробки є стратегічна відеогра у поєднанні жанрів менеджеру ресурсів та

текстової новели. Об’єктом дослідження є процес побудови архітектури ігрового застосунку, що буде гнучким до доповнень та реалізовуватиме заплановані ігрові механіки.

Для реалізації відеогри був використаний ігровий рушій Unity 2019.3.15f1 з мовою програмування C# в якості мови скриптингу, для роботи з програмним кодом та відладки використовувалася інтегрована середа розробки Microsoft Visual Studio 2019. При створенні візуальних елементів гри були використані графічні редактори Adobe Photoshop CC 2021 та Adobe Illustrator 2020, тривимірні об’єкти та їх анімації створювались з допомогою пакету для створення тривимірної комп'ютерної графіки Blender 2.91.2.

Можливі сфери застосування. Результати дослідження методів розробки відеоігор можна застосовувати при створенні нових ігрових проектів.

РОЗДІЛ 1. ВІДЕОІГРИ

1.1 Основні поняття

Відеогра – це програма, що запускається на електронному пристрої з метою створення інтерактивної системи в яку може грати користувач використовуючи певний різновид контролеру, такі як, джойстик, клавіатура чи сенсорний екран.

Жодне означення гри не може в повній мірі охарактеризувати всі аспекти даного поняття, проте розкладаючи відеоігри на атомарні елементи можна наблизитися до достатнього опису конкретної гри для її реалізації. До зіставних елементів відеоігор відносять поняття:

- гравців, їх кількості, методів взаємодії з грою та з іншими гравцями;
- ігрових цілей, що є причиною гравця до продовження ігрової сесії;
- ігрових механік та правил, що визначають засоби просування всередині гри, та описують ситуації, котрі підтримуються грою;
- ігрових ресурсів та механізмів управління ними;
- ігрового стану;
- рівня даних гри, та представлення ігрової інформації користувачеві;
- ігрових систем;
- загальних характеристик теми, жанру, сюжету, наративу гри.

Модифікації в хоча б одному атомарному елементі відеогри здатні повністю змінити враження від гри чи спосіб її роботи. Формалізація ігрового процесу допомагає виділяти на перший погляд неочевидні зв’язки між різними іграми і разом з тим спрощує розробку нових відеоігор, оскілки дозволяє повторно використовувати вже працюючі ігрові елементи чи підходи до їх розробки. Важко прогнозувати, як саме будуть працювати комбінації різних ігрових елементів, тому розробка відеоігор завжди несе у собі долю експерименту .

Ігрова механіка є набором правил і методів, що реалізують певним чином деяку частину інтерактивної взаємодії користувача та гри. Сукупність ігрових механік формують певну реалізацію деякого ігрового процесу. Для опису відеогри використовують поняття основних ігрових механік, що являють собою дії, котрі будуть доступні гравцю протягом усього ігрового процесу чи більшої його частини, а також другорядних механік, котрі стають доступними на деякий час чи при здобутті певних ігрових цілей. Завдяки такому підходу ігри здатні залишатися цікавими впродовж більшого проміжку часу, додаючи нових вражень гравцю.

Чітке представлення ігрового процесу в термінах ігрових механік дозволяє вирішувати питання комунікації в команді й формалізувати ігровий процес, що спрощує тестування продукту, завдяки прописаній очікуваній поведінці гри на певні дії користувача.

Одним із підходів до формалізації ігрового процесу є опис в термінах об’єктів, агентів та їх методів. Даний метод використовує об’єктно орієнтований підхід для розуміння які дії й поведінка мають бути доступними для певних класів. В даній концепції ігрові механіки є методами, що викликаються агентами з метою взаємодії з ігровим станом, більш детальний опис наведений у [2]. Оптимальним шляхом до розуміння механік як методів є їх формалізація з використанням дієслів й інших синтаксичних структур, таких як правила, що уточнюють вплив дієслів на дії, котрі відбуваються в грі. Наприклад, у грі Shadow of the Colossus можна знайти такі механіки як: лізти, їздити верхи, вдаряти, стрибати, стріляти з використанням лука і стріл, свистіти, хватати, бігати, плавати, ниряти. Всі ці методи агентів в ігровому світі формують простір можливостей заданий правилами.

Відеоігри, як і музичні чи літературні твори поділяють за жанрами, котрі визначають спільні риси чи основні ігрові механіки, присутні всередині жанру.

Також відеоігри класифікують за платформами на яких вони можуть запускатися. Багато відеоігор розробляються таким чином, щоб мати можливість запускати їх на декількох платформах без суттєвої зміни коду самої гри, такий підхід дозволяє покривати більшу аудиторію.

1.2 Стратегічні відеоігри та їх особливості

Стратегічні відеоігри виділяються як жанр в основному тим, що в них гравцю для перемогу необхідно використовувати стратегічне мислення, та планування можливих результатів дій у грі.

Відеоігри цього жанру зосереджені на управлінні ресурсами, що можуть являти собою бойові одиниці, різноманітні ігрові показники та характеристики, території, тощо. Для кожного типу ресурсів гравцю доступні прямі та не прямі способи їх управління. Під час не прямого управління користувач здійснюючи дії в грі може спричиняти зміну певних параметрів ресурсів у стані гри.

Основні цілі у стратегічних відеоіграх можуть бути різними, проте в загальному випадку вони зводяться до максимізації деякого абстрактного виду ресурсів, наприклад у грі «хрестики-нулики» ресурсом, що призводить до виграшу можна вважати заповнений символом гравця набір з трьох клітинок, що утворюють лінію по вертикалі, горизонталі чи діагоналі.

Опис ігрового процесу стратегічних відеоігор зводиться до розроблення набору ігрових механік та правил, за якими гравець може маніпулювати ігровими ресурсами.

РОЗДІЛ 2. ІДЕЯ ТА ОПИС ОБРАНОГО ІГРОВОГО ПРОЕКТУ

2.1 Ідея відеогри

В рамках цієї роботи має бути розроблена покрокова стратегічна відеогра в якій гравцю на кожному кроці буде надаватися можливість вибору відповідей з двох опцій, котрі є варіантами вирішення поточної ігрової ситуації. Такими виборами буде відбуватися просуватися по сюжету гри. Для кожного варіанту відповідей чи їх сукупності передбачений механізм непрямої зміни ігрових ресурсів. Для збільшення можливостей контролю над показниками ресурсів гравцю буде надана система управління спеціалізованими елементами, що витрачають одні ресурси для генерації інших на кожному кроці сюжетного вибору.

Для даної відеогри було обрано сюжет в якому гравець виступає в ролі власника науково-фантастичної підводної бази, що була побудована через підвищення рівня світового океану. В ході гри користувачу доведеться вирішувати різноманітні ситуації, що стосуються бази та її навколишніх територій, спілкуючись з різними неігровими персонажами.

2.2 Основні ігрові механіки

В даній відеогрі управління ресурсами є здебільшого непрямим. Виділимо набір ресурсів, з якими буде відбуватися взаємодія: ігрова валюта, показник щастя населення, показник порядку й охорони на станції, кількість жителів.

Ціллю гри є утримання значень ігрових ресурсів більшими за нуль й просування по сюжету.

Для розвитку сюжету використовується механіка покрокового показу подій. Підчас показу події, гравець має змогу оцінити ситуацію, та переглянути

можливі варіанти розв’язання ситуації, що склалася. Також гравець має змогу переглядати на які ресурси буде здійснений моментальний вплив при виборі того чи іншого варіанту відповіді. Таким чином було виділено наступний набір основних механік відеогри:

- перегляд поточних значень ресурсів;
- покроковий перегляд сюжетних подій;
- перегляд варіантів відповідей на сюжетні події;
- перегляд зміни параметрів перед вибором відповіді;
- обрання варіантів відповідей;

Зв’язки описаних ігрові механік, зображені на рисунку 2.1, де видно, що перегляд сюжетних подій включає в себе перегляд варіантів відповідей й може бути доповнений вибором варіанту відповіді.

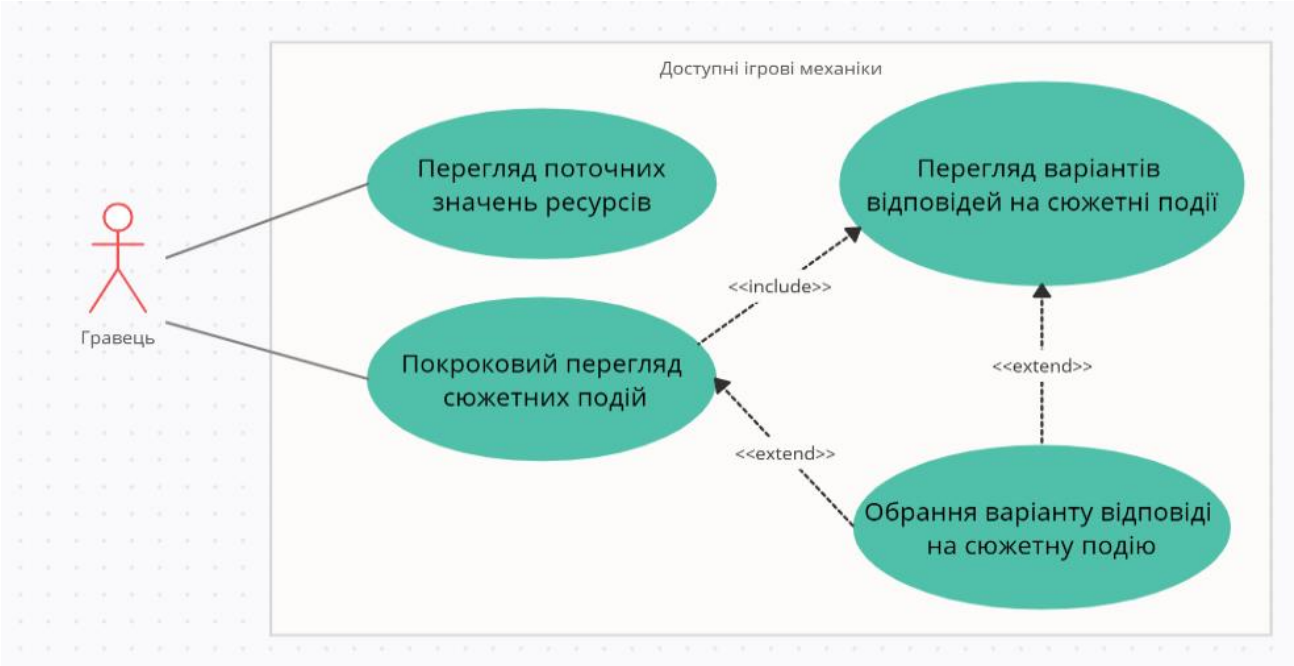

Рисунок 2.1 - Схематичне зображення ігрових механік на діаграмі прецедентів

2.3 Опис запланованих візуальних елементів та взаємодії гравця з грою

В якості візуального супроводу події будуть подаватися у вигляді тривимірних карток з зображеннями неігрових персонажів, з якими відбувається розмова. Текст опису ситуації та тексти відповідей відображаються в спеціальних полях на ігровому екрані. Параметри ігрових ресурсів відображаються як іконки умовного позначення виду ресурсу та числове значення ресурсу. Для відображення характеру змін ресурсів при виборі певного варіанту відповіді використовуються символи-стрілки з різним направленням, розміром і кольором.

Для взаємодії з сюжетом використовується перетягування карток з допомогою миші чи сенсорного екрану на сторону обраного варіанту відповіді.

На рисунку 2.2 зображене схематичне представлення запланованого інтерфейсу відеогри.

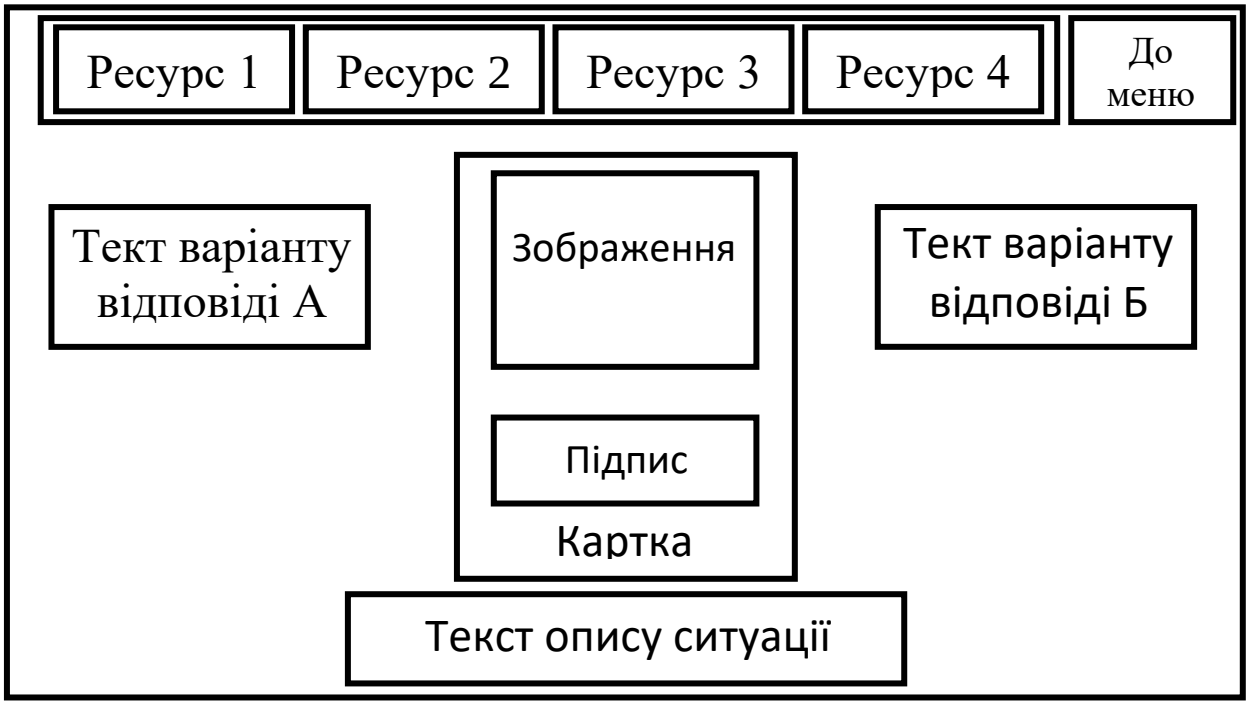

Рисунок 2.2 – схематичне представлення основного ігрового інтерфейсу

В верхній частині ігрового екрану будуть відображатися поточні значення ігрових ресурсів підводної бази, а також кнопка виклику меню гри, через яке можна перейти до головного меню гри чи змінити деякі налаштування.

В центрі екрану відображатиметься сюжетна картка з певним підписом та зображенням, що відноситься до сюжетного елементу чи деякого неігрового персонажу, з яким ведеться діалог.

По ліву і праву сторони від картки знаходяться тексти варіантів відповідей на поточну картку сюжету. Нижче від картки знаходиться текст опису сюжетної картки.

РОЗДІЛ 3. РОЗРОБКА ВІДЕОІГОР

3.1 Обрання інструментів розробки

Вибір інструменту розробки залежить від складності та масштабності проекту. Також важливим фактором є те, на яких платформах планується запускати гру. При створенні сучасних відеоігор використовують ігрові рушії, вони можуть бути створені самими розробниками відеогри, такими, що постачаються безкоштовно чи ліцензованими.

Використання ігрового рушія власного випуску вимагає багато часу на розробку самого рушія без гарантій роботи створених з його допомогою відеоігор на різних пристроях навіть в рамках однієї операційної системи, перевагою такого підходу є те, що розробник має змогу створювати і використовувати лише ті компоненти, котрі необхідні для конкретної відеогри.

Популярні безкоштовні ігрові рушії загального призначення такі як Unity та Unreal Engine містять у собі готові набори компонентів, що слугують каркасом для створення відеоігор. Unreal Engine є ігровим рушієм з відкритим програмним кодом, що дозволяє модифікувати його так як це потрібно розробнику, відключаючи чи взагалі видаляючи модулі, що не використовуються для конкретного проекту та дописуючи власні. Таким чином подібні готові ігрові рушії фактично прирівнюються до самописних в контексті здатності до модифікацій та налаштувань під конкретні проекти.

Ліцензовані ігрові рушії зазвичай використовуються великими командами і обираються під конкретні проектні вимоги. Зазвичай ліцензовані ігрові рушії містять у собі більший функціонал в порівнянні з безкоштовними, а також надають підтримку в розробці при виникненні проблем.

Якщо для реалізації проекту обираються безкоштовні ігрові рушії тоді найчастіше розробники обирають між двома лідерами ринку Unity та Unreal

Engine. Вибір між цими двома ігровими рушіями зазвичай базується на вимогах до конкретних ігор. Unity використовує мову програмування та віртуальну машину C#, що дозволяє запускати створені ігри на всіх платформах, що підтримують віртуальну машину C#, проте таке рішення уповільнює роботу створених відеоігор, тому проекти, що мають високі вимоги до швидкодії зазвичай розробляють з використанням Unreal Engine.

3.2 Ігровий цикл

Незалежно від обраних інструментів розробки та ігрових рушіїв, в основі будь-яких відеоігор знаходиться поняття ігрового циклу в різних варіаціях, в тілі якого виконується логіка гри, опрацювання вхідних сигналів від контролерів користувача, вивід зображень, аудіо, тощо.

Частота виконання ітерацій ігрового циклу найчастіше визначає частоту оновлення візуального відображення гри, щоб створити ілюзію плавного руху ігрових елементів цільовими значеннями кількості кадрів за секунду обирають 30, 60 та 90, оскільки саме такі значення є найбільш вживаними частотами оновлення екранів пристроїв гравців. В свою чергу цільова частота оновлення накладає вимоги до часу виконання кожної ітерації ігрового циклу з усіма його етапами, таким чином для частоти оновлення у 30 кадрів за секунду накладається ліміт на виконання однієї ітерації в 33 мс, для 60 кадрів за секунду – 16.6 мс. Такі показники суттєво обмежують візуальні та розрахункові можливості для відеоігор, оскільки ігрові продукти мають запускатися на широкому спектрі пристроїв. Питання оптимізації ігрового циклу є нагальним як для потужних платформ у вигляді настільних комп’ютерів, так і мобільних пристроїв.

Зазвичай використовується чітко визначена схема з порядком дій, що виконуються для забезпечення роботи ігрової програми.

Розглянемо спрощені етапи, що виконуються в тілі ігрового циклу на прикладі ігрового рушія Unity:

- перевірка вводу – виконується запит, що перевіряє чи був отриманий сигнал введення з ігрових контролерів користувача, якщо був отриманий то він додається до спеціалізованого буферу;
- виклик методів ініціалізації для новостворених ігрових об’єктів;
- обробка сигналів вводу збережених у буфері;
- виклик методів оновлення активних ігрових об’єктів;
- оновлення графічного відображення сцени;
- виклик методів знищення чи переходу в неактивний стан ігрових об’єктів.

3.3 Механізми управління ігровими сутностями

Для ігрових сутностей передбачені методи, що викликаються на різних етапах ігрового циклу, оскільки вони виконуються при кожній ітерації циклу, можна вважати, що кожен ігровий об’єкт має власний ігровий цикл, за який відповідають методи оновлення чи їх аналоги.

Кожна ігрова сутність представляє свій стан та методи його зміни. Для задання механізмів управління станом найчастіше використовують підходи послідовного виконання, скінченні автомати, системи подій.

Дані методи можуть бути скомбіновані з метою усунення власних недоліків. Наприклад часто використовується підхід послідовного виконання, всередині якого для реалізації кожної з механік використаний виділений скінченний автомат, а для комунікації з іншими ігровими сутностями реалізована система подій через яку відбувається відправка сигналів до об’єктів, що підписалися на певні типи подій.

3.3.1 Послідовне виконання

В тілі ігрового циклу певної сутності для реалізації ігрових механік до яких відноситься певна сутність має виконуватися набір дій, що змінюватимуть стан гри. Базовим підходом до управління зміною стану є послідовне виконання програмних операцій. Основним недоліком такого підходу є неможливість гарантії контролю поточного стану сутності.

Під час виклику методів інших сутностей із процесу послідовного виконання важко відстежувати мотивацію та джерело виклику, що часто призводить до надмірного використання шаблону програмування Singleton лише з метою синхронізації станів декількох об’єктів.

Розглянемо принцип послідового виконання ігрового циклу для ігрової сутності на прикладі гри «хрестики-нулики» для двох гравців на одному пристрої та сутності клітинка ігрового поля.

При кожній ітерації ігрового циклу, клітинка перевіряє чи не було здійснено натискання на неї, якщо було здійснено натискання відбувається перевірка внутрішнього стану клітинки, тобто чи був вже встановлений деякий символ у клітинці. Далі клітинка звертається деяким чином до ігрової сутності загального ігрового процесу, де дізнається символ поточного гравця, встановлює його в собі для відображення і викликає певний метод для повідомлення про встановлення символу в клітинку з заданими координатами.

При виклику методу встановлення символу в клітинку, сутність ігрового процесу записує дані про координати в пам’ять та виконує перевірку на виграш, у разі виграшу чи нічиєї переходить до процесу демонстрації результату, в іншому випадку змінює символ поточного гравця.

3.3.2 Скінченні автомати

Скінченний автомат це різновид автомату, що використовується для опису зміни стану об’єкта в залежності від поточного стану та інформації отриманої ззовні. Моделювання поведінки з використанням скінченних автоматів є одним із найстаріших та найбільш вивчених, згідно з [3]. Для використання скінченних автоматів, для ігрових сутностей виділяється скінчена множина станів, якою може бути описана їх поведінка. Далі будується модель скінченного автомату, що відображає переходи між станами ігрового об’єкту в залежності від зовнішніх змін, наприклад вводу користувача, зміни певного загального аспекту, поточних значень ресурсів, тощо. Конкретні приклади використання скінченних автоматів наведені в [4].

Недоліком такого підходу є необхідність створення нових станів і переходів із них та у них при додаванні нової ігрової логіки, тому даний підхід часто використовують для опису невеликих за об’ємом функціоналу ігрових сутностей. Частим прикладом використання скінченних автоматів для розробки відеоігор є задання станів та переходів між різними анімаціями, для подібного роду задач створюються редактори, які не потребують зміни коду для додавання і поєднання нових станів.

На прикладі гри «хрестики-нулики» та сутності клітинка ігрового поля, в неї було б задано наступний можливий набір станів:

- порожня клітинка
- заповнена клітинка

Зобразімо схематично (рисунок 3.1) переходи автомату клітинки ігрового поля. Овалами позначені стани з їх назвою всередині, стрілки відображають переходи. Умова переходу відображена на початку стрілки, а на кінці – дія, що відбувається при переході.

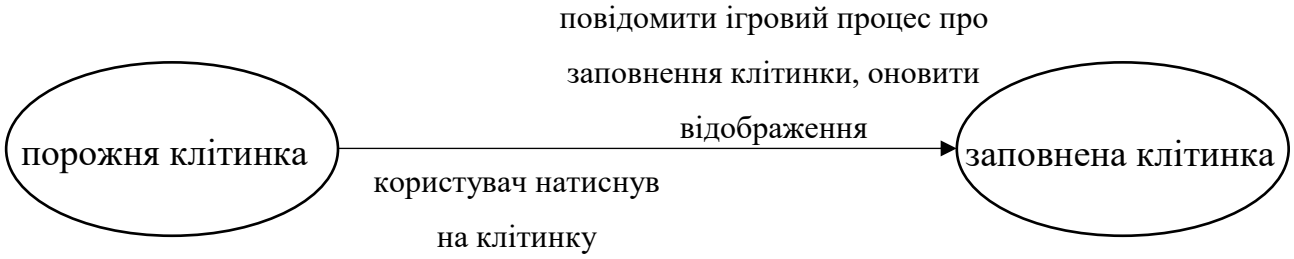

Рисунок 3.1 – схема переходу станів клітинки ігрового поля гри «хрестики-нулики»

Для сутності ігрового процесу можна виділити наступні стани:

- хід гравця-нуликів
- хід гравця-хрестиків
- перемога (візуальне відображення перемоги визначається символом останнього гравця, що ходив до переходу в стан перемоги)
- нічия

Переходи між цими станами характеризують основну ігрову логіку. На рисунку 3.2 схематично зображено автомат ігрового стану, де на стрілках переходів між станами через кому описані ігрові події що призводять до переходу та дія, що відбувається в додаток до здійснення переходу. Використання подібних надбудов над базовою версією скінченного автомату зумовлено необхідністю зміни інших сутностей із середини об’єкту, що керується скінченним автоматом. У даному прикладі, сутність ігрового процесу має мати змогу управляти клітинками та деякими елементами інтерфейсу користувача.

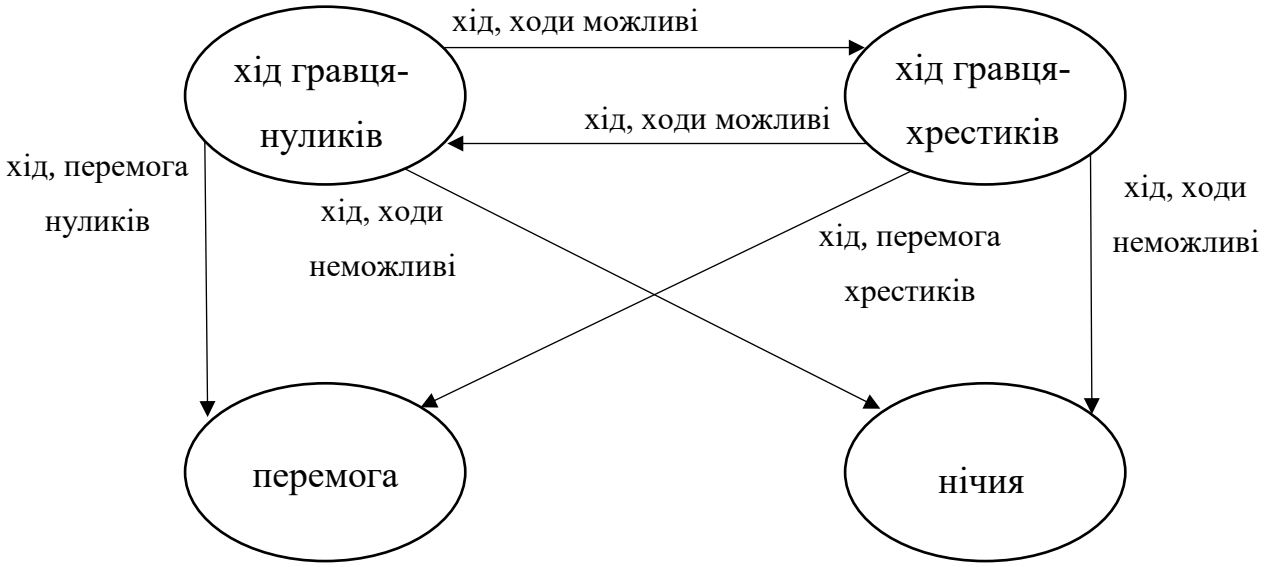

Рисунок 3.2 – схема переходу станів сутності ігрового процесу гри «хрестики-нулики»

Бачимо, що при такому підході окрім реалізації ігрової логіки, розробник може набагато зручніше створювати сутності візуального відображення оскільки для визначення поточного стану, котрий необхідно відобразити – достатньо переглянути значення стану відповідного скінченного автомату. Дані поточного стану є корисними для отримання інформації про те, які процеси необхідно запускати іншим ігровим сутностям, таким як анімації персонажів, характер поведінки неігрових ботів, зміна числових характеристик швидкості, величини заробітку ігрової валюти, тощо.

3.3.3 Системи подій

Подіями називають дії, що розпізнаються програмним забезпеченням та оброблюється використовуючи набір певних інструкцій. Такі події можуть бути викликані операційною системою, користувачем чи згенеровані викликом із

програми, що виконується. В об’єктно-орієнтовному програмуванні події найчастіше реалізуються на основі моделі делегатів [3].

Делегат – це спеціальний механізм, що представляє в собі посилання на певні методи з конкретним списком вхідних параметрів. В контексті розробки відеоігор на певний делегат можуть підписуватися ігрові сутності з метою виклику необхідної поведінки при виклику делегату.

Цей підхід реалізує шаблон проектування публікації-підписки, в якій делегати виступають в ролі публікацій, а методи, що зв’язані з делегатами – підписками.

Завдяки використанню механізму системи подій розроблена ігрова логіка стає масштабованою та динамічною до доповнення новим функціоналом, без необхідності змін чи додавання викликів в сутностях, котрі містять делегати. Частим доповненням до системи подій є використання фільтрації подій, що дозволяє підписникам приймати лише підмножину від усіх опублікованих повідомлень. Фільтрація повідомлень може здійснюватися підставі тем повідомлень, та змісті повідомлень. Під темою повідомлень маються на увазі іменовані канали, в яких помітка назви каналу слугує темою повідомлення, підписники будуть отримувати лише ті повідомлення, на теми яких вони були підписані, відповідальним за помітки на які може здійснюватися підписка є видавець. Системи, що використовують фільтрацію за змістом повідомлення покладають відповідальність за аналіз змісту повідомлень на підписників для відкидання чи реагування на отримане повідомлення [4].

Розглянемо роботу підходу системи подій на прикладі гри «хрестики-нулики». Основними ігровими сутностями є клітинки ігрової дошки, що містять набори делегатів, та сутність загального ігрового процесу, котра зберігає посилання на всі клітинки ігрового поля. Під час запуску гри загальна ігрова сутність підписується на події, котрі будуть викликатися клітинками ігрового

поля через механізм делегатів. Керування викликом методів зміни стану клітинок покладається на загальну ігрову сутність.

Клітинки ігрового поля містять делегати, котрі у змісті повідомлень зберігають посилання на клітинку котра надсилає повідомлення, деякий аналог ідентифікатора, наприклад, координати на дошці. Ключовим делегатом клітинок буде повідомлення про на обрання клітинки користувачем, що розповсюджується за умови відсутності символу в клітинці. В свою чергу сутність ігрового процесу реагує на повідомлення про обрання користувачем клітинки, виконує перевірку стану виграшу на ігровому полі й встановлює в ідентифіковану за повідомленням клітинку символ поточного гравця.

При подібній реалізації зв’язку між об’єктами, зникає необхідність модифікації програмного коду делегатів у випадку зміни програмних інтерфейсів у сутностей, що на них підписуються, а тому кожна сутність делегат може виступати у ролі ізольованого модуля, котрий можна тестувати у виділеному середовищі й у разі необхідності повторення ігрової поведінки для інших ігрових елементів безпечно переносити. Наприклад, створена сутність клітинок ігрового поля може бути повторно використана для ігрового поля симуляції логічної настільної гри го.

Використання системи подій дозволяє створювати каркас гри, над котрим якого під час розробки будуть додаватися нові ігрові елементи та другорядні механіки, що є модифікаціями основних з метою збільшення варіативності ігрового процесу [5].

Гарним прикладом використання системи подій э створення універсального контролеру вводу користувача, що містить делегати для кожного типу сигналів, що може бути передана користувачем грі [6].

На відміну від послідовного виконання та скінченних автоматів, система подій дозволяє усунути перевірку ігрових параметрів на кожній ітерації ігрового циклу й викликати делеговані методи тільки за необхідності, це покращує

загальний час виконання ітерації ігрового циклу, при цьому спрощує розуміння ігрової логіки при роботі з програмним кодом гри за відсутності надмірного використання конструкцій умовних переходів.

3.4 Ігровий стан

Ігровий стан це сукупність всієї інформації про поточний стан ігрових елементів та зв’язку між ними. Також поняття ігрового стану відносять й до окремих ігрових об’єктів, тоді воно приймає характер опису об’єкту в конкретний момент часу гри [7]. Стани ігрових об'єктів визначають їх поведінку під час гри, їх відображення та реакцію на виклики ззовні, такі як ввід користувача.

Задачею розробника відеоігор є реалізація системи, що здатна контролювати та у разі потреби відновлювати ігровий стан з заданою точністю, наприклад при повторному запуску гри.

Для чіткого представлення стану ігрових об’єктів використовують так званий рівень даних, що являє собою набір скалярних та структуровані змінних, котрі описують поточний стан ігрової сутності. Окрім поточного стану рівень даних також представляє характеристики ігрових сутностей, що зберігають своє значення між усіма чи частиною ігрових сесій, це можуть бути набори текстур, анімацій, строки текстових полів, ігрові властивості швидкості, розміру об’єкту, сили атаки, тощо. Коли стани ігрового об’єкту можуть бути чітко описані набором слів-характеристик, часто використовують змінні типу перечислень, наприклад для тривимірного анімованого представлення персонажу можуть використати набір із станів: біг, ходьба, стояння на місці, ходьба навприсядки. В цьому прикладі можна помітити, що часто для кращого зовнішнього відображення ігрового процесу необхідно мати можливість переходити між станами не моментально, а за допомогою певного роду інтерполяцій [8]. В

комплексних іграх, ігрові сутності можуть комбінувати у собі взаємодію декількох ігрових механік, в цьому випадку використовують декілька змінних типу перечислень, що характеризуватимуть стани для конкретних механік.

При розробці набору ігрових механік, ігровий дизайнер може формалізувати весь ігровий стан, що спрощує роботу розробників, оскільки це дозволяє розуміти які саме дані необхідні для представлення гри. Прикладом ігор в яких чітко видно ігровий стан є шахи, де сукупність позицій фігур, їх типів та кольорів разом з ідентифікатором гравця, що зараз ходить повністю характеризують поточний стан гри.

3.5 Об’єктно орієнтований підхід в контексті розробки відеоігор

Для представлення елементів відеогри зручно використовувати поняття ігрових сутностей та ігрових об’єктів. Такі поняття можуть бути чітко пов’язані з програмною реалізацією відеогри при використанні об’єктно орієнтовного підходу програмування. Ігрові сутності представляються класами, а ігрові об’єкти екземплярами класів. Класи містять у собі необхідні дані для представлення власного стану й набори методів для управління станом та взаємодії з іншими класами та об’єктами.

3.5.1 Методи взаємодії між об’єктами

В контексті об’єктно орієнтовного підходу, об’єкти обмінюються спеціальними повідомленнями, що зазвичай являють собою виклики відповідних методів. Такі виклики можуть бути прямими, тобто викликатися для певного об’єкту за посиланням на нього, та непрямими через механізми-посередники, наприклад, з використанням системи подій та делегатів. При збільшенні кількості типів ігрових сутностей підвищується складність встановлення механізмів

взаємодії між об’єктами. Виділимо два основні типи взаємодії ігрових об’єктів: прямий та ієрархічний.

Прямий метод взаємодії дозволяє встановлювати зв’язки об’єктів один з одним напряму, зберігаючи посилання на об’єкти з якими передбачена взаємодія за ігровою логікою, чи передаючи посилання на об’єкт, що викликає метод, як параметр викликаного методу. Цей метод є оптимальним рішенням для невеликих проектів чи прототипів відеоігор, оскільки дозволяє реалізувати заплановану ігрову логіку без затрат часу на створення додаткових методів контролю за взаємодією об’єктів. Недоліком такого підходу є те, що при необхідності додати в гру нові елементи логіки, будуть вноситися зміни в логіку одразу декількох об’єктів, з якими нові елементи можуть бути навіть не пов’язані логікою гри.

Для вирішення проблем гнучкості системи створюють додаткові сутності менеджерів, що відіграють роль комутаторів між об’єктами й додають рівень контролю за станом гри, оскільки мають доступ одразу до низки ігрових об’єктів та відповідно їх станів. Механізм взаємодії з сутностями-комутаторами передбачає використання прямих викликів методів чи використання системи подій.

При використанні системи подій як способу комутації об’єктів через посередників можна накладати додаткові умови на те в якому напрямку будуть утворюватися зв’язки між об’єктами, а саме, якщо представити зв’язки між об’єктами у вигляді графу, де об’єкти є вершинами, а зв’язки - ребрами, основною умовою є майже повна відсутність циклів й одно направленість зв’язків від обраних об’єктів-комутаторів. Умова на повну відсутність не є строгою, оскільки можливі випадки коли раціонально дозволити обмінюватися повідомленнями двом об’єктам напряму. Такий метод взаємодії між ігровими об’єктами називають ієрархічним.

Об’єкти, що знаходяться вище по ієрархії зберігають посилання на підлеглі об’єкти й підписуються на делегати підлеглих об’єктів, щоб забезпечити події, що мають відбутися за умови появи в системі певного повідомлення (рис. 3.5 а). Для однієї ієрархічної системи можуть використовувати набір з декількох об’єктів-комутаторів (рис. 3.5 б).

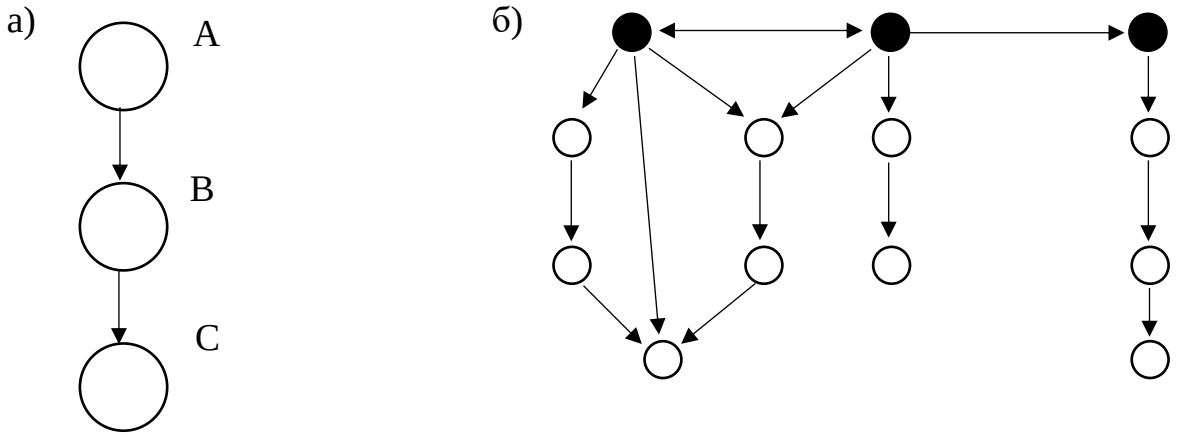

Рисунок 3.3 – Ієрархічні зв’язки ігрових об’єктів: а –ланка зв’язків з трьох об’єктів; б – зв’язки з більшою кількістю об’єктів

Ієрархічний метод взаємодії дозволяє чітко встановлювати порядок контролю та просування ігрових подій поміж об’єктів, разом з інформацію про стан гри, що можна отримати з об’єктів-комутаторів, надає повний контроль над ігровим станом.

3.5.2 Шаблони проектування для оптимізації ігрового процесу

Під час розробки відеоігор розробники мають пам’ятати про те, що кожна ітерація ігрового циклу має бути виконана за обмежений час, щоб надати користувачу кращий ігровий досвід без затримок, для цього використовують загальні підходи до оптимізації. Розглянемо приклади шаблонів проектування, що часто застосовуються в ігровій індустрії.

Шаблон проектування легковаговик (англ. flyweight pattern) дозволяє мінімізувати використання пам’яті через спеціалізований механізм створення об’єктів, при якому відбувається розподілення даних на скільки є можливим між подібними об’єктами. Реалізація цього механізму створює спеціалізовану структуру даних, що зберігає частини даних об’єктів, котрі є спільними і надає доступ до цих даних об’єктам тоді, коли це необхідно. Таким чином розробники можуть створювати велику кількість простих об’єктів не турбуючись про перенавантаження системи через нестачу пам’яті. Ігрові рушії часто містять у собі вбудовані реалізації шаблону легковаговик, що використовуються непомітно для розробника для представлення зображень, текстур, гліфів тексту, тощо.

При роботі з об’єктами котрі необхідно часто створювати й знищувати використовують шаблон проектування набір об’єктів (англ. object pull). Робота цього шаблону полягає в повторному використанні об’єктів, з метою зменшення кількості викликів методів їх ініціалізації й знищення, а також усунення необхідності частої активації механізму збору сміття, якщо такий присутній у мові програмування на якій розробляється відеогра. Структура шаблону являє собою чергу обгорнутих спеціалізованим класом, що перехоплює виклики створення та знищення обраних об’єктів й додає чи забирає об’єкти з черги. При запуску роботи даного механізму відбувається наповнення черги об’єктами до певного ліміту, далі якщо виникає потреба в створенні додаткових об’єктів, вони так само додаються у чергу, після чого стають доступними для використання.

3.5.3 Використання успадкування для представлення ігрових сутностей

Предметна область відеоігор часто надає розробникам природньо сформовані ієрархії успадкування ігрових сутностей. Це дозволяє створювати

програмні компоненти, котрі будуть використовуватися повторно в межах гри. Хоча успадкування ігрової логіки для створення різновидів однієї ігрової сутності є досить очевидним рішенням, успадкування носить важливу роль в створенні ієрархії структур даних, що використовуються у грі. Використовуючи підхід наслідування структур, що зберігають ігрові дані, можна створювати динамічні ігрові системи, в яких під час роботи відеоігрового застосунку буде присутня можливість заміни одного набору даних представлення об’єкту на інший.

3.6 Використання шаблону проектування «Модель-Представлення-Контролер»

Відеоігри є комплексними програмними засобами, що містять багато сутностей, зв’язки між котрими можуть ставати заплутаними в ході розробки продукту. Щоб забезпечити надійну й контрольовану структуру ігрового проекту використовують шаблон проектування «Модель-Представлення-Контролер». Ідея цього шаблону в контексті відеоігор полягає у відокремленні ігрових даних, тобто стану конкретних ігрових об’єктів, від логіки гри й візуального представлення. Одним із підходів до опису відеогри з допомогою даної моделі є представлення усіх елементів гри, що будуть візуально відображені користувачеві як класи представлення, логіка, що використовується для реалізації пов’язаних з візуальними елементами поміщується у класи-контролери, котрі в свою чергу використовують спеціальні класи-моделі для представлення стану ігрових об’єктів.

3.7 Модульна ігрова архітектура та тестування

Створення модульної ігрової архітектури полягає у виокремлені реалізованих ігрових механік таким чином, щоб їх можна було застосовувати повторно. Навіть якщо в грі не планується повторне використання деяких компонентів, такий підхід спрощує тестування коду ігрової програми, оскільки для перевірки коректності роботи певного модуля достатньо помістити його у середовище, що буде викликати методи й отримувати результати через встановлений механізм програмного інтерфейсу. Для модульних ігрових механік використовують тестові сцени, що представляють собою ігрову сцену, на які зазвичай зібрані усі ігрові механіки конкретної гри, для перевірки на коректність їх взаємодії.

3.8 Перехід від опису гри до її реалізації

Як було сказано раніше, відеоігри часто описують набором ігрових механік та правил, цей набір є наближенням до формалізації гри. З набору цих вимог можна здійснити поетапний перехід до більш конкретних вимог, котрі дозволяють перейти до програмної реалізації. Взявши за основу об’єктно орієнтований підхід, опишемо основні кроки цього переходу.

На початку ігрові дизайнери формулюють ідею гри, після чого задають набір ігрових механік, з яких складається ігровий процес, разом з цим, паралельно описується набір правил, за якими відбувається взаємодія між різними ігровими механіками.

Сукупність ігрових механік та правил трансформується у набір ігрових сутностей, що будуть реалізовувати ігрову логіку. Для сутностей описуються методи взаємодії одного з одним. На базі сукупності ігрових сутностей формують представлення рівня даних, що характеризує ігровий стан. Із представлення

аспектів ігрового стану виділяють частини, що можуть бути об’єднаними за логікою гри, для таких частин описуються ігрові об’єкти, що будуть їх представляти й зберігати у собі дані про частину ігрового стану. Після цього етапу починається реалізація логіки ігрових об’єктів, їх взаємодії.

У разі необхідності зміни певних ігрових механік чи правил, проходять по усім описаним етапам і у разі необхідності вносять корективи й оновлюють залежні елементи процесу. Схема переходів між етапами зображена на рисунку 3.6.

Рисунок 3.4 – Схема поетапного переходу від вимог гри до її реалізації

РОЗДІЛ 4. ПРОЕКТУВАННЯ ВІДЕОГРИ

4.1 Обрані підходи розробки

Для реалізації запланованого проекту було проаналізовано методи розробки відеоігор, що викладені у розділі 3 й обрано підходящі згідно вимог сформованих набором основних ігрових механік.

Контроль базового рівня ігрового циклу, а також низькорівневе управління вводом та виводом користувача буде покладено на обраний готовий ігровий рушій. Управління ігровим циклом для ігрових об’єктів буде здійснюватися за допомогою гібридного підходу послідовного виконання скінченних автоматів з комунікацією між об’єктами за допомогою системи подій.

Оскільки відеогра має досить багато елементів, що мають бути гнучкими до доповнення, було обрано ієрархічну схему комунікації між об’єктами з використанням набору класів-комутаторів.

В якості системи розробки було обрано ігровий рушій Unity.

4.2 Особливості роботи обраного ігрового рушія

Unity дозволяє створювати відеоігри для платформ Android, Windows, IOS, Smart TV, Web-Gl. Як середа розробки Unity пропонує графічний інтерфейс для створення та редагування структури проекту, та програмний інтерфейс для роботи з ігровою логікою на мові програмування C#.

Порядок виконання ігровий циклу Unity гарантується за рахунок того, що додатки розроблені з допомогою ігрового рушія запускаються на одному основному потоці. Використання програмного інтерфейсу Unity з не основних потоків не є можливим, для цього використовується механізм зворотного зв’язку, тобто усі сигнали отримані з інших потоків будуть оброблені ігровим циклом

Unity послідовно як і усі інші методи оновлення ігрових об’єктів. Такий підхід спрощує управління станом гри, оскільки розробнику не треба думати про можливі ситуації гонки потоків.

Мова програмування C# підтримує вбудований механізм створення делегатів й інтерфейси для використання систем подій, завдяки ключовим словам delegate та event. Для передачі ряду параметрів через систему подій використовується ключове слово delegate, воно дозволяє описати новий клас-функцію, котрий містить у собі дані про передані у нього параметри. Ключове слово event сигналізує про те, що на наступну подію можна підписатися, щоб отримувати зворотні виклики функцій реагування, переданих підписантами.

Основою для реалізації ігрової логіки в ігровому рушії Unity слугують класи, що успадковуються від класу MonoBehaviour. Ці класи є модулями, що можна додавати на абстрактні ігрові об’єкти GameObject. Клас MonoBehaviour надає низку методів для роботи з різними етапами ігрового циклу та життя ігрового об’єкту, такі як:

- Update – викликається при кожній ітерації ігрового циклу, коли об’єкт до якого застосований модуль та сам модуль активні;
- OnEnable – викликається коли модуль був увімкнений;
- OnDisable – викликається коли модуль був вимкнений;
- Start – викликається при першому включенні логіки ігрового об’єкту;
- Awake – викликається для всіх ігрових об’єктів з гарантією того, що при запуску гри будуть виконані методи Awake для всіх об’єктів перед тим як будуть відбуватися виклики інших методів контролю ігрового циклу.

Unity пропонує розробникові власноруч налаштовувати систему рендерінгу, визначати параметри якості зображення, обробки тіней, задавати правила постобробки, тощо. Для створення візуального стилю гри був обраний Universal Render Pipeline з додатковими налаштуваннями під мобільні

платформи. Такий конвеєр дозволяє створювати нові вершинні та фрагменті шейдери у вигляді коду і поєднувати їх використовуючи графічну систему вузлів.

4.3 Опис основних елементів обраної ігрової архітектури

Для управління грою було виділено наступні модулі:

- GameManager – представляє ядро логіки гри й реалізує у собі методи взаємодії ігрових об’єктів різних типів;
- InputManager – реалізує перевірку, аналіз та відправлення подій в залежності від вводу користувача, реалізований на базі рушія Unity;
- UIManager – відповідає за роботу графічного інтерфейсу користувача, оновлення відображення показників, сюжетних текстів, відповідей та вікон різних ігрових меню;
- AudioManager – відповідає за роботу аудіо, що присутнє у грі, зберігає дані про те які аудіо доріжки мають програватися в залежності від стану гри;
- DialogueSystem – система роботи з поточним станом сюжету, відповідальна за те, як ті чи інші картки впливають на ігровий стан при виборі сюжетних варіантів гравцем;
- SaveSystem – система збереження поточного стану гри, записує та завантажує дані про поточну ігрову колоду.

На рисунку 4.1 зображено напрям зв’язків між описаними модулями, що визначається напрямом виклику методів з інших сутностей. Зв’язки було вирішено утворювати таким чином, щоб модулі по максимуму взаємодіяли через модуль GameManager, з метою збільшення гнучкості системи.

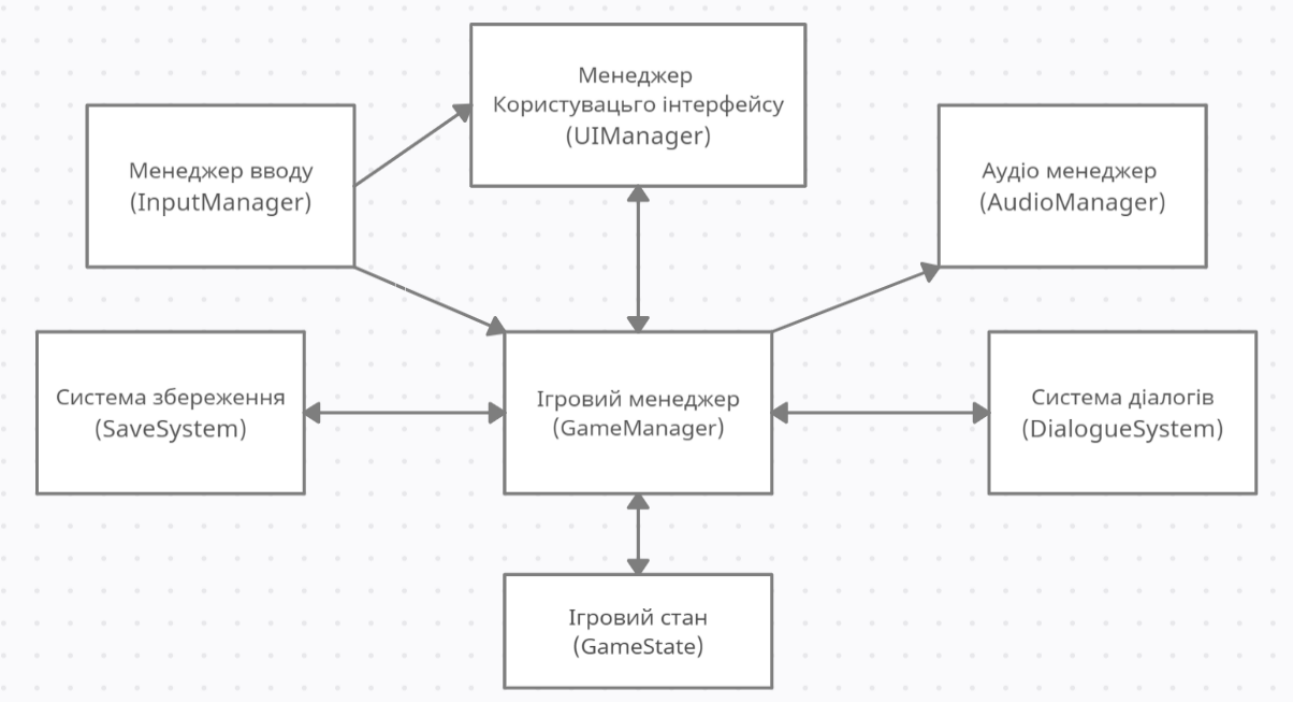

Рисунок 4.1 - схема направлення викликів між основними елементами ігрової системи

Кожен модуль є сукупністю вузько направлених ігрових елементів та класів, що реалізують конкретну ігрову логіку.

4.4 Забезпечення гнучкості проекту до доповнення

Ключовою ігровою механікою даного проекту є взаємодія з сюжетом, через вибір варіантів на сюжетних картках. Для забезпечення гнучкості до додавання нових сюжетних елементів, картки мають підтримувати деякий варіант скриптів, що можна підключати до вже існуючої логіки картки.

Такий підхід реалізується за допомогою інструменту Unity – ScriptableObject, котрий є класом, на який ігровий рушій зберігає посилання і має можливість запускати методи цього класу під час роботи гри.

РОЗДІЛ 5. РЕАЛІЗАЦІЯ ВІДЕОГРИ

5.1 Ігрові дані

Поточний ігровий стан визначається значеннями системи ігрових ресурсів та інформацією про поточний стан ігрової колоди, ці дані під час гри зберігаються у системі DialogueSystem.

Важливим аспектом ігрової програми є можливість збереження процесу сесії для продовження гри з того ж моменту або з найближчої контрольної точки. Запропонована карткова система є зручним представленням ходу сюжету оскільки дозволяє зберігати поточний стан гри на диск простим перезаписом посилань на карти поточної руки та ігрових значень-характеристик у файл.

Ігрові характеристики кількості людей, грошей, значення задоволення населення та готовності містяться у класах нащадках від класу Parameter, що слугує обгорткою числової змінної і реалізує інтерфейси необхідні для візуалізації змін значень відповідних характеристик.

Для групування даних про об’єкти сюжетної систему була створена об’єктна база даних, що під час роботи гри може за парою ключ-значення визначати необхідні ScriptableObject і отримувати посилання на них в оперативній пам’яті.

Щоб реалізувати таку базу даних, при додаванні нової зв’язки пари карт у колоді обраховується наступне значення хеш-функції:

$$DB_{value}(S) = sha256(concat(S.type, S.params))$$

Далі це значення вставляється разом з посиланням на серіалізоване значення у відсортовану множину існуючих ключів бази даних. За рахунок того, що ключі відсортовані, пошук здійснюється за $O(\log(n))$ бінарним пошуком. В свою чергу сама база даних є нащадком класу ScriptableObject, що дозволяє зберігати та оновлювати її без необхідності зборки усього проекту.

Дана схема дозволяє виконувати збереження ігрового стану після кожного кроку користувача.

Під час старту головної сцени гри, відбувається пошук у файловій системі на наявність файлів з даними про збережені сесії гри, якщо такий файл знайдений – відбувається спроба завантажити збережені дані, якщо спроба була не успішною, чи файлів збереження не знайдено, то розпочинається нова сесія гри.

5.2 Користувацький інтерфейс

Користувацький інтерфейс має відображати достатню інформацію гравцю про поточний стан гри, щоб користувач міг приймати ігрові рішення. В контексті обраної гри було реалізовано ігровий інтерфейс зображений на рисунку 5.1

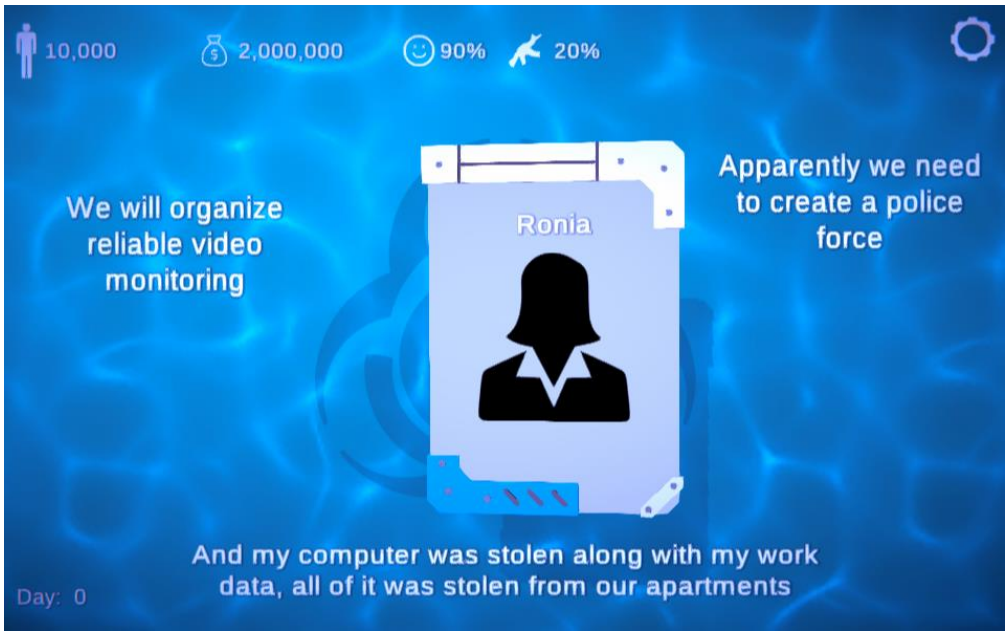

Рисунок 5.1 – користувацький інтерфейс гри

Ігровий рушій Unity надає можливість створення двовимірних графічних інтерфейсів, котрі можна налаштовувати для адаптивної поведінки на екранах з різним співвідношенням сторін. Інтерфейс гри складається з елементів

відображення поточних значень ресурсів, поточної сюжетної картки, текстів варіантів відповідей та запитання. Як видно на рис. 5.1, в грі поєднується використання двовимірних та тривимірних елементів. За це відповідальний модуль Unity - Canvas, котрий здатний створювати динамічні текстури, які розташовуються в тривимірному просторі.

Для оновлення відображення при зміні ігрового стану використовується система подій, розглянемо оновлення лічильників та їх анімацію при зміні значення (рис. 5.2).

```
public void Start()
{
   startColor = peopleCountText.color; // Taking any text

   GameEvents.current.onUpdateGameStateDisplay += UpdateDisplay;
   GameEvents.current.onVisualiseChange += VisualiseChange;
}

private void OnDestroy()
{
   GameEvents.current.onUpdateGameStateDisplay -= UpdateDisplay;
   GameEvents.current.onVisualiseChange -= VisualiseChange;
}

public void UpdateDisplay()
{
   GameState gs = GameState.current;
   peopleCountText?.SetText(gs.GetRepresentation(GameStateParameter.PEOPLE_COUNT));
   peopleHappinesText?.SetText(gs.GetRepresentation(GameStateParameter.PEOPLE_HAPPINES));
   MoneyText?.SetText(gs.GetRepresentation(GameStateParameter.MONEY));
   ArmyText?.SetText(gs.GetRepresentation(GameStateParameter.ARMY));
   DayNumberText?.SetText(gs.GetRepresentation(GameStateParameter.DAY_NUMBER));
}
```

Рисунок 5.2 – приклад підписки на події оновлення значень ігрових ресурсів

5.3 Ігрова логіка

Ігрова логіка проекту реалізується через сукупність об’єктів, пов’язаних в ієрархії обміну повідомленнями з сутністю GameManager.

Клас GameManager відповідальний за старт гри, виклик методів завантаження ігрової колоди із системи збереження в діалогову систему, перехід

між сценою головного ігрового меню та основною ігровою сценою, управління системою ресурсів.

5.3.1 Система управління ресурсами

В контексті обраних ігрових механік ігровими ресурсами є набір з чотирьох параметрів, щоб забезпечити оновлення значень параметрів та надсилання сигналів про їх оновлення була використана система подій. Для параметру загального вигляду був створений клас, в якому міститься поле поточного значення, при зміні котрого клас викликає делегат, прив’язаний до свого параметру, що ініціює оновлення усіх сутностей, що були підписані на даний параметр. Доступ до всіх ігрових ресурсів відбувається через спеціалізований клас GameState.

5.3.2 Діалогова система

Створення сюжетної системи потребує збереження даних про поточний стан гри та можливі варіанти розвитку з поточної точки сюжету. Було вирішено представляти сюжетну систему як однозв’язний графу розвитку подій з інформацією про подію у вершинах, на ребрах зберігаються умови відповідей для переходів у наступні вершини-стани та додатково списку посилань на вершини поточних сюжетних ліній (рис. 5.3).

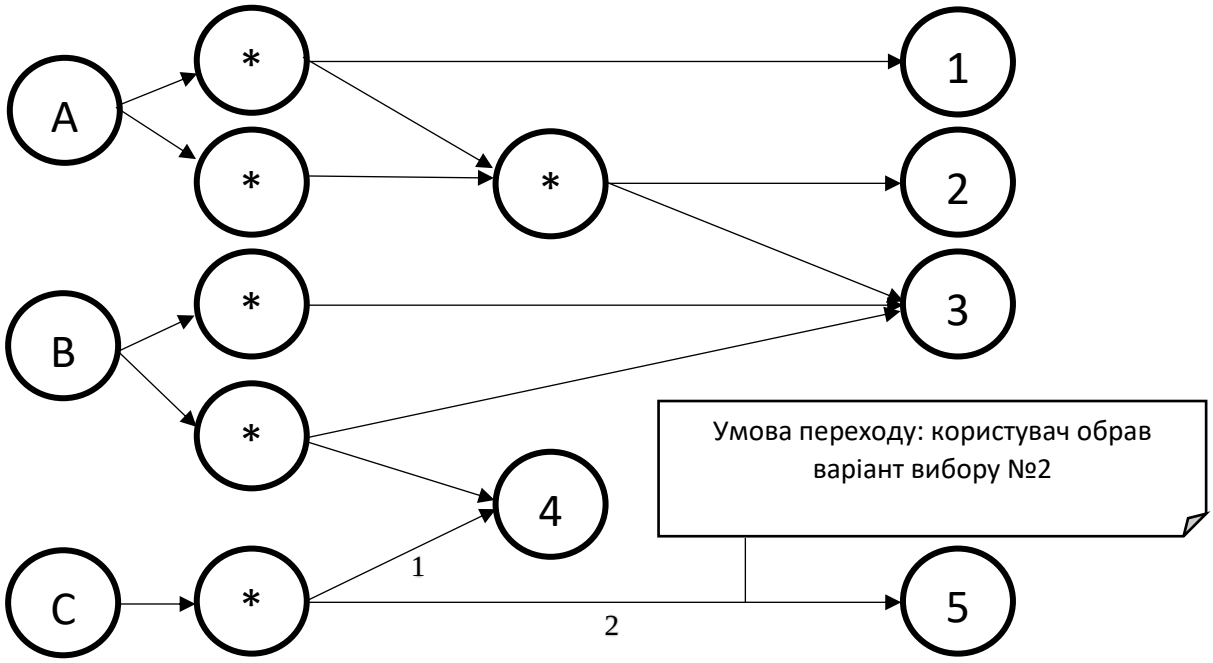

Рисунок 5.3 – представлення графу подій

На рис. 5.1 наведений приклад схематичного зображення сюжетної лінії, для старту якої мають бути досягнена хоча б одна з умов {A, B, C}. Переходячи по заданим на ребрах правилам переходів змінюється сюжет гри, доки не буде досягнена одна з термінальних вершин, позначених числами {1, 2, 3, 4, 5}.

Формалізуємо наведене вище представлення структури сюжету. Позначимо весь сюжет гри як $H = (V, E)$, де V – множина вершин-станів, E – множина ребер-переходів між станами. Позначимо поточний стан гри як S. Позначимо множину вже відвіданих вершин у системі як $C \in V$; Для зручності розробки та подальшої роботи сценаристів весь сюжет поділений на сюжетні лінії, відносно короткі відрізки сюжету, відокремлені за змістом, вони являють собою підграф:

$$G \in H;\ G = (V', E');\ V' \in V;\ E' \in E;$$

Задамо множину вхідну множину вершин-тригерів, що призводять до активації сюжетної лінії G як $T \in V'$.

Важливо помітити, що в даній системі не накладаються умови на відсутність циклічних переходів, оскільки певні події можуть мати повторюваний характер.

Введемо функцію активатор, що повертає один із можливих варіантів вибору, котрий обрав би абстрактний користувач під час гри.

$$u(S, C, v, e_1, e_2) \rightarrow e_{res}$$

У наведеному вище представленні e_1, e_2 це ребра можливих варіантів відповідей, в поточній системі їх, два, проте може бути більше, e_{res} це одне з вхідних в аргументи функції ребер. Позначимо функцію переходу по ребру графа як $n(G, c, e) \rightarrow c'$, де c це поточна вершина, а e – вихідне з неї ребро у вершину c'. Тоді один хід користувача можна описати наступним виразом:

$$s(G, c, u(S, C, c, e_1, e_2)) \rightarrow c'$$

Просування по сюжету гри складається з множини таких переходів, доки не буде досягнута деяка тупикова вершина, в такому разі можливий виграш чи програш користувача в залежності від даних гри, чи падіння одного з ресурсів гри S до мінімального значення, що означає неможливість переходу по сюжету.

Окрім даних потрібних для переходів в сюжетній системі, вершини та ребра сюжетного графу можуть містити у собі додаткову інформацію про зміни параметрів ігрової ситуації чи тригери певних візуальних чи додаткових сюжетних подій. Така інформація зберігається у вигляді об’єктів-функцій, що можуть визиватися: перед демонстрацією сюжетної вершини користувачу та після обрання певного варіанту відповіді.

5.3.3 Система взаємодії користувача з грою

Для перехоплення користувацького вводу використовується спеціалізований модуль вводу ігрового рушія – InputManger. Даний модуль на кожній ітерації ігрового циклу перевіряє буфер користувацького вводу на наявність подій кліку чи дотику, після чого оброблену інформацію можуть використовувати інші ігрові сутності.

Інструментом управління сюжетними картками є їх перетягування на ліву чи праву сторони екрану. Ігровий простір представляє собою стіл, для простоти будемо вважати, що він лежить у площині YOX. Картка має перебувати над столом впродовж усієї взаємодії користувача і певним чином відображати своє переміщення. Для відображення переміщення будемо змінювати центр об’єкту карти та Ейлерів кут в залежності від напряму переміщення від початкової точки створення карти.

Для визначення дотику користувача до тривимірних об’єктів сцени використовується метод посилання пробних променів (англ. Ray Casting) [9][10] з точки на проективній площині поточної камери рендеру, котрій відповідає точка дотику на екрані чи місце кліку мишею. Якщо якийсь промінь перетнув тривимірний об’єкт виконується перевірка на те чи є цей об’єкт необхідним для даної операції. Якщо був обраний необхідний об’єкт (карта) то фіксується відмітка внутрішнього стану відображення картки про початок перетягування. Коли було зафіксовано дотик до об’єкту, у пам’ять заносяться дані про початкове положення та трансформацію повороту щоб у випадку завершення процедури перетягування можна було повернути об’єкт у початкове положення.

Для обрання одного із варіантів відповіді користувач має перетягнути карту на сторону вибраного варіанту і відпустити карту. Щоб зафіксувати знаходження об’єкту карти над варіантом відповіді виконується процедура RayCast з точки центру карти, спроектованої на площину камери, для пошуку

пересічення променів з об’єктами варіантів відповідей. У випадку коли таких пересічень не знайдено карта повертається у вихідне положення запам’ятоване раніше.

5.4 Створення інструментів ігрового рушія

Ігровий рушій Unity надає можливість створення власних графічних інтерфейсів надбудов редактора шляхом використання спеціалізованого програмного інтерфейсу.

Основною частиною взаємодії ігрового дизайнера з проектом є створення сюжетних ліній, для спрощення цієї роботи було реалізовано надбудову для ігрового рушія Unity, для створення, збереження і додавання нових сюжетних карток, та колод.

Створений редактор надає можливість редагування наступних полів карток:

- Поміток групи, до якої відноситься картка – використовується для керування появи, пошуку, видалення картки в колоді.
- Посилань на представлення персонажу картки.
- Опису картки, що являє собою текст події.
- Текстів варіантів відповідей.
- Посилань на скрипти, що мають виконуватися при появі картки, її зникненні та при виборі певного варіанту відповіді.
- Опису змін ресурсів, що відбудеться при обранні варіанту відповіді.
- Подій управління колодою.

Оскільки редактор, що розробляється буде працювати зі статичними даними карток були використані можливості класу Editor та серіалізації. Для нового представлення серіалізований об’єкт проходить обробку за типом на

назвами полів, після чого з наявних реалізацій графічного інтерфейсу обирається одна і демонструється у редакторі Unity (рис. 5.4).

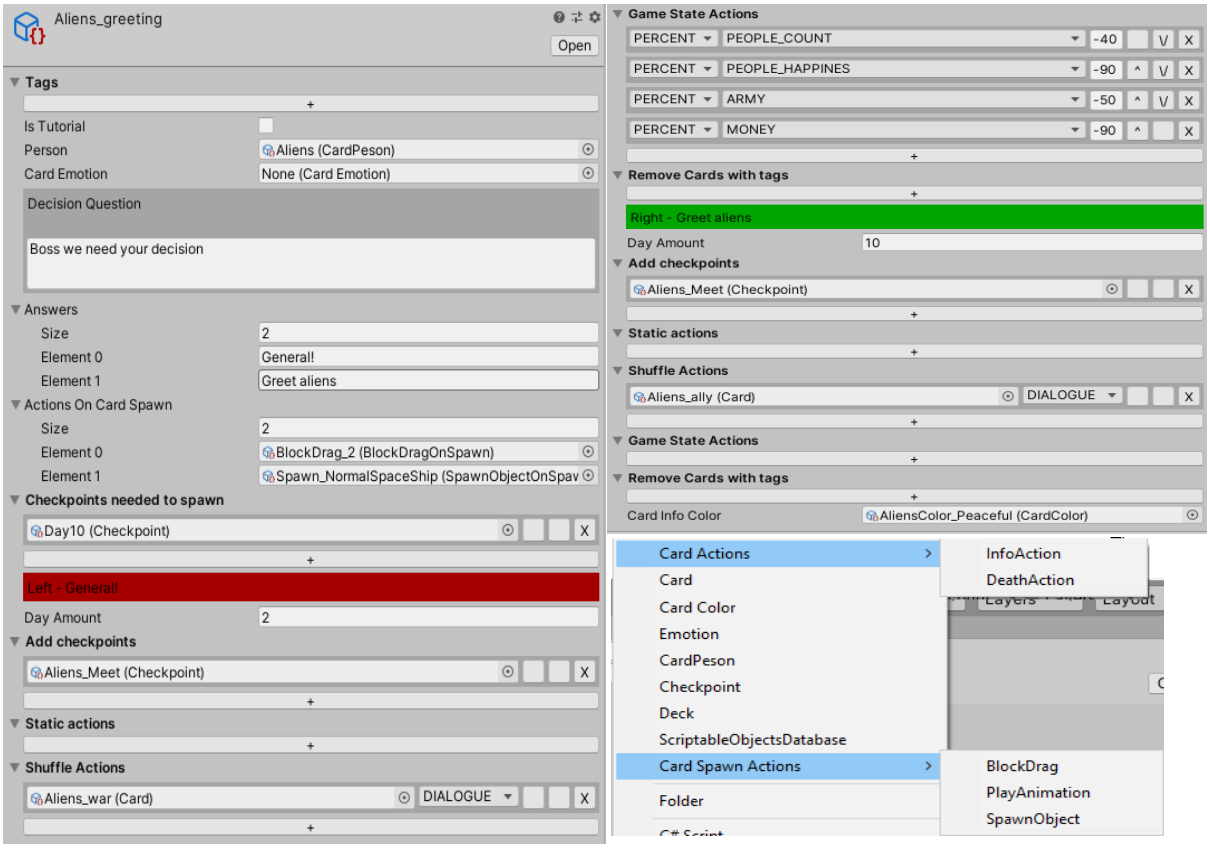

Рисунок 5.4 - інтерфейс для редагування сюжетних карток

ВИСНОВКИ

В даній роботі було описано підхід до формалізації елементів, що утворюють відеоігоровий продукт, описано та проаналізовано загальні підходи до розробки відеоігор та розроблено ігровий проект з елементами стратегії на базі проведеного аналізу. Приділено увагу етапам створення відеоігор, внутрішньому принципу роботи ігрових програм, описано шлях переходу від ігрових механік і правил до конкретної програмної реалізації.

Виконаний аналіз вимог гри і в ході розробки проведений опис створених ігрових елементів, мотивації до вигляду їх реалізації. Ця робота може слугувати основою для створення нових відеоігрових проектів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1]. itch.io [Електронний ресурс] Режим доступу до ресурсу: https://itch.io

[2]. Sicart M. Defining Game Mechanics [Електронний ресурс] / M. Sicart – 2008. – Режим доступу до ресурсу: http://gamestudies.org/0802/articles/sicart

[3]. David R. Finite State Machines / R. David – 2005. – 2 с.

[4]. Wagstaff B. State Machines in Games [Електронний ресурс] / B. Wagstaff – 2013. – Режим доступу до ресурсу: https://www.gamedev.net/articles/programming/general-and-gameplay-programming/state-machines-in-games-r2982/

[5]. Bee C.P. A High Level Design of the Sub-Farm Event Handler [Електронний ресурс] / C.P. Bee, M. Caprini, P.Y. Duval, D. Francis, D. Laugier, G. Mornacchi, C. von Praun, Z. Qian and F. Touchard – 1997. – Режим доступу до ресурсу: https://web.archive.org/web/20070211111543/http://atddoc.cern.ch/Atlas/Notes/061/Note061-1.html

[6]. Великовский Н. EventBus — Система событий для Unity [Електронний ресурс] / Н. Великовский – 2020. – Режим доступу до ресурсу: https://habr.com/ru/post/527418/

[7]. Clingerman G.W. The State of Things [Електронний ресурс] /G.W. Clingerman – 2008. – Режим доступу до ресурсу: http://www.xnadevelopment.com/tutorials/thestateofthings/thestateofthings.shtml

[8]. Shuller D. How to Build a JRPG: A Primer for Game Developers [Електронний ресурс] / D. Schuller – 2013. – Режим доступу до ресурсу: https://gamedevelopment.tutsplus.com/articles/how-to-build-a-jrpg-a-primer-for-game-developers--gamedev-6676#state

[9]. Westermann R Efficiently using graphics hardware in volume rendering applications. In Proceedings of SIGGRAPH 98 / R. Westermann, T. Ertl – 1998. – 169–178 с.

[10]. Mora B. A new object-order ray-casting algorithm / B. Mora, Jean-Pierre Jessel, R. Caubet – 2002. – 203-210 с.