Національний університет України
(повна назва вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повна назва інституту, назва факультету (відділення))

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

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(рівень вищої освіти)

на тему: *Розроблення інтерактивної візуальної новели засобами Ren'Py.*

Виконав: студент 2 курсу групи КНС-21
Спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Грімняк А.А.
(прізвище та ініціали)
Керівник Розумовський М.П.
(прізвище та ініціали)
Керівник Прошик Ю. С.
(прізвище та ініціали)
Рецензент Часковський О.Т.
(прізвище та ініціали)

Львів – 2025

Національний політехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Борецька І. Б.
"10" червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Грімнаку Андрію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення інтерактивної візуальної новели засобами Ren'Py*
керівники роботи *ас. кафедри КН Розумовський М.П., к.ф.-м.н., доц. кафедри КН Процик Ю.С.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "15" листопада 2024 року
№ С-882
2. Термін подання студентом роботи 10 червня 2025 р.
3. Вихідні дані до роботи Розроблення інтерактивної візуальної новели з механікою розвитку відносин між персонажами засобами Python/Ren'Py
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація до диплому
6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літературних даних.*	*19.11.2024 – 09.12.2024*	Виконано
2.	*Розділ 1. Стан проблемної області.*	*10.12.2024 – 06.01.2025*	Виконано
3.	*Розділ 2. Інформаційне та математичне забезпечення.*	*07.01.2025 – 03.02.2025*	Виконано
4.	*Розділ 3. Програмне та технічне забезпечення.*	*04.02.2025 – 05.05.2025*	Виконано
5.	*Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.*	*06.05.2025 – 09.06.2025*	Виконано
6.	*Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.*	*10.06.2025*	Виконано

Студент _____ *Грімак А. А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Розумовський М.П.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Процик Ю.С.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота містить 64 сторінок, 33 ілюстрацій, 1 додаток, 11 джерел.

У дипломному проєкті представлено процес розробки інтерактивної візуальної новели з механікою розвитку відносин між персонажами засобами Python і Ren'Py. Основна мета проєкту — створення інтерактивного ігрового середовища, у якому рішення гравця впливають на розвиток сюжету та взаємини між героями. У роботі проаналізовано особливості жанру візуальних новел, реалізовано календарну систему та поділ ігрового часу на сегменти, що визначають доступні дії гравця. Розроблено систему збереження прогресу, змінних відносин, енергії та навчальних подій. Проєкт ілюструє можливості використання рушія Ren'Py для створення нелінійних інтерактивних сценаріїв з елементами симулятора життя.

Ключові слова: *візуальна новела, Ren'Py, Python, ігрова механіка, розвиток відносин, інтерактивний сюжет, симулятор життя.*

ABSTRACT

The Bachelor's thesis consists of 64 pages, 33 illustrations, 1 appendix, and 11 sources.

The diploma project presents the development process of an interactive visual novel with a relationship-building mechanic using Python and Ren'Py. The main goal is to create an interactive game environment where the player's choices influence the storyline and character relationships. The work analyzes the features of the visual novel genre, implements a calendar system and segmented game time that determines the player's available actions. A system for saving progress, tracking relationship variables, energy, and educational events was developed. The project demonstrates the capabilities of the Ren'Py engine for creating nonlinear interactive narratives with life simulation elements.

Keywords: *visual novel, Ren'Py, Python, game mechanics, relationship development, interactive story, life simulation.*

ТЕХНІЧНЕ ЗАДАННЯ

Розробити програмний продукт у вигляді візуальної новели, в якому реалізовано симуляцію життя з розподілом часу та обмеженими ресурсами.

Розробка програмного продукту передбачає реалізацію наступних функціональних складових:

- Реалізувати календарну систему з розподілом гри на ігрові дні, кожен з яких поділений на часові відрізки;
- Створити декілька кінцівок та їх варіацій, отримання яких напряму залежить від дій гравця та його параметрів;
- Розробити ігрові події (сесія, яку гравець повинен закрити) та системи статусу (енергія, стрес, репутація викладачів);
- Реалізувати ігрові дії інтерактивними методами;

Використати стек технологій: Python з фреймворком Ren'Py.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

VN – Visual Novel – візуальна новела

SDK – Software Development Kit – програмний комплект для розробки

GUI – Graphical User Interface – графічний інтерфейс користувача

UI – User Interface – інтерфейс користувача

HUD – Heads-Up Display – екранна панель зі статусом

RNG – Random Number Generator – генератор випадкових чисел

ВСТУП

У сучасному цифровому світі ігрова індустрія стрімко розвивається, охоплюючи різноманітні жанри та технології. Одним із популярних жанрів, що поєднує в собі літературну розповідь, інтерактивність та візуальне оформлення, є візуальні новели. Такі ігри дають змогу гравцеві впливати на розвиток сюжету через прийняття рішень, занурюючи його у світ персонажів і подій.

Об’єктом дослідження є створення інтерактивних історій з використанням сучасних інструментів програмування та сценарного дизайну.

Предметом дослідження є реалізація візуальні новели з використанням технологій Python та рушію Ren'Py.

Метою роботи є розробка інтерактивної візуальної новели з системою розвитку відносин між персонажами, елементами симуляції повсякденного життя та інтерактивним навчальним компонентом.

Завдання полягає в розробці візуальної новели. У межах проєкту реалізувати календарну систему, механіку впливу виборів на сюжет, параметри персонажа (енергія, успішність), систему збереження та кілька альтернативних кінцівок, що формуються в залежності від дій гравця.

Практичне значення зумовлена зростаючим попитом на інтерактивні розповіді, а також прагненням до розширення інструментарію програмної реалізації ігрових механік у навчальному та розважальному контекстах.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Сучасний стан ринку ігрової індустрії

Ігрова індустрія є однією з найдинамічніших і найприбутковіших галузей цифрової економіки. За останнє десятиліття вона перетворилася з нішової сфери розваг на глобальну культурну та економічну силу. За даними аналітичних компаній, таких як Newzoo та Statista [1], загальний обсяг світового ринку відеоігор у 2024 році сягнув 187.7 мільярдів доларів США, а кількість активних гравців сягнула понад 3 мільярди осіб у всьому світі (Рисунок 1.1).

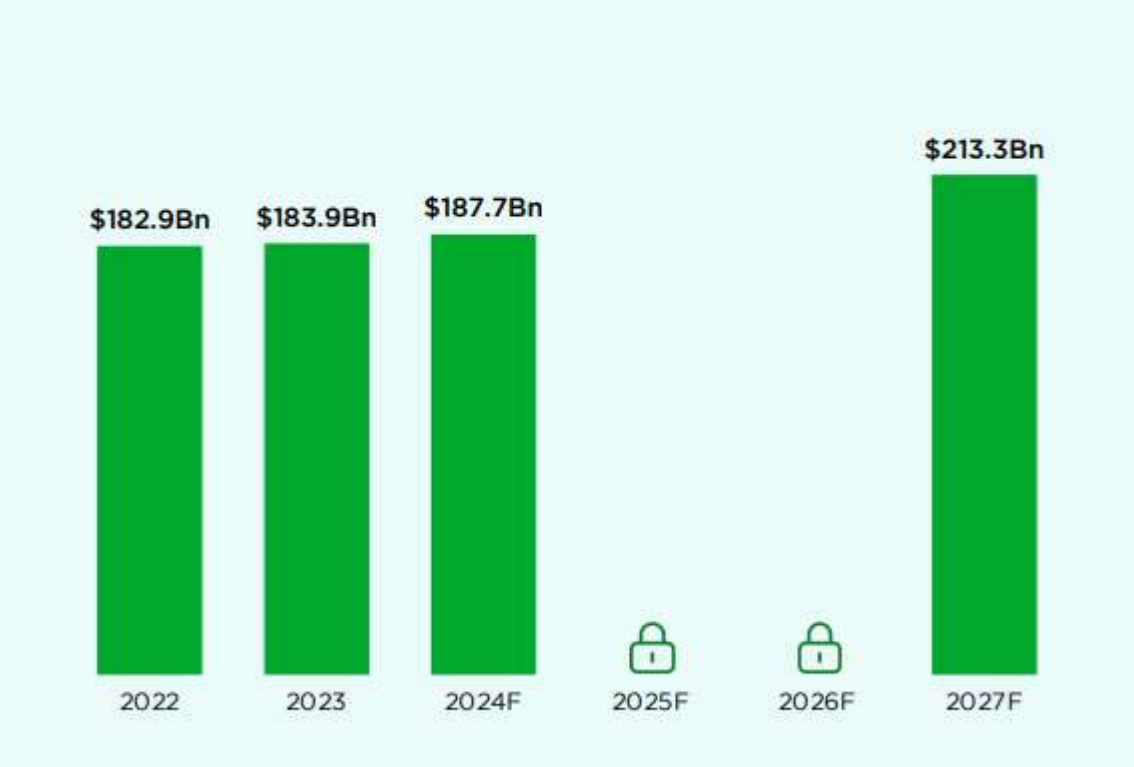


Рисунок 1.1 – Розмір глобального ринку ігрової індустрії

Ринок відеоігор охоплює численні платформи: ПК, консолі, мобільні пристрої, браузерні ігри (Рисунок 1.2), а також окремі жанри, які постійно розвиваються. Серед них особливу нішу займають візуальні новели, які активно розвиваються переважно на ПК та мобільних пристроях. Цей жанр особливо популярний в Японії, Південній Кореї та все більше набуває поширення у західних країнах.

Рисунок 1.2 – Кількість геймерів у світі

Популярність візуальних новел пояснюється їхньою здатністю поєднувати глибоку емоційну історію з можливістю впливу на розвиток сюжету. Вони не потребують значних технічних ресурсів, що робить їх доступними для інді-розробників. Застосування спеціалізованих рушіїв, таких як Ren'Py, спрощує процес створення інтерактивних сценаріїв навіть для користувачів із базовими знаннями програмування.

Крім того, зростає інтерес до ігор, орієнтованих на персоналізований досвід, де вибір гравця має значення, а персонажі взаємодіють із ним як повноцінні особистості. Саме тому ігри з елементами симуляції життя, романтичних взаємин і нелінійного сюжету активно приваблюють широку аудиторію, включаючи як молодь, так і дорослих гравців.

Отже, сучасний стан ігрової індустрії створює сприятливі умови для розробки візуальних новел як актуального напряму в інтерактивному наративному дизайні. Наявність готових рушіїв, відкритість інструментів і попит на емоційно насичені

історії забезпечують високий потенціал для реалізації подібних проєктів у навчальному й комерційному середовищі.

1.2. Тенденції та жанри візуальних новел

Візуальні новели — це жанр інтерактивних ігор, що поєднує текстовий наратив, візуальні елементи та, у багатьох випадках, озвучення персонажів. Основною особливістю таких ігор є нелінійна структура сюжету, яка дозволяє гравцеві впливати на події шляхом вибору варіантів відповіді чи дії. На відміну від класичних книг або навіть більшості комп’ютерних ігор, візуальні новели активно використовують розгалуження сценаріїв та множинні кінцівки, що робить їх близькими до інтерактивного кіно.

Станом на сьогодні спостерігається активний розвиток жанрового розмаїття у візуальних новелах [10]. Найпоширенішими жанрами є:

- Романтичні новели: гравець розвиває стосунки з одним або кількома персонажами. Такий тип сюжетів часто містить систему "відносин" або "прихильності".
- Драматичні історії: сюжет акцентує увагу на психологічному розвитку персонажів, емоційних конфліктах та складних моральних виборах.
- Фентезі та наукова фантастика: сюжет розгортається у вигаданих світах, часто з додатковими ігровими механіками (магія, бойові сцени, головоломки).
- Детективи та трилери: основна увага приділяється розслідуванню, пошуку підказок та розгадуванню таємниць.
- Жахи: використовуються візуальні та аудіоефекти для створення напруженої атмосфери.

Серед сучасних тенденцій у розробці візуальних новел можна виділити:

- Гейміфікація ігрового процесу: додавання додаткових механік (менеджмент ресурсів, прокачка параметрів, мінігри).
- Симуляція життя: поєднання візуальної новели з елементами симулятора повсякденності — розклад дня, обмеження часу, управління енергією чи навичками.

- Емоційна глибина та соціальні теми: усе більше розробників піднімають питання ідентичності, психології, етики та міжособистісних стосунків.
- Інтерактивний наратив на мобільних платформах: адаптація візуальних новел для смартфонів, що дозволяє ширшій аудиторії взаємодіяти з історіями в зручному форматі.
- Використання рушіїв з відкритим кодом: доступність інструментів сприяє поширенню інді-проєктів та аматорських ігор високої якості.

Завдяки таким тенденціям візуальні новели перетворюються на багатогранний інструмент для створення інтерактивного контенту — як розважального, так і навчального. Вони демонструють, що навіть за відсутності складної 3D-графіки чи екшен-геймплею, гравців приваблює можливість впливу на сюжет та розвиток емоційного зв’язку з персонажами.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз альтернативних рушіїв

На сучасному етапі розробки інтерактивних історій доступно чимало ігрових рушіїв, що дозволяють створювати візуальні новели різної складності — від простих сценаріїв до проєктів із розгалуженою логікою, графікою та анімацією. Кожен рушій має свої переваги та обмеження, які слід враховувати при виборі інструменту для реалізації конкретного проєкту.

Нижче подано короткий огляд найпоширеніших рушіїв, які використовуються для створення візуальних новел:

1. Ren'Py

Одним із найпопулярніших рішень є Ren'Py — рушій з відкритим кодом, що спеціалізується саме на створенні візуальних новел (Рисунок 2.1) [2]. Він поєднує власну просту сценарну мову з мовою програмування Python, що дозволяє реалізовувати складні сценарії, внутрішню логіку, змінні, умовні переходи, трекінг виборів гравця, систему збережень та інше.

Ren'Py підтримує додавання графіки, звукових ефектів, фонової музики, а також адаптується під різні платформи (Windows, Linux, macOS, Android) [9].

Його основною перевагою є низький поріг входу, активна спільнота та велика кількість навчальних матеріалів. Водночас, для складних візуальних елементів або кастомних інтерфейсів можуть знадобитися знання Python.

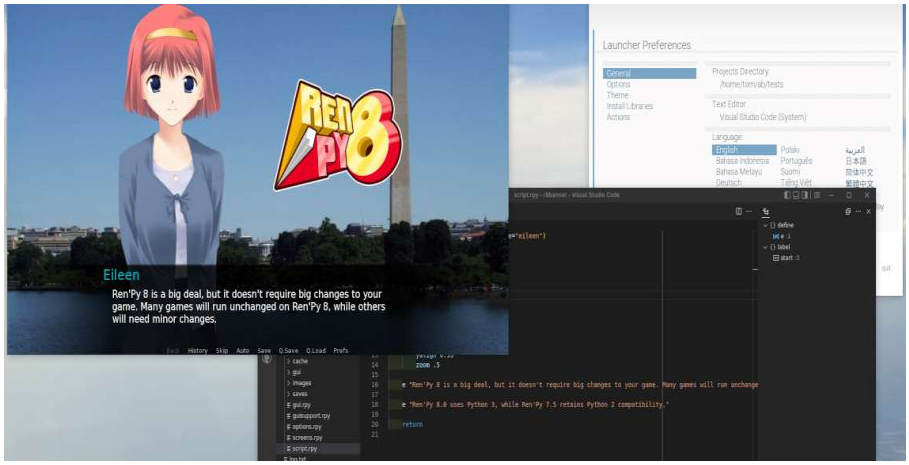


Рисунок 2.1 – Інтерфейс Ren'Py

2. Unity з додатками типу Fungus або Naninovel

Ще одним варіантом є Unity, універсальний ігровий рушій, який у поєднанні з відповідними плагінами (наприклад, Fungus або Naninovel) дозволяє створювати візуальні новели з високим ступенем кастомізації (Рисунок 2.2).

Unity підтримує 2D та 3D графіку, анімацію, візуальні ефекти та дозволяє інтегрувати додаткові механіки — мініігри, симуляції, інвентарі тощо. Однак розробка у Unity передбачає знання мови C# та складніший процес налаштування, а також більші розміри фінального продукту.

Цей рушій підходить для проєктів, де візуальна новела є частиною більшого геймплейного досвіду або коли потрібна широка технічна свобода [5].

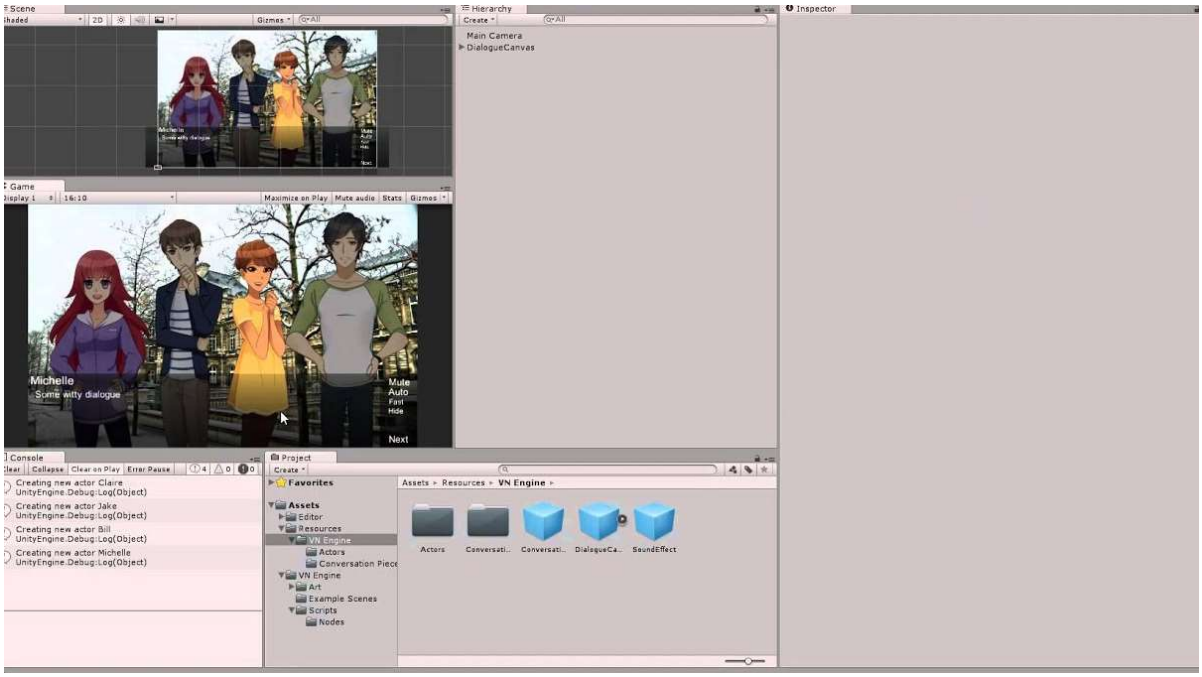


Рисунок 2.2 – Інтерфейс Unity

3. TyranoBuilder

Для початківців або невеликих проєктів зручним варіантом є TyranoBuilder (Рисунок 2.3).

Це рушій із графічним інтерфейсом, який дозволяє створювати візуальні новели без програмування. Його функціональність базується на JavaScript і HTML, що дає змогу експортувати проєкти для запуску в браузері або на мобільних пристроях [6].

Однак можливості кастомізації в TyranoBuilder досить обмежені, а для реалізації складної логіки доводиться вдаватися до ручного редагування коду. Це робить рушій привабливим для швидкої розробки простих історій, але менш ефективним у реалізації нестандартного геймплею.

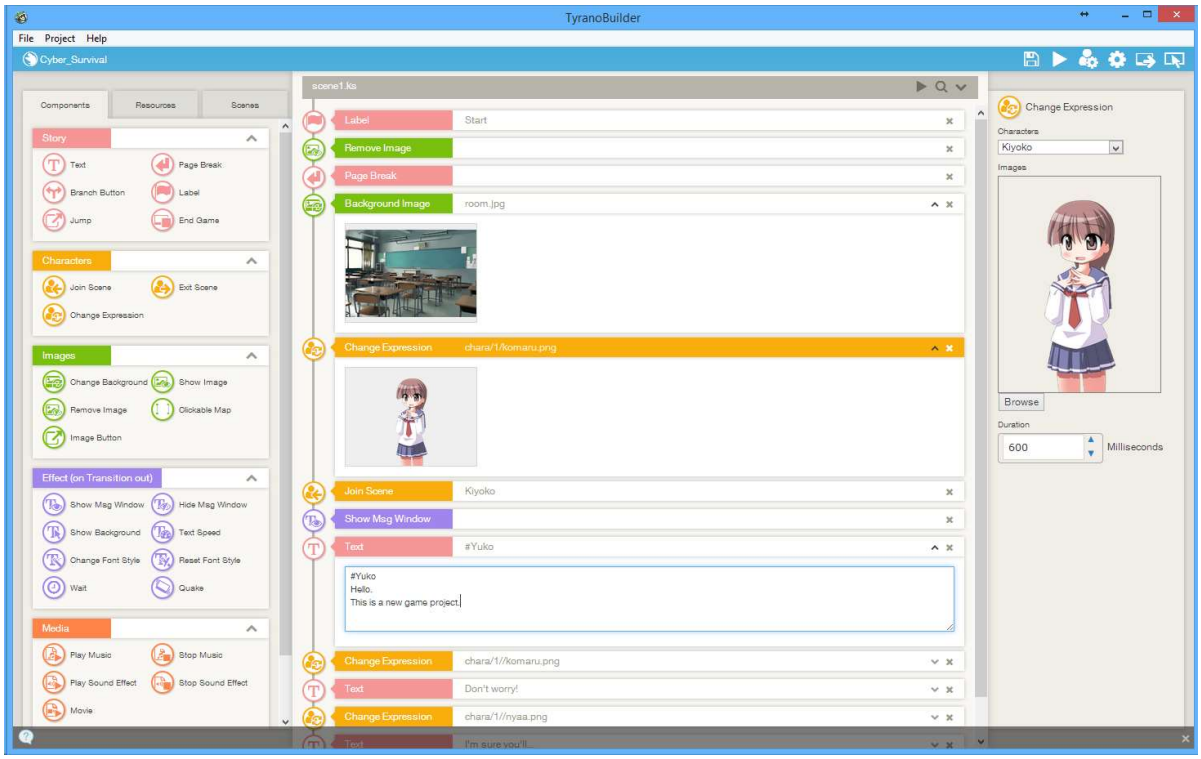


Рисунок 2.3 – Інтерфейс TyranoBuilder

4. Visual Novel Maker

Схожим за принципом є Visual Novel Maker, який розроблений компанією Degica — творцями RPG Maker (Рисунок 2.4).

Він має зручний інтерфейс, а також дозволяє редагувати внутрішню логіку за допомогою JavaScript. До переваг належить підтримка шаблонів сцен, бібліотека персонажів, автоматизовані анімації та мультиплатформеність. Проте програмний продукт є комерційним і має певні обмеження у вільному поширенні готових проєктів, що може бути критичним для інді-розробників [7].

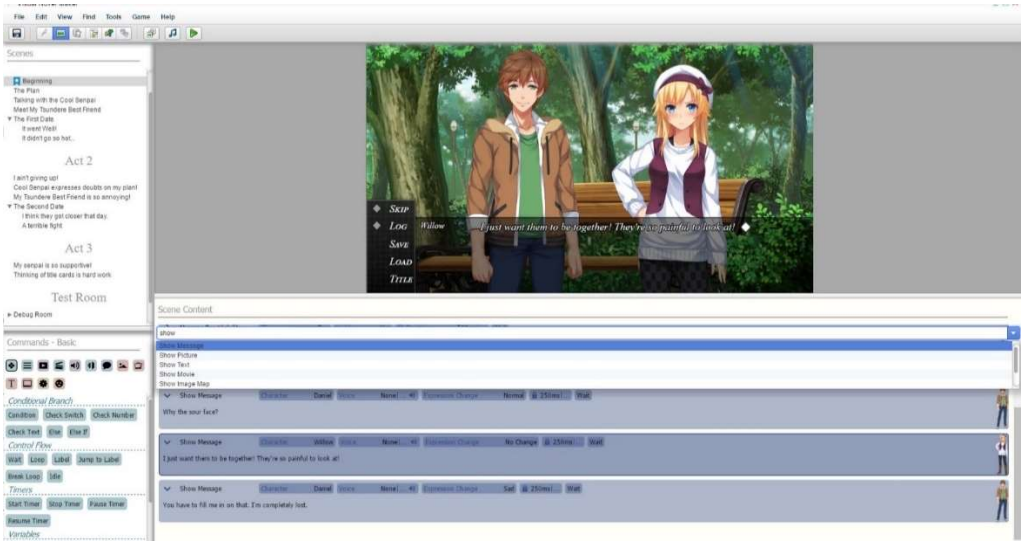


Рисунок 2.4 – Інтерфейс Visual Novel Maker

5. Twine

Для текстоцентричних історій варто згадати Twine — інструмент для створення інтерактивної літератури у форматі HTML [8].

Його основна перевага — надзвичайна простота використання, можливість створення нелінійних сценаріїв через вузлову систему та миттєвий запуск у браузері. Хоча Twine і не орієнтований на класичні візуальні новели (він не має вбудованої підтримки графіки, звуку або інтерфейсів), його можна розширити за допомогою CSS та JavaScript.

Twine є добрим варіантом для реалізації діалогових структур і складних сюжетів без акценту на візуальному оформленні (Рисунок 2.5).

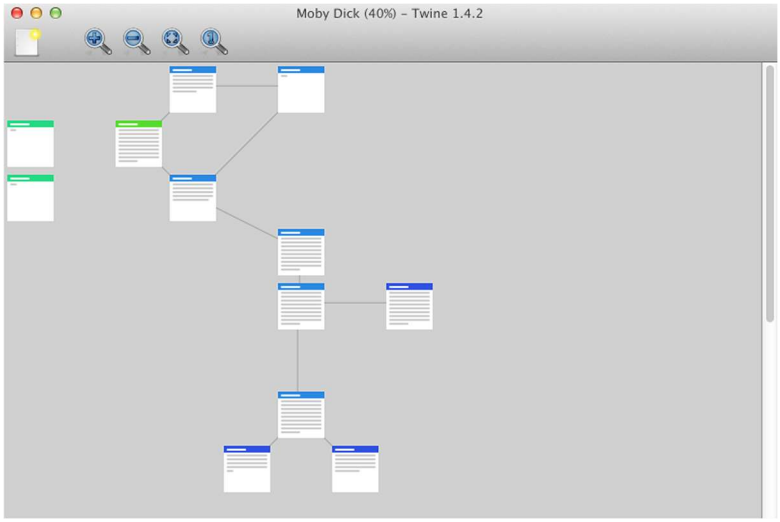


Рисунок 2.5 – Інтерфейс Twine

Серед розглянутих рушіїв Ren'Py залишається найбільш придатним для створення візуальних новел, орієнтованих на сюжет, розвиток стосунків між персонажами та навчальні елементи. Його поєднання простоти, гнучкості та активної спільноти забезпечує зручне середовище для як новачків, так і досвідчених розробників. Інші рушії можуть бути доцільнішими у випадках, коли проєкт потребує складних анімацій, 3D-графіки чи більш специфічних механік.

2.2. Мова програмування Python

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією [4]. Розроблена на початку 1990-х років Гвідо ван Россумом.

Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, аспектно-орієнтована та функціональна.

Python підтримує динамічну типізацію, тобто, тип змінної визначається лише під час виконання. З базових типів слід зазначити підтримку цілих чисел довільної довжини і комплексних чисел. Python має багату бібліотеку для роботи з рядками, зокрема, кодованими в юнікоді.

З колекцій Python підтримує кортежі (tuples), списки (масиви), словники (асоціативні масиви) і від версії 2.4, множини.

Система класів підтримує множинне успадкування і метапрограмування. Будь-який тип, включаючи базові, входить до системи класів, й за необхідності можливе успадкування навіть від базових типів.

Дизайн мови Python побудований навколо об'єктно-орієнтованої моделі програмування. Реалізація ООП в Python є елегантною, потужною та добре продуманою, але разом з тим, достатньо специфічною в порівнянні з іншими об'єктно-орієнтованими мовами.

Можливості та особливості:

- Класи є одночасно об'єктами з усіма нижче наведеними можливостями.
- Успадкування, в тому числі множинне.
- Поліморфізм (всі функції віртуальні).
- Інкапсуляція (два рівні — загальнодоступні та приховані методи і поля). Особливість — приховані члени доступні для використання та помічені як приховані лише особливими іменами.
- Спеціальні методи, що керують життєвим циклом об'єкта: конструктори, деструктори, розподільники пам'яті.
- Перевантаження операторів (усіх, крім is, '.', '=' і символьних логічних).
- Властивості (імітація поля за допомогою функцій).
- Управління доступу до полів (емуляція полів і методів, частковий доступ тощо).
- Методи для управління найпоширенішими операціями (істиннісне значення, len(), глибоке копіювання, серіалізація, ітерація по об'єкту, …)
- Метапрограмування (управління створенням класів, тригери на створення класів, та ін)
- Повна інтроспекція.
- Класові та статичні методи, класові поля.
- Класи, вкладені у функції та інші класи

2.3. Ren’Py

2.3.1. Сценарна мова Ren’Py

Сценарна мова Ren'Py є доменно-специфічною мовою, що призначена для опису діалогів, подій, переходів між сценами та інтерактивних виборів [3]. Її основні

конструкції інтуїтивно зрозумілі та дозволяють швидко створювати нелінійний сюжет.

Основні елементи мови (Рисунок 2.6):

- label — маркує окремі частини сценарію, до яких можна звертатися.
- menu — дозволяє реалізовувати вибори гравця, що впливають на подальший розвиток сюжету.
- show, hide, scene — команди для управління зображеннями персонажів і фону.
- say — конструкція для виведення реплік персонажів.
- jump, call, return — управління переходами між частинами сюжету.
- if, elif, else — умовні оператори, які дозволяють змінювати хід сюжету залежно від значень змінних.

```
label start:
    scene bg classroom
    show eileen happy
    "Привіт, я – Ейлін!"
    "Ласкаво просимо до Ren'Py!"
    return
```

Рисунок 2.6 – Приклад сцени

Сцени створюються наступним чином:

- Сцена створюється за допомогою команди label name.
- Після цього, додається задній фон для сцени, використовуючи команду show bg name
- Далі, при необхідності на екран виводиться персонаж за допомогою команди show character
- Усі діалоги, монологи та наратив описується за допомогою тексту всередині лапок. Кожен новий рядок тексту відображається як нова репліка.
- Якщо гравцю необхідно обрати варіант подій, використовується конструкцію menu (Рисунок 2.7).

- Переходи між сценами відбуваються за допомогою команд call name або jump name. Call викликає наступну сцену як функцію, після відтворення якої гравець повертається до сцени, яка зробила виклик. Jump виконує “стрибок” з однієї сцени на іншу, не завершуючи її до кінця.
- Для завершення сцени, використовується return. Якщо завершена сцена є останньою в черзі – гра закінчується і виконується повернення на головний екран.

```
menu:
    "Що ти хотів би придбати?"
    "Енергетики (75 гривень)": ···
    "Обід (180 гривень)": ···
    "Конспекти (300 гривень)": ···
    "Шпаргалки (1000 гривень)": ···
    "Вийти": ···
```

Рисунок 2.7 – Приклад застосування menu

2.3.2. Структура проєкту та організація файлів

Проєкт, реалізований у середовищі Ren'Py, має чітко визначену файлову структуру, яка дозволяє впорядкувати сценарні файли, графіку, музику та інші ресурси. Структура каталогу проєкту сприяє зручності розробки, тестування та підтримки гри.

Після створення проєкту Ren'Py автоматично формує базову структуру, до якої в процесі розробки додаються додаткові каталоги й файли відповідно до потреб проєкту (Рисунок 2.8). Проєкт розпочинається з кореневої папки, яка містить файли-застосунки папки renpy та lib з необхідними системними файлами та бібліотеками, і папку game. Папка game – це основна директорія, де розміщується весь контент гри, а саме файли-скрипти, зображення, аудіо- та відеофайли тощо.

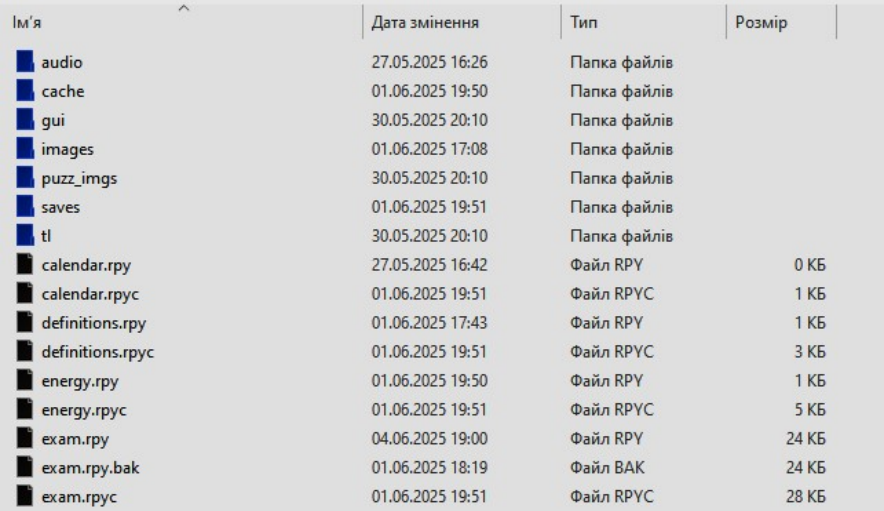
game	01.06.2025 19:53	Папка файлів	
lib	08.12.2024 4:02	Папка файлів	
renpy	08.12.2024 4:02	Папка файлів	
log.txt	01.06.2025 19:53	Текстовий докум...	2 КБ
SchoolLife.exe	08.12.2024 4:02	Застосунок	102 КБ
SchoolLife.py	08.12.2024 4:02	Python Source File	9 КБ
SchoolLife.sh	08.12.2024 4:02	SH Source File	2 КБ

Рисунок 2.8 – Структура кореневої папки

При першопочатковому створенні проєкту, створюються файли script.rpy, options.rpy, gui.rpy та screens.rpy:

- script.rpy — це головний сценарний файл, який містить основну частину сюжету та керування ігровою логікою.
- options.rpy — це файл, який містить налаштування назви гри, авторів, іконок, параметрів вікна та GUI.
- gui.rpy — це файл, що відповідає за візуальне оформлення інтерфейсу користувача: шрифт, кольори, положення елементів меню тощо.
- screens.rpy — файл, в якому визначаються зовнішній вигляд та поведінка екранів гри (головне меню, меню збережень, діалогові вікна тощо).

При необхідності, розробник може змінювати ці файли та додавати нові – Ren'Py читає усі файли з розширеннями .rpy та .rpyc при запуску гри (Рисунок 2.9).



Ім'я	Дата змінення	Тип	Розмір
audio	27.05.2025 16:26	Папка файлів	
cache	01.06.2025 19:50	Папка файлів	
gui	30.05.2025 20:10	Папка файлів	
images	01.06.2025 17:08	Папка файлів	
puzz_imgs	30.05.2025 20:10	Папка файлів	
saves	01.06.2025 19:51	Папка файлів	
tl	30.05.2025 20:10	Папка файлів	
calendar.rpy	27.05.2025 16:42	Файл RPY	0 КБ
calendar.rpyc	01.06.2025 19:51	Файл RPYC	1 КБ
definitions.rpy	01.06.2025 17:43	Файл RPY	1 КБ
definitions.rpyc	01.06.2025 19:51	Файл RPYC	3 КБ
energy.rpy	01.06.2025 19:50	Файл RPY	1 КБ
energy.rpyc	01.06.2025 19:51	Файл RPYC	5 КБ
exam.rpy	04.06.2025 19:00	Файл RPY	24 КБ
exam.rpy.bak	01.06.2025 18:19	Файл BAK	24 КБ
exam.rpyc	01.06.2025 19:51	Файл RPYC	28 КБ

Рисунок 2.9 – Структура папки game

В папкці game також містяться папки images, gui та saves. Папка images містить зображення, що використовуються в грі, а саме фони, зображення персонажів, та будь-які інші зображення, які будуть використовуватися у грі. Папка gui містить зображення елементів інтерфейсу. Аналогічно до файлів-сценаріїв, розробник може створювати додаткові папки з різноманітними ресурсами, які Ren'Py самостійно знайде в директорії гри.

Папка saves автоматично створюється при запуску гри та містить усі збереження гравця, що були створені в грі.

2.3.3. Математичні моделі та логіка проєкту

Хоча проєкт належить до жанру візуальних новел, він включає ряд логічних та математичних моделей, що забезпечують нелінійність сюжету, розвиток відносин між персонажами, відстеження стану гравця та внутрішньоігровий прогрес.

Гра поділена на ігрові дні та часові проміжки (ранок, день, вечір). Поточний день і час зберігаються у змінних, що дозволяє організувати події за розкладом, вводити обмеження на доступність дій або персонажів.

Це дозволяє планувати події, наприклад: "Персонаж доступний лише ввечері другого дня", або "Подія стається на 5-й день".

Для підвищення реалізму ігрового процесу та глибшої симуляції повсякденного життя студента в грі реалізовано три ключові ресурси:

- енергія — відображає фізичний стан гравця,
- стрес — емоційне навантаження,
- гроші — фінансовий ресурс, який дозволяє купувати речі чи отримувати додаткові можливості.

Енергія зменшується під час виконання активностей, таких як навчання та робота. Якщо рівень енергії падає до низького значення, деякі дії стають недоступними.

Стрес накопичується в результаті складних активної діяльності, наприклад роботі та навчанні. Коли стрес накопичується до критичної відмітки, гравець на наступний день отримує погану кінцівку, в якій герой “вигорає” та вирішує закінчити навчання. Гравець повинен знаходити способи його зниження — відпочинок.

Гроші є універсальним ресурсом, що витрачається на різні покупки. Отримати гроші можна шляхом підробітку у вільний час.

Взаємозв’язок трьох ресурсів створює баланс між роботою, навчанням, соціальними зв’язками та особистим благополуччям гравця. Це підсилює ігрову стратегію та робить проходження більш різноманітним, наближаючи геймплей до симулятора життя.

Ключові вибори гравця зберігаються у булевих або числових змінних, що дозволяє повертатися до них у подальших сценах та враховувати їх у сюжеті.

Таким чином, у проєкті реалізована логіка поведінки персонажів, відстеження станів та модель впливу вибору гравця, що забезпечує варіативність і повторюваність проходження. Ці механіки створюють ефект живої взаємодії з героями, що є ключовим елементом жанру симулятора стосунків.

Гравець, відвідуючи лекції викладачів з певної дисципліни, будує “репутацію” з цим викладачем. При досягненні певної межі, викладач стає більш лояльнішим до головного героя, що дозволить йому відкрити нові можливості в подальшому.

На фінальному етапі проходження гри, після завершення внутрішньоігрового семестру, гравцеві потрібно скласти сесію з кількох дисциплін. Механіка сесії розроблена як міні-гра у форматі тестування, де користувач відповідає на запитання з кількома варіантами відповідей. Кожна правильна відповідь приносить певну кількість балів.

Структура сесії виглядає наступним чином: сесія складається з 5 іспитів з дисциплін, які герой вивчав на протязі гри. Кожен іспит складається з 5 питань, які мають 4 варіанти відповідей. Якщо гравець відповів правильно – він отримує бал, якщо ж він відповів неправильно – ні. Після цього, до цих балів додаються додаткові, які гравець зміг отримати від його іншої діяльності. Для проходження іспиту, гравцю необхідно отримати хоча б половину з максимальних балів.

Після здачі кожного з іспитів, робляться підсумки. Якщо головний герой пройшов хоча б 3 із 5 іспитів успішно, сесія вважається успішно пройденою, та гравцю показується хороша кінцівка. Якщо ж було пройдено успішно менше 3 іспитів, гравця відправляють на перездачу, на якій йому необхідно буде перездати іспити з незадовільним результатом. Після цього, гравцю ще раз покажуть результати іспитів, і якщо він досі не здав достатню кількість іспитів – він отримує погану кінцівку.

Після здачі сесії, на основі їх результатів гравець може отримати стипендію. Отримання стипендії залежить від декількох факторів:

- Чи гравець успішно закрив сесію з першої спроби;
- Чи гравець успішно здав усі іспити;
- Чи гравець отримав достатній середній бал зі всіх іспитів.

Якщо гравець зміг дотриматися всіх цих умов, йому нараховують стипендію, яку він зможе витратити, як йому заманеться.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ.

3.1. Вимоги до технічного та програмного забезпечення

У процесі розробки та експлуатації інтерактивної візуальної новели важливо врахувати як вимоги до середовища розробки, так і до кінцевого пристрою користувача, на якому буде запускатися гра. Також визначаються необхідні компоненти програмного забезпечення.

Для повноцінної роботи над проєктом розробнику необхідні такі компоненти:

- Операційна система: Windows 10 або новіша, macOS 10.14+, Linux (будь-який дистрибутив з підтримкою Python);
- Ren'Py SDK: версія 8.1 або новіша;
- Текстовий редактор: Visual Studio Code;
- Процесор: Intel Core i5 або потужніший;
- Оперативна пам’ять: від 8 ГБ;
- Графічна карта: NVIDIA GTX 950 або інтегрована з підтримкою OpenGL 3.0+;
- Місце на диску: 1 ГБ або більше (враховуючи ресурси та резервні копії);
- Роздільна здатність екрану: 1440x900 або вища для роботи з інтерфейсами.

Зібрана гра оптимізована для запуску на пристроях з наступними мінімальними характеристиками:

- Процесор: 2-ядерний Intel/AMD з частотою 2.0 GHz;
- Оперативна пам’ять: 4 ГБ і більше;
- Відеокарта: з апаратним прискоренням OpenGL 3.0 або вище;
- Диск: SSD для швидкого завантаження сцен і збережень;
- Операційна система: Windows 10/11;
- Роздільна здатність екрана: 1440×900.

3.2. Опис структури, логіки та програмної реалізації

3.2.1. Структура проєкту

Структура проєкту поділяється на логічні модулі, які забезпечують керованість, масштабованість та зручність у розробці.

Основні компоненти гри поділені на наступні файли в каталозі game:

- calendar.rpy – містить основну логіку календарної системи та екран відображення дати
- definitions.rpy – містить ініціалізацію певних об’єктів, таких як персонажі та медіа-об’єкти (задні фони, музика, звуки тощо)
- energy.rpy – задає логіку енергії та стресу, а також створює відповідні екрани
- exam.rpy – відповідальний за логіку сесії в кінці гри, питання на кожному із іспитів, та кінцівки після сесії.
- gui.rpy – описує основний дизайн інтерфейсу
- lectures.rpy – містить логіку проведення лекцій та репутації викладачів
- money.rpy – задає логіку магазину, оплату проживання в гуртожитку, нарахування стипендії та нарахування грошей
- numbers.rpy – міні-гра, що відбувається при роботі днем
- options.rpy – файл з налаштуваннями проєкту
- puzzles.rpy – міні-гра, що відбувається при роботі вечером
- screens.rpy – файл з основними екранами проєкту
- script.rpy – містить основну логіку гри
- study.rpy – описує процес навчання вдома/в бібліотеці

3.2.2. Основна логіка гри

Ігрова логіка побудована навколо симуляції життя студента впродовж обмеженого періоду (14 днів до початку сесії), під час якого гравець приймає рішення, що впливають на ресурси та розвиток. У центрі — взаємодія між сценарієм, механікою часу, ресурсами гравця та реакцією ігрового світу.

Гра поділена на ігрові дні, кожен з яких включає 4 часових відрізки:

- Ранок — автоматична подія (підготовка до навчання або сон),
- День — навчання,

- Після навчання — вибір дій (робота, зустріч, підготовка),
- Вечір — відпочинок, дозвілля або повторна дія.

У кожному слоті гравець має можливість обирати дії з декількох запропонованих варіантів. Вибір дії впливає на:

- зміни внутрішніх ресурсів (енергії, стресу, грошей),
- внутрішню статистику для фінального іспиту,
- подальший хід подій та доступність сцен.

Дії гравця впливають на його ресурси:

- Енергія зменшується при роботі, навчанні або активному проведенні часу. Відновлюється сном або відпочинком. Низький рівень енергії блокує певні дії
- Стрес підвищується під час інтенсивних занять і зменшується при відпочинку. Високий рівень стресу може спричинити кінець гри.
- Гроші потрібні для покупки розхідників та заробляються через підробіток.

Візуальна новела реалізована на базі сценарних вузлів (label), між якими гравець переміщується через вибори. Деякі сцени доступні лише при досягненні певних умов (наприклад, достатня підготовка або конкретний день гри).

Кожен вибір має наслідки: змінює ресурси, відкриває або блокує наступні сцени, модифікує внутрішні змінні, що враховуються при фінальних іспитах або епілозі гри.

У фіналі гравець проходить кілька іспитів. Результати залежать від рівня підготовки та залишкових ресурсів. Іспити подаються як тести з варіантами відповідей. Успішне проходження відкриває кращий фінал.

3.2.3. Реалізація календарної системи

Календарна система є одним із ключових елементів ігрової логіки, оскільки дозволяє структуровано організувати сюжет, впливати на ігрову динаміку та обмежувати дії гравця у часі. В рамках проєкту календар використовується для відліку часу до іспитів, визначення подій на певні дні та контролю доступності сюжетних сцен.

Гра триває 14 ігрових днів, розбитих на чотири часових відрізки: ранок, день, після навчання, вечір. Внутрішня реалізація календаря базується на таких змінних

- current_day – визначає номер ігрового дня з початку гри
- day_index – вказує на день тижня
- time_of_day – показує часовий відрізок
- max_days – максимальна кількість днів до іспитів

Кожен день – це цикл, що виглядає наступним чином (Рисунок 3.1).

```
label day_loop:

    call day_info

    call time_segment("ранок")
    if current_day > max_days:
        $ exams_tried = True
        $ exam_scores = {}
        call exam_session
    elif passed_count < 3 and exams_tried == True:
        call retake_exams
    else:
        call time_segment("день")
        if day_index < 5:
            call time_segment("після навчання")
        else:
            call time_segment("день")
        call time_segment("вечір")
```

Рисунок 3.1 – Структура дня

Тут, спочатку викликається сцена day_info, в якій розпочинається новий день та дається інформація, скільки днів лишилось до сесії. Після цього, якщо сесія ще не наступила, викликаються сцени time_segment з агрументом, який відповідає за часовий відрізок, що буде використаний в сцені. В залежності від нього, гравець або відвідує пари, або проводить час як йому заманеться. Якщо наступила пора сесії, гравець переноситься на сцену exam_session, де буде відбуватись

Зміна дня відбувається після завершення вечірнього слоту. Після кожного повного дня викликається функції, що відповідають за оновлення календаря (Рисунок 3.2).

```
$ current_day += 1
$ hud_date = update_hud_date()
$ day_index = (current_day - 1) % 7
jump day_loop
```

Рисунок 3.2 – Зміна дня

Вони змінюють змінюють день на наступний, оновлюють дату на ігровому екрані та змінюють день тижня.

Для зручності гравця передбачено екран-календар, на якому відображаються поточна дата та день тижня (Рисунок 3.3).

```
screen hud():
    frame:
        xalign 0.98
        yalign 0.02
        padding (10, 10)
        background "#0008"

        vbox:
            spacing 3
            text "[hud_date]" size 22 color "#fff"
            text "Час: [time_of_day]" size 20 color "#ccc"
```

Рисунок 3.3 – HUD календаря

Інтерфейс реалізований через кастомний екран (screen hud), що використовує рамку (frame) з вертикальною коробкою (vbox), в якій час відображається у форматі: День тижня, дата; Час: часовий відрізок (Рисунок 3.4).

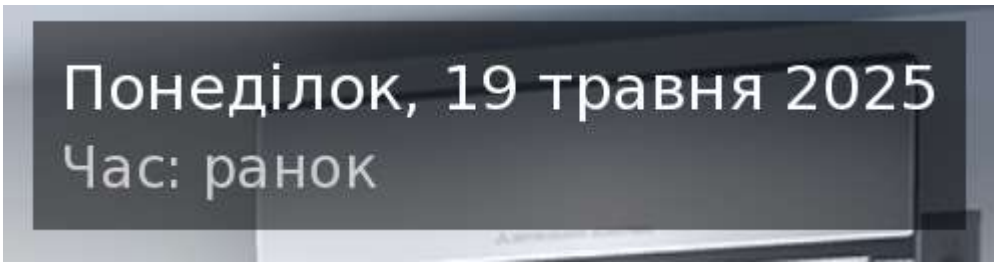


Рисунок 3.4 – Вигляд календаря

Після завершення останнього (14-го) дня запускається сцена складання іспитів. Календар виконує роль "таймера", який підштовхує гравця планувати свої дії заздалегідь і ефективно розподіляти ресурси.

3.2.4. Реалізація системи ресурсів

Система ресурсів є важливою частиною ігрової логіки, що впливає на ігровий процес, прийняття рішень і розвиток подій. У розробленій грі реалізовано три основні ресурси: енергія, стрес та гроші. Вони змінюються в залежності від дій гравця, відображають внутрішній стан персонажа та можуть відкривати або блокувати певні ігрові можливості (Рисунок 3.5).

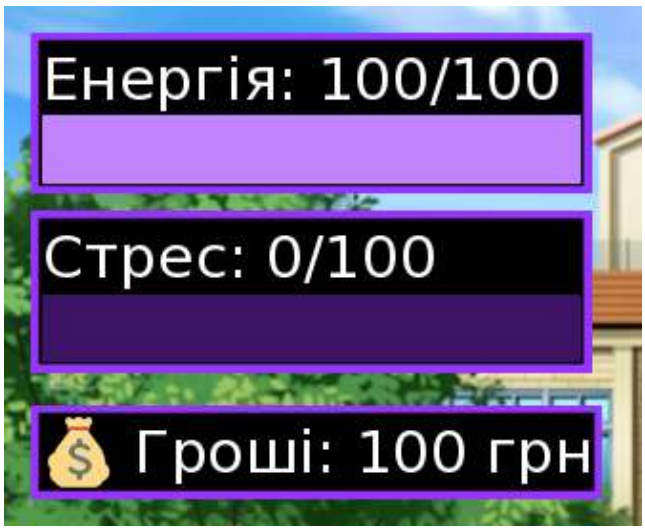


Рисунок 3.5 – HUD ресурсів гравця

Енергія показує рівень фізичної та ментальної витривалості персонажа. Вона витрачається під час навчання, роботи та відвідування лекцій, і відновлюється під час сну або відпочинку. Якщо рівень енергії низький, гравець не може виконувати дії, що потребують витрат, до моменту відпочинку, а також не може зосередивсь на лекціях та під час навчання, втрачаючи корисну інформацію, яку він зможе застосувати на іспиті.

Стрес відображає емоційний стан персонажа. Він накопичується під час перевантаження навчанням та роботою, і знижується внаслідок відпочинку. Високий рівень стресу викликає погану кінцівку, в якій головний герой вирішує призупинити своє навчання.

Гроші використовуються для різноманітних предметів, які можуть підвищити рівень енергії та підготовки до іспитів, а також оплатити проживання в гуртожитку. Якщо у гравця немає достатньої кількості коштів, він не зможе купляти нові предмети, а також буде змушений виселитись з гуртожитку (що зменшує кількість вільного часу). Гроші можна заробити, працюючи кур’єром (у денний час), виконуючи лабораторні лабораторні роботи молодшокурсникам (у вечірній час), а також отримуючи стипендію за успіхи у навчанні.

Усі ці ресурси зберігаються як змінні, а їх інтерфейс оновлюється динамічно за допомогою екрану HUD, який відображається у верхній частині екрана та автоматично оновлюється після кожної дії (Рисунок 3.6):

```
screen hud_money():
    frame:
        xalign 0.05
        yalign 0.25
        has vbox
        text "💰 Гроші: [player_money] грн"
screen energy_bar():
    frame:
        xalign 0.05
        yalign 0.05
        has vbox
        text "Енергія: [energy]/100"
        bar value energy range 100 xmaximum 200

screen stress_bar():
    frame:
        xalign 0.05
        yalign 0.15
        has vbox
        text "Стрес: [stress]/100"
        bar value stress range 100 xmaximum 200
```

Рисунок 3.6 – Екрани ресурсів

Система ресурсів створює умови для стратегічного планування: гравець не може виконувати всі дії поспіль, тому змушений розставляти пріоритети. Наприклад, обрати між додатковим підробітком, відпочинком і навчанням.

Це дозволяє забезпечити варіативність проходження, підвищує реіграбельність та робить геймплей більш динамічним і насиченим.

3.2.5 Реалізація системи іспитів та репутації

Іспити є фінальним ігровим викликом, що підсумовує вибір гравця, його підготовку протягом усієї гри та використання внутрішніх ресурсів. Механіка іспитів покликана не лише перевірити ігрові показники, а й залучити гравця до коротких інтерактивних тестів, які створюють ілюзію реального заліку чи сесії.

У грі передбачено кілька дисциплін, кожна з яких завершується іспитом (Рисунок 3.7). Підготовка до них відбувається поступово: гравець може відвідувати лекції, вивчати матеріал самостійно або ігнорувати предмет, що вплине на фінальний результат. Оцінка з кожної дисципліни складається з 2 частин: правильних відповідей, які гравець дав під час тестування, а також з додаткової підготовки до кожного з іспитів. Додаткова підготовка може включати в себе:

- Відвідування лекцій;
- Самостійна підготовка в бібліотеці днем;

- Читання лекційного матеріалу вечором;
- Покупки конспектів у старшокурсників.

```
label exam_session:
    $ time_of_day = "день"
    scene bg classroom day with fade

    "Наступний іспит: Англійська"
    call exam("Англійська") from _call_exam

    "Наступний іспит: ООП"
    call exam("ООП") from _call_exam_1

    "Наступний іспит: Дискретна математика"
    call exam("Дискретна математика") from _call_exam_2

    "Наступний іспит: Комп'ютерні мережі"
    call exam("Комп'ютерні мережі") from _call_exam_3

    "Наступний іспит: Патерни проєктування"
    call exam("Патерни проєктування") from _call_exam_4

    jump exams_final
```

Рисунок 3.7 – Послідовність іспитів

Гравець може накопичити до трьох додаткових балів з кожної дисципліни таким способом.

Кожен іспит складається з 5 питань, кожне з яких має 4 варіанти відповідей, з яких тільки одна – правильна (Рисунок 3.8). За кожну правильну відповідь гравець отримує бал, за кожну неправильну – ні. Таким чином, гравець може набрати 5 балів за відповіді на іспиті, і 3 додаткових бали за підготовку. Отже, загалом можна отримати 8 балів за іспит, з яких необхідно набрати хоча б половину, щоб успішно здати іспит. Незалежно від результату, після завершення гравець зразу почне роботу над наступним іспитом, повторюючи процес.

```
screen exam_question_screen(question):
    modal True
    tag exam

    frame:
        xalign 0.5
        yalign 0.5
        padding (20, 20)
        background "#222a"
        has vbox

        text question["question"] size 30 color "#fff"

        for i, answer in enumerate(question["choices"]):
            button:
                text answer
                action Return(i)
                xminimum 300
                padding (10, 5)
```

Рисунок 3.8 – Екран запитань

Окрім навчання, гравець може виконати наступні дії, щоб додатково гарантувати здачу іспиту:

- Використати шпаргалку на іспиті, щоб автоматично відповісти на всі тестові запитання
- Мати достатню репутацію з викладачем, який зможе “підтягнути” оцінку

Якщо гравець має хоча б одну куплену шпаргалку, він може використати її під час іспиту (Рисунок 3.9).

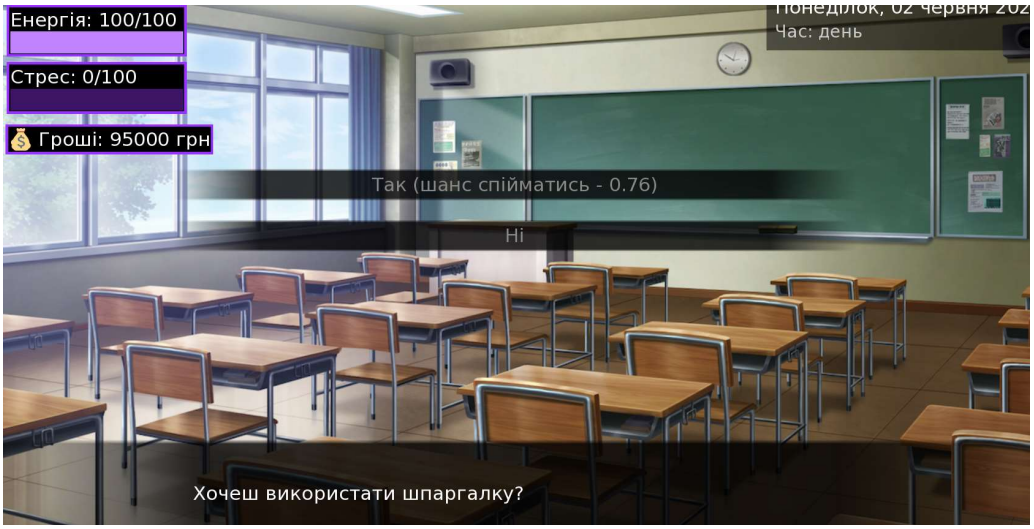


Рисунок 3.9 – Використання шпаргалки

Використання шпаргалки має певний шанс, що гравець буде “спійманий”. Шанс цього генерується випадково за допомогою RNG, але його можна зменшити, відвідуючи лекцію і збільшуючи репутацію з викладачем (Рисунок 3.10). Якщо гравець успішно використав її – усі питання на іспиті будуть зараховані, як правильні. Якщо ж він був спійманим – він автоматично провалює іспит, отримуючи 0 балів.

```
"Ти згадуєш, що в тебе є шпаргалка, яку ти можеш використати"
python:
    loyalty_decrease = min(0.5, 0.075 * lecture_attendance[subject])
    random_chance = round(random.random(), 2)
    caught_chance = max(round(random_chance - loyalty_decrease, 2), 0.05)
menu:
    "Хочеш використати шпаргалку?"
    "Так (шанс спійматись - [caught_chance])":
        $ cheat_sheets -= 1
        "Ти намагаєшся витягнути шпаргалку та непомітно списати..."

        python:
            got_caught = random.random() <= caught_chance
        if got_caught:
            "Ой-ой! Шпору знайшли!"
            python:
                scholarship_allpassed = False
                score = 0
                passed = score >= total * 0.5
                exam_results[subject] = passed
                exam_scores[subject] = score
            "Тебе одразу вигнали з іспиту та відправили на перездачу..."
        else:
            $ score = 5
            "Фух! Повезло непомітно списати всі основні тестові питання!"

    "Ні":
        "Ти видихаєш і починаєш роботу над іспитом..."

        python:
            for q in questions:
                choice = renpy.call_screen("exam_question_screen", q)
                if choice == q["answer"]:
                    score += 1
```

Рисунок 3.10 – Логіка використання шпаргалки

Гравець може будувати репутацію з викладачами, відвідуючи лекції з високим запасом енергії. Кожне відвідування лекції збільшує репутацію з викладачем даної дисципліни на 1 бал. Якщо гравець набрав хоча б 5 балів, він отримує лояльність даного викладача, що зменшує шанси бути пійманим на списуванні. Окрім цього, даний викладач в кінці іспиту збільшить кількість отриманих балів, що підвищить загальну оцінку (Рисунок 3.11).

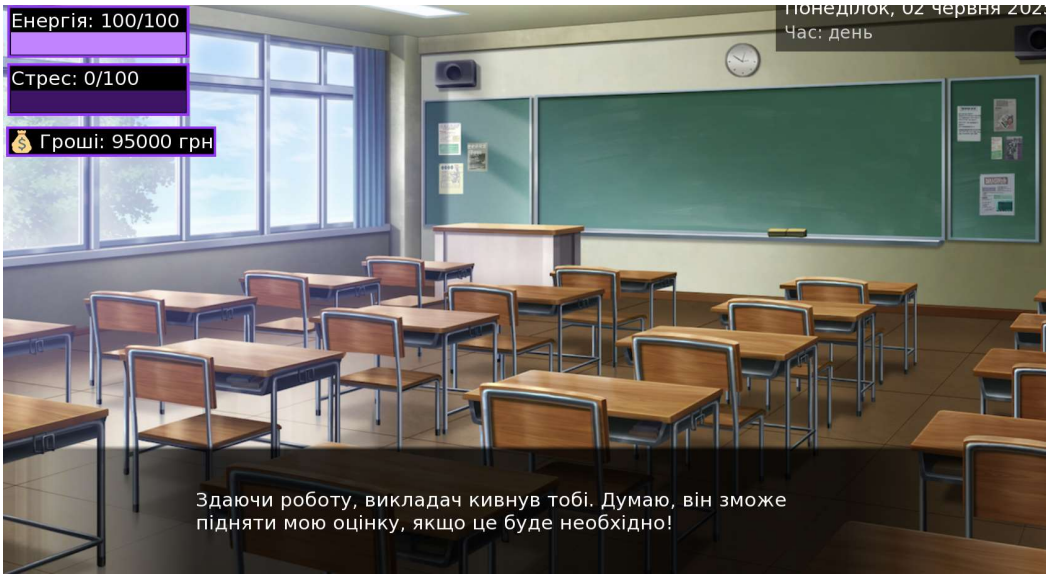


Рисунок 3.11 – Результати високої репутації

Після завершення усіх 5 іспитів, ідуть підсумки цієї сесії (Рисунок 3.12):

- Якщо гравець успішно здав хоча б 3 іспити, для нього сесія буде вважатись успішно завершеною, після чого йде хороша кінцівка;
- Якщо гравець склав менше трьох іспитів, його відправляють на перездачу тих іспитів, які він не зміг здати;
- Якщо ж після перездачі все одно не було здано хоча б 3 іспити, гравцю показується погана кінцівка, де його відраховують з університету.

```
label exams_final:

    scene bg school corridor day with fade
    "Твій підсумок сесії:"

    python:
        for subject in exam_data.keys():
            if exam_results.get(subject, False):
                renpy.say(None, f"✔ {subject} — ЗДАНО")
                passed_count += 1
            else:
                renpy.say(None, f"✘ {subject} — НЕ ЗДАНО")

    if passed_count >= 3:
        "Ти здав щонайменше 3 іспити. Вітаємо!"
        call check_scholarship from _call_check_scholarship
        call check_dormpayment from _call_check_dormpayment
        jump good_ending
    else:
        "Ти здав менше 3 іспитів. Треба перездати провалені..."
        "Втомленим, ти зразу же повертаєшся до гуртожитку..."
        jump retake_exams_prep
```

Рисунок 3.12 – Логіка підсумків сесії

Після успішної здачі сесії, гравець може отримати стипендію за успіхи в навчанні (Рисунок 3.13). Для цього йому необхідно виконати наступні умови:

- Успішно здати всі іспити з першої спроби;
- Отримати середній бал в 6.5 зі всіх іспитів.

```
label check_scholarship:

    $ average_score = sum(exam_scores.values()) / len(exam_scores) if exam_scores else 0
    $ avg_high = average_score >= 6.5

    if avg_high and scholarship_allpassed:
        $ scholarship_active = True
        $ player_money += 3000
        "Вітаємо! Ти отримав стипендію за високі результати. +3000 грн!"
    else:
        "Цього разу ти не отримаєш стипендію. Але наступного семестру все може бути інакше!"
    return
```

Рисунок 3.13 – Логіка видачі стипендії

Якщо обидві умови виконані, гравець отримує велику грошову винагороду.

3.2.6 Реалізація міні-ігор

Міні-ігри є інтерактивними ігровими подіями, що розбавляють основний наратив і додають елемент змагання та динаміки. У межах гри реалізовано дві міні-гри, які пов’язані з робочою діяльністю головного героя: денна робота кур'єром та вечірня підробітка з виконання лабораторних робіт. Обидві міні-ігри надають гравцеві можливість заробити гроші та перевірити швидкість мислення, реакцію чи логічне мислення.

Механіка міні-гри кур’єра заснована на швидкому реагуванні та послідовності дій [11]. На екрані з'являється карта міста, на якій у випадковому порядку розміщено 10-15 чисел від 1 до 10-15. Завдання гравця — натиснути на всі числа в правильному порядку за обмежений час (Рисунок 3.14). Для ускладнення, числа періодично обертаються (змінюють орієнтацію), що ускладнює пошук і впізнавання цифр. За кожне успішно натиснуте число, гравець отримує по 50 гривень, і якщо користувач доставив усі замовлення, він додатково отримує 200 гривень, що дозволяє гравцю отримати до 950 гривень за робочу зміну.

Механіка реалізована через інтерактивний екран screen, у якому генеруються кнопки з випадковими позиціями, кожна з яких перевіряє правильність обраного порядку. Для анімації обертання було створено власну анімацію обертання мовою ATL.

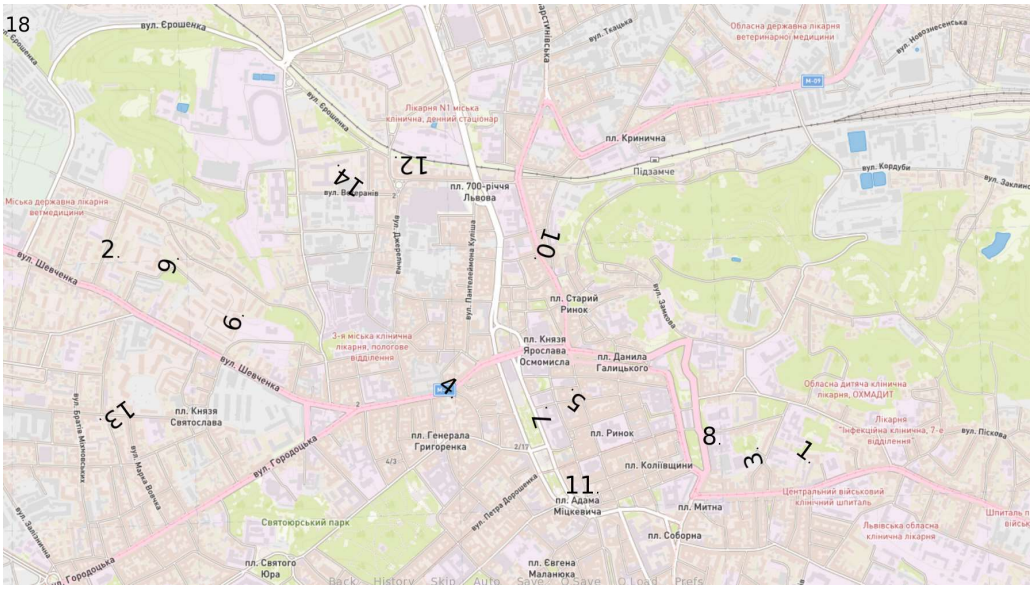


Рисунок 3.14 – Міні-гра «Кур’єр»

Перед початком гри відбувається ініціалізація змінних. Згенеровано список чисел від 1 до випадкового значення (в межах 10–14), які розміщуються у випадкових координатах на мапі. Кожному числу відповідає словник, що містить номер кнопки (b_number), текст для відображення (b_value), координати (b_x_pos, b_y_pos) та стан видимості (b_to_show) (Рисунок 3.15).

```
$ numbers_buttons = []
$ buttons_values = []

python:
    for i in range (1, renpy.random.randint (10, 15) ):
        buttons_values.append (str(i) )

python:
    for i in range (0, len(buttons_values) ):
        numbers_buttons.append ( {"b_number":i, "b_value":buttons_values[i], "b_x_pos":(renpy.random.randint (10, 100))*10, "b_y_pos

$ game_timer = 20
```

Рисунок 3.15 – Ініціалізація даних в «Кур'єрі»

На екран виводиться спеціально створений screen numbers_scr, де для кожного елемента реалізовано кнопку з відповідним числом (Рисунок 3.16). Ці кнопки мають однакові розміри, позиціонуються відповідно до заданих координат і мають ефект обертання, який задається за допомогою ATL-трансформації roto_transform.

Для реалізації таймера використано компонент timer, який кожну секунду перевіряє, чи завершено гру: якщо всі кнопки з числами вже натиснуті – повертається результат "win", якщо час вичерпано – "lose". У противному випадку змінна game_timer зменшується на одиницю.

Коли гравець натискає на кнопку, перевіряється її правильність: якщо натиснуто правильне наступне число – кнопка зникає (b_to_show = False). У випадку помилки (тобто гравець натискає число не за порядком), залишок часу зменшується на одну секунду.

```
transform roto_transform (roto_var):
    rotate roto_var
    rotate_pad False

screen numbers_scr:

    timer 1 action [Return("smth"), If( game_timer>1, If( numbers_buttons[-1]["b_to_show"] == False, Return("win"),
    text "[game_timer]" size 25 xpos 10 ypos 10 color "#000"


    for each_b in sorted(numbers_buttons, key=lambda b: b["b_number"], reverse=True):
        if each_b["b_to_show"]:
            $ text_var = each_b["b_value"]
            $ i = each_b["b_number"] - 1
            button:
                text '[text_var]{size=18}.{/size}' size 30 align (0.5, 0.55) color "#000"
                xminimum 100 xmaximum 100
                yminimum 100 ymaximum 100
                xpos each_b["b_x_pos"]
                ypos each_b["b_y_pos"]
                anchor (0.5, 0.5)
                action If (i == -1, SetDict(numbers_buttons[each_b["b_number"] ], "b_to_show", False),
                    If (numbers_buttons[i]["b_to_show"] == False,
                        SetDict(numbers_buttons[each_b["b_number"] ], "b_to_show", False),
                        SetVariable("game_timer", game_timer-1) )  )
                at roto_transform (renpy.random.randint (0, 10)*36)
```

Рисунок 3.16 – Основна логіка в «Кур'єрі»

Після завершення гри за будь-яким з двох сценаріїв (win або lose), відбувається розрахунок кількості вдалих доставок – це кількість натиснутих кнопок (тобто таких, що стали b_to_show = False). Гравець отримує грошову винагороду: по 50 гривень за кожну доставку, а у випадку повного проходження – додатковий бонус у 200 гривень. Отримані кошти додаються до змінної player_money.

У випадку програшу гравець отримує лише часткову винагороду, що стимулює до повторного проходження міні-гри з метою підвищення ефективності. Така механіка сприяє реіграбельності та формує відчуття досягнення.

Робота з лабораторними - це міні-гра, яка є логічною головоломкою, що імітує процес виконання складного завдання — складання пазлу. Гравцеві потрібно зібрати випадково згенерований пазл із частин у правильному порядку (Рисунок 3.17). Фрагменти пазла розміщені хаотично, і гравцю необхідно перетягувати та розставляти їх на правильні місця.

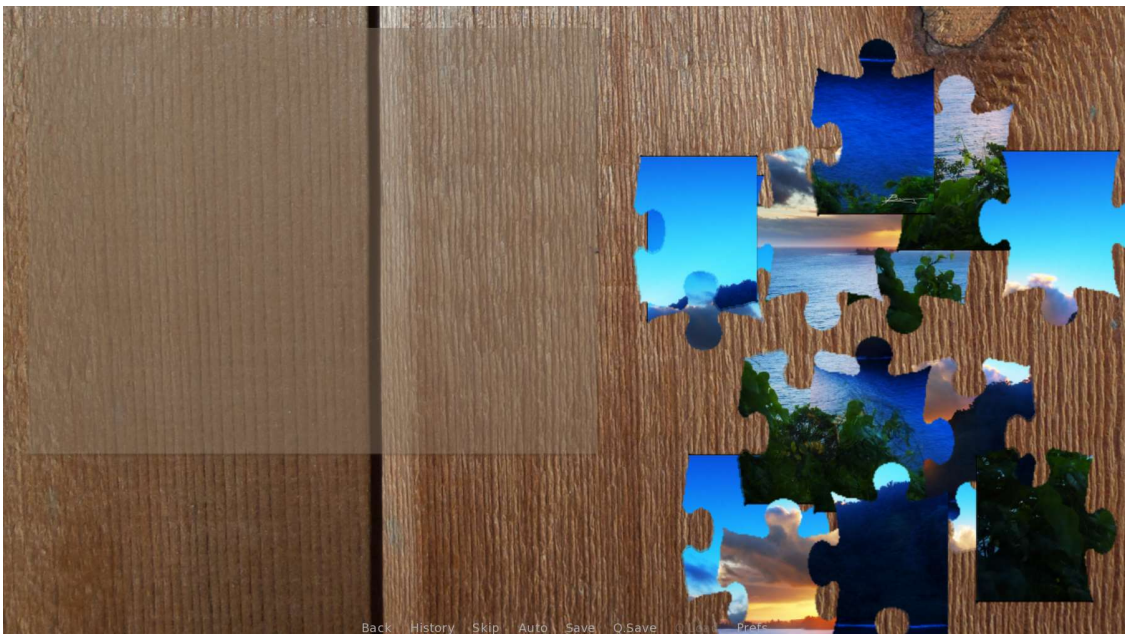


Рисунок 3.17 – Міні-гра «Робота з лабораторними»

Складність пазла – випадкова: його розмір та зображення генерується випадково між 3x3 та 6x6, а зображення випадковим чином вибирається із одного з доступних. В залежності від розмірності пазла, гравець може отримати до 360 гривень.

При завантаженні, спочатку йде підготовка зображення (Рисунок 3.18). Зображення завантажується, та перевіряються його розміри та розміри поля для збору, після чого зображення збільшується, зменшується або лишається таким же.

```
python:

    img_width, img_height = renpy.image_size(chosen_img)

    if img_width >= img_height and img_width > puzzle_field_size:
        img_scale_down_index = round( (1.00 * puzzle_field_size / img_width), 6)
        img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
        img_width = int(img_width * img_scale_down_index)
        img_height = int(img_height * img_scale_down_index)

    elif img_height >= img_width and img_height > puzzle_field_size:
        img_scale_down_index = round( (1.00 * puzzle_field_size / img_height), 6)
        img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
        img_width = int(img_width * img_scale_down_index)
        img_height = int(img_height * img_scale_down_index)

    else:
        img_to_play = chosen_img

    x_scale_index = round( (1.00 * (img_width/grid_width)/active_area_size), 6)
    y_scale_index = round( (1.00 * (img_height/grid_height)/active_area_size), 6)


    mainimage = im.Composite((int(img_width+(grip_size*2)*x_scale_index), int(img_height+(grip_size*2)*y_scale_index)),
```

Рисунок 3.18 – Генерація зображення пазлу

Після цього йде процес побудови фрагментів пазлу (Рисунок 3.19). Для початку, генерується пустий список jigsaw_grid. Потім для кожної клітинки сітки розміру пазлу до нього додається чотириелементний масив [9,9,9,9]. Ці чотири позиції згодом будуть заповнені значеннями, що позначають форму кожної сторони фрагмента:

- 0 — рівний край (без виступів),
- 1 — виступ,
- 2 — впадина.

Після цього йде процес побудови фрагментів пазлу (Рисунок 3.14). Для початку, генерується пустий список jigsaw_grid. Потім для кожної клітинки сітки розміру пазлу до нього додається чотириелементний масив [9,9,9,9]. Ці чотири позиції згодом будуть заповнені значеннями, що позначають форму кожної сторони фрагмента:

Далі другий подвійний цикл проходить по тій же сітці, але тепер по черзі заповнює значення для кожного боку.

Для верхньої грані, якщо поточна клітинка не знаходиться у першому рядку (not in top_row), то її верхня грань повинна допасовуватися до нижньої грані сусіднього фрагмента над нею. Тобто якщо у попереднього елементу (grid_width*(i−1)+j) внизу ([2]) стоїть 1 (виступ), то у цього фрагмента зверху призначається 2 (впадина), і навпаки; для елементів у першому рядку верхня грань встановлюється в 0.

Для правої грані, якщо поточна клітинка не в останньому стовпці (not in right_column), їй випадково присвоюється 1 або 2 (виступ чи впадина). Для крайнього правого стовпця — 0.

Нижня та ліва грань обчисляються аналогічно правій та верхній граням відповідно.

```python
python:

    img_width, img_height = renpy.image_size(chosen_img)

    if img_width >= img_height and img_width > puzzle_field_size:
        img_scale_down_index = round( (1.00 * puzzle_field_size / img_width), 6)
        img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
        img_width = int(img_width * img_scale_down_index)
        img_height = int(img_height * img_scale_down_index)

    elif img_height >= img_width and img_height > puzzle_field_size:
        img_scale_down_index = round( (1.00 * puzzle_field_size / img_height), 6)
        img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
        img_width = int(img_width * img_scale_down_index)
        img_height = int(img_height * img_scale_down_index)

    else:
        img_to_play = chosen_img

    x_scale_index = round( (1.00 * (img_width/grid_width)/active_area_size), 6)
    y_scale_index = round( (1.00 * (img_height/grid_height)/active_area_size), 6)


    mainimage = im.Composite((int(img_width+(grip_size*2)*x_scale_index), int(img_height+(grip_size*2)*y_scale_index)),
```

Рисунок 3.19 – Генерація фрагментів пазлу

Після цього для кожного фрагмента застосовуються маски виступів/впадин, обернені під відповідним кутом (im.Rotozoom()), щоб відповідати сторонам.

Для пересування фрагментів гравцем використовується функція piece_dragged() (Рисунок 3.20).

Дана функція отримує параметри drags (що тягнеться) і drop (куди це переміщується). Після цього, вона порівнює координати імені фрагмента та зони призначення. Якщо координати збігаються, тобто фрагмент поставлено у правильне місце, він "приклеюється" до відповідної позиції та стає неперетягуваним (draggable = False) і візуально центрується анімацією snap().

За допомогою placedlist перевіряється, чи всі частини пазлу вже поставлені на місце. Якщо так — функція повертає True, що вказує на завершення головоломки.

```
def piece_dragged(drags, drop):

    if not drop:
        return


    p_x = drags[0].drag_name.split("-")[0]
    p_y = drags[0].drag_name.split("-")[1]
    t_x = drop.drag_name.split("-")[0]
    t_y = drop.drag_name.split("-")[1]

    a = []
    a.append(drop.drag_joined)
    a.append((drags[0], 3, 3))
    drop.drag_joined(a)

    if p_x == t_x and p_y == t_y:

        my_x = int(int(p_x)*active_area_size*x_scale_index)-int(grip_size*x_scale_index)+puzzle_field_offset
        my_y = int(int(p_y)*active_area_size*y_scale_index)-int(grip_size*y_scale_index)+puzzle_field_offset
        drags[0].snap(my_x,my_y, delay=0.1)
        drags[0].draggable = False
        placedlist[int(p_x),int(p_y)] = True

        for i in range(0, grid_width):
            for j in range(0, grid_height):
                if placedlist[i,j] == False:
                    return
        return True
    return
```

Рисунок 3.20 – Функція пересування фрагментів пазлу

Після завершення головоломки, зображення відображається без поділу на частини та гравець отримує винагороду залежно від кількості частин: 10 * grid_width * grid_height.

3.2.7 Реалізація збереження та завантаження

Ren’Py за замовчуванням надає користувацький інтерфейс для збереження та завантаження стану гри. У ігровому меню є кнопки «Зберегти» та «Завантажити», які відкривають відповідні екрани зі списком слотів збережень (Рисунок 3.21).

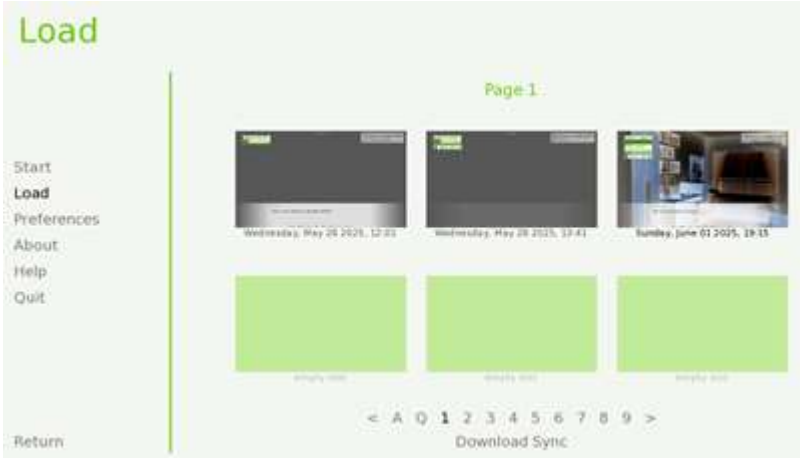


Рисунок 3.21 – Екран завантаження

Крім того, Ren’Py підтримує швидке збереження (Quick Save) і швидке завантаження (Quick Load). Ці дії реалізовані в Screen Language як QuickSave і

QuickLoad. За допомогою QuickSave можна негайно зберегти гру в одну з «швидких» позицій, циклічно використовуючи налаштовану кількість слотів (за замовчуванням 10 слотів для швидких збережень). Аналогічно, дія QuickLoad негайно завантажує останній автозбережений чи швидкий слот.

Крім того, у Ren'Py реалізоване автозбереження (autosave). За замовчуванням Ren'Py робить автозбереження у фоновому режимі при певних подіях гри: наприклад, при настанні варіанту вибору або при введенні тексту гравцем. Таким чином гра автоматично зберігає кілька попередніх станів без втручання гравця, а старі слоти перезаписуються у циклі.

Ren'Py зберігає ігровий стан у вигляді серіалізованих файлів у спеціальній директорії на комп'ютері гравця. За допомогою модуля pickle Python, Ren'Py перетворює внутрішній стан гри (змінні, стани екранів, інші об'єкти) на потік байтів. Зазвичай ці байти зберігаються у файл з розширенням .save. У цих файлах також міститься доповнення у форматі JSON, що може включати ім'я збереження, знімок екрана, час збереження тощо. Директорія збережень залежить від платформи: за замовчуванням Ren'Py використовує системну директорію для ігрових даних. У цій папці зберігаються всі файли збережень гри та файл персистентних даних. Наприклад, гравець може знайти набір файлів із розширенням .save та додатковий файл з іменем persistent, який містить глобальні дані.

_reload-2-LT1.save	07.06.2025 17:35	Файл SAVE	154 КБ
1-1-LT1.save	28.05.2025 12:01	Файл SAVE	75 КБ
1-2-LT1.save	28.05.2025 13:41	Файл SAVE	82 КБ
1-3-LT1.save	01.06.2025 19:15	Файл SAVE	90 КБ
auto-1-LT1.save	07.06.2025 18:22	Файл SAVE	155 КБ
auto-2-LT1.save	07.06.2025 18:13	Файл SAVE	122 КБ
auto-3-LT1.save	07.06.2025 17:57	Файл SAVE	116 КБ
auto-4-LT1.save	07.06.2025 17:57	Файл SAVE	108 КБ
auto-5-LT1.save	07.06.2025 17:57	Файл SAVE	156 КБ
auto-6-LT1.save	07.06.2025 17:27	Файл SAVE	151 КБ
auto-7-LT1.save	07.06.2025 17:18	Файл SAVE	150 КБ
auto-8-LT1.save	07.06.2025 17:18	Файл SAVE	130 КБ
auto-9-LT1.save	07.06.2025 17:17	Файл SAVE	130 КБ
auto-10-LT1.save	07.06.2025 17:17	Файл SAVE	128 КБ
navigation.json	11.06.2025 0:47	JSON Source File	17 КБ
persistent	07.06.2025 18:22	Файл	10 КБ

Рисунок 3.22 – Директорія збережень

Persistent-дані – це особливий тип даних, який зберігається незалежно від конкретного збереження гри. Ren’Py надає об’єкт persistent, поля якого автоматично серіалізуються при завершенні роботи гри і завантажуються щоразу на старті гри. На відміну від звичайних змінних гри, persistent не «відкочується» при завантаженні з іншого слоту: його дані є глобальними для всієї гри. Якщо в поле persistent нічого не встановлено, за замовчуванням воно дорівнює None.

Таким чином, persistent – поля можна вважати загальними глобальними змінними, значення яких переживають перезапуски гри.

Найчастіше персистентні дані використовуються для зберігання налаштувань гравця та глобального прогресу, які не належать до одного проходження гри. Наприклад, сюди можна записувати гучність музики, вибрану мову інтерфейсу або те, чи переглянуто початкові вступи. Іншим поширеним прикладом є система досягнень чи відкриття галерей. Аналогічним чином у поле persistent можна зберігати будь-які інші глобальні стани, наприклад системні опції гри чи список усіх отриманих гравцем нагород – щоразу, коли гра завершиться, Ren’Py збереже їх в файл persistent і відновить при наступному старті.

ВИСНОВКИ

У процесі виконання дипломного проєкту було розроблено інтерактивну візуальну новелу з елементами симулятора життя, яка включає календарну систему, систему ресурсів, відносини між персонажами, міні-ігри та іспити. Проєкт реалізовано за допомогою мови програмування Python та рушія Ren'Py, що забезпечили необхідний функціонал для створення нелінійного сюжету, складної логіки гри та зручної інтеграції мультимедійних елементів.

У ході роботи було проаналізовано стан ринку візуальних новел, сучасні тенденції та існуючі рушії для їх створення. Після порівняльного аналізу альтернативних платформ було обрано Ren’Py як найбільш відповідний інструмент для реалізації сюжетно-орієнтованого проєкту з необхідною інтерактивністю.

У межах гри реалізовано:

- календарну систему, що структурує ігровий час на дні та часові блоки;
- механіку ресурсів (енергія, стрес, гроші), які впливають на доступні дії гравця;
- систему навчання та іспитів, що базується на накопиченні знань та інтерактивному тестуванні;
- міні-ігри, що відтворюють робочі підробітки гравця та забезпечують фінансовий прогрес;

Реалізований проєкт має модульну архітектуру, що дозволяє масштабувати його у майбутньому, додавати нові гілки сюжету, міні-ігри або персонажів без значного перегляду існуючої структури. Гра орієнтована на молодіжну аудиторію, яка цікавиться сюжетно-орієнтованими симуляторами, і може слугувати основою для подальшої комерціалізації або використання в освітньо-розважальних цілях.

Таким чином, поставлені завдання були успішно виконані, а розроблений програмний продукт відповідає вимогам до сучасних візуальних новел за функціональністю, логікою й ігровим процесом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Newzoo. Global Games Market Report 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://newzoo.com/resources/trend-reports/newzoos-global-games-market-report-2024-free-version
2. Ren'Py [Електронний ресурс] – Режим доступу до ресурсу: https://www.renpy.org/
3. Ren'Py Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.renpy.org/doc/html/
4. Python 3 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/
5. Naninovel [Електронний ресурс] – Режим доступу до ресурсу: https://naninovel.com/
6. TyranoBuilder [Електронний ресурс] – Режим доступу до ресурсу: https://tyranobuilder.com/
7. Visual Novel Maker [Електронний ресурс] – Режим доступу до ресурсу: https://www.visualnovelmaker.com/
8. Twine [Електронний ресурс] – Режим доступу до ресурсу: https://twinery.org/
9. Ciesla R. Game Development with Ren'Py: Introduction to Visual Novel Games Using Ren'Py, TyranoBuilder, and Twine. Leanpub, 2021, 370 с.
10. VNDev Wiki. Genre. [Електронний ресурс] – Режим доступу до ресурсу: https://vndev.wiki/Genre
11. Sweigart A. The Big Book of Small Python Projects: 81 Easy Practice Programs, 2021, 423 с.

ДОДАТКИ

calendar.rpy

```
default game_day = 0
default hud_date = ""
default current_day = 1
default day_index = (current_day) % 7
default time_of_day = "ранок"
default max_days = 14
init python:
    import datetime

    GAME_START_DATE = datetime.date(2025, 5, 19)

    def get_game_date():
        date = GAME_START_DATE + datetime.timedelta(days=game_day)
        weekday = date.strftime("%A")
        weekday_ua = {
            "Monday": "Понеділок",
            "Tuesday": "Вівторок",
            "Wednesday": "Середа",
            "Thursday": "Четвер",
            "Friday": "П’ятниця",
            "Saturday": "Субота",
            "Sunday": "Неділя",
        }[weekday]

        formatted = date.strftime("%d %B %Y")
        formatted_ua = convert_months(formatted)

        return weekday_ua, formatted_ua

    def convert_months(date_str):
        months = {
            "January": "січня", "February": "лютого", "March": "березня",
            "April": "квітня", "May": "травня", "June": "червня",
            "July": "липня", "August": "серпня", "September": "вересня",
            "October": "жовтня", "November": "листопада", "December": "грудня"
        }
        for en, ua in months.items():
            if en in date_str:
                return date_str.replace(en, ua)
        return date_str

    def update_hud_date():
        weekday, formatted = get_game_date()
        return f"{weekday}, {formatted}"

screen hud():
    frame:
        xalign 0.98
        yalign 0.02
        padding (10, 10)
        background "#0008"
        vbox:
            spacing 3
            text "[hud_date]" size 22 color "#fff"
            text "Час: [time_of_day]" size 20 color "#ccc"
```

definitions.rpy

```
define e = Character("Максим")
image bg city day = "Backgrounds/City3_Morning.png"
image bg city evening = "Backgrounds/City3_Night.png"
image bg classroom day = "Backgrounds/Classroom1_Morning1.png"
image bg school corridor day = "Backgrounds/School_Hallway1_1.png"
image bg school outside day = "Backgrounds/SchoolOutside_Cloudy1.png"
image bg library study = "Backgrounds/LibraryStudyArea_Morning1.png"
image bg dorms day = "Backgrounds/StudentdormitoryInside_Morning.png"
image bg dorms evening = "Backgrounds/StudentdormitoryInside_Night.png"
```

energy.rpy

```
init python:
    def change_energy(amount):
        global energy
```

```
        energy = max(0, min(100, energy + amount))
    def change_stress(amount):
        global stress
        stress = max(0, min(150, stress + amount))

default energy = 100
default stress = 0

screen energy_bar():
    frame:
        xalign 0.05
        yalign 0.05
        has vbox
        text "Енергія: [energy]/100"
        bar value energy range 100 xmaximum 200

screen stress_bar():
    frame:
        xalign 0.05
        yalign 0.15
        has vbox
        text "Стрес: [stress]/100"
        bar value stress range 100 xmaximum 200
```

exam.rpy

```
init python:
    exam_data = {
        "Англійська": [
            {
                "question": "What is the past tense of 'go'?",
                "choices": ["goed", "went", "gone", "going"],
                "answer": 1,
                "explanation": "'Went' is the correct past tense form of the verb 'go'."
            },
            {
                "question": "Choose the correct form: She _____ to the gym every day.",
                "choices": ["go", "goes", "going", "gone"],
                "answer": 1,
                "explanation": "With third-person singular subjects, the verb takes an 's': 'She goes to the gym.'"
            },
            {
                "question": "What is the plural form of 'child'?",
                "choices": ["childs", "children", "childes", "child"],
                "answer": 1,
                "explanation": "'Children' is the irregular plural of 'child'."
            },
            {
                "question": "Translate to English: 'Я люблю читати книги.'",
                "choices": ["I love to read books.", "I loves reading books.", "I loving read books.", "I loved to
reading books."],
                "answer": 0,
                "explanation": "'I love to read books' is the correct and natural English translation."
            },
            {
                "question": "Which word is an adjective?",
                "choices": ["run", "beautiful", "quickly", "apple"],
                "answer": 1,
                "explanation": "'Beautiful' describes a noun and is an adjective."
            }
        ],
        "ООП": [
            {
                "question": "Що таке інкапсуляція?",
                "choices": [
                    "Приховування внутрішніх деталей об’єкта",
                    "Наслідування властивостей класу",
                    "Поліморфізм методів",
                    "Визначення інтерфейсу"
                ],
                "answer": 0,
                "explanation": "Інкапсуляція дозволяє приховати реалізацію і відкривати лише необхідні інтерфейси."
            },
            {
                "question": "Що таке поліморфізм?",
                "choices": [
                    "Можливість об'єктів мати багато форм",
```

```
                "Приховування даних",
                "Створення об'єктів",
                "Використання шаблонів проектування"
            ],
            "answer": 0,
            "explanation": "Поліморфізм дозволяє викликати методи, не знаючи точного типу об’єкта."
        },
        {
            "question": "Який патерн використовується для створення єдиного екземпляру класу?",
            "choices": ["Factory", "Singleton", "Observer", "Decorator"],
            "answer": 1,
            "explanation": "Singleton гарантує, що існує лише один екземпляр класу."
        },
        {
            "question": "Що таке наслідування?",
            "choices": [
                "Відношення між класами, коли один клас успадковує властивості іншого",
                "Процес створення об’єктів",
                "Метод обробки помилок",
                "Взаємодія між інтерфейсами"
            ],
            "answer": 0,
            "explanation": "Наслідування дозволяє створити новий клас на основі вже існуючого."
        },
        {
            "question": "Який із цих патернів є поведінковим?",
            "choices": ["Adapter", "Strategy", "Singleton", "Factory"],
            "answer": 1,
            "explanation": "Strategy дозволяє змінювати поведінку об'єкта під час виконання."
        }
    ],
    "Дискретна математика": [
        {
            "question": "Що таке множина?",
            "choices": [
                "Колекція унікальних елементів",
                "Впорядкований список",
                "Функція від одного числа до іншого",
                "Матриця з числами"
            ],
            "answer": 0,
            "explanation": "Множина — це сукупність елементів, де не допускається повторення."
        },
        {
            "question": "Яке твердження є істинним для логічного оператора 'AND'?",
            "choices": [
                "True AND False = False",
                "True AND False = True",
                "False AND False = True",
                "False AND True = True"
            ],
            "answer": 0,
            "explanation": "Оператор 'AND' повертає True тільки якщо обидва операнди є True."
        },
        {
            "question": "Що таке граф?",
            "choices": [
                "Набір вершин і ребер",
                "Послідовність чисел",
                "Функція від двох змінних",
                "Множина рівнянь"
            ],
            "answer": 0,
            "explanation": "Граф складається з вузлів (вершин), з'єднаних ребрами."
        },
        {
            "question": "Яке з наведених тверджень є тавтологією?",
            "choices": [
                "p OR NOT p",
                "p AND NOT p",
                "p IMPLIES q",
                "p AND q"
            ],
            "answer": 0,
            "explanation": "'p OR NOT p' завжди істинне незалежно від значення p."
        },
        {
```

```
                "question": "Що таке бінарне відношення?",
                "choices": [
                    "Підмножина декартового добутку двох множин",
                    "Функція одного аргументу",
                    "Послідовність операцій",
                    "Матриця з двійковими числами"
                ],
                "answer": 0,
                "explanation": "Бінарне відношення визначається як пара елементів з двох множин."
            }
        ],
        "Комп'ютерні мережі": [
            {
                "question": "Що таке IP-адреса?",
                "choices": [
                    "Унікальний ідентифікатор пристрою в мережі",
                    "Тип мережевого кабелю",
                    "Протокол передачі файлів",
                    "Мережевий комутатор"
                ],
                "answer": 0,
                "explanation": "IP-адреса — це унікальний числовий ідентифікатор, що призначається кожному пристрою
в комп’ютерній мережі для його ідентифікації та взаємодії."
            },
            {
                "question": "Який протокол відповідає за передачу електронної пошти?",
                "choices": ["HTTP", "FTP", "SMTP", "DNS"],
                "answer": 2,
                "explanation": "SMTP (Simple Mail Transfer Protocol) — це стандартний протокол для надсилання
електронної пошти між серверами."
            },
            {
                "question": "Що таке DNS?",
                "choices": [
                    "Система доменних імен",
                    "Протокол бездротового зв’язку",
                    "Тип IP-адреси",
                    "Мережевий кабель"
                ],
                "answer": 0,
                "explanation": "DNS (Domain Name System) перетворює зрозумілі доменні імена (наприклад, google.com)
у IP-адреси, які розуміють комп’ютери."
            },
            {
                "question": "Який рівень моделі OSI відповідає за маршрутизацію?",
                "choices": [
                    "Фізичний",
                    "Канальний",
                    "Мережевий",
                    "Прикладний"
                ],
                "answer": 2,
                "explanation": "Мережевий рівень OSI (Network Layer) відповідає за маршрутизацію пакетів між різними
мережами, наприклад, з використанням IP."
            },
            {
                "question": "Що таке MAC-адреса?",
                "choices": [
                    "Аппаратний адрес пристрою",
                    "IP-адреса сервера",
                    "Тип мережевого протоколу",
                    "Назва маршрутизатора"
                ],
                "answer": 0,
                "explanation": "MAC-адреса — це унікальний фізичний ідентифікатор мережевого інтерфейсу пристрою,
зазвичай зашитий у його апаратне забезпечення."
            }
        ],
        "Патерни проєктування": [
            {
                "question": "Що таке Singleton?",
                "choices": [
                    "Патерн, що гарантує існування лише одного екземпляра класу",
                    "Патерн для створення сімейства об’єктів",
                    "Патерн для групування об’єктів",
                    "Патерн для видалення об’єктів"
                ],
```

```
                "answer": 0,
                "explanation": "Патерн Singleton дозволяє створити лише один екземпляр класу, який буде спільним для всього застосунку, що корисно, наприклад, для керування налаштуваннями."
            },
            {
                "question": "Що таке 'Фабричний метод'?",
                "choices": [
                    "Патерн для створення об’єктів без вказівки точного класу",
                    "Патерн для доступу до об’єктів",
                    "Патерн для обробки помилок",
                    "Патерн для сортування даних"
                ],
                "answer": 0,
                "explanation": "Фабричний метод (Factory Method) дозволяє створювати об’єкти через інтерфейси, не вказуючи їх конкретний клас, що підвищує гнучкість системи."
            },
            {
                "question": "Що таке 'Наблюдач' (Observer)?",
                "choices": [
                    "Патерн, що повідомляє підписників про зміни стану",
                    "Патерн для створення об’єктів",
                    "Патерн для логування",
                    "Патерн для запуску потоків"
                ],
                "answer": 0,
                "explanation": "Observer дозволяє об'єктам (спостерігачам) автоматично оновлюватися, коли стан іншого об'єкта змінюється, що зручно для реалізації взаємодії."
            },
            {
                "question": "Що таке 'Декоратор'?",
                "choices": [
                    "Патерн, що динамічно додає поведінку об’єкту",
                    "Патерн для видалення об’єктів",
                    "Патерн для створення об’єктів",
                    "Патерн для групування об’єктів"
                ],
                "answer": 0,
                "explanation": "Декоратор дозволяє динамічно додавати нову поведінку до об’єктів без зміни їхньої структури, зберігаючи принцип відкритості/закритості."
            },
            {
                "question": "Що таке 'Фасад'?",
                "choices": [
                    "Простий інтерфейс до складної системи",
                    "Об’єкт для спостереження",
                    "Метод створення об’єктів",
                    "Алгоритм сортування"
                ],
                "answer": 0,
                "explanation": "Фасад (Facade) надає спрощений інтерфейс для доступу до складної підсистеми, приховуючи її складність від користувача."
            }
        ]
    }
    exam_results = {}
    exam_days = ["Англійська", "ООП", "Дискретна математика", "Комп'ютерні мережі", "Патерни проєктування"]
    exam_scores = {day: 0 for day in exam_days}
    exam_current_question = 0
    exam_current_day_index = 0
    exam_correct_count = 0
    passed_count = 0

    exam_scores = {}
    scholarship_allpassed = True
    scholarship_active = False

    caught_chance = 0.9

default prep_scores = {
    "Англійська": 0,
    "ООП": 0,
    "Дискретна математика": 0,
    "Комп'ютерні мережі": 0,
    "Патерни проєктування": 0,
    }
default exams_tried = False
```

```
screen exam_question_screen(question):
    modal True
    tag exam

    frame:
        xalign 0.5
        yalign 0.5
        padding (20, 20)
        background "#222a"
        has vbox

        text question["question"] size 30 color "#fff"

        for i, answer in enumerate(question["choices"]):
            button:
                text answer
                action Return(i)
                xminimum 300
                padding (10, 5)

label exam_session:
    $ time_of_day = "день"
    scene bg classroom day with fade

    "Наступний іспит: Англійська"
    call exam("Англійська") from _call_exam

    "Наступний іспит: ООП"
    call exam("ООП") from _call_exam_1

    "Наступний іспит: Дискретна математика"
    call exam("Дискретна математика") from _call_exam_2

    "Наступний іспит: Комп'ютерні мережі"
    call exam("Комп'ютерні мережі") from _call_exam_3

    "Наступний іспит: Патерни проєктування"
    call exam("Патерни проєктування") from _call_exam_4

    jump exams_final

label exam(subject):
    $ questions = exam_data[subject]
    $ score = 0
    $ total = len(questions) + 3
    $ got_caught = False

    "Перед тим як розпочати іспит, ти бачиш викладача."
    if teacher_loyalty[subject]:
        "Він киває тобі. Можливо, не дарма ти відвідував пари..."
    else:
        "Ти не часто бачив цього викладача, тож розраховуй лише на себе."

    if cheat_sheets > 0:
        "Ти згадуєш, що в тебе є шпаргалка, яку ти можеш використати"
        python:
            loyalty_decrease = min(0.5, 0.075 * lecture_attendance[subject])
            random_chance = round(random.random(), 2)
            caught_chance = max(round(random_chance - loyalty_decrease, 2), 0.05)
        menu:
            "Хочеш використати шпаргалку?"
            "Так (шанс спійматись - [caught_chance])":
                $ cheat_sheets -= 1
                "Ти намагаєшся витягнути шпаргалку та непомітно списати..."

                python:
                    got_caught = random.random() <= caught_chance
                if got_caught:
                    "Ой-ой! Шпору знайшли!"
                    python:
                        scholarship_allpassed = False
                        score = 0
                        passed = score >= total * 0.5
                        exam_results[subject] = passed
                        exam_scores[subject] = score
                    "Тебе одразу вигнали з іспиту та відправили на перездачу..."
                else:
```

```
                    $ score = 5
                    "Фух! Повезло непомітно списати всі основні тестові питання!"

            "Ні":
                "Ти видихаєш і починаєш роботу над іспитом..."

                python:
                    for q in questions:
                        choice = renpy.call_screen("exam_question_screen", q)
                        if choice == q["answer"]:
                            score += 1

    else:
        python:
            for q in questions:
                choice = renpy.call_screen("exam_question_screen", q)
                if choice == q["answer"]:
                    score += 1


    if prep_scores[subject] > 0 and not got_caught:
        $ score += prep_scores[subject]
        "Твоя підготовка до іспиту дозволила тобі відповісти на [prep_scores[subject]] із 3 додаткових питань!"
    if teacher_loyalty[subject] and not got_caught:
        "Здаючи роботу, викладач кивнув тобі. Думаю, він зможе підняти мою оцінку, якщо це буде необхідно!"
        python:
            score = min(score + 3, total)
    python:
        passed = score >= total * 0.5
        exam_results[subject] = passed
        exam_scores[subject] = score
    "Іспит з [subject]. Результат: [score]/[total]"

    if passed:
        "Ти успішно здав іспит!"
    else:
        $ scholarship_allpassed = False
        "Іспит провалено. Доведеться брати талон..."


    return


label exams_final:

    scene bg school corridor day with fade
    "Твій підсумок сесії:"

    python:
        for subject in exam_data.keys():
            if exam_results.get(subject, False):
                renpy.say(None, f"✅ {subject} — ЗДАНО")
                passed_count += 1
            else:
                renpy.say(None, f"❌ {subject} — НЕ ЗДАНО")

    if passed_count >= 3:
        "Ти здав щонайменше 3 іспити. Вітаємо!"
        call check_scholarship from _call_check_scholarship
        call check_dormpayment from _call_check_dormpayment
        jump good_ending
    else:
        "Ти здав менше 3 іспитів. Треба перездати провалені..."
        "Втомленим, ти зразу же повертаєшся до гуртожитку..."
        jump retake_exams_prep

label retake_exams_prep:
    $ time_of_day = "вечір"
    scene bg dorms evening with fade
    "Перед перездачею ти можеш повторити матеріал з будь-якої дисципліни."

    menu:
        "Оберіть дисципліну для повторення:"
        "Англійська":
            $ selected_subject = "Англійська"
        "ООП":
            $ selected_subject = "ООП"
```

```
        "Дискретна математика":
            $ selected_subject = "Дискретна математика"
        "Комп'ютерні мережі":
            $ selected_subject = "Комп'ютерні мережі"
        "Патерни проєктування":
            $ selected_subject = "Патерни проєктування"

    $ lecture_question = get_next_lecture(selected_subject)
    $ question_text = lecture_question["question"]
    $ correct_index = lecture_question["answer"]
    $ correct_answer = lecture_question["choices"][correct_index]

    if prep_scores[selected_subject] < 3:
        $ prep_scores[selected_subject] += 1
    "Ти пройшовся по питанням з [selected_subject]."
    "[lecture_question['question']]"
    "[lecture_question['explanation']]"

    $ lecture_question = get_next_lecture(selected_subject)
    "[lecture_question['question']]"
    "[lecture_question['explanation']]"

    "Підовчивши матеріал додатково, ти вирішив піти спати."
    return

label retake_exams:

    python:
        second_chances = {}
        failed_subjects = [subject for subject in exam_data.keys() if not exam_results.get(subject, False)]

        for subject in failed_subjects:
            renpy.say(None, f"Повторна спроба іспиту з {subject}...")
            renpy.call("exam", subject)
            second_chances[subject] = exam_results[subject]

        passed_count = sum(1 for passed in exam_results.values() if passed)

    if passed_count >= 3:
        "Ти зміг покращити результат і склав сесію!"
        call check_scholarship from _call_check_scholarship_1
        call check_dormpayment from _call_check_dormpayment_1
        jump good_ending
    else:
        "На жаль, друга спроба також була невдалою."
        jump bad_ending


label good_ending:
    "Після важкого семестру та сесії ти все ж впорався!"
    "Твоє студентське життя продовжується!"
    $ renpy.full_restart()
    return


label bad_ending:
    "Нажаль, ти не склав сесію навіть по талону..."
    "Доведеться якось викручуватись, але, ймовірно, з університету тебе все ж таки виженуть..."
    $ renpy.full_restart()
    Return
```

lectures.rpy

```
init python:
    visited_lectures = {subject: set() for subject in exam_data.keys()}

init python:
    import random

    def get_next_lecture(subject):
        total_questions = len(exam_data[subject])
        visited = visited_lectures[subject]

        if len(visited) == total_questions:
            # Всі лекції вже відвідані — скидаємо
            visited_lectures[subject] = set()
            visited = set()
```

```
        remaining = [i for i in range(total_questions) if i not in visited]
        selected_index = random.choice(remaining)
        visited_lectures[subject].add(selected_index)
        return exam_data[subject][selected_index]

default lecture_attendance = {"Англійська": 0, "ООП": 0, "Дискретна математика": 0, "Комп'ютерні мережі": 0,
"Патерни проєктування": 0}
default teacher_loyalty = {"Англійська": False, "ООП": False, "Дискретна математика": False, "Комп'ютерні мережі":
False, "Патерни проєктування": False}

label lecture:
    $ lecture_question = get_next_lecture(selected_subject)
    $ question_text = lecture_question["question"]
    $ correct_index = lecture_question["answer"]
    $ correct_answer = lecture_question["choices"][correct_index]

    scene bg classroom day with fade

    "Ти прийшов на лекцію з [selected_subject]."

    if energy >= 40:
        $ change_energy(-40)
        if prep_scores[selected_subject] < 3:
            $ prep_scores[selected_subject] += 1
        "Викладач пояснює питання, що буде винесено на іспит:"
        "[lecture_question['question']]"
        "[lecture_question['explanation']]"
        "Викладач" "Це важлива тема. Обов’язково зверніть увагу на це на іспиті."

        $ lecture_attendance[selected_subject] += 1
        "В кінці пари у вас лишився час щоб поспілкуватись з викладачем на різні теми"
        if lecture_attendance[selected_subject] >= 5:
            "Здається, викладач тебе запам'ятав та буде відноситись до тебе лояльніше!"
            $ teacher_loyalty[selected_subject] = True
    else:
        $ change_energy(+10)
        "Ти був занадто втомлений і заснув на лекції..."

    return
```

money.rpy

```
init python:
    import random

    def show_random_explanations(subject, number):
        questions = exam_data.get(subject, [])
        random.shuffle(questions)
        selected = questions[:number]

        for q in selected:
                question_text = q["question"]
                correct_index = q["answer"]
                correct_answer = q["choices"][correct_index]
                explanation = q.get("explanation", "Немає пояснення.")

                renpy.say(None, f"{explanation}")

default player_money = 100
default cheat_sheets = 0
default player_livesInDorms = True

screen hud_money():
    frame:
        xalign 0.05
        yalign 0.25
        has vbox
        text "💰 Гроші: [player_money] грн"

label shop:
    if time_of_day == "вечір":
        scene bg city evening with dissolve
    else:
        scene bg city day with dissolve
    $ done = False
    while not done:
        menu:
```

```
            "Що ти хотів би придбати?"
            "Енергетики (75 гривень)":
                "Допоможуть трохи зосередитись і відновити невелику кількість енергії"
                if player_money >= 75:
                    menu:
                        "Бажаєш придбати і випити?"
                        "Так":
                            $ change_energy(+15)
                            $ player_money -= 75
                            "Ти купив енергетик і випив його. Ти почуваєш прилив сил!"
                        "Ні":
                            pass
                else:
                    "В тебе недостатньо грошей на нього..."
            "Обід (180 гривень)":
                "Хороший обід, яким можна насититись і відновити достатню кількість енергії"
                if player_money >= 180:
                    menu:
                        "Бажаєш придбати та поїсти?"
                        "Так":
                            $ change_energy(+45)
                            $ player_money -= 180
                            "Ти купив розкішний обід та поїв. Ти почуваєш прилив сил!"
                        "Ні":
                            pass
                else:
                    "В тебе недостатньо грошей на нього..."
            "Конспекти (300 гривень)":
                "Конспекти від старшокурсників, які містять 3 випадкових пояснення на питання з іспиту"
                if player_money >= 300:
                    menu:
                        "Бажаєш придбати та почитати?"
                        "Так":
                            $ player_money -= 300
                            menu:
                                "З якої дисципліни ти хочеш придбати конспект?"
                                "Англійська":
                                    $ prep_scores["Англійська"] = 3
                                    $ show_random_explanations("Англійська", 3)
                                "ООП":
                                    $ prep_scores["ООП"] = 3
                                    $ show_random_explanations("ООП", 3)
                                "Дискретна математика":
                                    $ prep_scores["Дискретна математика"] = 3
                                    $ show_random_explanations("Дискретна математика", 3)
                                "Комп'ютерні мережі":
                                    $ prep_scores["Комп'ютерні мережі"] = 3
                                    $ show_random_explanations("Комп'ютерні мережі", 3)
                                "Патерни проєктування":
                                    $ prep_scores["Патерни проєктування"] = 3
                                    $ show_random_explanations("Патерни проєктування", 3)
                        "Ні":
                            pass
                else:
                    "В тебе недостатньо грошей на нього..."
            "Шпаргалки (1000 гривень)":
                "Шпаргалка, яку можна незамітно принести на один із іспитів і успішно його здати... Якщо тебе не
замітять"
                if player_money >= 1000:
                    menu:
                        "Бажаєш придбати? В тебе на даний момент [cheat_sheets] шпаргалок."
                        "Так":
                            $ player_money -= 1000
                            $ cheat_sheets += 1
                            "Ти купив шпаргалку. Ти зможеш її застосувати на іспиті!"
                        "Ні":
                            pass
                else:
                    "В тебе недостатньо грошей на них..."
            "Вийти":
                $ done = True
    return

label check_scholarship:

    $ average_score = sum(exam_scores.values()) / len(exam_scores) if exam_scores else 0
    $ avg_high = average_score >= 6.5
```

```
    if avg_high and scholarship_allpassed:
        $ scholarship_active = True
        $ player_money += 3000
        "Вітаємо! Ти отримав стипендію за високі результати. +3000 грн!"
    else:
        "Цього разу ти не отримаєш стипендію. Але наступного семестру все може бути інакше!"
    return

label check_dormpayment:
    "Настав час оплатити проживання в гуртожитку..."
    if player_money >= 3000:
        $ player_livesInDorms = True
        $ player_money -= 3000
        "На щастя, грошей виявилось достатньо! Можна продовжувати жити в ньому до наступної сесії!"
    else:
        $ player_livesInDorms = False
        "Нажаль, коштів виявилось замало... Доведеться повертатись додому..."
        "Відтепер, в робочі дні весь день буде займати поїздка додому..."
    return
```

numbers.rpy

```
transform roto_transform (roto_var):
    rotate roto_var
    rotate_pad False

screen numbers_scr:

    key "K_LEFT" action Hide("nonexistent_screen")
    key "K_RIGHT" action Hide("nonexistent_screen")
    key "K_UP" action Hide("nonexistent_screen")
    key "K_DOWN" action Hide("nonexistent_screen")
    key "K_RETURN" action Hide("nonexistent_screen")
    key "K_KP_ENTER" action Hide("nonexistent_screen")

    timer 1 action [Return("smth"), If( game_timer>1, If( numbers_buttons[-1]["b_to_show"] == False, Return("win"),
SetVariable("game_timer", game_timer-1) ), Return("lose") ) ] repeat True
    text "[game_timer]" size 25 xpos 10 ypos 10 color "#000"


    for each_b in sorted(numbers_buttons, key=lambda b: b["b_number"], reverse=True):
        if each_b["b_to_show"]:
            $ text_var = each_b["b_value"]
            $ i = each_b["b_number"] - 1
            button:

                text '[text_var]{size=18}.{/size}' size 30 align (0.5, 0.55) color "#000"
                xminimum 100 xmaximum 100
                yminimum 100 ymaximum 100
                xpos each_b["b_x_pos"]
                ypos each_b["b_y_pos"]
                anchor (0.5, 0.5)
                action If (i == -1, SetDict(numbers_buttons[each_b["b_number"] ], "b_to_show", False),
                    If (numbers_buttons[i]["b_to_show"] == False,
                        SetDict(numbers_buttons[each_b["b_number"] ], "b_to_show", False),
                        SetVariable("game_timer", game_timer-1) )  )
                at roto_transform (renpy.random.randint (0, 10)*36)


label numbers_game:

    if time_of_day == "вечір":
        scene bg city evening with dissolve
    else:
        scene bg city day with dissolve

    $ numbers_buttons = []
    $ buttons_values = []

    python:
        for i in range (1, renpy.random.randint (10, 15) ):
            buttons_values.append (str(i) )
]


    python:
```

```
        for i in range (0, len(buttons_values) ):
            numbers_buttons.append  (  {"b_number":i,  "b_value":buttons_values[i],  "b_x_pos":(renpy.random.randint
(10, 100))*10, "b_y_pos":(renpy.random.randint (15, 60))*10, "b_to_show":True} )

    "Твоя ціль - об'їхати всі пункти доставки по порядку (починаючи з 1)."

    $ game_timer = 20

    window hide

    hide screen hud
    hide screen energy_bar
    hide screen stress_bar
    hide screen hud_money

    scene city_map

    show screen numbers_scr
    label loop:
        $ result = ui.interact()
        $ game_timer = game_timer
        if result == "smth":
            jump loop

    if result == "lose":
        hide screen numbers_scr
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)

        if time_of_day == "вечір":
            scene bg city evening with dissolve
        else:
            scene bg city day with dissolve
        window show
        show screen hud
        show screen stress_bar
        show screen energy_bar
        show screen hud_money

        $ work_done = 0
        python:
            temp = ""

            last_false_index = -1
            for i in range(len(numbers_buttons) - 1, -1, -1):
                if numbers_buttons[i]["b_to_show"] == False:
                    last_false_index = i
                    break

            work_done = last_false_index + 1

        "Ти встиг доставити їжу лише [work_done] клієнтам..."
        $ player_money += 50 * work_done
        "Ти заробив [50 * work_done] гривень!"
        return

    if result == "win":
        hide screen numbers_scr
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)
        $ renpy.pause (0.1, hard = True)

        if time_of_day == "вечір":
            scene bg city evening with dissolve
        else:
            scene bg city day with dissolve
        window show
        show screen hud
        show screen stress_bar
        show screen energy_bar
        show screen hud_money

        $ work_done = 0
        python:
```

```
            temp = ""

            last_false_index = -1
            for i in range(len(numbers_buttons) - 1, -1, -1):
                if numbers_buttons[i]["b_to_show"] == False:
                    last_false_index = i
                    break

            work_done = last_false_index + 1

        "Ти встиг доставити їжу всім клієнтам!"
        $ player_money += 50 * work_done + 200
        "Ти заробив [50 * work_done + 200] гривень!"
        return
```

puzzles.rpy

```
default grid_width = 3
default grid_height = 3
define puzzle_field_size = 650
define puzzle_field_offset = 30
define puzzle_piece_size = 450
define grip_size = 75
define active_area_size = puzzle_piece_size - (grip_size * 2)


init python:

    def piece_dragged(drags, drop):

        if not drop:
            return


        p_x = drags[0].drag_name.split("-")[0]
        p_y = drags[0].drag_name.split("-")[1]
        t_x = drop.drag_name.split("-")[0]
        t_y = drop.drag_name.split("-")[1]

        a = []
        a.append(drop.drag_joined)
        a.append((drags[0], 3, 3))
        drop.drag_joined(a)

        if p_x == t_x and p_y == t_y:

            my_x = int(int(p_x)*active_area_size*x_scale_index)-int(grip_size*x_scale_index)+puzzle_field_offset
            my_y = int(int(p_y)*active_area_size*y_scale_index)-int(grip_size*y_scale_index)+puzzle_field_offset
            drags[0].snap(my_x,my_y, delay=0.1)
            drags[0].draggable = False
            placedlist[int(p_x),int(p_y)] = True

            for i in range(0, grid_width):
                for j in range(0, grid_height):
                    if placedlist[i,j] == False:
                        return
            return True
        return


screen jigsaw:

    key "rollback" action [[]]
    key "rollforward" action [[]]

    add im.Scale("puzz_imgs/_puzzle_field.png", img_width, img_height) pos(puzzle_field_offset, puzzle_field_offset)

    draggroup:

        for i in range(0, grid_width):
            for j in range(0, grid_height):
                $ name = "%s-%s"%(i,j)
                $ my_x = i*int(active_area_size*x_scale_index)+puzzle_field_offset
                $ my_y = j*int(active_area_size*y_scale_index)+puzzle_field_offset
                drag:
                    drag_name name
                    child        im.Scale("puzz_imgs/_blank_space.png",        int(active_area_size*x_scale_index),
int(active_area_size*y_scale_index) )
```

```
                draggable False
                xpos my_x ypos my_y


        for i in range(0, grid_width):
            for j in range(0, grid_height):
                $ name = "%s-%s-piece"%(i,j)
                drag:
                    drag_name name
                    child imagelist[i,j]
                    #droppable False
                    dragged piece_dragged
                    xpos piecelist[i,j][0] ypos piecelist[i,j][1]




image puzzle_background = "puzz_imgs/_20130709_192009.jpg"

label puzzle:
    python:

        img_width, img_height = renpy.image_size(chosen_img)


        if img_width >= img_height and img_width > puzzle_field_size:
            img_scale_down_index = round( (1.00 * puzzle_field_size / img_width), 6)
            img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
            img_width = int(img_width * img_scale_down_index)
            img_height = int(img_height * img_scale_down_index)

        elif img_height >= img_width and img_height > puzzle_field_size:
            img_scale_down_index = round( (1.00 * puzzle_field_size / img_height), 6)
            img_to_play = im.FactorScale(chosen_img, img_scale_down_index)
            img_width = int(img_width * img_scale_down_index)
            img_height = int(img_height * img_scale_down_index)

        else:
            img_to_play = chosen_img

        x_scale_index = round( (1.00 * (img_width/grid_width)/active_area_size), 6)
        y_scale_index = round( (1.00 * (img_height/grid_height)/active_area_size), 6)


        mainimage                    =                    im.Composite((int(img_width+(grip_size*2)*x_scale_index),
int(img_height+(grip_size*2)*y_scale_index)),(int(grip_size*x_scale_index),          int(grip_size*y_scale_index)),
img_to_play)



        top_row = []
        for i in range (0, grid_width):
            top_row.append(i)

        bottom_row = []
        for i in range (0, grid_width):
            bottom_row.append(grid_width*(grid_height-1)+i)

        left_column = []
        for i in range (0, grid_height):
            left_column.append(grid_width*i)

        right_column = []
        for i in range (0, grid_height):
            right_column.append(grid_width*i + (grid_width-1) )


        jigsaw_grid = []
        for i in range(0,grid_height):
            for j in range (0,grid_width):
                jigsaw_grid.append([9,9,9,9])   # [top, right, bottom, left]

        for i in range(0,grid_height):
            for j in range (0,grid_width):
```

```
                if grid_width*i+j not in top_row:
                    if jigsaw_grid[grid_width*(i-1)+j][2] == 1:
                        jigsaw_grid[grid_width*i+j][0] = 2
                    else:
                        jigsaw_grid[grid_width*i+j][0] = 1
                else:
                    jigsaw_grid[grid_width*i+j][0] = 0

                if grid_width*i+j not in right_column:
                    jigsaw_grid[grid_width*i+j][1] = renpy.random.randint(1,2)
                else:
                    jigsaw_grid[grid_width*i+j][1] = 0

                if grid_width*i+j not in bottom_row:
                    jigsaw_grid[grid_width*i+j][2] = renpy.random.randint(1,2)
                else:
                    jigsaw_grid[grid_width*i+j][2] = 0

                if grid_width*i+j not in left_column:
                    if jigsaw_grid[grid_width*i+j-1][1] == 1:
                        jigsaw_grid[grid_width*i+j][3] = 2
                    else:
                        jigsaw_grid[grid_width*i+j][3] = 1
                else:
                    jigsaw_grid[grid_width*i+j][3] = 0


        piecelist = dict()
        imagelist = dict()
        placedlist = dict()
        testlist = dict()

        for i in range(0,grid_width):
            for j in range (0,grid_height):
                piecelist[i,j]     =     [int(renpy.random.randint(0,     int(config.screen_width     *     0.9     -
puzzle_field_size))+puzzle_field_size), int(renpy.random.randint(0, int(config.screen_height * 0.8)))]

                temp_img            =            im.Crop(mainimage,            int(i*active_area_size*x_scale_index),
int(j*active_area_size*y_scale_index), int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index))

                imagelist[i,j] = im.Composite(
        (int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index)),
        (0,0),  im.AlphaMask(temp_img,  im.Scale(im.Rotozoom("puzz_imgs/_00%s.png"%(jigsaw_grid[grid_width*j+i][0]),
0, 1.0), int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index))),
        (0,0),  im.AlphaMask(temp_img,  im.Scale(im.Rotozoom("puzz_imgs/_00%s.png"%(jigsaw_grid[grid_width*j+i][1]),
270, 1.0), int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index))),
        (0,0),  im.AlphaMask(temp_img,  im.Scale(im.Rotozoom("puzz_imgs/_00%s.png"%(jigsaw_grid[grid_width*j+i][2]),
180, 1.0), int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index))),
        (0,0),  im.AlphaMask(temp_img,  im.Scale(im.Rotozoom("puzz_imgs/_00%s.png"%(jigsaw_grid[grid_width*j+i][3]),
90, 1.0), int(puzzle_piece_size*x_scale_index), int(puzzle_piece_size*y_scale_index)))
        )
                placedlist[i,j] = False



    show puzzle_background as puzzle_bg
    call screen jigsaw
    jump win

label win:
    show black as puzzle_bg
    show expression img_to_play as win_img at truecenter with dissolve

    window show
    show screen hud
    show screen stress_bar
    show screen energy_bar
    show screen hud_money

    "Ти успішно написав конспект!"
    $ player_money += 10 * grid_width * grid_height
    "Ти заробив [10 * grid_width * grid_height] гривень та пішов спати!"
    hide puzzle_bg
    hide win_img
    return # end of puzzle (return to game)
```

script.rpy

```
label start:
    $ current_day = 1
    $ max_days = 14
    $ game_day = 0


    $ hud_date = update_hud_date()

    show screen hud
    show screen stress_bar
    show screen energy_bar
    show screen hud_money

    e "Новий день... Починаємо життя студента!"

    call day_loop from _call_day_loop
    return

label day_loop:

    call day_info from _call_day_info

    call time_segment("ранок") from _call_time_segment
    if current_day >= max_days:
        $ exams_tried = True
        $ exam_scores = {}
        call exam_session from _call_exam_session
    elif passed_count < 3 and exams_tried == True:
        call retake_exams from _call_retake_exams
    else:
        call time_segment("день") from _call_time_segment_1
        if day_index < 5:
            call time_segment("після навчання") from _call_time_segment_2
        else:
            call time_segment("день") from _call_time_segment_3
        call time_segment("вечір") from _call_time_segment_4

    $ current_day += 1
    $ game_day += 1
    $ hud_date = update_hud_date()
    $ day_index = (current_day - 1) % 7
    $ change_energy(+5)

    jump day_loop


label day_info:
    $ time_of_day = "ранок"
    $ weekday, formatted_date = get_game_date()
    $ days_left = max_days - current_day + 1

    scene bg dorms day with fade

    e "Сьогодні [weekday], [formatted_date]."
    if days_left > 1:
        e "До сесії залишилося [days_left] днів."
    else:
        e "Сьогодні день сесії."
    return

label stress_ending:
    "Ти прокинувся, але більше не відчуєш, що ти - це ти"
    "В погоні за навчанням та грошима, ти втратив душевний спокій"
    "Щоденні пари, робота, іспити, сесія, дедлайни - усе це набридло"
    "Нехай горить цей диплом вогнем, ти тепер будеш шукати себе в цьому світі"
    "На цьому твоє студентське життя закінчується, хочаб доти, доки ти не знайдеш те, що тобі знову сподобається"
    $ renpy.full_restart()
    return

label time_segment(t):
    $ time_of_day = t

    if time_of_day == "ранок":
        if stress >= 100:
            jump stress_ending
        elif day_index < 5:
            "Я поснідав, зібрав речі та пішов в універ..."
```

```
        else:
            "Я накрився ковдрою та ліг спати далі..."
            $ change_energy(+5)
            $ change_stress(-5)

    elif time_of_day == "день" and day_index < 5:
        scene bg school outside day with fade
        "Яку лекцію ти хочеш відвідати сьогодні?"
        menu:
            "Обрати предмет"
            "Англійська":
                $ selected_subject = "Англійська"
            "ООП":
                $ selected_subject = "ООП"
            "Дискретна математика":
                $ selected_subject = "Дискретна математика"
            "Комп'ютерні мережі":
                $ selected_subject = "Комп'ютерні мережі"
            "Патерни проєктування":
                $ selected_subject = "Патерни проєктування"
        $ change_stress(+15)
        jump lecture
    else:

        if time_of_day == "вечір":
            scene bg dorms evening with fade
        elif time_of_day == "після навчання" and day_index < 5:
            scene bg school outside day with fade
        else:
            scene bg dorms day with fade

        menu activity_choice:
            e "Зараз [time_of_day]. Що робитимемо?"
            "Сходити в бібліотеку" if time_of_day == "день" or time_of_day == "після навчання":
                menu:
                    "Яку дисципліну хочеш вивчити?"
                    "Англійська":
                        $ selected_subject = "Англійська"
                    "ООП":
                        $ selected_subject = "ООП"
                    "Дискретна математика":
                        $ selected_subject = "Дискретна математика"
                    "Комп'ютерні мережі":
                        $ selected_subject = "Комп'ютерні мережі"
                    "Патерни проєктування":
                        $ selected_subject = "Патерни проєктування"
                $ change_stress(+10)
                jump study

            "Почитати лекцію" if time_of_day == "вечір":
                menu:
                    "Яку дисципліну хочеш вивчити?"
                    "Англійська":
                        $ selected_subject = "Англійська"
                    "ООП":
                        $ selected_subject = "ООП"
                    "Дискретна математика":
                        $ selected_subject = "Дискретна математика"
                    "Комп'ютерні мережі":
                        $ selected_subject = "Комп'ютерні мережі"
                    "Патерни проєктування":
                        $ selected_subject = "Патерни проєктування"
                $ change_stress(+10)
                jump study

            "Зробити лабораторні знайомим" if time_of_day == "вечір" and energy >= 10:
                $ change_energy(-10)
                $ change_stress(+5)
                "Ти вирішив допомогти молодшим курсам виконати лабораторні роботи."

                centered "Загрузка...{nw}"

                $ grid_width = renpy.random.randint(3, 6)
                $ grid_height = renpy.random.randint(3, 6)
                $  chosen_img  =  renpy.random.choice(['4298785909_ccbfe3b453_o.jpg',  '5567411233_8ae94bd497_o.jpg',
'Colorful_Lake_3.JPG', 'image.jpg', 'Red_eyed_tree_frog_edit2.jpg'])
```

```
                window hide
                hide screen hud
                hide screen stress_bar
                hide screen energy_bar
                hide screen hud_money

                call puzzle from _call_puzzle

            "Попрацювати кур'єром" if (time_of_day == "день" or time_of_day == "після навчання") and energy >= 30:
                $ change_energy(-30)
                $ change_stress(+5)
                "Ти пішов підробляти кур'єром доставки їжі."
                call numbers_game from _call_numbers_game

            "Прогулятись" if time_of_day == "день" or time_of_day == "після навчання" or time_of_day == "вечір":
                if time_of_day == "вечір":
                    scene bg city evening with fade
                else:
                    scene bg city day with fade
                "Ти вирішив відпочити та прогулятись по місту з друзями..."
                $ change_stress(-5)
                $ change_energy(renpy.random.randint(5, 30))

            "Купити щось":
                "Ти вийшов на вулицю щоб купити щось..."
                call shop from _call_shop
                jump activity_choice

            "Піти поспати" if time_of_day == "вечір":
                "Ти вирішив піти поспати раніше."
                $ change_stress(-15)
                $ change_energy(+15)
    return


label show_day_info:
    $ time_of_day = "ранок"
    $ weekday, formatted_date = get_game_date()
    $ days_left = max_days - current_day + 1

    scene bg dorms day with fade

    e "Сьогодні [weekday], [formatted_date]."
    if days_left > 1:
        e "До сесії залишилося [days_left] днів."
    elif passed_count < 3:
        e "Сьогодні я маю перездати іспити..."
    else:
        e "Сьогодні день сесії."

    return
```

study.rpy

```
        label study:

            if time_of_day == "день" or time_of_day == "після навчання":
                scene bg library study with fade
                "В бібліотеці ти почитав книги "
            else:
                scene bg dorms evening with fade
                "Ти відкрив на ноутбуці на лекцію з [selected_subject]."

            if energy >= 20:
                $ change_energy(-20)
                if prep_scores[selected_subject] < 3:
                    $ prep_scores[selected_subject] += 1
                "Відкривши, ти знаходиш таку інформацію:"
                $ show_random_explanations(selected_subject, 1)
            else:
                $ change_energy(+5)
                "Ти прочитав її, але не зміг нічого запам'ятати через втому..."
            return
```