Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 «Інженерія програмного забезпечення»

На тему: *Розробка програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення*

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-21-1
_____ *Брільов Є. Ю.*

Керівник кваліфікаційної роботи _____ */ Стрюк А. М. /*

Завідувач кафедри _____ */ Стрюк А. М. /*

Кривий Ріг
2025

Криворізький національний університет
Факультет: *Інформаційних технологій*
Кафедра: *Моделювання та програмного забезпечення*
Освітньо-кваліфікаційний рівень: *бакалавр*
Спеціальність: *121 "Інженерія програмного забезпечення"*

ЗАТВЕРДЖУЮ
Зав. кафедри
_____*Стрюк А. М.*
«___»_____ 2025__ р.

ЗАВДАННЯ
на кваліфікаційну роботу
студенту групи ІПЗ-21-1 Брільову Єгору Юрійовичу

1. Тема: *Розробка програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення* затверджено наказом по КНУ №119 від «10» квітня 2025 р.
2. Термін подання студентом закінченого проекту «20» червня 2025 р.
3. Вихідні дані по роботі: *Пояснювальна записка: 81 сторінки, 34 рисунків, 1 додаток, 12 використаних у роботі джерел*
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): актуальність розробки програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення. Проектування програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення. Розробка програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення.
5. Перелік графічного демонстраційного матеріалу: Приклади існуючих програм. Діаграма потоків даних. Діаграми використання. Блок-схеми основних алгоритмів. Структура бази даних. Приклади роботи розробленої програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	*Формулювання мети та задач роботи*	*13.02.2025-20.02.2025*
2	*Аналіз інформаційних джерел*	*21.02.2025-09.03.2025*
3	*Визначення вимог до програмного забезпечення*	*10.03.2025-18.03.2025*
4	*Розробка функціональної схеми*	*19.03.2025-23.03.2025*
5	*Розробка алгоритмів*	*24.03.2025-31.03.2025*
6	*Розробка бази даних*	*01.04.2025-14.04.2025*
7	*Розробка програмного забезпечення*	*15.04.2025-13.05.2025*
8	*Тестування програмного забезпечення*	*14.05.2025-20.05.2025*
9	*Оформлення пояснювальної записки*	*21.05.2025-12.06.2025*
10	*Розробка демонстраційних матеріалів*	*13.06.2025-17.06.2025*

Дата видачі завдання: «_____»_____ 2025 р.
Студент _____ Брільов Є. Ю.
Керівник роботи _____ Стрюк А. М.

РЕФЕРАТ

Кваліфікаційна робота містить 81 сторінку, 35 рисунків, 12 джерел, 1 додаток.

Метою кваліфікаційної роботи є проєктування та реалізація програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення.

Відповідно до мети в роботі визначено актуальність розробки комп'ютерної гри з історії інженерії програмного забезпечення. Досліджено становлення інженерії програмного забезпечення. Вивчено освітній потенціал комп’ютерних ігор в жанрі візуальних новел. Спроєктовано сюжетну структуру візуальної новели, присвячену історії інженерії програмного забезпечення.

В проєктній частині виконано проєктування сюжетної структури гри, її основних алгоритмів, побудовані UML-діаграми, що відображають режими її використання, взаємодію користувача з інтерфейсом.

Виконано добір засобів розробки комп'ютерної гри з історії інженерії програмного забезпечення. Детально розглянуто процес створення ігрових сцен та персонажів. Наведено приклади роботи гри в різних режимах.

Створена гра може бути корисна всім, хто цікавиться історією створення інженерії програмного забезпечення, історичними постатями, які пов’язані з нею. Гру можна рекомендувати як інтерактивний матеріал до курсу «Основи інженерії програмного забезпечення».

ABSTRACT

The qualification work comprises 81 pages, 35 illustrations, 12 sources, and 1 appendix.

The purpose of the qualification work is to design and implement software for a computer game on the history of software engineering.

In accordance with the purpose, the work defines the relevance of developing a computer game on the history of software engineering. The formation of software engineering has been studied. The educational potential of computer games in the visual novel genre has been examined. A storyline structure for a visual novel dedicated to the history of software engineering has been designed.

In the project section, the storyline structure of the game and its main algorithms have been developed; UML diagrams reflecting usage modes and user interaction with the interface have been constructed.

Development tools for the computer game on the history of software engineering were selected. The process of creating game scenes and characters is considered in detail. Examples of the game functioning in various modes are provided.

The created game can be useful for everyone interested in the history of software engineering development and historical figures associated with it. The game can be recommended as interactive material for the course "Fundamentals of Software Engineering."

ЗМІСТ

ВСТУП

Інженерія програмного забезпечення — це одна з наймолодших галузей інженерії, яка почала активно розвиватися у другій половині XX століття. Її виникнення стало відповіддю людства на нові виклики, пов’язані із швидким розвитком комп’ютерних технологій і необхідністю створення більш складних, масштабних та надійних програмних систем. Історія появи цієї дисципліни тісно переплітається з кризою програмного забезпечення, яка виникла у 60-х роках минулого століття.

У той час розвитку комп’ютерної техніки супроводжувала експоненціальна потреба у створенні програмного забезпечення. Зростання обчислювальної потужності давало можливість розробляти дедалі складніші та масштабніші програми, які могли автоматизувати бізнес-процеси, наукові дослідження, державне управління і навіть військові операції. Однак підходи до написання та організації програмного коду не встигали за новими викликами, що привело до низки проблем: постійних затримок у розробці, перевищення бюджету, низької продуктивності програм, складності в підтримці і високої ймовірності помилок.

Ця ситуація отримала назву "кризи програмного забезпечення". Вона стала очевидною саме в 60-х роках, коли через стрімке зростання попиту на нові технології багато проектів зазнавали невизначеності та втрачали контроль над розробкою. Крім того, відсутність стандартизованих методів проектування і управління процесом створення програм викликала хаос у багатьох компаніях та організаціях. Саме тоді стало зрозуміло, що підходи до програмування необхідно переглянути та систематизувати, перетворивши цю діяльність з ремісництва на справжню інженерну дисципліну.

У рамках міжнародної конференції НАТО в 1968 році вперше прозвучала концепція "інженерії програмного забезпечення". Її метою було розроблення нових принципів, методів і технологій, які б забезпечували якісніший, надійніший та більш організований процес створення програмних

продуктів. В цей момент почалося формування теоретичної бази дисципліни: стандартизація процесів розробки, впровадження моделей життєвого циклу програми, розроблення підходів до тестування і забезпечення якості, а також створення засобів управління проектами.

Таким чином, інженерія програмного забезпечення стала важливою відповіддю на кризу, спричинену недостатньою адаптацією до зростаючого попиту. Сьогодні вона є центральним елементом у створенні цифрових рішень, забезпечуючи планування, розробку, тестування та підтримку програмного коду за допомогою комплексних, науково обґрунтованих підходів. Інженерія програмного забезпечення стала фундаментом, без якого сучасні інформаційні технології не змогли б досягнути такого рівня розвитку.

Історія створення інженерії програмного забезпечення дуже цікава. Але вивчати її за протоколами конференції 1968 року або іншими матеріалами доволі нудно. Особливо для студентів, що тільки стикаються з цією спеціальністю. Тому виникла ідея розкрити історію зародження інженерії програмного забезпечення у формі комп’ютерної гри в стилі візуальної новели.

Тож метою кваліфікаційної роботи є проєктування та реалізація програмного забезпечення комп'ютерної гри з історії інженерії програмного забезпечення.

1 АКТУАЛЬНІСТЬ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ ГРИ З ІСТОРІЇ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Дослідження становлення інженерії програмного забезпечення

З появою перших комп’ютерів програмування не вважалося серйозною професійною діяльністю і сприймалося як простий процес обслуговування комп’ютерної техніки. Програмісти виконували завдання, що полягали в написанні простих операційних інструкцій для машин, які були громіздкими та могли виконувати лише обмежений набір функцій. Проте, із зростанням складності завдань і необхідності автоматизації все більшої кількості процесів стало очевидним, що створення програмного забезпечення вимагає значно більше, ніж просте маніпулювання машиною. Виникла потреба в ґрунтовних знаннях математики, логічному й алгоритмічному мисленні, які забезпечували здатність вирішувати складні задачі ефективно та оптимально.

Додатково до цього розвиток програмного забезпечення виявив залежність від здібностей організовувати великі обсяги даних, створювати реалістичні симуляції, формувати інтерактивні елементи та забезпечувати точність розрахунків, що вимагає комбінування теоретичних знань із практичними навичками. Ця нова потреба дала поштовх до виникнення і стрімкого розвитку комп’ютерних наук як окремої сфери знань. Університети почали формувати відповідні освітні програми, створювались спеціалізовані інститути для досліджень в галузі алгоритмів, теорії обчислень та штучного інтелекту.

Але не зважаючи на це, у 60-х роках виникло явище, яке отримало назву «криза програмного забезпечення» (рис. 1.1). Це явище було пов’язане з різким зростанням складності комп’ютерних програм, а також зі значним розширенням сфери їх застосування у науці, промисловості, бізнесі та державному управлінні. Основною проблемою стало те, що методи і підходи, які використовувалися для розробки програм, були недостатньо

структурованими і не враховували потреби у масштабованості, надійності та ефективності. Відсутність системного підходу призводила до численних помилок у коді, перевищення термінів розробки, а також до фінансових втрат. Інженери-програмісти зіштовхувалися зі складністю управління обсягами робіт та координації між членами команд, що поступово перетворило процес створення програмного забезпечення на один з найбільш трудомістких і непередбачуваних аспектів технологічної сфери. Це явище спонукало до перегляду підходів до розробки програмного забезпечення, що в результаті сприяло виникненню нових методологій, таких як структурне програмування, а згодом і об’єктно-орієнтоване програмування, а також комплексних систем методів управління проектами.

Рисунок 1.1 – Ілюстрації в популярній літературі, присвячені кризі програмного забезпечення

Криза програмного забезпечення, що виникла в 1960-х роках, стала ключовим викликом для індустрії, який вимагав перегляду підходів до розробки програмних продуктів. На той час у інших галузях інженерії вже

існували систематизовані методології, стандарти та принципи, які забезпечували контроль над процесами виробництва, знижували ризики та підвищували якість продукту. Це підштовхнуло розробників програмного забезпечення до пошуку аналогічних рішень у своїй сфері, що дало початок ідеї створення "інженерії програмного забезпечення". Увага почала концентруватися на систематизації процесу розробки, введенні чітких вимог, розробці етапів життєвого циклу програмного забезпечення, організації командної роботи та використанні автоматизованих інструментів.

Вперше на загал про появу нової галузі знань – інженерії програмного забезпечення – було проголошено на історичній конференції “Software Engineering”, яка відбулася у жовтні 1968 року в мальовничому містечку Гарміш-Партенкірхен, Німеччина (рис. 1.2). Ця конференція, організована під егідою НАТО, мала вирішальне значення для формування концептуальних основ сучасного підходу до розробки програмного забезпечення як дисципліни, що поєднувала технічну майстерність, системне мислення та управління процесами. Учасники заходу обговорювали ключові проблеми, серед яких були забезпечення надійності, керованості та ефективності програмних систем, що ставали все більш складними. Саме тоді вперше було визнано, що створення програмного забезпечення потребує інженерного підходу, де методичність та стандарти відігравали ключову роль. Ця подія стала важливим кроком у переході від інтуїтивного програмування до системного проектування, поклавши основу для виникнення цілої галузі, яка надалі розвивалася в геометричній прогресії, охоплюючи методології, інструменти та практики, які сьогодні стали невід'ємною частиною світової технологічної індустрії.

Рисунок 1.2 – Фотографія з конференції «Software Engineering» 1968 року

Конференція НАТО 1968 року відіграла ключову роль у визнанні необхідності формалізації та професіоналізації розробки програмного забезпечення. Введення терміну "програмна інженерія" стало початком нової ери, де акцент змістився від індивідуальної майстерності до систематичних процесів, методів та інструментів. Обговорювані проблеми (масштаб, складність, надійність, вартість, документація, підтримка) та запропоновані рішення (модульність, специфікація, ієрархічне структурування, стандартизація компонентів, тестування) залишаються актуальними і сьогодні, демонструючи далекоглядність учасників конференції. Документ підкреслює усвідомлення того, що програмне забезпечення стало невід'ємною частиною складних систем і вимагає такого ж інженерного підходу, як і апаратне забезпечення.

Звіт про конференцію НАТО 1968 року з програмної інженерії охоплює низку критичних проблем у розробці та обслуговуванні програмного забезпечення, які обговорювали експерти з усього світу. Зокрема, висвітлюються труднощі з дотриманням графіків та специфікацій у великих проєктах, а також питання освіти майбутніх інженерів програмного забезпечення. Значну увагу приділено виробничим аспектам, включаючи

технічні інструменти, тестування, оцінку продуктивності та управління проєктами. Учасники також обговорювали суперечливе питання окремого ціноутворення на програмне забезпечення та важливість зворотного зв'язку з користувачами для вдосконалення систем.

Звіт про конференцію «Software Engineering» 1968 року є цінним документом не лише з історичної точки зору, але й для навчання майбутніх програмістів, адже він окреслює фундаментальні концепції, які стали основою сучасної програмної індустрії. Ця конференція, що часто вважається моментом народження терміна «Software Engineering», відобразила нагальні проблеми, з якими галузь стикалася на той час, такі як невизначеність процесів розробки, неефективне управління проектами, складність в масштабуванні програмного забезпечення й недостатня увага до перевірки якості кінцевого продукту. У звіті були підняті питання стандартизації методів роботи з кодом, необхідність впровадження чітких управлінських структур, а також більш строгий контроль над всіма етапами життєвого циклу програмного забезпечення.

Для сучасних програмістів цей звіт є важливим джерелом навчання, оскільки на його основі можна простежити еволюцію підходів до розробки програмного забезпечення та зрозуміти шлях, який пройшла індустрія від хаотичних методів до впорядкованих процесів. Крім того, аналіз документу дає змогу зрозуміти причини, через які були підняті ті чи інші питання, а також побачити, наскільки актуальними деякі з них лишаються й нині. Це дозволяє не лише використовувати набуті знання, але й критично оцінювати сучасні проблеми, адаптуючи запропоновані тоді ідеї до реалій XXI століття і пошуку інноваційних рішень у сфері програмування.

Але вивчати історію появи інженерії програмного забезпечення за сухими офіційними документами може здатися доволі нудним завданням, особливо для тих, хто тільки починає знайомитися з цією сферою. Тому для адаптації матеріалу до сучасних інтерактивних форматів є доцільним викласти

цю історію у вигляді захоплюючої комп'ютерної гри, наприклад, у жанрі візуальної новели.

У такій грі історія галузі оживає через взаємодію гравця з персонажами, які символізують ключові постаті, події чи концепції в історії програмної інженерії. Так гравець занурюється в атмосферу 1960-х років, коли постала криза "програмного забезпечення" і коли поняття інженерії програмного забезпечення вперше вийшло на світову арену завдяки знаменитим конференціям НАТО.

Персонажі гри могли б відображати реальних історичних постатей – учасників конференції – та діяти як наставники, що розповідають про базові концепції, на яких формується інженерія програмного забезпечення.

Завдяки такому формату історія інженерії програмного забезпечення стане не просто хронологією подій, а живою інтерактивною пригодою. Це не лише сприятиме популяризації галузі, але й залучатиме нових ентузіастів до вивчення технологій через цікавий та доступний спосіб подання матеріалу.

Тож нашою задачею має стати створення комп’ютерної гри в жанрі візуальної новели, що знайомить з формуванням інженерії програмного забезпечення.

1.2 Освітній потенціал комп’ютерних ігор в жанрі візуальних новел

Візуальні новели, як жанр комп’ютерних ігор, мають значний освітній потенціал. Цей формат поєднує інтерактивні елементи з глибокими сюжетами, що дозволяє створювати навчальні середовища, здатні залучити як юних, так і дорослих користувачів. Вони надають можливість інтегрувати такі методи навчання, як гейміфікація та сторітелінг, створюючи умови для ефективного засвоєння матеріалу.

Гейміфікація – це прискорення залучення користувача через використання ігрових механік, таких як система досягнень, рівнів, завдань та винагород. У контексті візуальних новел цей підхід стимулює гравця активно брати участь у навчальному процесі, вирішуючи завдання, приймаючи

рішення й досліджуючи сюжетні лінії. Наприклад, історичні візуальні новели можуть пропонувати гравцям виконувати ролі реальних історичних персонажів або переживати ключові події минулого, прокладаючи власний шлях через них. Таким чином, гравець не лише сприймає факти, а й активно взаємодіє з контекстом, що сприяє глибшому розумінню інформації.

Сторітелінг є ще одним сильним аспектом візуальних новел. Через потужний та емоційно насичений сюжет гра може передавати важливі знання й життєві уроки. Наприклад, візуальна новела з фокусом на соціальних проблемах може допомогти користувачу зрозуміти складні аспекти людських відносин, моральні дилеми або наслідки певних дій. У навчальному контексті сюжети таких новел можуть бути побудовані на основі історичних подій, наукових концепцій або культурних реалій, що забезпечує не лише навчання, а й емоційне залучення.

Крім того, візуальні новели дають можливість адаптувати навчальний контент до конкретної аудиторії. Наприклад, для дітей сюжети можуть бути більш барвистими й інтерактивними, а для студентів чи дорослих – включати складні теми, глибокі моральні роздуми й деталізовану інформацію.

Таким чином, візуальні новели, завдяки унікальному поєднанню інтерактивності, графіки та глибокого сюжету, можуть бути ефективним і цікавим інструментом в освітньому процесі. Це новий спосіб розвивати критичне мислення, емпатію та творче бачення, роблячи навчання не лише продуктивним, а й захопливим.

Можна виділити декілька важливих переваг подання навчального матеріалу у вигляді візуальних новел:

– Візуальні новели можуть створювати симуляції різних ситуацій, наприклад, історичних подій, професійних сценаріїв або наукових експериментів, що допомагає студентам краще зрозуміти і запам'ятати інформацію.

– Можливість робити вибір та впливати на сюжет візуальної новели робить навчання більш захоплюючим та цікавим, що в свою чергу підвищує залученість студентів.

– Візуальні новели можна використовувати у різних форматах, таких як навчальні ігри, інструкції або навіть мобільні додатки, що дозволяє адаптувати їх до конкретних навчальних потреб.

– Візуальні новели можуть надавати студентам можливість відчути навчальний матеріал на особистому рівні, що сприяє кращому засвоєнню інформації.

Одним з яскравих прикладів освітніх візуальних новел є гра Attentat 1942 (рис. 1.3). Це історично точна пригодницька гра, розказана очима людей, які пережили Другу світову війну. В цій грі на вас чекають багато моральних дилем і екзистенційних переживань на шляху до розкриття складної історії вашої родини. Сценарій написаний і розроблений професійними істориками.

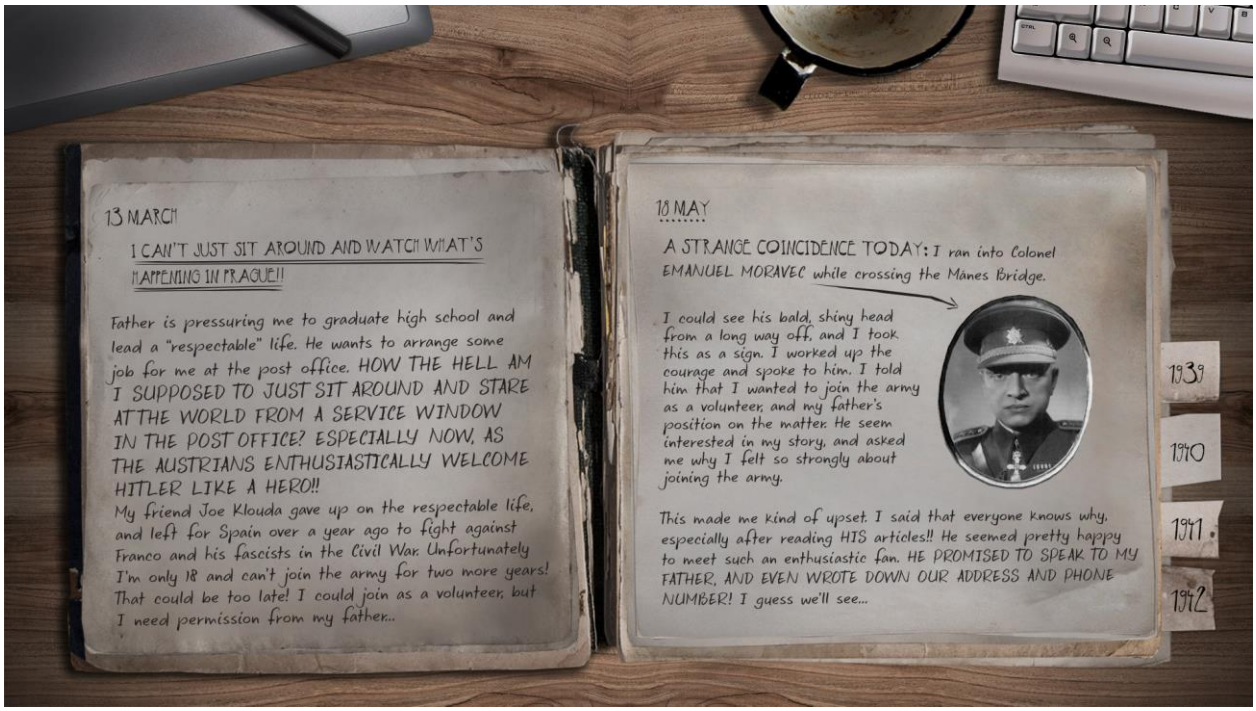

Рисунок 1.3 – Attentat 1942

За сюжетом гри 1942 рік. Рейнхард Гайдрих, правитель нацистсько-окупованих чеських земель, призначений Гітлером, був убитий. Відплата нацистів була жорстокою. Ваш дідусь став однією з жертв і був відправлений

до концентраційного табору. Але чому? Яку роль він зіграв у нападі? Чому він не розповів про це своїй родині? Чи був він хоробрим, чи необачним?

Інтерактивна історія "Attentat 1942" пропонує:

- глибокі розгалужені діалоги, де ваш вибір має значення;

- інтерактивні комікси та спогади, переховуйте листівки опору від гестапо або втікайте з нацистської в'язниці;

- рідкісні оцифровані кіноматеріали та історичні предмети.

У зв'язку з тим, що історія Другої світової війни є болісною для багатьох народів, цим подіям присвячено і багато інших ігор. Наприклад, Venti Mesi (рис. 1.4).

Рисунок 1.4 – Venti Mesi

Ця гра, – це збірка історій про італійський опір та визволення від фашизму. 20 доленосних місяців (з вересня 1943 року по квітень 1945 року), 20 історій, заснованих на реальних подіях Другої світової війни в Міланській агломерації (Сесто-Сан-Джованні та його околиці), 20 різних точок зору на історію італійської демократії.

Іншим прикладом навчальної візуальної новели є гра «Grey Plague» (рис. 1.5).

Рисунок 1.5 – Grey Plague

Grey Plague в ігровій формі знайомить гравця з біологією, анатомією, причинами виникнення деяких хвороб.

Інша історична гра – Golden Threads – знайомить нас з історією еміграції китайців до Нової Зеландії (рис. 1.6).

За сюжетом – середина 1800-х років. Ви зростаєте в селі на півдні Китаю. Але пригоди ваблять вас за кордон, і ви можете повернутися багатим – або й ні.

Гра натхнена реальними історичними постатями та подіями, досліджує та святкує 175 років життя китайців у Новій Зеландії. Схожа на щоденник «втрачених і знайдених речей». Як гравець ви приймаєте ключові змістовні життєві рішення, поринаєте у життя китайських мігрантів 19-го століття до Нової Зеландії та створюєте власну історію.

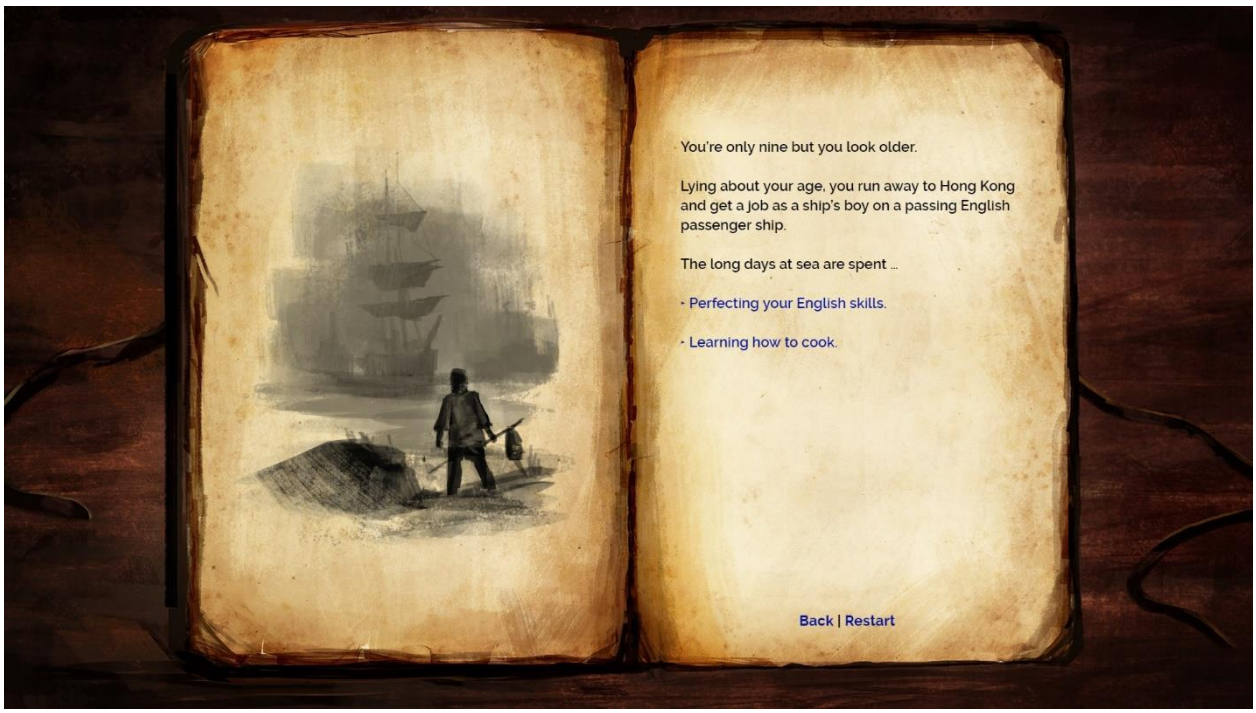

Рисунок 1.6 – Golden Threads

Проте, не лише історію вивчають за допомогою візуальних новел. Є доволі оригінальна гра, що навчає, наприклад, хімії. Це – AP Chemistry (рис. 1.7).

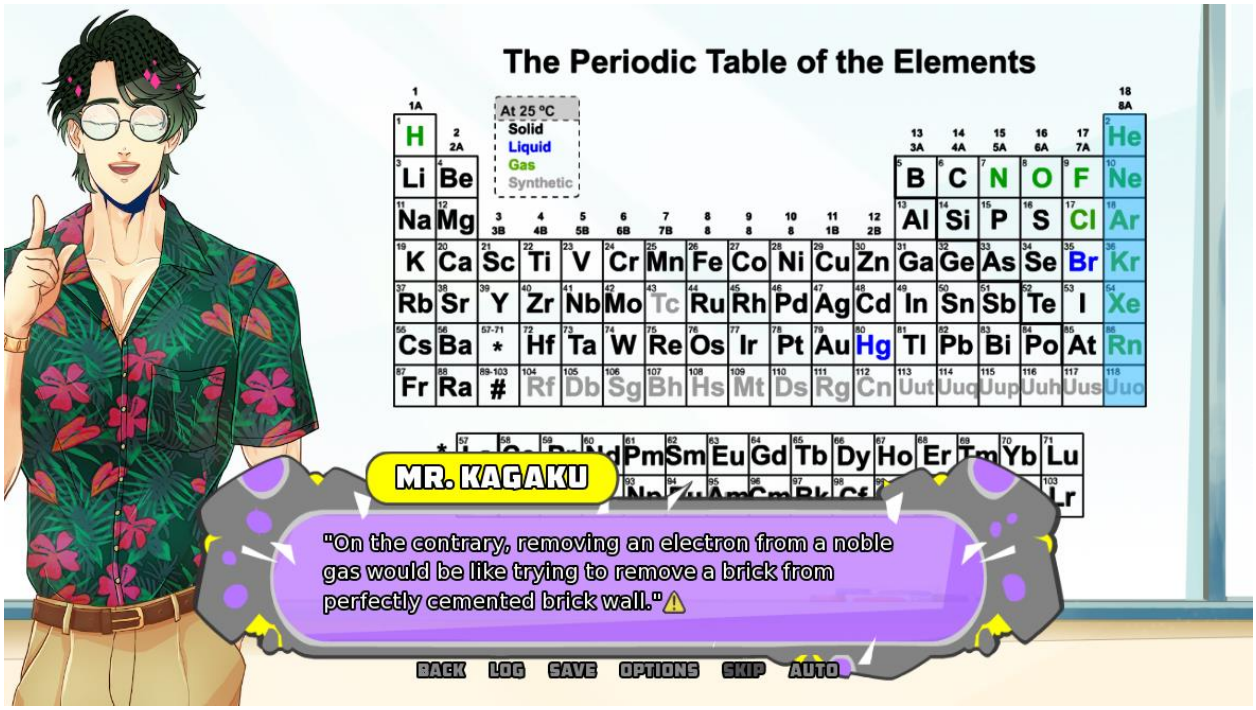

Рисунок 1.7 – AP Chemistry

Таким чином ми бачимо, що комп’ютерні ігри в жанрі візуальної новели доволі часто використовуються для навчання та висвітлення історичних подій. Тож саме такий жанр є найбільш підходящим для нашої задачі.

Тепер нам необхідно детальніше спроєктувати сюжет гри, та продумати персонажів.

1.3 Проєктування сюжетної структури візуальної новели

Спроєктуємо сюжетну структуру гри відповідно до хронології подій конференції 1968 року та основних її задач.

1. Вступ/Акт 1: Криза.

– Гравець знайомиться зі світом розробки програмного забезпечення до 1968 року.

– Представлення проблем масштабу, надійності, оцінок, хаосу в проектах.

– Отримання запрошення або можливість відвідати першу конференцію НАТО з програмної інженерії.

– Визнання терміну "програмна інженерія" та мети конференції.

2. Акт 2: Виклики та Обговорення

– Гравець "відвідує" різні секції конференції або обговорює проблеми з різними експертами/персонажами.

– Обговорення проблем Проектування (відсутність дисципліни, складності, послідовності, інтеграції з апаратним забезпеченням).

– Обговорення проблем Виробництва (управління, кадри, інструменти, оцінки).

– Обговорення проблем Обслуговування (документація, підтримка, розповсюдження, тестування).

– Обговорення проблем Освіти та кадрів.

– Можливість вибору, на якій проблемі зосередитися або до якої робочої групи приєднатися (Проектування, Виробництво, Обслуговування).

3. Акт 3: Пошук Рішень та Майбутнє

– Гравець досліджує запропоновані рішення та методології, представлені різними персонажами.

– Вивчення ідей ієрархічного проектування та рівнів абстракції (Дікстра, Ренделл).

–◦Ознайомлення з концепцією програмних компонентів та індустрії компонентів (Макілрой).

–◦Обговорення важливості симуляції та тестування (Грехем, Перліс).

– Розгляд практик управління, використання інструментів, документації (Бемер, Селіг).

–◦Вибір гравцем певного підходу або комбінації підходів для вирішення проблем.

–◦Завершення може показувати, як обрані підходи впливають на успіх проекту гравця або майбутнє програмної інженерії.

Основними рушіями сюжету в візуальних новелах є: використання діалогів, внутрішніх монологів головного героя, візуальних сцен (наприклад, що ілюструють зростання складності коду, роботу команди, взаємодію з машиною, конференц-зал) та можливостей вибору дій або відповідей гравця.

Ключовими дійовими особами, з якими буде взаємодіяти герой гри, є учасники конференції, що обговорювали базові засади інженерії програмного забезпечення:

д’Агапеєфф (d’Agapeyeff): Активний учасник дискусій, що висловлює думки про специфікації, дизайн, управління проєктами, ціноутворення програмного забезпечення та масове виробництво компонентів.

Бабкок (Babcock): Учасник дискусій, що висловлює занепокоєння щодо розподілу відповідальності за технічне обслуговування між користувачем та виробником, а також про адекватну перевірку систем перед випуском.

Бартон (Barton): Учасник дискусій, що критикує ідею масового виробництва компонентів програмного забезпечення, наголошуючи на важливості мислення про представлення даних, а не тільки про написання коду.

Бауер (Bauer): Учасник дискусій, що коментує термінологію та концепції програмної інженерії.

Бемер, Р. В. (Bemer, R.W.): Співавтор робочого документа "Класифікація предметної області" (Classification of subject matter) та автор "Контрольного списку для планування виробництва програмних систем" (Checklist for planning software system production). Активний учасник дискусій.

Бакстон (Buxton): Учасник дискусій, що коментує різні аспекти програмної інженерії, включаючи зростання програмного забезпечення та документацію.

Девід, Д. (David, D.): Автор робочого документа "Деякі думки про виробництво великих програмних систем" (Some thoughts about production of large software systems). Наголошує на стратегії побудови на підсистемі та дизайні до кодування. Активний учасник дискусій.

Дейкстра, Е. В. (Dijkstra, E.W.): Автор робочого документа "Контроль складності через ієрархічне впорядкування функцій та змінних" (Complexity controlled by hierarchical ordering of function and variability). Відомий своїми роботами з методології програмування та структурного програмування. Активний учасник дискусій.

Ендрес, А. (Endres, A.): Учасник дискусій, що висловлює застереження щодо ідеї масового виробництва компонентів.

Фрейзер, А. Дж. (Fraser, A.G.): Співавтор робочого документа "Класифікація предметної області" (Classification of subject matter). Учасник дискусій, що коментує різні аспекти дизайну та виробництва програмного забезпечення.

Галлер (Galler): Учасник дискусій, що коментує випуск та розповсюдження програмних систем, ціноутворення та стандарти.

Геню (Genuys): Учасник дискусій, що коментує термінологію та необхідність попередніх версій систем для навчання персоналу.

Гілл, С. (Gill, S.): Автор робочого документа "Думки про послідовність написання програмного забезпечення" (Thoughts on the sequence of writing

software). Учасник дискусій, що коментує різні аспекти розробки та тестування.

Жиллет, Дж. (Gillette, J.): Автор робочого документа "Засоби допомоги у виробництві підтримуваного програмного забезпечення" (Aids in the production of maintainable software). Наголошує на модульності, специфікації та загальності. Учасник дискусій.

Гленні, А. Е. (Glennie, A.E.): Співавтор робочого документа "Класифікація предметної області" (Classification of subject matter). Учасник дискусій.

Грехем, Р. М. (Graham, R.M.): Учасник дискусій, що коментує використання високорівневих мов для дизайну систем та ідею каталогу компонентів.

Харр, Дж. (Harr, J.): Автор графіків та даних про дизайн та виробництво програмного забезпечення для електронних комутаційних систем.

Кьолер, Х. (Köhler, H.): Автор робочого документа про технічне обслуговування та розповсюдження програм.

Коленс (Kolence): Учасник дискусій, що коментує вартість, оцінки та системну оцінку.

Летелієр, Д. (Letellier, D.): Автор робочого документа про тестування та дизайн програмних пакетів.

Ллевелін, А. І. (Llewelyn, A.I.): Співавтор робочого документа про тестування комп'ютерного програмного забезпечення.

МакІлрой, М. Д. (McIlroy, M.D.): Доповідач, що представляє доповідь про "Масове виробництво компонентів програмного забезпечення". Активний учасник дискусій.

Нэш, Н. (Nash, N.): Автор даних та ілюстрацій з робочого документа "Деякі проблеми у виробництві великомасштабних програмних систем".

Нейр (Naur): Учасник дискусій, що коментує дизайн, специфікації та документацію.

Оплер, А. (Opler, A.): Співавтор робочого документа "Класифікація предметної області" (Classification of subject matter). Активний учасник дискусій.

Пол (Paul): Учасник дискусій, що коментує докази правильності алгоритмів.

Перліс, А. Й. (Perlis, A.J.): Доповідач, що виступає з основною доповіддю на конференції. Активний учасник дискусій, коментує різні аспекти програмної інженерії, освіту та ідею компонентів.

Пінкертон, Т. Б. (Pinkerton, T.B.): Автор робочого документа про моніторинг продуктивності та оцінку систем.

Ван дер Поель (Van der Poel): Автор робочого документа, де представлено невелику програму асемблера.

Ренделл, Б. (Randell, B.): Співавтор робочого документа "До методології проектування комп'ютерних систем" (Towards a methodology of computing system design). Активний учасник дискусій, що коментує підходи "зверху-вниз" та "знизу-вгору", а також термін "програмна інженерія".

Росс (Ross): Учасник дискусій, що коментує ідею інтегрованих пакетів та автоматизованої програмної технології, зокрема у банківській сфері.

Шольтен, К. С. (Scholten, C.S.): Співпрацював з Е. В. Дейкстрою над розробкою примітивів секвенування.

Селіг, Ф. (Selig, F.): Автор робочого документа про стандарти документації.

Сміт (Smith): Учасник дискусій, що коментує розпливчасті терміни в дизайні програмного забезпечення та відсутність дисципліни.

Вікенс, Р. Ф. (Wickens, R.F.): Співавтор робочого документа про тестування комп'ютерного програмного забезпечення. ([216], стор. 189)

Віле, Х. (Wiehle, H.): Співавтор робочого документа "Класифікація предметної області" (Classification of subject matter).

Цюрхер (Zurcher): Співавтор робочого документа, де використовується термін "рівень абстракції".

Окресливши основний сюжет та персонажів гри, ми можемо перейти до детального етапу розробки, який включає проєктування програмного забезпечення, створення основних алгоритмів та дизайн.

Дизайну гри приділяється особлива увага, адже він формує естетику світу, атмосферу та стиль. Тут важливо врахувати тематику сюжету, характер персонажів, а також створити детально продуманий ігровий простір. Дизайн може містити розробку локацій, дизайн інтерфейсу користувача (UI/UX), анімації та візуальні ефекти. Крім того, варто подбати про звукові ефекти та музичний супровід, які доповнюють глибину емоційного занурення в гру.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ ГРИ З ІСТОРІЇ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування сюжетної структури гри

Проєктування програмного забезпечення комп’ютерної гри з історії інженерії програмного забезпечення ми почнемо з розробки загальної сюжетної структури.

Ми вже визначили, що в сюжеті нашої гри буде три акти:

1. Акт 1 – Вступ. Введення в проблематику розробки програмного забезпечення, визначення основних проявів кризи програмного забезпечення.

2. Акт 2 – Участь в конференції 1968 року. Спілкування з учасниками. Обговорення шляхів подолання кризи програмного забезпечення. Окреслення основних засад та задач інженерії програмного забезпечення.

3. Акт 3 – Імплементація здобутих знань. Гравець намагається в своїй компанії запровадити інженерні засади в розробці програмного забезпечення.

Відповідно до цієї загальної структури, ми спроєктували детальну структуру (рис. 2.1) з деталізацію сцен та локацій, де будуть відбуватись події гри.

Відповідно до цієї схеми, події гри починаються у вигаданій американській компанії «Software Services». Наш герой – провідний програміст компанії. Його команда працює над великим програмним проєктом, але стикається з численими труднощами:

– постійні зміни в коді та архітектурі призводять до того, що терміни завершення проєкту регулярно відкладаються;

– через необхідність пришвидшити проєкт, залучено більше програмістів. Це збільшило кошторис проєкту, але не пришвидшило його;

– постійно в коді виявляються числені помилки, через що доводиться переробляти компоненти знову і знову, а ці переробки порушують цілісність системи.

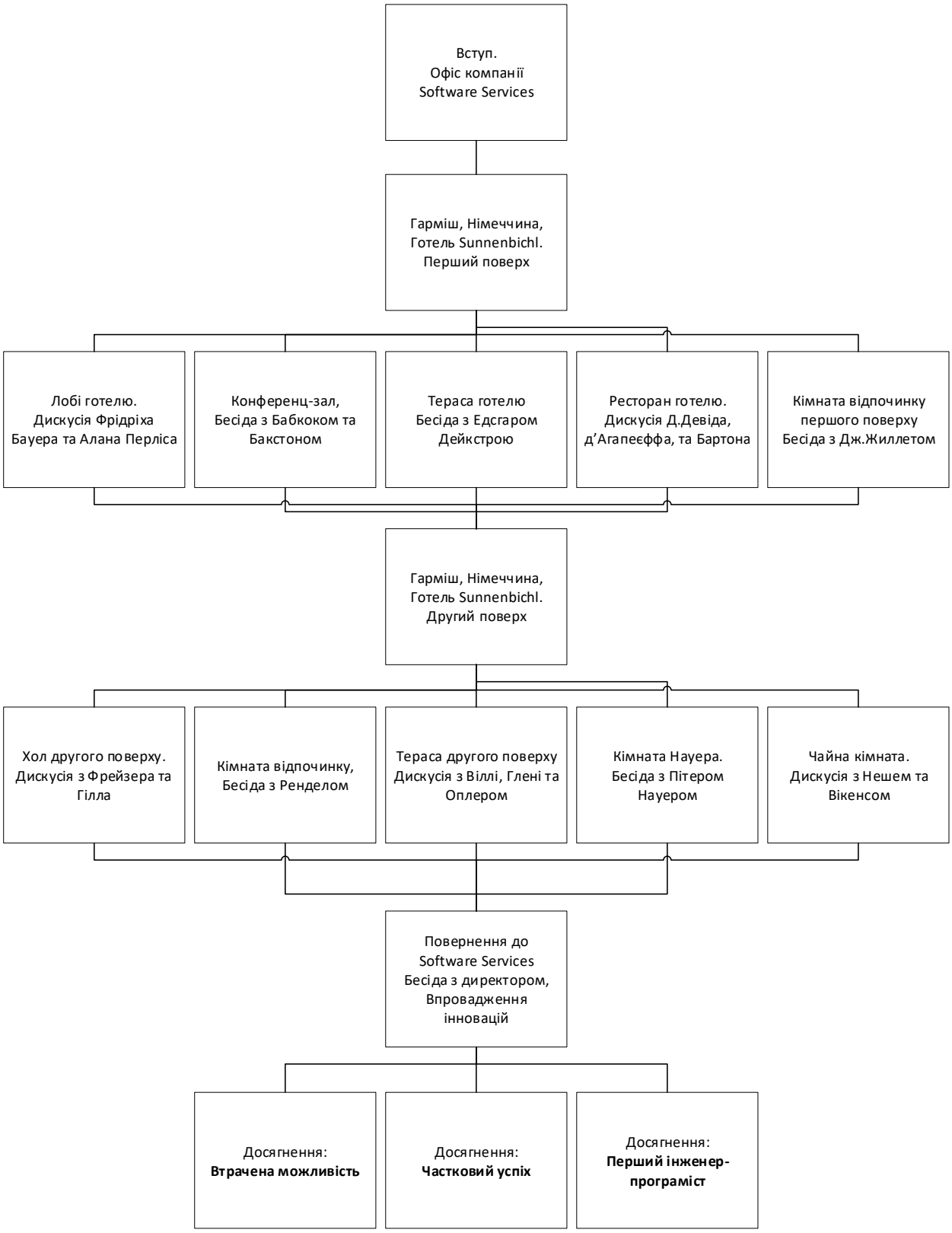

Рисунок 2.1 – Сюжетна структура гри

Гра починається з зустрічі головного героя та директора компанії, під час якої вони обговорюють всі проблеми, що виникли під час роботи. Доходять висновку, що основною причиною проблем є відсутність системного підходу до розробки ПЗ, як це, наприклад, відбувається в машинобудуванні або будівництві, де добре працюють інженерні підходи.

Несподівано директор згадує, що отримав листа з інформацією про те, що в Європі, в Німеччині, місті Гарміш має відбутись велика конференція під егідою НАТО, що має провокативну назву «Інженерія програмного забезпечення». Директор приймає рішення відрядити головного героя на цю конференцію, щоб він особисто дізнався, як учасники конференції

В наступній сцені дії переносяться в місто Гарміш, де герой знаходить готель Sonnenbichl, в якому і проходить конференція Software Engineering.

В цьому готелі герой може відвідувати такі локації, як лобі готелю, терасу, конференц-зал, ресторан, кімнату відпочинку та інші. В цих локаціях герой зустрічає різних учасників конференції, з якими може поспілкуватись, подискутувати. Для зручності гри локації розділені по поверхах готелю.

Після того, як герой поспілкувався з усіма учасниками, провів дискусії та дебати, він може повернутись в рідну компанію.

Опинившись знову в компанії Software Services герой разом з директором обговорюють зміни, що потрібно запровадити для імплементації інженерних підходів в розробку програмного забезпечення.

В цьому діалозі гравцю потрібно обирати правильні варіанти відповідей з запропонованих дій. За кожен правильний вибір гравцю нараховуються бали. Всього можна набрати під час відповідей 10 балів. В залежності від того, скільки балів набрав гравець, він отримує одне з можливих досягнень та завершень гри:

10-9 балів – Досягнення «Перший інженер-програміст». Гравець правильно або майже правильно імплементує інженерні засади в роботу програмістів своєї компанії.

5-7 балів – Досягнення «Частковий успіх». Гравець змінює більшість підходів до розробки програм, але не досягає повноцінного успіху через половинчастість дій.

0-4 бали – Досягнення «Втрачена можливість». Гравець був неуважним на конференції і не зміг імплементувати інженерні засади в роботу своєї компанії.

Таким чином останній акт виконує фактично роль контроля знань про основні інженерні принципи в розробці програмного забезпечення, а також дає гравцю зворотний зв'язок – чи зрозумів він основну проблематику, що обговорювалась на конференції 1968 року та основні принципи, що покладено в інженерію програмного забезпечення.

2.2 Проєктування основних алгоритмів гри

З алгоритмічної точки зору найбільш цікавими механіками гри є організовані в ній діалоги. В цілому діалоги можна розділити на три групи:

– звичайні діалоги, в яких присутні вибори реплік, що впливають на певну відповідь опонента;

– розширені діалоги, під час яких важливо задати всі питання зі списку та отримати всі відповіді;

– діалоги з оцінкою, під час яких гравець набирає бали за правильні відповіді.

Блок схема звичайного діалогу показана на рисунку 2.2. В такому діалозі гравець має обрати один з варіантів репліки, на що отримає ту чи іншу відповідь опонента. В такому діалозі може бути передбачений додатковий функціонал, наприклад, активація того чи іншого логічного прапору, що вплине на подальшу гру. Але до самого діалогу гра вже не повертається.

Наступним прикладом є розширені діалоги, блок-схема якого показана на рисунку 2.3.

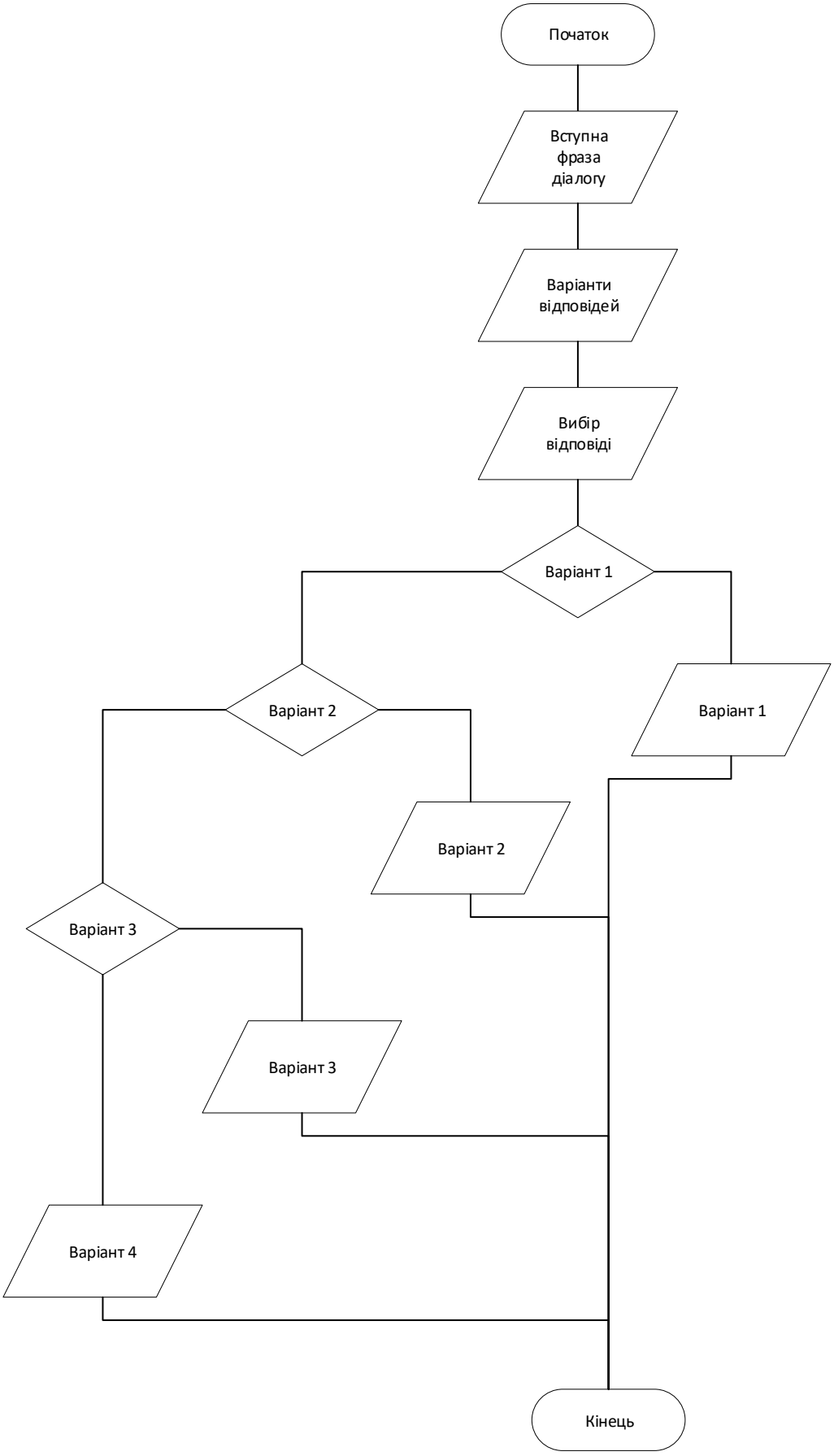

Рисунок 2.2 – Блок-схема алгоритму простого діалогу

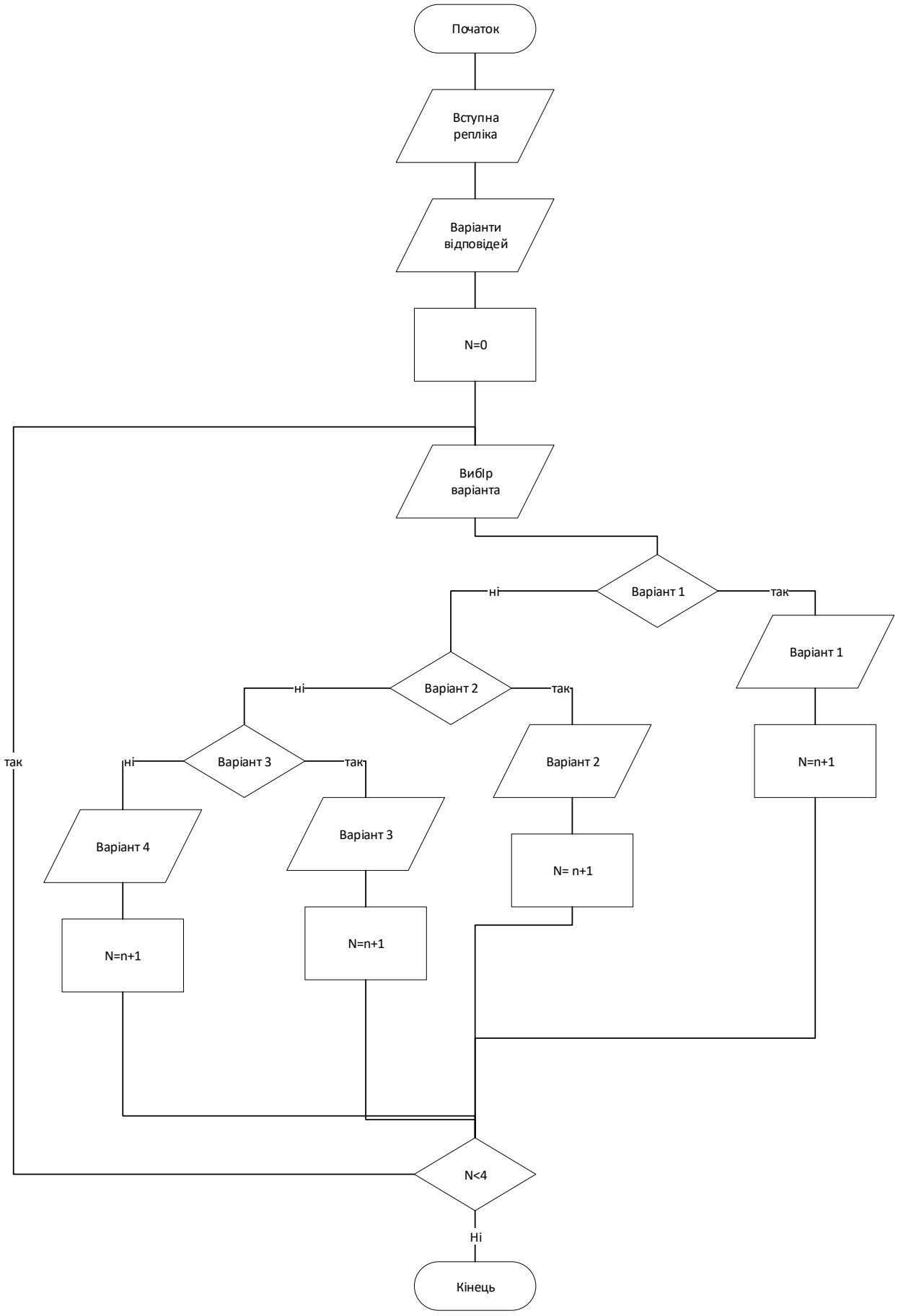

Рисунок 2.3 – Блок-схема розширеного діалогу

Такий діалог має певний цикл. Гравець може знову і знову повертатись до списку питань, щоб опитати опонента. Передбачено підрахунок циклів питань-відповідей. По завершенню циклу, коли вже всі питання задані, діалог завершується і гра продовжується далі.

Блок-схема діалогу з оцінками представлена на рисунку 2.4.

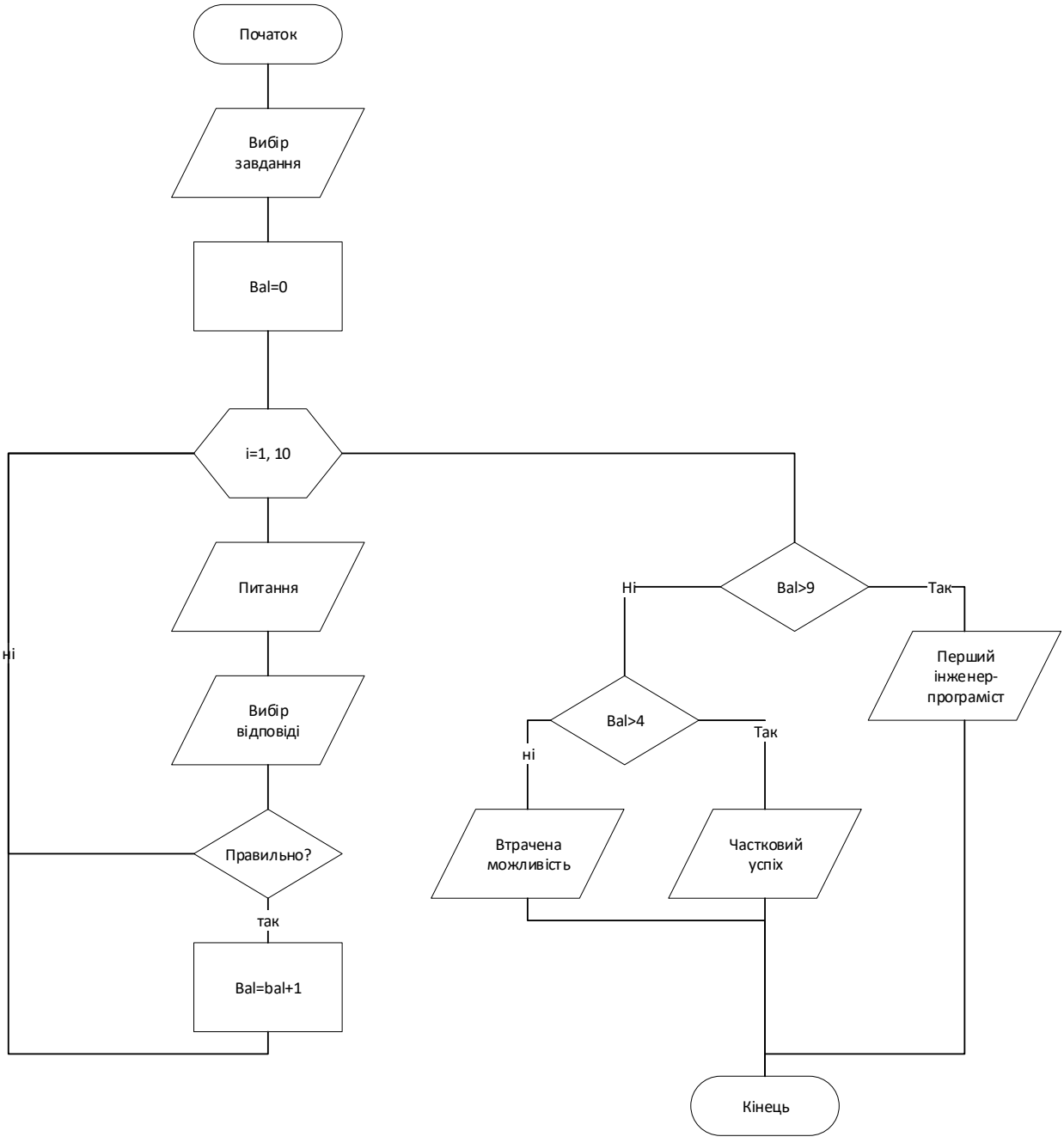

Рисунок 2.4 – Блок-схема алгоритму фінального рішення

В такому діалозі гравець має обирати з низки відповідей правильну відповідь на поставлене питання. За правильну відповідь гравцю мають

нараховуватись бали. В залежності від набраних балів гравець після такого діалогу отримує той чи інший результат.

На рисунку 2.4 показано приклад такого діалогу для фінального рішення, що приймає гравець, обираючи які зміни він має імплементувати для своєї команди, на основі отриманих під час гри знань. В результаті такого діалогу в залежності від набраних балів, можна відкрити одне з досягнень гри:

– перший інженер програміст;

– частковий успіх;

– втрачена можливість.

2.3 Проєктування діаграми використання

Окрім самого геймплею, для гри надзвичайно важливим елементом є інтерфейс оболонки, який забезпечує взаємодію гравця з грою, створюючи зручне та інтуїтивно зрозуміле середовище. Інтерфейс оболонки дозволяє виконувати базові функції, як-от запуск гри, управління її параметрами, а також забезпечує доступ до широкого спектра налаштувань і опцій, які підлаштовують користувацький досвід під індивідуальні потреби. З його допомогою гравець може створювати збереження прогресу, завантажувати їх у будь-який потрібний момент, змінювати візуальні та звукові налаштування, налаштовувати управління або мову інтерфейсу, переглядати статистику гри та навіть взаємодіяти з додатковими режимами. У сучасних іграх якісний інтерфейс оболонки є одним із ключових чинників, що впливають на загальне враження від гри, оскільки він сприяє її доступності, привабливості та зручності використання для всіх категорій гравців, незалежно від їхнього досвіду.

Таким чином необхідно приділити увагу проєктуванню такої оболонки. Проєктування почнемо з проєктування діаграми використання (рис. 2.5).

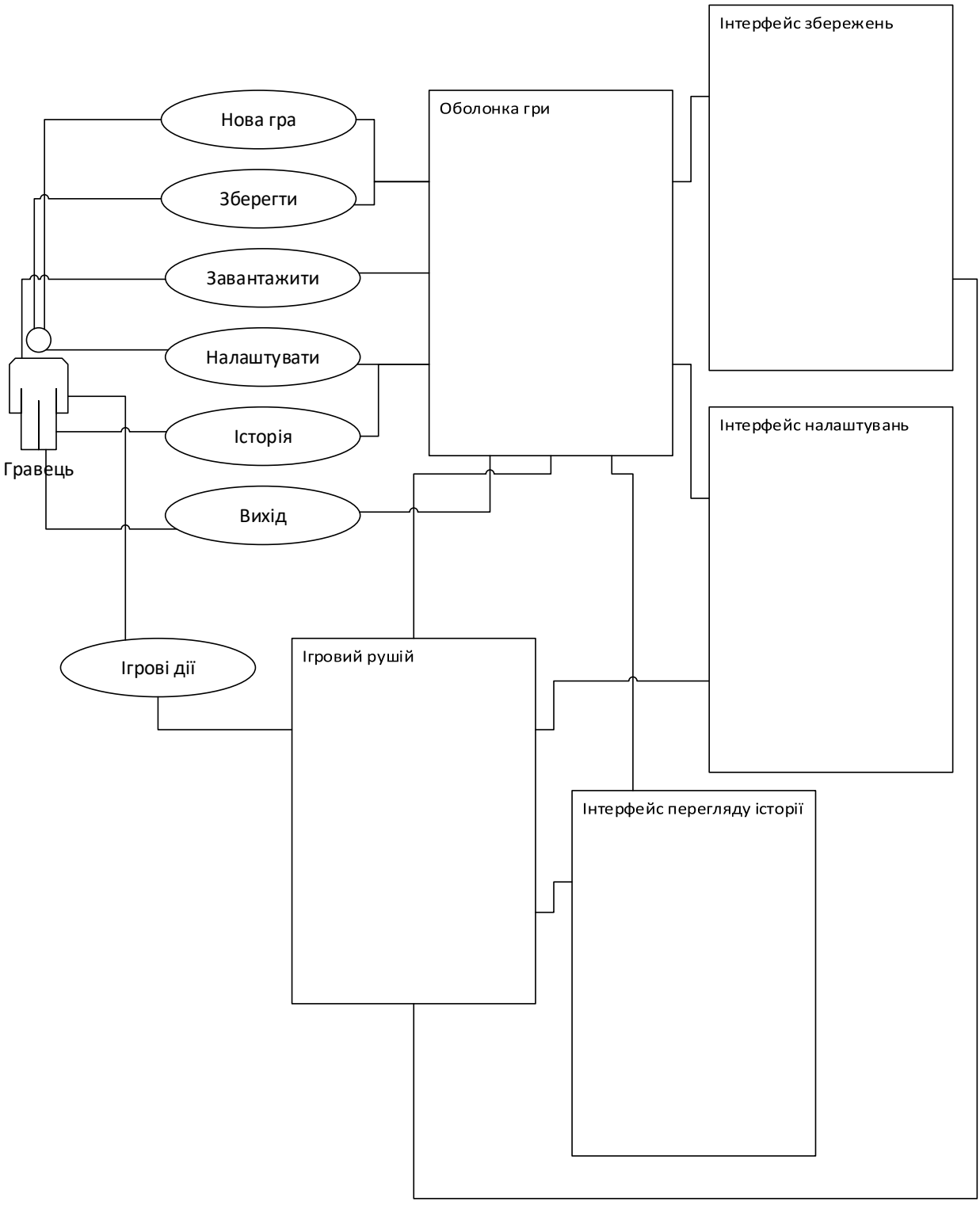

Рисунок 2.5 – Діаграма використання ігрової оболонки

На цій діаграмі представлено основні підсистеми нашої гри:

– Інтерфейсна оболонка, що надає гравцю доступ до всіх інших підсистем;

– Інтерфейс збережень, через який гравець може робити збереження поточного стану гри або завантажувати попередні збереження;

– Інтерфейс налаштувань, в якому гравець може змінювати налаштування гри та певні параметри;

– Інтерфейс історії гри, в якій гравець може переглянути всі пройдені діалоги, обрані варіанти відповідей та репліки інших персонажів. Фактично, така історія складається в окреме оповідання, як «пише» сам гравець;

– Ігровий рушій – підсистема, в якій відбувається сама гра, виконуються ігрові дії, відображаються сцени та персонажі.

Завершивши проєктування ми можемо перейти до безпосередньої розробки гри.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ ГРИ З ІСТОРІЇ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Добір засобів розробки комп'ютерної гри з історії інженерії програмного забезпечення

У зв’язку з тим, що візуальні новели наразі стали доволі популярним жанром сюжетних комп’ютерних ігор, розробники мають доступ до широкого спектру загальнодоступних рушіїв для їх створення. Ці рушії надають всі необхідні інструменти для формування інтерактивного ігрового досвіду: можливість створення діалогів, побудова нелінійного сюжету, інтеграція високоякісного візуального контенту, а також реалізація системи виборів, що впливають на розвиток історії.

Серед найвідоміших рушіїв можна виділити такі платформи, як Ren'Py, Unity з відповідними пакетами для створення новел, TyranoBuilder, Visual Novel Maker і навіть Godot. Кожен з цих інструментів має свої особливості, можливості та рівень складності, що дозволяє розробникам обрати оптимальний варіант залежно від їхніх навичок та потреб проекту. Завдяки простоті використання, доступності та активній спільноті розробників, створення візуальних новел стає можливим як для професійних студій, так і для незалежних авторів, що сприяє розширенню жанру та розмаїттю історій, представлених у цифровому форматі.

TyranoBuilder — це універсальний і зручний у використанні інструмент для розробки візуальних новел, який дозволяє творцям будь-якого рівня майстерності проєктувати та публікувати захоплюючі, сюжетно орієнтовані ігри (рис. 3.1). Розроблений з акцентом на доступність, TyranoBuilder поєднує у собі інтерфейс перетягування елементів (drag-and-drop) із набором потужних функцій, що робить його чудовим вибором як для новачків, так і для досвідчених розробників у жанрі візуальних новел.

Рисунок 3.1 – TyranoBuilder

Програмне забезпечення дозволяє користувачам створювати інтерактивні історії шляхом упорядкування діалогів, виразів персонажів, фонів, звуків та інших елементів у простий спосіб. Його інтуїтивний інтерфейс мінімізує потребу у складному кодуванні, водночас забезпечуючи гнучкість для тих, хто бажає застосувати більш просунуте програмування. TyranoBuilder підтримує скрипти на TyranoScript та JavaScript, що дозволяє технічно грамотним користувачам налаштовувати проєкти відповідно до їхніх творчих задумів.

Однією з визначних характеристик TyranoBuilder є його сумісність із різними платформами. Ігри, створені за допомогою цього інструменту, можна експортувати на платформи Windows, macOS, iOS, Android і HTML5, що дозволяє розробникам ділитися своїми роботами з широкою аудиторією. Ця функція особливо цінна для незалежних творців, які прагнуть максимізувати охоплення своїх проєктів без необхідності долати складнощі розробки для конкретних платформ.

Інструмент постачається з попередньо завантаженими зразками ресурсів, такими як спрайти персонажів, фони, звукові ефекти та музика, що допомагає творцям швидко розпочати свої проєкти. Також підтримується імпорт власних матеріалів, що дає розробникам повний контроль над естетикою та сюжетом їхніх ігор. Незалежно від того, чи використовуються

аніме-стиль, піксель-арт або фотореалістична графіка, TyranoBuilder забезпечує зручну інтеграцію цих елементів.

TyranoBuilder також відзначається легкістю створення розгалужених сюжетних ліній — ключової особливості візуальних новел. Письменники можуть без зусиль розробляти складні дерева рішень та альтернативні фінали, збагачуючи досвід гравця шляхом пропонування різних шляхів і результатів. Вбудована функція попереднього перегляду дозволяє легко тестувати та усувати помилки в цих точках вибору, забезпечуючи плавний розвиток сюжету.

Крім того, програмне забезпечення включає такі функції, як налаштовані текстові вікна, анімовані рухи персонажів, синхронізація з озвученням і спеціальні ефекти для покращення розповіді. Розробники можуть інтегрувати музичні плейлисти, навколишні звуки та динамічні переходи для повного занурення гравців у атмосферу гри.

Для розробників, які прагнуть підтримки від спільноти, TyranoBuilder пропонує ресурси, такі як навчальні посібники, форуми та керівництва користувача для оптимізації проєктів і вирішення проблем. Його доступна вартість робить програму зручною для широкого кола користувачів — від аматорів до майбутніх професіоналів.

У підсумку, TyranoBuilder виділяється як один із найбільш доступних і водночас потужних інструментів для створення візуальних новел, що дозволяє розповідачам зосередитися на своїх історіях, надаючи при цьому професійний досвід розробки ігор. Незалежно від того, чи створюєте ви романтичну історію, захопливий детектив чи драму, TyranoBuilder має всі інструменти, необхідні для втілення вашої історії в життя.

Visual Novel Maker — це універсальне й потужне програмне забезпечення, спеціально розроблене для створення візуальних новел — інтерактивних, сюжетно-орієнтованих ігор, які часто вирізняються глибокими наративами, варіативними виборами та вражаючим художнім оформленням. Цей інструмент, випущений компанією Degica Games, слугує доступною, але

водночас функціонально багатою платформою як для початківців, так і для професійних розробників, які прагнуть втілити свої творчі ідеї в життя.

Рисунок 3.2 – Visual Novel Maker

Програма пропонує зручний інтерфейс, що спрощує процес розробки гри, водночас забезпечуючи широкий спектр розширених функцій для налаштування. Підтримка функції "перетягування" (drag-and-drop) дозволяє легко розміщувати сцени та персонажів, завдяки чому творці можуть більше зосереджуватися на сюжеті, а не на технічних аспектах. Visual Novel Maker включає вбудовані можливості сценарного програмування на JavaScript для тих, хто хоче реалізувати унікальні механіки або розширити функціонал двигуна понад стандартні можливості. Це робить програму придатною як для новачків, так і для досвідчених програмістів.

Однією з ключових особливостей Visual Novel Maker є його бібліотека ресурсів, яка включає широкий спектр спрайтів персонажів, фонів, музичних композицій, звукових ефектів і елементів графічного інтерфейсу. Ці ресурси забезпечують швидкий старт для новачків, які можуть одразу приступити до створення проєктів без потреби у зовнішніх матеріалах. Для тих, хто прагне додати індивідуальності, програма дозволяє інтегрувати власні художні роботи, анімації та аудіофайли, надаючи творцям повний контроль над естетикою і атмосферою їхньої гри.

Ще однією важливою особливістю Visual Novel Maker є підтримка розгалужень сюжету та прийняття рішень, які є невід’ємною частиною

ігрового процесу візуальних новел. Розробники можуть створювати складні історії з кількома шляхами розвитку та кінцівками за допомогою інструментів візуалізації, що спрощують побудову наративних структур і дозволяють легко відслідковувати, як вибори персонажів впливають на хід сюжету. Це забезпечує захопливий та багаторазово відтворюваний ігровий досвід.

До того ж, Visual Novel Maker підтримує мультиплатформенний випуск. Ігри, створені за допомогою цього програмного забезпечення, можна експортувати у формати для ПК (Windows, macOS та Linux), вебу (HTML5) і мобільних платформ (iOS та Android). Ця гнучкість дозволяє розробникам досягати широкої аудиторії, незалежно від її вподобаного ігрового пристрою.

Для тих, хто зацікавлений у монетизації своїх творінь, Visual Novel Maker включає інструменти для локалізації, що полегшує переклад ігор різними мовами і сприяє залученню глобальної аудиторії. Крім того, підтримка Steam Workshop дає розробникам змогу ділитися своїми роботами, спілкуватися з творчою спільнотою та обмінюватися ресурсами чи ідеями.

Також Visual Novel Maker має активну спільноту користувачів та ентузіастів, які регулярно діляться навчальними матеріалами, інструкціями та ресурсами. Існує безліч форумів й онлайн-спільнот, що допомагають розробникам вирішувати складнощі чи знаходити натхнення для спільної творчості.

Підсумовуючи, Visual Novel Maker є комплексним рішенням для створення візуально привабливих, сюжетно орієнтованих ігор. Завдяки поєднанню інтуїтивного дизайну, можливостей налаштування та підтримки різних платформ, це популярний вибір серед авторів, що прагнуть втілити свої творчі задуми і створювати захопливі пригоди у світі візуальних новел та інтерактивної літератури.

Godot, офіційно відомий як Godot Engine, є могутньою та універсальною платформою для розробки ігор з відкритим доступом, створеною для розробки як 2D, так і 3D ігор. Випущений під ліцензією MIT, він набув широкої популярності серед розробників завдяки своїй гнучкості, простоті

використання та повній відсутності роялті. Спочатку розроблений Хуаном Лінієтським і Аріелем Манзуром у 2014 році, цей рушій перетворився на потужний інструмент із активною спільнотою, яка активно сприяє його розвитку та документації.

Рисунок 3.3 – Godot

Godot пропонує унікальний, інтуїтивний підхід до розробки ігор. Його архітектура, заснована на вузлах, є ключовою для створення проєктів, дозволяючи розробникам структурувати сцени та логіку гри у форматі ієрархії. Вузли можуть представляти об'єкти, джерела світла, звуки, фізичні сутності, скрипти або інтерфейси, і розробники можуть комбінувати вузли для створення складної поведінки. Така модульність робить Godot особливо зручним для початківців, водночас надаючи достатньо потужності для професійного використання.

Система сцен є однією з найбільш видатних особливостей Godot. Сцена може представляти повний рівень гри, персонажа або навіть багаторазовий компонент, такий як меню. Сцени можуть складатися з численних вузлів і бути вкладеними для створення складних дизайнів. Така філософія дизайну

дозволяє розробникам вибудовувати рівні та механіку гри в зрозумілий і масштабований спосіб.

Godot підтримує кілька мов програмування, але його рідна мова GDScript є найбільш поширеною. GDScript є легкою, схожою на Python мовою, оптимізованою для розробки ігор безпосередньо в рушії. Godot також підтримує C#, VisualScript (візуальну мову програмування на основі вузлів) і C++ для тих, хто потребує високорівневої кастомізації або оптимізації продуктивності.

Godot відмінно працює як з 2D, так і з 3D розробкою ігор. У сфері 2D розробники отримують доступ до інструментів, таких як точне зіткнення пікселів, гнучкий редактор плиткових карт і шейдери для створення стилізованих ефектів. Для 3D ігор Godot пропонує вдосконалені функції рендерингу, включаючи глобальну освітленість, відображення в реальному часі та ефекти пост-обробки. Останні оновлення, особливо з виходом Godot 4, включають рендеринг на основі вулкану, що покращує графічну якість і продуктивність.

Godot забезпечує просте розгортання на різних платформах, що дозволяє розробникам експортувати свої ігри для таких операційних систем і платформ, як Windows, macOS, Linux, Android, iOS, HTML5, а також консолей, таких як Xbox, PlayStation і Nintendo Switch (хоча підтримка консолей часто потребує сторонніх послуг або спеціальних рішень).

Godot має у своєму складі широкі візуальні інструменти для редагування, такі як редактор сцен, редактор анімацій і графовий редактор шейдерів. Ці інструменти дозволяють розробникам швидко створювати прототипи та шліфувати свої проєкти без необхідності повністю покладатися на код, що робить його фаворитом серед інді-студій і сольних розробників.

Однією з ключових сильних сторін Godot є його жвава та постійно зростаюча спільнота. Тисячі користувачів сприяють створенню плагінів, ресурсів, навчальних посібників і виправлень, забезпечуючи безцінні ресурси як для новачків, так і для професіоналів. Його відкритий характер дозволяє

кожному покращувати або розширювати рушій, щоб відповідати конкретним потребам.

Бібліотека ресурсів Godot слугує центром для безкоштовних і платних активів, зокрема шейдерів, спрайтів, шаблонів і скриптів, скорочуючи час розробки багатьох проєктів. Крім того, форуми, сервери Discord і групи у соціальних мережах сприяють співпраці та навчанню всередині спільноти Godot.

У порівнянні з провідними рушіями галузі, такими як Unity і Unreal Engine, Godot часто цінується за свою легкість, простоту в освоєнні та відсутність дорогих ліцензійних угод. Хоч він може поступатися Unreal Engine у високоякісній графіці чи Unity у величезному ринку плагінів, Godot процвітає там, де простота й економічна ефективність є пріоритетом. Для інді-розробників, ентузіастів і викладачів Godot пропонує менш громіздкий, більш доступний спосіб входу в світ розробки ігор.

Випуск Godot 4.0 став важливим кроком вперед для рушія, запровадивши значні покращення, такі як підтримка Vulkan, підвищена продуктивність, покращені системи освітлення та переписані основні компоненти. Оновлення ліквідувало попередні обмеження, роблячи Godot життєздатним вибором навіть для проєктів AAA-рівня. Основні переписування й оптимізації дозволили розробникам використовувати рушій на новому рівні.

Godot є трансформаційним інструментом у світі розробки ігор, пропонуючи відкриту, інклюзивну і високофункціональну екосистему для творців будь-якого рівня підготовки. Чи ви створюєте свій перший платформер у стилі піксель-арт, чи розробляєте масштабну 3D RPG, Godot забезпечує інфраструктуру, підтримку та творчу свободу, необхідну для втілення вашого бачення в життя. Його філософія відкритого коду разом із інноваційним дизайном гарантують його постійний розвиток і актуальність у змінному ландшафті розробки ігор.

Ren'Py — це потужний, гнучкий і надзвичайно популярний рушій для створення візуальних новел, який використовується для створення інтерактивних розповідей, симуляцій та ігор. Він відомий своєю простотою й доступністю, що робить його чудовим вибором як для новачків, так і для досвідчених розробників, які хочуть зосередитися на проєктах, орієнтованих на розповіді, без необхідності глибокого занурення в складні вимоги до програмування. Ren'Py побудований на Python — одному з найпопулярніших і найбільш дружніх для новачків мов програмування, і об'єднує Python-скрипти з власною інтуїтивною мовою, яка спрощує створення візуальних новел.

Рисунок 3.4 – Ren’Py

Однією з ключових особливостей Ren'Py є його здатність працювати з широким спектром мультимедійних матеріалів, таких як текст, зображення, музика, звукові ефекти та відео. Ця універсальність дозволяє розробникам створювати багаті й захоплюючі досвіди, адаптовані до їх творчого бачення. Платформа підтримує функції, такі як розгалужені діалоги, вибір гравця, персоналізація персонажа та складні сюжетні лінії, які є важливими для створення захопливих історій. Завдяки широким можливостям для

інтерактивного сторітелінгу, Ren'Py забезпечує глибоке залучення гравців у взаємодію з персонажами та сюжетом, створюючи досвіди, які здаються особистими й значущими.

Ren'Py також оснащений інструментами для оптимізації процесу розробки. Його система написання діалогів проста, але потужна, дозволяючи авторам зосередитися на створенні своїх історій без заглиблення в технічні деталі. Рушій включає в себе такі функції, як переходи, анімації, а також можливість інтеграції мініігор для поліпшення різноманітності ігрового процесу. Крім того, Ren'Py підтримує локалізацію, що дозволяє розробникам легко перекладати свої ігри на кілька мов для охоплення глобальної аудиторії.

Ще одним важливим аспектом привабливості Ren'Py є його відкритий код і сильна підтримка спільноти. Завдяки відкритості рушія він є безкоштовним для використання і отримує внески від розробників з усього світу. Існує величезна кількість навчальних матеріалів, посібників та форумів, які допомагають авторам освоїти систему та вирішувати технічні проблеми, формуючи співпрацю й знижуючи бар’єр для входу новачків. Велика спільнота також ділиться бібліотеками, матеріалами та модулями, які можна легко впровадити для прискорення виробничого процесу.

Ren'Py є кросплатформним, тобто ігри, створені з його допомогою, можуть бути експортовані для роботи на різних пристроях, включаючи Windows, macOS, Linux і навіть мобільні платформи, такі як Android та iOS. Це дозволяє розробникам охоплювати широку аудиторію й забезпечувати доступність своїх ігор для користувачів різних баз.

Загалом, Ren'Py є важливим інструментом для тих, хто цікавиться створенням візуальних новел або ігор, орієнтованих на розповідь. Його простота у використанні, широкі можливості й підтримка спільноти роблять його улюбленим серед незалежних розробників і авторів, які прагнуть втілити свої ідеї в життя. Незалежно від того, чи створюєте ви романтичну візуальну новелу, інтерактивний детектив чи епізодичну симуляційну гру, Ren'Py

пропонує необхідні інструменти для перетворення вашого творчого бачення на професійний і завершений продукт.

З перерахованих рушіїв безкоштовними є лише Ren’Py та Godot. Але Godot, хоча і є дуже потужним ігровим рушієм, не орієнтований саме на візуальні новели і гра, в якій багато діалогів та тексту не буде так привабливо на ньому реалізована.

Тож, зважаючи на переваги та недоліки досліджених рушіїв ми зупинили свій вибір саме на рушії Ren’Py, як такому, що найбільш підходить для нашої задачі.

3.2 Створення ігрових сцен та персонажів

Під час створення комп'ютерної гри з історії інженерії програмного забезпечення ми намагались відтворити сцени та персонажі з максимальною історичною точністю. В архівах інтернету ми знайшли підбірку фотографій, зроблених під час конференції. Проте, ці фотографії доволі низької якості, тож безпосередньо використати їх для побудови сцен гри було неможливо. Тож, ми вирішили скористатись сучасними засобами штучного інтелекту, щоб вибудувати потрібні нам сцени, максимально наближені до автентичних. З поправкою, звичайно, на ігрову умовність.

Так в архіві було знайдено фото з фасадом готелю, в якому проходила конференція (рис. 3.5).

Пошук в інтернеті показав, що цей готель існує і досі, щоправда тепер називається Grand Hotel Sonnenbichl (рис. 3.6). Проте нам вдалось знайти його загальне фото 60-х років, яку ми стилізували та використали як одну із цен в грі (рис. 3.7).

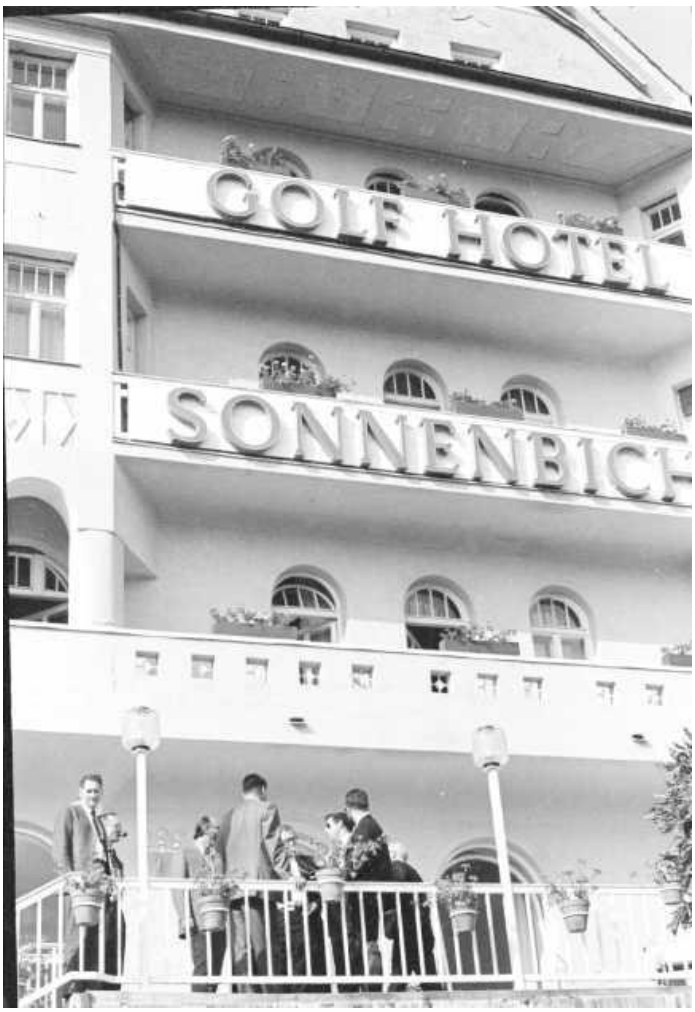

Рисунок 3.5 – Golf Hotel Sonnenbichl, 1968 р.

Рисунок 3.6 – Grand Hotel Sonnenbichl. Сучасний вигляд

Рисунок 3.7 – Стилізоване зображення готелю, що було використано в грі

Фотографій старого інтер’єру готелю теж не збереглось. Проте, в інтернеті знайшлись сучасні фотографії різних локацій, у тому числі і того конференц-залу, в якому проходила конференція 1968 року.

Тож ми взяли сучасне фото цього конференц-залу (рис. 3.8) і за допомогою генеративної моделі DALL-E створили його стилізацію під вигляд в 60-х роках (рис. 3.9).

Аналогічним чином були оброблені і інші фотографії інтер’єру для сцен в різних локаціях.

Деякі ж локації, які за задумом гри були цілком вигаданими, були повністю згенеровані штучним інтелектом.

Рисунок 3.8 – Сучасний вигляд конференц-залу готелю

Рисунок 3.9 – Стилізація конференц-залу під вигляд в 60-х за допомогою штучного інтелекту

Також засоби штучного інтелекту нам допомогли з відтвореннями історичних персонажів.

Так в архівах конференції зберіглась фотографія Фрідріха Бауера, який виступає на конференції (рис. 3.10).

Рисунок 3.10 – Фото Фрідріха Бауера під час виступу на конференції 1968 р.

За допомогою штучного інтелекту ми спочатку покращили це фото та додали кольорів (рис. 3.11). А потім створили на її основі ігрового персонажа (рис. 3.12).

Рисунок 3.11 – Покращена версія фотографії Фрідріха Бауера

Рисунок 3.12 – Ігровий персонаж на основі образа Фрідріха Бауера

Аналогічним чином ми створили і інших впізнаваних персонажів, прототипами яких були відомі особистості, учасники конференції, такі як Едсгар Дейкстра, Анал Перліс, Пітер Науер та ін.

Деякі персонажі, що в грі є вигаданими та не мають історичних прототипів, були повністю згенеровані за допомогою штучного інтелекту.

3.3 Приклади роботи комп'ютерної гри з інженерії програмного забезпечення

Гра починається з завантаження ігрової оболонки та відображення стартової сторінки, на якій розташовано головне меню (рис. 3.13).

З головного меню ми можемо перейти до інших підсистем, такі як Завантаження гри, Налаштування, Довідка тощо. Ну а також можемо почати нову гру.

Рисунок 3.13 – Стартова сторінка гри

На рисунку 3.14 показано приклад типового діалогу в момент, коли гравцю необхідно обрати свою репліку.

Рисунок 3.14 – Типовий діалог в грі

В залежності від вибору гравця в діалогах буде змінюватись поведінка, реакція та репліки інших персонажів (рис. 3.15).

Рисунок 3.15 – Приклад ігрової сцени зі зміною міміки персонажа

Також в грі реалізовані дискусії з декількома персонажами (рис. 3.16) та діалоги з контрольними питаннями (3.17), правильні відповіді на які надають можливість гравцю рухатись далі по сюжету.

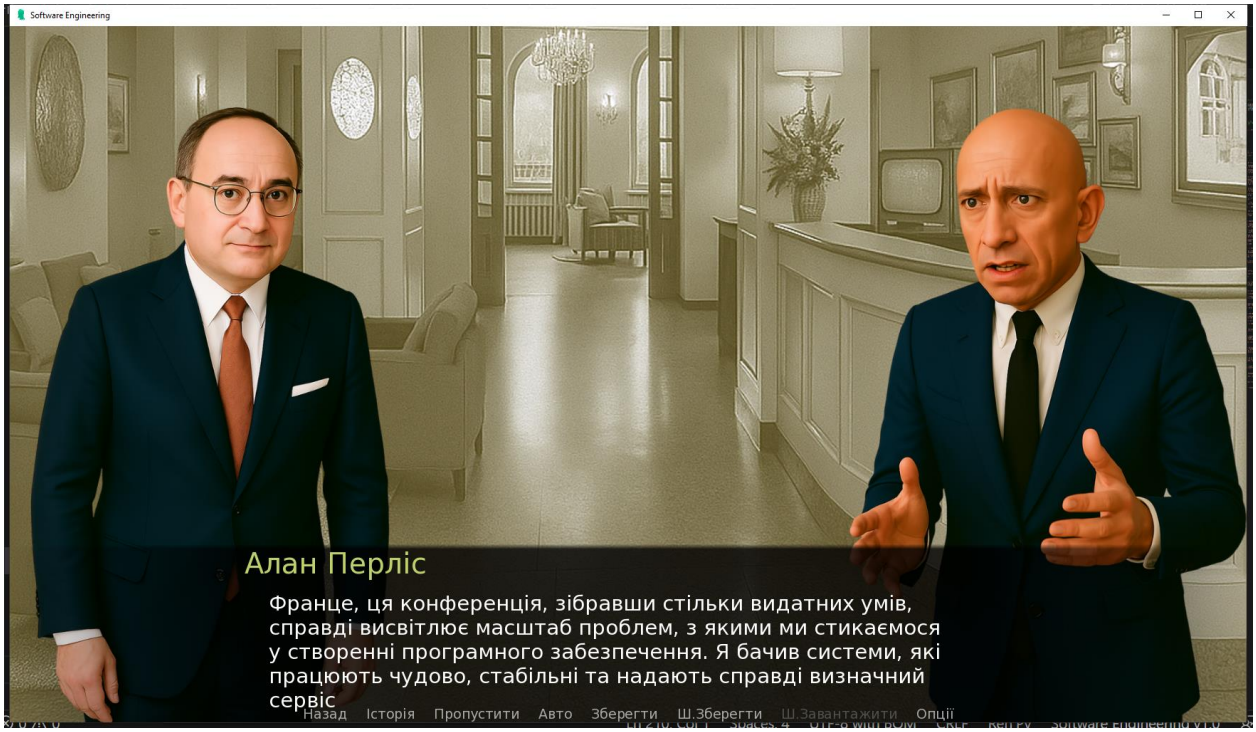

Рисунок 3.16 – Приклад ігрової дискусії з декількома персонажами

Рисунок 3.17 – Приклад діалогу з контрольними питаннями

В деяких ігрових ситуаціях передбачено введення даних з клавіатури, наприклад, як в ситуації, коли гравець задає своє ім’я в грі (рис. 318).

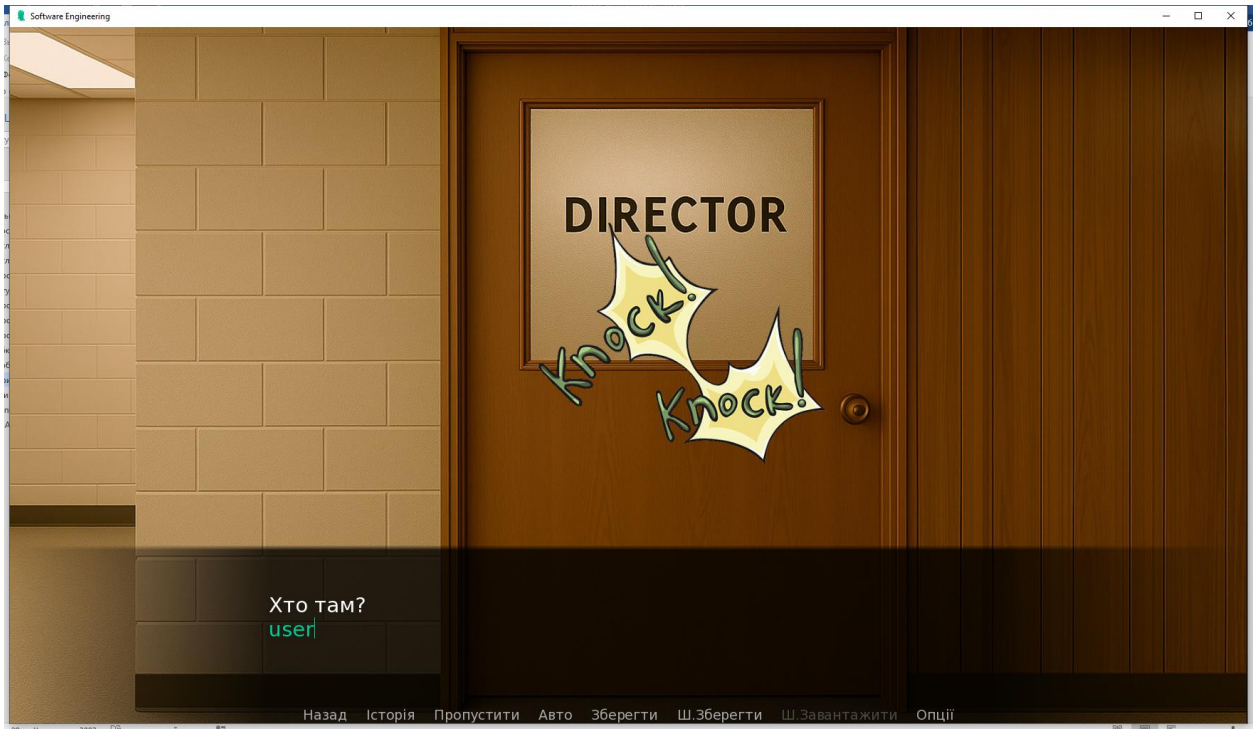

Рисунок 3.18 – Приклад введення даних гравцем з клавіатури

В грі багато діалогів та текстових повідомлень. Запам’ятати їх може бути доволі складно. Але, щоб відтворити будь-який фрагмент діалогу, що відбувся, немає потреби повертатись до збережень в грі. Передбачено режим «Історії» (рис. 3.19), в якому можна переглянути всі діалоги та репліки, що були з самого початку гри.

Така «Історія» зрештою складається в цілісне оповідання з діалогів та окремих реплік.

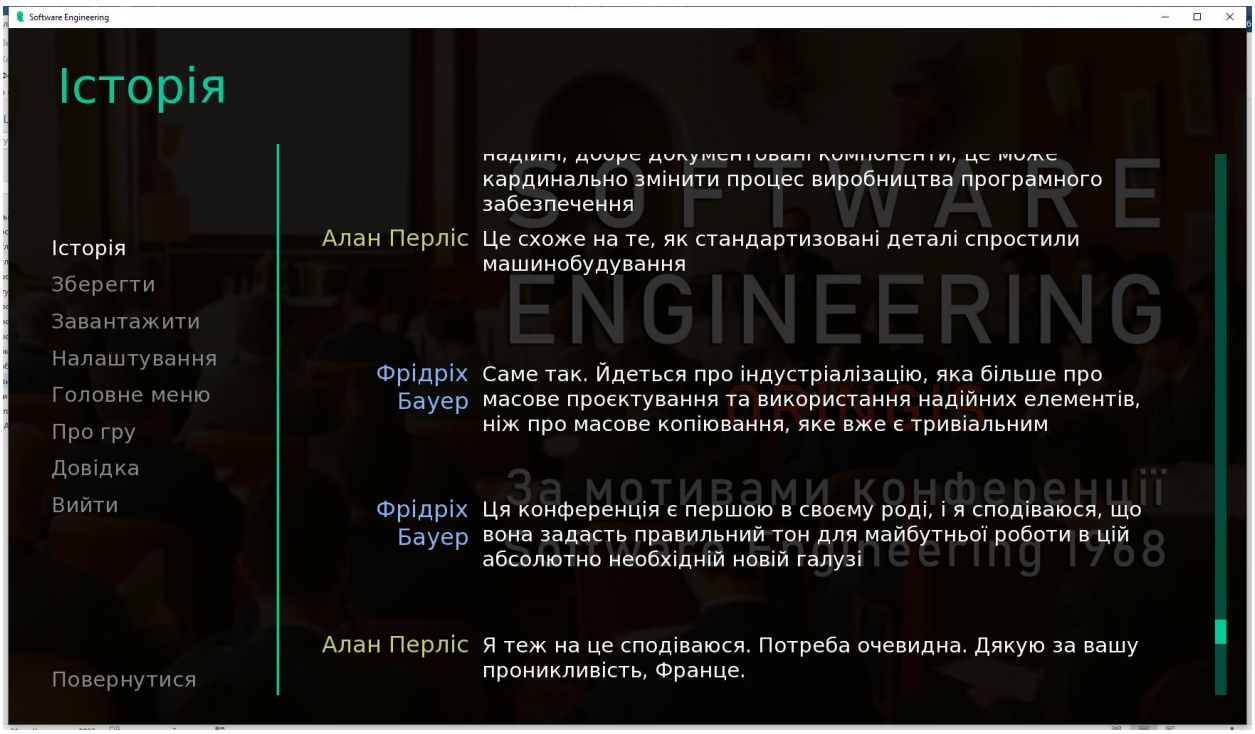

Рисунок 3.19 – Режим історії

Проте, якщо потрібно якусь частину гри пройти знову, або є необхідність припинити гру та повернутись до неї пізніше, реалізовано механізм збережень та завантажень (рис. 3.20).

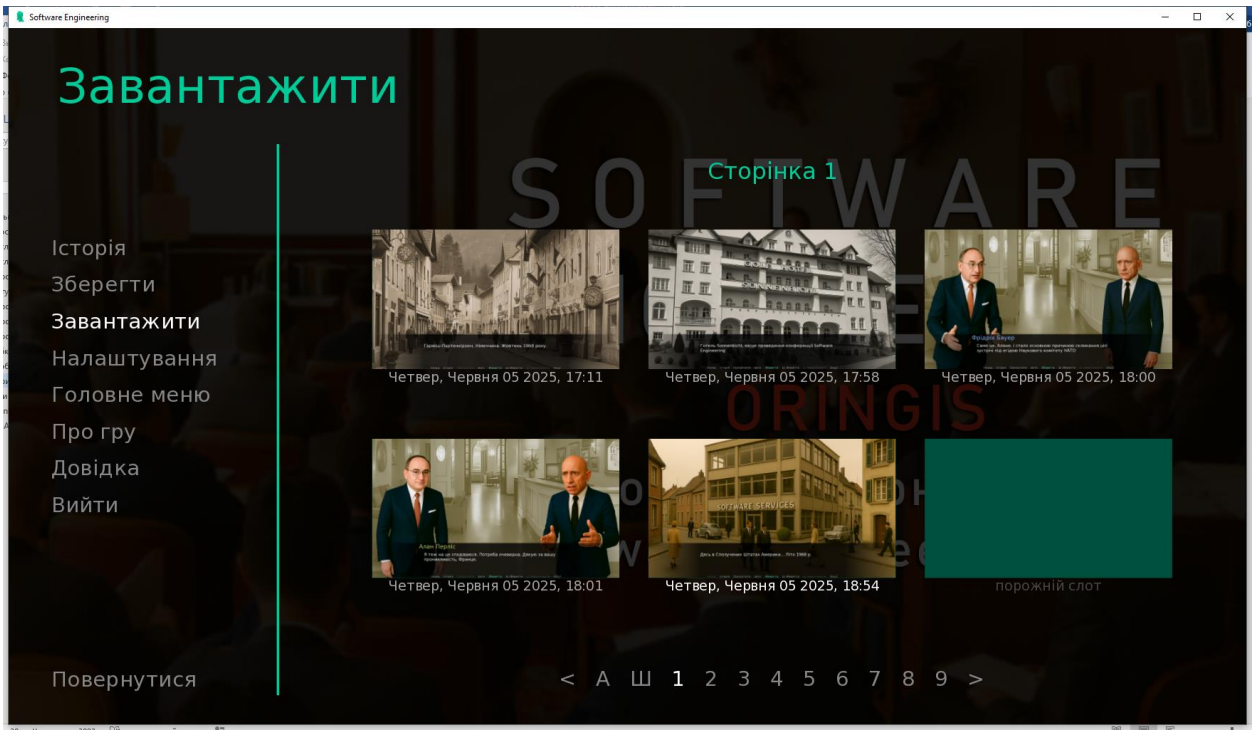

Рисунок 3.20 – Екран збереження

Для того, щоб зорієнтуватись в принципах управління грою та основних функціях ігрової оболонки, реалізовано вікно Довідки (рис. 3.21).

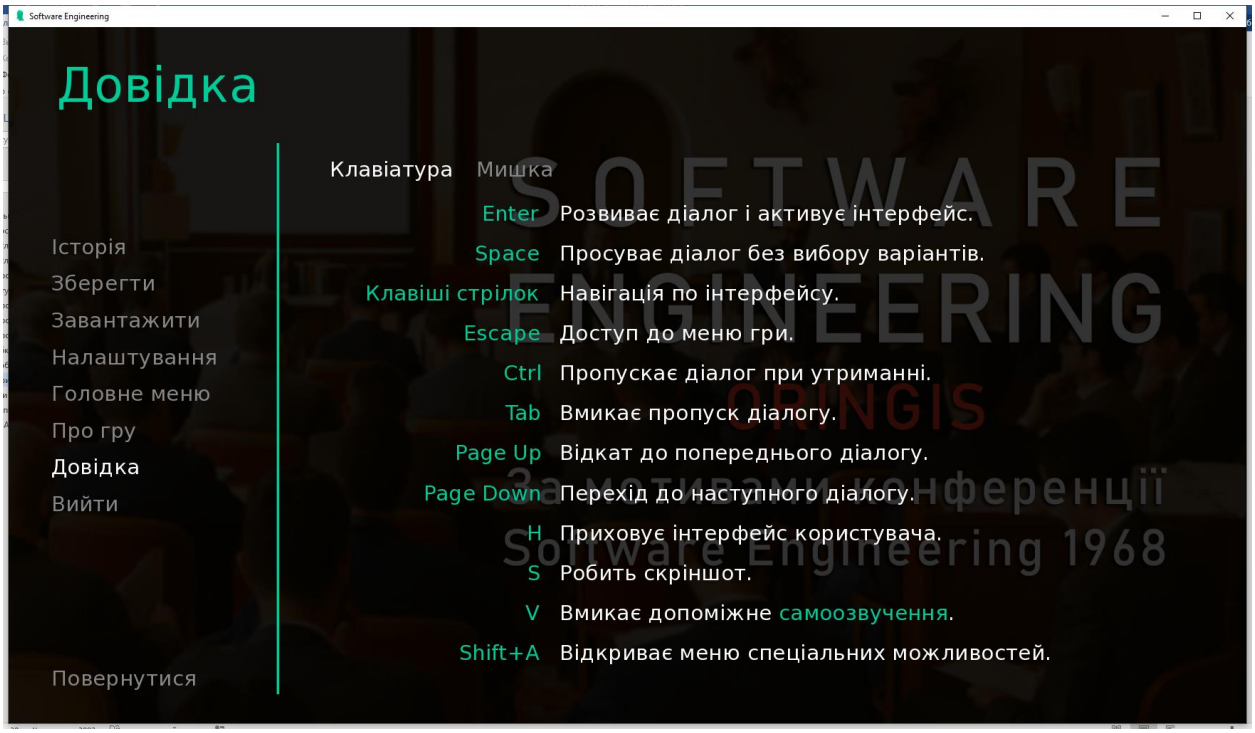

Рисунок 3.21 – Довідка по грі

А також гру можна налаштувати під свої вподобання (рис. 3.22).

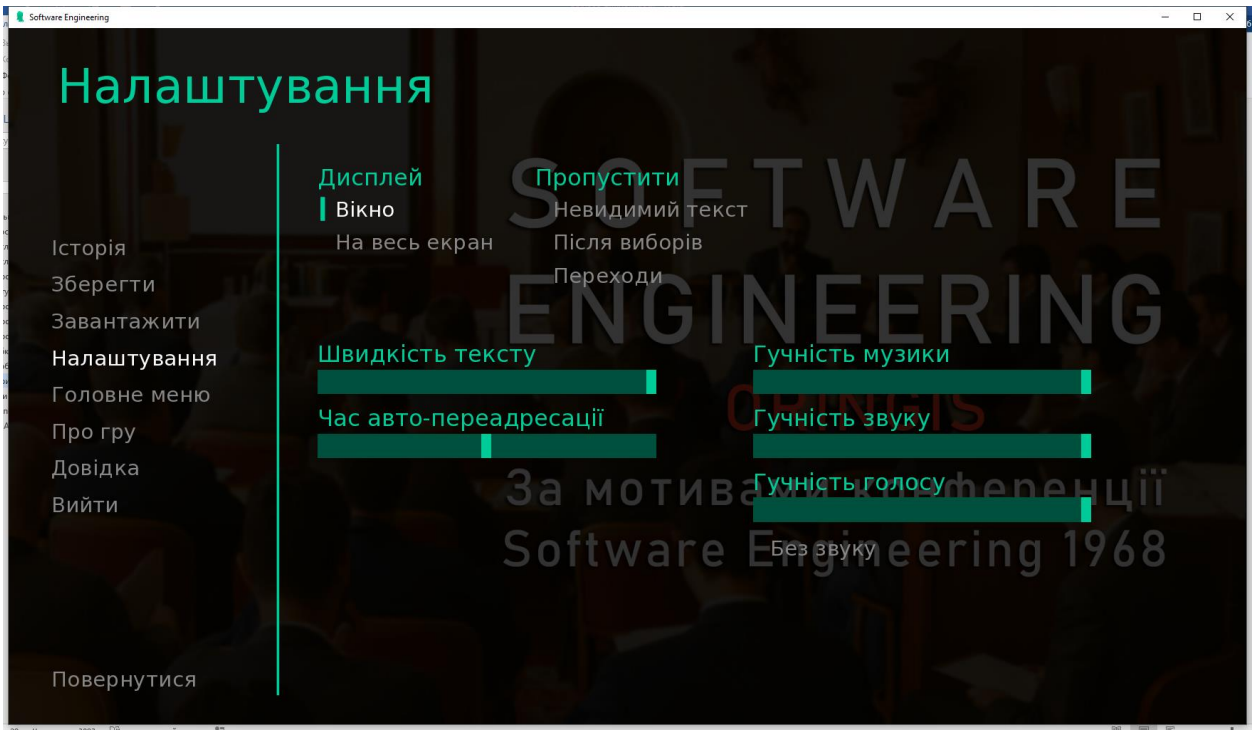

Рисунок 3.22 – Налаштування гри

Таким чином, згідно з завданням ми розробили комп'ютерну гри з історії інженерії програмного забезпечення, що заснована на реальних подіях.

Гра має привабливий зовнішній вигляд, містить велику кількість візуального та текстового матеріалу. Пропонує гравцю сюжетну історію з великою кількістю виборів, можливих діалогів з персонажами.

Створена гра може бути корисна всім, хто цікавиться історією створення інженерії програмного забезпечення, історичними постатями, які пов’язані з нею.

ВИСНОВКИ

Відповідно до завдання, ми розробили інноваційну комп’ютерну гру, що присвячена історії розвитку інженерії програмного забезпечення та базується на реальних історичних подіях. Основна мета гри – створення захоплюючого навчального середовища, яке дозволяє гравцям не лише дізнатися про ключові події, але й поринути у світ видатних особистостей і їх внеску в становлення галузі.

Гра вирізняється сучасним і привабливим дизайном, який забезпечує інтуїтивний інтерфейс і візуальне задоволення для користувачів. Вона наповнена багатим контентом, включаючи інтерактивні ілюстрації, анімовані сцени, а також текстовий матеріал, що доповнює загальну атмосферу. Сюжетна лінія гри ретельно спланована і містить багато варіантів вибору, що дозволяє гравцеві впливати на розвиток подій. Крім того, передбачено діалоги з різними персонажами, які можуть надавати корисну інформацію і розширювати уявлення про епоху.

Розроблений продукт орієнтований на широке коло користувачів: від студентів і викладачів, які цікавляться історичним аспектом програмного забезпечення, до розробників і ентузіастів технічних наук. Гра може стати додатковим джерелом мотивації для молоді, яка мріє займатися програмуванням, показуючи їм, як прості ідеї перетворювались на технологічні революції.

Окрім розважальної функції, планується використовувати гру як інтерактивний навчальний інструмент, доповнення до курсу «Основи інженерії програмного забезпечення», що допоможе студентам закріпити отримані знання через ігровий досвід. Це дозволить не лише краще розуміти теоретичний матеріал, а й опановувати практичні аспекти в контексті історичних подій.

Таким чином, наш продукт прагне стати не просто джерелом інформації, але також інструментом для захоплюючого навчання, допомагаючи гравцям розвивати критичне мислення, аналізувати дані і приймати рішення на основі реальних історичних фактів у контексті розвитку інженерії програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Has the Software Crisis Passed? – URL: https://medium.com/@ryancohane/has-the-software-crisis-passed-d45ce975a1e7
2. Øygardslia K., Weitze C. L., Shin J. The Educational Potential of Visual Novel Games: Principles for Design // Replaying Japan, Vol. 2, 2020. – URL: https://ritsumei.repo.nii.ac.jp/record/13382/files/rcgs_2_%EF%BE%83%E3%83%BBgardslia.pdf
3. Attentat 1942. – URL: https://charlesgames.itch.io/attentat1942
4. Venti Mesi / Twenty Months. – URL: https://wearemuesli.itch.io/ventimesi
5. Grey Plague – URL: https://zephyo.itch.io/grey-plague
6. Golden Threads – URL: https://allanxia.itch.io/golden-threads
7. AP Chemistry – URL: https://xxmissarichanxx.itch.io/apc
8. NATO Software Engineering Conference – URL: https://web.archive.org/web/20120501060221/http://homepages.cs.ncl.ac.uk/brian.randell/NATO/N1968/index.html
9. TyranoBuilder Visual Novel Studio – URL: https://store.steampowered.com/app/345370/TyranoBuilder_Visual_Novel_Studio/?l=Ukrainian
10. Visual Novel Maker – URL: https://store.steampowered.com/app/495480/Visual_Novel_Maker/
11. Your free, open-source game engine – URL: https://godotengine.org/
12. What is Ren'Py? – URL: https://www.renpy.org/

Додаток А

Текст програми

```
    define dr = Character("Директор", color="#b3840d",
image="director.png")
    define gg = Character("[name]", color="#fcf976")
    define br = Character("Фрідріх Бауер", color="#87b4f7",
image="bauer.png")
    define ps = Character("Алан Перліс", color="#bfd16d",
image="perlis.png")
    define da = Character("Едсгар Дейкстра", color="#07a774",
image="dijkstra.png")



    label start:

        # додати файл (з назвою "bg room.png" або "bg room.jpg")
до
        # каталогу images, щоб показати його.


        scene softs01 with Dissolve(3)
        "Десь в Сполучених Штатах Америки... Літо 1968 р."

        scene softs02 with Dissolve(3)
        "Офіс компанії Software Services"

        scene softs03 with Dissolve(3)

        scene door0 with Dissolve(3)

        scene door1 with Dissolve(3)
        "Тук-тук..."

        scene door2
        dr "Хто там?"
        python:
            name = renpy.input("Хто там?")
            name = name.strip() or "Файний Програміст"
        scene door1
        dr "О, [name]! Я вас чекав. Заходьте!"

        scene kabinet2 with Dissolve(1)
        show director

        dr "Отже, [name]. Знову затримка по термінах? І
кошторис... краще навіть не дивитися. Скільки ми вже б'ємося над
цим проєктом?"
```

menu:
"Так, шеф. На жаль. Ми... ми намагаємося. Але проблеми накопичуються, кожне виправлення породжує нові помилки.":
dr "Намагаєтеся? Звісно, ви намагаєтеся. Я бачу, що команда працює понаднормово. Але результату немає."
"Нам потрібно ще трохи часу":
dr "Ми вже тричі зсували дедлайн. Скільки вже можна?"
"Боюсь, що доведеться збільшити бюджет проєкту":
dr "Наш кошторис і так виріс вдвічі з початку роботи!"

dr "Ми обіцяли замовнику стабільну систему, яка буде легко розширюватися та підтримуватися. А що ми маємо? Щось, що... як ви там це називаєте?"
gg "Ну... складно керувати процесом, коли над ним працює стільки людей"
gg "Бракує єдиного підходу, дисципліни"
gg "Кожен пише код по-своєму. Коли додаємо нову функцію, руйнується щось інше"
gg "Тестування займає стільки часу, що здається, ніби ми ніколи не закінчимо"
gg "Документація... її або немає, або вона не відповідає реальності"

show director sad
dr "Так, я чув про це. Неприбуткове болото – хтось так назвав розробку великого програмного забезпечення. "
dr "Витрати на програмування вже не менші за витрати на 'залізо'. А кінцевий продукт... Не відповідає обіцянкам"
dr "Наші замовники вже купують це як 'акт віри', сподіваючись, що воно якось запрацює."

gg "Складається враження, що... ми просто не знаємо, як це робити правильно. Як будувати такі складні системи"
gg "Немає чітких принципів, стандартів. Це більше схоже на... кустарне виробництво, ніж на будівництво мосту чи літака."

dr "Саме так! 'Кустарне виробництво' – влучне слово. Ми ніби 'ліпимо' програми з чогось 'м'якого', яке постійно змінює форму. Потрібно зробити його 'твердішим', надійним. Перетворити з желе на кристали'"
dr "Але як? Де знайти ці принципи?"

gg "Я не знаю. Наші програмісти... вони талановиті, але їхній рівень підготовки дуже нерівномірний"
gg "Більшість просто вміє писати код, але не розуміє всього процесу створення великої системи."

dr "Це теж проблема. Нам потрібні... фахівці іншого типу. Не просто кодери. Люди, які розуміють архітектуру, процеси, надійність."

show director sad2
dr "А ось... Дивно. Прийшов лист від колеги з Європи. НАТО... Науковий комітет..."

gg "НАТО? Військові? Вони теж стикаються з такими проблемами?"
dr "Очевидно. Вони планують конференцію. У Гарміші, Німеччина, в жовтні. Дивно, що вони цим займаються. Але... ще дивніша назва."

gg "Назва? Яка?"
show director wow
dr "Програмна Інженерія (Software Engineering)"

gg "Програмна... Інженерія? Що це означає? Програмне забезпечення – це ж... логіка, алгоритми, математика. Яка тут може бути інженерія? Як у будівництві чи механіці?"
dr "Ось і я думаю! Вони пишуть, що навмисно обрали фразу «програмна інженерія» як провокаційну"
dr "Вона має підкреслити необхідність базувати виробництво програмного забезпечення на типах теоретичних основ та практичних дисциплін, що є традиційними в усталених галузях інженерії"

gg "Отже... вони бачать у цьому щось схоже на будівництво мостів? З розрахунками, стандартами, етапами виробництва?"
dr "Схоже на те. Вони говорять про успішне проєктування, виробництво та супровід корисних програмних систем як про практичну проблему значної складності та важливості "
dr "Саме ті проблеми, про які ти щойно говорив."

gg "Інженерія... програмного забезпечення. Це звучить... незвично. Але, можливо... можливо, в цьому є сенс?"
gg "Якщо інші галузі змогли розробити дисципліни для будівництва величезних складних об'єктів, чому ми не можемо зробити те саме для програм?"

dr "Саме так. Вони вважають, що інженерія є більш плідною областю для пошуку філософії, яка може сприяти нашим підприємствам, ніж науки чи математика у класичному сенсі"
dr "Йдеться про застосування принципів, про створення надійних конструкцій, про передбачуваність"

gg "Надійність... Нам цього так бракує. І передбачуваність термінів, кошторису..."

show director smile
dr "Ось що. Це може бути саме те, що нам потрібно. Новий погляд. Нова дисципліна."

gg "Як... стандартні компоненти, про які говорив МакІлрой? Як у механіці, коли можна взяти готову деталь, а не створювати її щоразу заново?"

dr "Саме так! Йдеться про індустріалізацію, про масове проєктування. Це перша конференція з такою назвою. Вони хочуть задати правильний тон для майбутньої роботи в цій абсолютно необхідній новій галузі"

gg "Звучить... обнадійливо. І лячно водночас. Чи можливо це?"

dr "Ми повинні це з'ясувати. [name], ти глибоко розумієш наші поточні проблеми. Ти бачиш, де болить найбільше. Ти повинен поїхати на цю конференцію"

gg "Я? До Гарміша?"

dr "Так. Саме ти. Послухай, що ці люди говорять. Поспілкуйся з ними. З'ясуй, що вони мають на увазі під 'програмною інженерією'. Чи є у них відповіді на наші проблеми? Чи справді можна перетворити наше 'желе' на 'кристали'?"

dr "Ця конференція може стати важливим кроком, поворотним моментом для всієї галузі, і ми повинні бути в курсі. Дізнайся все, що зможеш."

scene black with Dissolve(3)

scene garmish with Dissolve(3)

"Гарміш-Партенкірхен, Німеччина. Жовтень 1968 року."

scene sonnenbic with Dissolve(3)

"Готель Sonnenbichl, місце проведення конференції Software Engineering"

scene hotel_lobby with Dissolve(3)
show bauer at left
with dissolve
show perlis at right
with dissolve
show perlis talk
with dissolve

ps "Франце, ця конференція, зібравши стільки видатних умів, справді висвітлює масштаб проблем, з якими ми стикаємося у створенні програмного забезпечення. Я бачив системи, які працюють чудово, стабільні та надають справді визначний сервіс"

ps "Але, на жаль, це не універсальна картина. Існує безліч проєктів, у яких задіяні великі команди програмістів, але, здається, їм бракує того, кого ми могли б назвати інженером програмного забезпечення"

ps "Вони функціонують так, ніби не знають, як підійти до будівництва програмного забезпечення систематично"

show perlis
with dissolve
show bauer talk

```
        with dissolve
        br "Саме це, Алане, і стало основною причиною скликання цієї зустрічі під егідою Наукового комітету НАТО"
        show bauer attention
        with dissolve
        br " Ми визнаємо, що виникла практична проблема значної складності та важливості: успішне проєктування, виробництво та супровід корисних програмних систем"
        show bauer talk
        with dissolve
        br "Важливість цього стає дедалі очевиднішою, оскільки вимоги до таких систем лише зростатимуть"
        br "Наслідки — низька продуктивність, поганий дизайн, нестабільність та невідповідність обіцянок і результатів — вже торкаються значних верств нашого суспільства"

        show bauer
        with dissolve
        show perlis talk
        with dissolve
        ps "Проблеми особливо помітні у великомасштабних проєктах "
        ps "Як я вже згадував раніше, підхід 'спроб і помилок', який може спрацювати для невеликої команди з трьох-чотирьох осіб, повністю розвалюється, коли йдеться про сотню людей, через відсутність контролю"
        show perlis attention
        with dissolve
        ps "Нам потрібні дисципліна та структурований підхід, який дозволить керувати цією складністю"

        show perlis
        with dissolve
        show bauer attention
        with dissolve
        br "Ось чому ми навмисно обрали фразу «програмна інженерія» як провокаційну"
        show bauer talk
        with dissolve
        br "Вона покликана підкреслити необхідність базувати виробництво програмного забезпечення на типах теоретичних основ та практичних дисциплін, що є традиційними в усталених галузях інженерії"
        br "Подібно до того, як будівельник спирається на принципи цивільного будівництва, а виробник автомобілів — на машинобудування"
        show bauer attention
        with dissolve
        br "Наша мета цього тижня — перетворити наші «м'які вироби» (mushyware) на «прошивку» (firmware), перетворити наші продукти з желе на кристали"
        show bauer talk
        with dissolve
```

br "Ми повинні серйозно сприймати як наукові, так і інженерні компоненти програмного забезпечення, але саме на останніх має бути наша концентрація"

show bauer
with dissolve
show perlis talk
ps "З желе на кристали – це дуже влучний опис бажаного перетворення. Але, Франце, чим, на вашу думку, ця програмна інженерія відрізняється від, можливо, 'системної інженерії', про яку ми вже говоримо?"
ps "І чи має вона достатньо спільного з традиційною інженерною освітою, як вона визначається сьогодні в США чи Західній Європі, щоб вважатися окремою інженерною дисципліною?"

show perlis
with dissolve
show bauer talk
with dissolve
br "Я вважаю, що інженерія є більш плідною областю для пошуку філософії, яка може сприяти нашим підприємствам, ніж науки чи математика у класичному сенсі "
br "Мова йде про застосування принципів, про створення надійних конструкцій, про передбачуваність."
br "Поточні проблеми виникають, оскільки певні класи систем ставлять перед нами вимоги, які перевищують наші поточні можливості, наші теорії та методи проєктування та виробництва"
show bauer attention
with dissolve
br "Нам потрібен інженерний підхід, щоб відповідати цим вимогам! "

show bauer
with dissolve
show perlis talk
with dissolve
ps "І це безпосередньо стосується питання кадрів. Проблеми посилюються через нерівномірний рівень підготовки персоналу"
ps "Ми повинні серйозно поставити собі питання, чи не було б добре створити галузь інженерії, що займається обчислювальною технікою як такою"
ps "Нам потрібні фахівці, які не просто пишуть код, а розуміють весь процес створення складних систем"
ps "Потрібні працівники, які є правильно навченими, мотивованими та, певною мірою, взаємозамінними"

show perlis
with dissolve
show bauer talk
with dissolve

br "Цілком згоден. Питання освіти нерозривно пов'язане з інженерією програмного забезпечення — вона не може існувати без програмних інженерів"

br "Дослідницькі та освітні установи мають відігравати ключову роль у формуванні цієї дисципліни та підготовці кадрів"

br "Ми маємо знайти основні принципи, 'будівельні блоки' нашого ремесла, зробити їх достатньо зрозумілими, щоб викладати їх на ранніх етапах освіти, як це робиться в інших інженерних галузях"

show bauer
with dissolve
show perlis talk
with dissolve

ps "Так, як МакІлрой говорив про стандартні програмні компоненти. Якщо ми зможемо розробити і стандартизувати надійні, добре документовані компоненти, це може кардинально змінити процес виробництва програмного забезпечення"

ps "Це схоже на те, як стандартизовані деталі спростили машинобудування"

show perlis
with dissolve
show bauer talk
with dissolve

br "Саме так. Йдеться про індустріалізацію, яка більше про масове проєктування та використання надійних елементів, ніж про масове копіювання, яке вже є тривіальним"

show bauer attention
with dissolve

br "Ця конференція є першою в своєму роді, і я сподіваюся, що вона задасть правильний тон для майбутньої роботи в цій абсолютно необхідній новій галузі"

show bauer
with dissolve
show perlis talk
with dissolve

ps "Я теж на це сподіваюся. Потреба очевидна. Дякую за вашу проникливість, Франце."

show perlis
with dissolve
show bauer talk
with dissolve

br "Вам дякую, Алане. Сподіваюся, тиждень обговорень принесе конкретні результати."

show perlis
with dissolve
show bauer
with dissolve

- Дейкстра

```
        scene hotel_terace2 with Dissolve(3)
        show dijkstra
        with dissolve

        gg "Професоре Дейкстра? Вибачте, що турбую вас під час перерви... Я слухав вашу доповідь про контроль складності... Це... це було дуже глибоко."
        da "Слухали? Це вже щось. Багато хто приходить на такі заходи лише щоб відмітити свою присутність, чи, можливо, посміятися над проблемами інших. Але чи зрозуміли?"
        gg "Я... я намагаюся. Моя команда працює над великою системою, і ми загрузли. Постійні помилки, затримки, все складніше зрозуміти, як компоненти взаємодіють. Ваша ідея про ієрархічне впорядкування... здається багатообіцяючою."
        da "Здається багатообіцяючою? "
        show dijkstra attention
        with dissolve
        da "Це не обіцянка, це фундаментальний принцип."
        da "Ви не можете подолати складність, не розбивши її на керовані частини, на шари абстракції."
        show dijkstra
        with dissolve
        da "Наче будувати будинок – ви не починаєте з виготовлення цеглин і вікон одночасно, а потім сподіваєтеся, що вони складуться в щось стійке. Є послідовність рішень."
        gg "Ми намагалися... розділити роботу. Але все одно виникають залежності, які ми не передбачили. Коли одна частина змінюється, ламається інша."
        show dijkstra attention
        with dissolve
        da "Саме так! Тому що ви, мабуть, розділили її не за логічною структурою, а за... не знаю... зручністю на даний момент?"
        show dijkstra
        with dissolve
        da "Чи, можливо, за тим, як організована ваша команда? Структура системи часто дзеркалить структуру групи, яка її створює, на краще чи на гірше."
        da "Але справжня, надійна структура має випливати з природи самої проблеми, а не з тимчасової зручності чи організаційної схеми."

        gg "Але як знайти цю 'правильну' структуру? Ми починаємо з вимог користувача..."
        da "Вимоги користувача – це лише початок, і часто вони неповні або суперечливі."
        da "Справжнє проєктування починається, коли ви розумієте суть проблеми достатньо глибоко, щоб вибудувати надійний логічний фундамент."
        da "е як визначити, що має бути в 'шарі 0'. Це рішення має далекосяжні наслідки. І якщо цей фундамент хибний, усе, що ви будуєте зверху, буде нестабільним."
```

gg "Ми багато часу приділяємо тестуванню, намагаючись знайти всі помилки..."

da "Тестування! (Легкий сарказм у голосі) Ви намагаєтеся встановити якість після того, як продукт створено?"

da "Це як намагатися навчити літати літак, який вже впав зі скелі."

da "Якість не можна 'втестувати' в систему. Вона має бути вбудована в процес проєктування."

da "Можливість переконатися в правильності продукту тісно переплетена з самим процесом проєктування."

da "Якщо ви не можете логічно обґрунтувати коректність на кожному кроці, ніяке тестування не дасть вам справжньої впевненості. Ви просто виявите деякі симптоми глибинних структурних проблем."

gg "Логічно обґрунтувати... Ви маєте на увазі... як математичний доказ? Це здається... непрактичним для великої системи."

da "Слово 'доказ' викликає у людей 'ментальну гикавку', я знаю. Назвіть це 'засвідчення логічної коректності', якщо вам так зручніше."

da "Справа не лише в ідеальному доказі, а в дисципліні мислення, яку він приносить. Це вибудовування системи шарами, де коректність кожного шару залежить від коректності шарів під ним."

show dijkstra attention
with dissolve

da "Це процес проєктування, який робить можливим переконатися в правильності."

show dijkstra
with dissolve

gg "Але... це вимагає так багато уваги до деталей, до... документації?"

da "Документації? (Киває, тепер його тон стає менш критичним, більш наставницьким) Так. Не 'документації' як гори паперу, яку ніхто не читає і яка застаріває в момент друку"

da "А як переддокументації. Опису того, що має робити код, перш ніж ви почнете писати, як."

da "Це не додатковий тягар! Це інструмент, який змушує вас зрозуміти, що ви робите!"

da "У мене був студент, який не міг написати просту умову циклу, поки я не змусив його словами описати, що саме цей фрагмент коду мав робити."

da "Він не розумів власний код, бо думав лише про 'як', а не про 'що'."

gg "Я... я ніколи не думав про це так. Ми сприймали документацію як щось, що робиться 'потім', якщо залишається час..."

show dijkstra attention
with dissolve

da "І ось де ваша біда!"

```
        da "'Потім' – це ніколи! І навіть якщо ви її зробите, вона не допоможе вам у процесі створення, лише може (можливо) допомогти комусь розібратися в ваших помилках пізніше."
        show dijkstra
        with dissolve
        da "Якщо ви не можете чітко описати логіку та структуру, ймовірно, ви самі її до кінця не розумієте. А якщо ви її не розумієте, як вона може бути коректною чи надійною?"


        gg "Отже... справа не просто в тому, щоб розбити систему на частини, а в тому, як саме це зробити, і в постійній перевірці логічної коректності на кожному кроці, починаючи з найнижчих шарів..."
        gg "І... документувати, що робиш, перш ніж робити?"
        da "Ви починаєте розуміти. Це не 'фіт-енд-трай' процес для сотні людей. Це дисциплінований підхід."
        da "Це перетворення 'м'якого забезпечення' (mushyware) на 'тверде' (firmware), з 'желе' на 'кристали'."
        da "Це вимагає мислення інженера, а не тільки програміста, який 'грає в ігри'. Це важко. Особливо, коли вимоги виходять за межі поточного стану технологій."
        da "Не сподівайтеся, що це буде легко. Якщо б було легко, у нас не було б цих проблем, цієї... 'кризи', як її дехто називає."
        da "Але це можливо. Починайте з фундаменту. Будуйте шарами. Думайте. І записуйте свої думки, перш ніж писати код. Це заощадить вам значно більше часу та нервів у довгостроковій перспективі."

        gg "Дякую, професоре. Дуже дякую. Це... це змінює погляд."
        da "Сподіваюся. Тепер ідіть і спробуйте застосувати. Теорія без практики залишається лише теорією."


        return


    ################################################################################
    ## Ініціалізація
    ################################################################################

    ## Оператор init offset змушує оператори ініціалізації в цьому файлі
    ## виконуватися перед операторами init в будь-якому іншому файлі.
    init offset = -2

    ## Виклик gui.init скидає стилі до розумних значень за замовчуванням і
    ## встановлює ширину та висоту гри.
```

```
    init python:
        gui.init(1920, 1080)



    ##########################################################################
####################
    ## Змінні конфігурації GUI
    ##########################################################################
####################


    ##                                                                Кольори
##########################################################################
#####
    ##
    ## Кольори тексту в інтерфейсі.

    ## Колір акценту, який використовується в інтерфейсі для
позначення та виділення
    ## тексту.
    define gui.accent_color = '#00cc99'

    ## Колір, який використовується для текстової кнопки, коли
вона не вибрана і не
    ## наведена.
    define gui.idle_color = '#888888'

    ## Дрібний колір використовується для дрібного тексту, який
має бути яскравішим/
    ## темнішим, щоб досягти того самого ефекту.
    define gui.idle_small_color = '#aaaaaa'

    ## Колір, який використовується для кнопок і смуг, на які
наводяться.
    define gui.hover_color = '#66e0c1'

    ## Колір, який використовується для текстової кнопки, коли
вона вибрана, але не
    ## в фокусі. Кнопку вибрано, якщо це поточний екран або
значення параметра.
    define gui.selected_color = '#ffffff'

    ## Колір текстової кнопки, якщо її неможливо вибрати.
    define gui.insensitive_color = '#8888887f'

    ## Кольори, що використовуються для незаповнених частин
стовпчиків. Вони не
    ## використовуються безпосередньо, а використовуються під час
повторного
    ## створення файлів зображень стовпчика.
    define gui.muted_color = '#00513d'
    define gui.hover_muted_color = '#007a5b'
```

```
    ## Кольори, які використовуються для тексту діалогу та вибору
меню.
    define gui.text_color = '#ffffff'
    define gui.interface_text_color = '#ffffff'


    ##          Шрифти          та          розміри          шрифтів
################################################

    ## Шрифт, який використовується для тексту в грі.
    define gui.text_font = "DejaVuSans.ttf"

    ## Шрифт, який використовується для імен символів.
    define gui.name_text_font = "DejaVuSans.ttf"

    ## Шрифт, який використовується для тексту поза грою.
    define gui.interface_text_font = "DejaVuSans.ttf"

    ## Розмір звичайного тексту діалогу.
    define gui.text_size = 33

    ## Розмір імен персонажів.
    define gui.name_text_size = 45

    ## Розмір тексту в інтерфейсі користувача гри.
    define gui.interface_text_size = 33

    ## Розмір міток в інтерфейсі користувача гри.
    define gui.label_text_size = 36

    ## Розмір тексту на екрані сповіщень.
    define gui.notify_text_size = 24

    ## Розмір назви гри.
    define gui.title_text_size = 75


    ##          Головне          та          ігрове          меню
#####################################################

    ## Зображення, які використовуються для головного та ігрового
меню.
    define gui.main_menu_background = "gui/Conf01.jpg"
    define gui.game_menu_background = "gui/Conf01.jpg"


    ##                                                          Діалог
#############################################################
######
    ##
    ## Ці змінні керують тим, як діалог відображатиметься на
екрані один рядок за
```

```
    ## раз.

    ## Висота текстового поля, що містить діалог.
    define gui.textbox_height = 278

    ## Розташування текстового поля вертикально на екрані. 0,0 –
верх, 0,5 – центр,
    ## 1,0 – низ.
    define gui.textbox_yalign = 1.0


    ## Розташування імені персонажа, що говорить, відносно
текстового поля. Це може
    ## бути ціла кількість пікселів зліва чи зверху або 0,5 до
центру.
    define gui.name_xpos = 360
    define gui.name_ypos = 0

    ## Горизонтальне вирівнювання імені персонажа. Це може бути
0,0 для вирівнювання
    ## по лівому краю, 0,5 для вирівнювання по центру та 1,0 для
вирівнювання по
    ## правому краю.
    define gui.name_xalign = 0.0

    ## Ширина, висота та межі поля, що містять ім’я персонажа,
або None, щоб
    ## автоматично змінити його розмір.
    define gui.namebox_width = None
    define gui.namebox_height = None

    ## Межі поля, що містять ім'я персонажа, у порядку зліва,
зверху, справа, знизу.
    define gui.namebox_borders = Borders(5, 5, 5, 5)

    ## Якщо True, фон поля імен буде мозаїкою, якщо False, фон
вікна імен буде
    ## масштабовано.
    define gui.namebox_tile = False


    ## Розташування діалогу відносно текстового поля. Це може
бути ціла кількість
    ## пікселів відносно лівого чи верхнього краю текстового поля
або 0,5 до центру.
    define gui.dialogue_xpos = 402
    define gui.dialogue_ypos = 75

    ## Максимальна ширина тексту діалогу в пікселях.
    define gui.dialogue_width = 1116

    ## Горизонтальне вирівнювання тексту діалогу. Це може бути
0,0 для вирівнювання
```

```
    ## по лівому краю, 0,5 для вирівнювання по центру та 1,0 для вирівнювання по
    ## правому краю.
    define gui.dialogue_text_xalign = 0.0


    ##                                                                Кнопки ##############################################################################
    ##
    ## Ці змінні разом із файлами зображень у gui/button контролюють аспекти
    ## відображення кнопок.

    ## Ширина та висота кнопки в пікселях. Якщо немає, Ren'Py обчислює розмір.
    define gui.button_width = None
    define gui.button_height = None

    ## Межі з кожного боку кнопки в порядку зліва, зверху, справа та знизу.
    define gui.button_borders = Borders(6, 6, 6, 6)

    ## Якщо True, фонове зображення буде мозаїчно. Якщо False, фонове зображення
    ## буде лінійно масштабовано.
    define gui.button_tile = False

    ## Шрифт, який використовується кнопкою.
    define gui.button_text_font = gui.interface_text_font

    ## Розмір тексту, який використовується кнопкою.
    define gui.button_text_size = gui.interface_text_size

    ## Колір тексту кнопки в різних станах.
    define gui.button_text_idle_color = gui.idle_color
    define gui.button_text_hover_color = gui.hover_color
    define gui.button_text_selected_color = gui.selected_color
    define gui.button_text_insensitive_color = gui.insensitive_color

    ## Горизонтальне вирівнювання тексту кнопки. (0,0 ліворуч, 0,5 центр, 1,0
    ## праворуч).
    define gui.button_text_xalign = 0.0


    ## Ці змінні перекривають налаштування для різних типів кнопок. Перегляньте
    ## документацію gui, щоб дізнатися про типи доступних кнопок і для чого кожна з
    ## них використовується.
    ##
```

```
    ## Ці налаштування використовуються стандартним інтерфейсом:

    define gui.radio_button_borders = Borders(27, 6, 6, 6)

    define gui.check_button_borders = Borders(27, 6, 6, 6)

    define gui.confirm_button_text_xalign = 0.5

    define gui.page_button_borders = Borders(15, 6, 15, 6)

    define gui.quick_button_borders = Borders(15, 6, 15, 0)
    define gui.quick_button_text_size = 21
    define          gui.quick_button_text_idle_color          =
gui.idle_small_color
    define        gui.quick_button_text_selected_color        =
gui.accent_color

    ## Ви також можете додати власні налаштування, додавши змінні
з правильними
    ## назвами. Наприклад, ви можете розкоментувати наступний
рядок, щоб встановити
    ## ширину кнопки навігації.

    # define gui.navigation_button_width = 250


    ##                            Кнопки                           вибору
##############################################################################
    ##
    ## Кнопки вибору використовуються в ігрових меню.

    define gui.choice_button_width = 1185
    define gui.choice_button_height = None
    define gui.choice_button_tile = False
    define gui.choice_button_borders = Borders(150, 8, 150, 8)
    define gui.choice_button_text_font = gui.text_font
    define gui.choice_button_text_size = gui.text_size
    define gui.choice_button_text_xalign = 0.5
    define gui.choice_button_text_idle_color = "#cccccc"
    define gui.choice_button_text_hover_color = "#ffffff"
    define gui.choice_button_text_insensitive_color = "#444444"


    ##              Кнопки              слотів              файлів
##############################################################################
    ##
    ## Кнопка слота файлів — це особливий вид кнопки. Він містить
ескіз зображення
    ## та текст, що описує вміст гнізда збереження. Слот
збереження використовує
    ## файли зображень у gui/button, як і інші типи кнопок.

    ## Кнопка збереження слота.
```

```
    define gui.slot_button_width = 414
    define gui.slot_button_height = 309
    define gui.slot_button_borders = Borders(15, 15, 15, 15)
    define gui.slot_button_text_size = 21
    define gui.slot_button_text_xalign = 0.5
    define          gui.slot_button_text_idle_color          =
gui.idle_small_color
    define      gui.slot_button_text_selected_idle_color      =
gui.selected_color
    define      gui.slot_button_text_selected_hover_color     =
gui.hover_color

    ## Ці змінні керують розташуванням і відстанню між різними
елементами інтерфейсу
    ## користувача.
    define config.thumbnail_width = 384
    define config.thumbnail_height = 216

    ## Ширина та висота мініатюр, що використовуються слотами для
збереження.
    define gui.file_slot_cols = 3
    define gui.file_slot_rows = 2


    ##          Позиціонування          та          інтервали
################################################
    ##
    ## Ці змінні керують розташуванням і відстанню між різними
елементами інтерфейсу
    ## користувача.

    ## Розташування лівої сторони навігаційних кнопок відносно
лівої сторони екрана.
    define gui.navigation_xpos = 60

    ## Вертикальна позиція індикатора пропуску.
    define gui.skip_ypos = 15

    ## Вертикальне положення екрана сповіщень.
    define gui.notify_ypos = 68

    ## Інтервал між пунктами меню.
    define gui.choice_spacing = 33

    ## Кнопки в розділі навігації головного та ігрового меню.
    define gui.navigation_spacing = 6

    ## Контролює розмір інтервалу між параметрами.
    define gui.pref_spacing = 15

    ## Контролює відстань між кнопками налаштувань.
    define gui.pref_button_spacing = 0
```

```
    ## Відстань між кнопками сторінки файлу.
    define gui.page_spacing = 0

    ## Відстань між слотами файлів.
    define gui.slot_spacing = 15

    ## Позиція тексту головного меню.
    define gui.main_menu_text_xalign = 1.0


    ##                                                              Рамки
################################################################################
    ##
    ## Ці змінні керують виглядом рамок, що можуть містити компоненти інтерфейсу
    ## користувача, коли накладання або вікно відсутні.

    ## Загальні рамки.
    define gui.frame_borders = Borders(6, 6, 6, 6)

    ## Рамка, яка використовується як частина екрана підтвердження.
    define gui.confirm_frame_borders = Borders(60, 60, 60, 60)

    ## Рамка, який використовується як частина екрана пропуску.
    define gui.skip_frame_borders = Borders(24, 8, 75, 8)

    ## Рамка, яка використовується як частина екрана сповіщень.
    define gui.notify_frame_borders = Borders(24, 8, 60, 8)

    ## Чи слід фонові рамки розміщувати плиткою?
    define gui.frame_tile = False


    ##     Панелі,     смуги     прокрутки     та     повзунки
########################################
    ##
    ## Це керує виглядом і розміром смуг, смуг прокрутки та повзунків.
    ##
    ## GUI за замовчуванням використовує лише повзунки та вертикальні смуги
    ## прокручування. Усі інші смужки використовуються лише на екранах, написаних
    ## автором.

    ## Висота горизонтальних смуг, смуг прокрутки та повзунків. Ширина вертикальних
    ## смуг, смуг прокручування та повзунків.
    define gui.bar_size = 38
    define gui.scrollbar_size = 18
    define gui.slider_size = 38
```

```
    ## True якщо зображення панелей мають бути мозаїками. False,
якщо вони повинні
    ## бути лінійно масштабовані.
    define gui.bar_tile = False
    define gui.scrollbar_tile = False
    define gui.slider_tile = False

    ## Горизонтальні межі.
    define gui.bar_borders = Borders(6, 6, 6, 6)
    define gui.scrollbar_borders = Borders(6, 6, 6, 6)
    define gui.slider_borders = Borders(6, 6, 6, 6)

    ## Вертикальні межі.
    define gui.vbar_borders = Borders(6, 6, 6, 6)
    define gui.vscrollbar_borders = Borders(6, 6, 6, 6)
    define gui.vslider_borders = Borders(6, 6, 6, 6)

    ## Що робити з непрокручуваними смугами прокручування в gui.
"hide" приховує їх,
    ## а «Нічого» показує їх.
    define gui.unscrollable = "hide"


    ##                                                        Історія
##########################################################################
#####
    ##
    ## На екрані історії відображається діалог, який гравець уже
закрив.

    ## Кількість блоків історії діалогів, яких Ren'Py збереже.
    define config.history_length = 250

    ## Висота запису на екрані історії або «Нічого», щоб зробити
висоту змінною за
    ## рахунок продуктивності.
    define gui.history_height = 210

    ## Положення, ширина та вирівнювання мітки, що вказує ім’я
персонажа, що
    ## говорить.
    define gui.history_name_xpos = 233
    define gui.history_name_ypos = 0
    define gui.history_name_width = 233
    define gui.history_name_xalign = 1.0

    ## Позиція, ширина та вирівнювання тексту діалогу.
    define gui.history_text_xpos = 255
    define gui.history_text_ypos = 3
    define gui.history_text_width = 1110
    define gui.history_text_xalign = 0.0
```

```
    ##                                 Режим                                 NVL
################################################################################
###
    ##
    ## На екрані режиму NVL відображається діалог, який вимовляють
персонажі режиму
    ## NVL.

    ## Межі фону заднього плану режиму NVL.
    define gui.nvl_borders = Borders(0, 15, 0, 30)

    ## Ren'Py відобразить максимальну кількість записів у режимі
NVL. Якщо потрібно
    ## показати більше записів, найстаріший запис буде видалено.
    define gui.nvl_list_length = 6

    ## Висота запису в режимі NVL. Встановіть значення None, щоб
записи динамічно
    ## регулювали висоту.
    define gui.nvl_height = 173

    ## Інтервал між записами режиму NVL, коли gui.nvl_height має
значення None, і
    ## між записами режиму NVL і меню режиму NVL.
    define gui.nvl_spacing = 15

    ## Положення, ширина та вирівнювання мітки, що вказує ім’я
персонажа, що
    ## говорить.
    define gui.nvl_name_xpos = 645
    define gui.nvl_name_ypos = 0
    define gui.nvl_name_width = 225
    define gui.nvl_name_xalign = 1.0

    ## Позиція, ширина та вирівнювання тексту діалогу.
    define gui.nvl_text_xpos = 675
    define gui.nvl_text_ypos = 12
    define gui.nvl_text_width = 885
    define gui.nvl_text_xalign = 0.0

    ## Положення, ширина та вирівнювання тексту nvl_thought
(тексту, який вимовляє
    ## персонаж nvl_narrator.)
    define gui.nvl_thought_xpos = 360
    define gui.nvl_thought_ypos = 0
    define gui.nvl_thought_width = 1170
    define gui.nvl_thought_xalign = 0.0

    ## Позиціонування nvl menu_buttons.
    define gui.nvl_button_xpos = 675
    define gui.nvl_button_xalign = 0.0
```

```
    ##                                                          Локалізація
##############################################################################
#

    ## Це визначає, де дозволений розрив рядка. Стандартне значення підходить для
    ## більшості мов. Список доступних значень можна знайти за адресою https://
    ##       www.renpy.org/doc/html/style_properties.html#style-property-language

    define gui.language = "unicode"


    ##############################################################################
####################
    ## Мобільні пристрої
    ##############################################################################
####################

    init python:

        ## Це збільшує розмір швидких кнопок, щоб їх було легше торкатися на
        ## планшетах і телефонах.
        @gui.variant
        def touch():

            gui.quick_button_borders = Borders(60, 21, 60, 0)

        ## Це змінює розмір і відстань між різними елементами gui, щоб вони були
        ## видимими на телефонах.
        @gui.variant
        def small():

            ## Розміри шрифту
            gui.text_size = 45
            gui.name_text_size = 54
            gui.notify_text_size = 38
            gui.interface_text_size = 45
            gui.button_text_size = 45
            gui.label_text_size = 51

            ## Налаштунки розташування текстового поля.
            gui.textbox_height = 360
            gui.name_xpos = 120
            gui.dialogue_xpos = 135
            gui.dialogue_width = 1650

            ## Зміна розміру і відстані між різними речами.
            gui.slider_size = 54
```

```
gui.choice_button_width = 1860
gui.choice_button_text_size = 45

gui.navigation_spacing = 30
gui.pref_button_spacing = 15

gui.history_height = 285
gui.history_text_width = 1035

gui.quick_button_text_size = 30

## Макет кнопки файлу.
gui.file_slot_cols = 2
gui.file_slot_rows = 2

## NVL-режим.
gui.nvl_height = 255

gui.nvl_name_width = 458
gui.nvl_name_xpos = 488

gui.nvl_text_width = 1373
gui.nvl_text_xpos = 518
gui.nvl_text_ypos = 8

gui.nvl_thought_width = 1860
gui.nvl_thought_xpos = 30

gui.nvl_button_width = 1860
gui.nvl_button_xpos = 30
```