МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РОЗРОБКА 2D ГРИ З ПРОЦЕДУРНОЮ ГЕНЕРАЦІЄЮ СВІТУ, МЕХАНІКОЮ ЗБОРУ РЕСУРСІВ, СИСТЕМОЮ ТОРГІВЛІ ТА СТВОРЕННЯ ПРЕДМЕТІВ»**

Виконав: студент 4 курсу, групи 6.1222-з-с2
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
П.О. Севастьянова
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Миронова Н.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Севастьяновій Поліні Олексіївні
(прізвище, ім'я та по-батькові)

1. Тема роботи Проєктування та розробка 2D гри з процедурною генерацією світу, механікою збору ресурсів, системою торгівлі та створення предметів

керівник роботи Миронова Наталя Олексіївна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1742-с

2. Строк подання студентом роботи 15.05.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Додатки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи.
2. Проєктування системи
3. Розробка та реалізація ігрових механік

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) 0
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.12.2025	
2.	Збір вихідних даних.	21.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	15.04.2026	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи бакалавра.	15.05.2026	
7.	Захист кваліфікаційної роботи.	29.05.2026	

Студент _____ П.О. Севастьянова
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.О. Миронова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та розробка 2D гри з процедурною генерацією світу, механікою збору ресурсів, системою торгівлі та створення предметів»: 74 с., 29 рис., 5 табл., 12 джерел, 3 додатки.

ГЕЙМДИЗАЙН, ІГРОВИЙ РУШІЙ GODOT, ПРОЦЕДУРНА ГЕНЕРАЦІЯ ВМІСТУ, СИСТЕМНИЙ АНАЛІЗ, СЦЕНАРІЙ ВЗАЄМОДІЇ, УПРАВЛІННЯ СТАНАМИ, ЦИФРОВА ОБРОБКА ЗОБРАЖЕНЬ.

Об’єкт дослідження – процес розробки ізометричної 2D гри в жанрі пригодницького симулятора з елементами менеджменту ресурсів.

Предмет дослідження – механіки процедурної генерації ігрового середовища, алгоритми комбінування інгредієнтів за колірними коефіцієнтами та системи взаємодії гравця з ігровим світом.

Мета роботи – розробити та реалізувати ігровий прототип, що поєднує процедурну генерацію локацій, систему збору ресурсів, механіку варіння зіль та ігрову економіку для досягнення фінальної сюжетної мети.

Метод дослідження – використання ігрового рушія Godot Engine та скриптової мови GDScript для програмування логіки, методи процедурної генерації мап, створення піксель-арт графіки в ізометричній проекції за допомогою Aseprite, тестування ігрових механік.

У ході виконання роботи було спроектовано та розроблено прототип комп’ютерної гри, де гравець виступає в ролі відьмочки. Основний акцент зроблено на створенні гнучкої системи алхімії: реалізовано дві механіки змішування компонентів на основі тегів та оброблення інгредієнтів.

Розроблено модуль процедурної генерації карти, де персонаж переміщується по осі X, для збору інгредієнтів для сцени з приготуванням зілля. Реалізовано систему торгівлі, яка дозволяє гравцеві торгувати з npc прогресувати через купівлю матеріалів. Особливу увагу приділено

математичному моделюванню алхімічної системи змішування інгредієнтів та реалізації алгоритмів процедурної генерації вмісту для створення нескінченного ігрового світу.

Одержані результати підтверджують ефективність обраних інструментів для створення комплексних ігрових механік у 2D середовищі.

SUMMARY

Bachelor's Qualifying Theses "Design and development of a 2D game with procedural world generation, resource collection mechanics, a trading system, and item crafting": 74 p., 29 fig., 5 tables, 12 sources, 3 supplements.

GAME DESIGN, GODOT GAME ENGINE, PROCEDURAL CONTENT GENERATION, SYSTEM ANALYSIS, INTERACTION SCENARIO, STATE MANAGEMENT, DIGITAL IMAGE PROCESSING.

The object of research is the process of developing an isometric 2D game in the genre of an adventure simulator with elements of resource management.

The subject of research is the mechanics of procedural generation of the game environment, algorithms for combining ingredients based on color coefficients, and systems of player interaction with the game world.

The purpose of the work is to design and implement a game prototype that combines procedural generation of locations, a resource collection system, potion brewing mechanics, and a game economy to achieve the final narrative goal

The research method involves the use of the Godot Engine and the GDScript scripting language for programming logic, methods of procedural map generation, creation of pixel-art graphics in isometric projection using Aseprite, and testing of game mechanics.

In the course of the work, I designed and developed a prototype of a computer game in which the player takes on the role of a witch. The main focus is on creating a flexible alchemy system: two mechanics for mixing components based on tags and ingredient processing were implemented.

I developed a procedural map generation module where the character moves along the X-axis to collect ingredients for the potion-making scene. A trading system was implemented, allowing the player to trade with NPCs and progress through purchasing materials. Special attention was given to the mathematical

modeling of the alchemy ingredient-mixing system and the implementation of procedural content generation algorithms to create an infinite game world.

The obtained results confirm the effectiveness of the selected tools for creating complex game mechanics in a 2D environment.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 6
Скорочення та умовні познаки 7
Вступ..... 11
1 Теоретичні основи..... 13
1.1Аналіз індустрії інді ігор та жанрових особливостей 13
1.1.1 Визначення та роль інді проєктів у сучасній гейм індустрії... 13
1.1.2 Аналіз пригодницьких ігор з елементами крафту та торгівлі. 14
1.2 Аналіз ігрових рушіїв для розробки проєкту..... 15
1.2.1 Вибір ігрового рушія 15
1.2.2 Ігровий рушій Godot..... 20
1.3 Обґрунтування вибору мови програмування..... 23
1.3.1 Огляд мови GDScript та її інтеграція з вузловою системою ... 24
1.3.2 Порівняльний аналіз GDScript та C# 26
1.4 Аналіз допоміжного програмного забезпечення 28
1.4.1 Створення графічний ресурсів Aseprite..... 28
1.5 Аналіз допоміжного програмного забезпечення 31
1.5.1 Створення графічний ресурсів Aseprite..... 32
1.5.1.1 Послідовна лінійна генерація 32
1.5.1.2 Генерація на основі сегментів 33
1.5.1.3 Зональна сегментація з динамічним кроком..... 34
1.5.2 Порівняльний аналіз шумових функцій та зональної сегментації 35
2 Проєктування системи..... 37
2.1 Робота в Aseprite 37

2.2 Проєктування сцен 39
2.2.1 Реалізація сцени процедурної генерації ігрового світу 40
2.2.2 Система обробки інгредієнтів та створення зілля 44
2.2.3 Система обробки замовленнь 53
2.2.4 Збереження даних у форматі JSON 56
3 Розробка та реалізація ігрових механік 59
3.1 Інструменти та загальна характеристика середовища розробки 59
3.2 Графічне оформлення та візуальна складова 60
3.2.1 Розробка ігрових спрайтів та персонажа 60
3.2.2 Розробка ігрових спрайтів та персонажа 60
3.3 Реалізація ігрового процесу та взаємодій 62
3.3.1 Система станів персонажа та інгредієнтів 62
3.3.2 Модифікатори предметів та система вимог 63
3.3.3 Модифікатори предметів та система вимог 63
3.3.4 Проєктування системи замовлень та торгівлі 64
3.4 Організація збереження ігрового прогресу 65
Висновки 67
Перелік посилань 70
Додаток А Діаграма загальна схема взаємодії сцен 71
Додаток Б Діаграма схема створення світу 722
Додаток В Діаграма структура даних для збереження 733

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

GDScript	–	мова програмування рушія Godot.
GUI	–	графічний інтерфейс користувача.
ПГ	–	процедурна генерація вмісту.
IDE	–	JSON Web Token, формат токена для безпечної передачі інформації.
ПК	–	персональний комп’ютер
ПЗ	–	програмне забезпечення.
ТЗ	–	техічне завдання.
ООП	–	об’єктно-орієнтоване програмування).
Side scroller	–	жанр гри, у якому огляд відбувається збоку, а персонаж переміщується по платформі.
Node	–	базовий об’єкт ієрархії Godot.
Inheritance	–	механізм створення нових сцен/класів на основі існуючих.
Scene	–	автономний ієрархічний модуль (клас) у рушії Godot.
Sprite	–	двовимірне растрове зображення.
JSON	–	формат обміну та збереження даних.
Delta	–	інтервал часу між кадрами в ігровому циклі.
Collision	–	система визначення зіткнень об’єктів.

ВСТУП

Актуальність теми дослідження зумовлена стрімким зростанням індустрії інді ігор та постійним пошуком нових підходів до поєднання процедурної генерації вмісту з глибокими механіками взаємодії. Багато сучасних розробок стикаються з проблемою швидкої втрати інтересу гравця через одноманітність ігрового процесу тому розробка унікальних систем таких як колірна алхімія та динамічне відкриття рецептів є актуальним завданням для створення продукту з високим потенціалом для повторного проходження. Необхідність розробки даного прототипу гри також зумовлена пошуком балансу між випадковою генерацією світу та структурованим прогресом персонажа через збір ресурсів та торгівлю.

Архітектура Godot включає в себе об’єктно орієнтоване проєктування з його гнучкою системою сцен та ієрархією вузлів що дозволяє складати або збирати сцени подібно до вкладених префабів де будь яка сцена може бути зброєю персонажем предметом дверима чи рівнем [1]. Це дозволяє успадковувати та розширювати ці сцени що допомагає будувати проєкти так щоб їх структура відповідала архітектурі гри а зміни в батьківських об’єктах миттєво оновлювали всі похідні елементи.

Метою роботи є створення ігрового прототипу 2D гри у жанрі платформера який інтегрує механіки процедурної генерації локацій збирання ресурсів систему виготовлення зіль на основі колірних коефіцієнтів та ігрову економіку для досягнення сюжетної мети.

Для досягнення поставленої мети необхідно виконати такі завдання як аналіз особливостей створення 2D ігор з платформерним переміщенням та вибір інструментарію розробки а також розробка модуля процедурної генерації мапи з розподілом ресурсів. До завдань також належить реалізація системи інвентарю та механіки збору інгредієнтів створення алгоритму алхімії що базується на комбінуванні основних кольорів та впровадження торгової

системи. Важливою частиною роботи є побудова сюжетної лінії з фінальною метою відкриття сімейного сейфу та розробка візуального стилю проекту в техніці піксель арт.

Об’єктом дослідження є процес розробки 2D гри у жанрі пригодницького симулятора з платформерним переміщенням та елементами менеджменту ресурсів. Предметом дослідження виступають алгоритми процедурної генерації ігрового середовища та механіки комбінаторного створення ігрових предметів.

У роботі запропоновано та реалізовано систему алхімічного крафту де властивості продукту визначаються математичним співвідношенням кольорів інгредієнтів що додає елемент стратегічного планування. Спроектований прототип демонструє цілісну екосистему де процедурна генерація світу безпосередньо впливає на економічні можливості гравця залежно від його рішень. Результати роботи мають практичне значення та можуть бути використані як база для створення комерційних проектів у жанрі пригодницьких ігор. Окремі модулі такі як процедурна генерація мапи та система крафту можуть бути адаптовані для інших проектів на базі ігрового рушія Godot.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Аналіз індустрії інді ігор та жанрових особливостей

1.1.1 Визначення та роль інді проєктів у сучасній гейм індустрії

Незалежна розробка ігор, або інді сегмент, займає особливе місце в сучасній цифровій економіці завдяки здатності до швидкої адаптації та впровадження експериментальних ігрових механік. На відміну від великих корпоративних проєктів, які орієнтовані на масового споживача та використовують перевірені часом, але стандартизовані шаблони, інді проєкти створюються невеликими колективами або соло розробниками.

У межах даної роботи інді формат обрано через можливість зосередити обмежені ресурси на якості ігрової логіки та авторському візуальному стилі, уникаючи складних технологічних рішень, що притаманні масовому промисловому виробництву ігор.

Для детальнішого системного аналізу інді сектору доцільно виділити такі стратегічні особливості:

- творча автономія: відсутність зовнішнього фінансового та маркетингового тиску з боку видавців дозволяє розробнику самостійно визначати вектор розвитку проєкту, що є критично важливим для створення нестандартних систем, таких як процедурна генерація біомів з різною щільністю ресурсів;
- доступність технологій: розвиток сучасних засобів розробки з відкритим доступом значно знижує поріг входу в індустрію. Це дозволяє розробникам мінімізувати стартові витрати та зосередитися на проектуванні архітектури продукту, а не на закупівлі дороговартісних ліцензій;

– гнучкість ігрового циклу: незалежні проєкти мають змогу концентруватися на опрацьованості ключових механік, що забезпечує високу якість ігрового процесу навіть при обмежених людських ресурсах;
– авторська ідентичність: використання стилізованих візуальних рішень сприймається аудиторією як усвідомлений вибір автора, що дозволяє створити цілісну атмосферу проєкту без необхідності копіювання фотореалістичних стандартів промислового виробництва.

Роль інді ігор полягає у підтримці різноманітності контенту та постійному оновленні стандартів взаємодії з гравцем. Для розробника це означає необхідність аналізу потреб користувачів та вибору таких методів розробки, що дозволяють створити продукт з потенціалом для повторного проходження при раціональному використанні часових та технічних ресурсів [2].

1.1.2 Аналіз пригодницьких ігор з елементами крафту та торгівлі

Основою ігрового процесу є тісна взаємодія між дослідженням навколишнього середовища та процесом створення нових ігрових об’єктів. Дослідження територій дозволяє гравцеві знаходити первинні ресурси, тоді як система синтезу виступає інструментом для їхньої глибокої переробки. Такий підхід створює безперервний цикл, де кожна дія має чітку мету та результат, що важливо для підтримки інтересу користувача протягом всієї гри.

Механіка збору інгредієнтів стимулює гравця активно вивчати різні зони ігрової карти. Оскільки ігровий світ поділений на різні біоми, такі як ліси, рівнини та магічні зони, кожен з них містить специфічні матеріали. Гравець змушений не просто переміщатися по карті, а планувати складні маршрути для пошуку рідкісних рослин або мінералів. Це перетворює навігацію світом на самостійну стратегічну задачу, де успіх залежить від знання особливостей кожної локації. Центральною частиною проєкту є система синтезу, яку в ігровій

індустрії називають крафтом. На відміну від стандартних систем, де створення предметів відбувається за готовими шаблонами, у даній роботі розглядається модель експериментального синтезу. Кожен інгредієнт володіє набором унікальних властивостей, що виражені через систему тегів та колірних коефіцієнтів. Процес варіння зілля стає аналітичним дослідженням: гравець комбінує матеріали, спостерігаючи за тим, як змінюються параметри кінцевого продукту залежно від обраного складу.

Розвиток у грі базується на накопиченні досвіду та знань про ігрові механіки. Гравець поступово відкриває властивості нових інгредієнтів, знаходить найбільш ефективні комбінації та вчиться створювати складні еліксири. Кожне успішне відкриття наближає персонажа до досягнення головної сюжетної мети — створення унікального зілля, здатного відкрити сімейний сейф. Таким чином, система крафту є не лише допоміжною механікою, а ключовим елементом прогресу та розповіді.

1.2 Аналіз ігрових рушіїв для розробки проєкту

На етапі вибору інструментарію було розглянуто три технології, які дозволяють створювати 2D ігри. Кожен рушій має свою філософію побудови ігрового простору та обробки даних.

1.2.1 Вибір ігрового рушія

У процесі вибору програмного забезпечення для реалізації ігрового проєкту було проведено аналіз сучасних ігрових рушіїв, серед яких найбільшу увагу було приділено таким рішенням, як Unity, Cocos2d та Godot. Дані рушії є широко відомими у сфері розробки ігор і активно використовуються як у професійному середовищі, так і серед незалежних розробників. Кожен із них

має свої особливості, переваги та недоліки, що впливають на вибір інструменту залежно від типу проєкту, його складності та вимог до реалізації.

Першим було розглянуто Unity як один із найпопулярніших рушіїв у сучасній індустрії. Його основною перевагою є універсальність, оскільки він дозволяє створювати як двовимірні, так і тривимірні ігри. У складі рушія вже присутня велика кількість вбудованих систем, які відповідають за рендеринг графіки, обробку фізичних взаємодій, роботу зі звуком, анімацію та інші аспекти функціонування гри. Це дозволяє реалізовувати складні проєкти без необхідності створювати базові механізми з нуля. Крім того, Unity має інтуїтивно зрозумілий графічний інтерфейс із гнучкими інструментами візуального редагування, що дозволяє зручно керувати об’єктами та ресурсами проєкту (рис. 1.1).

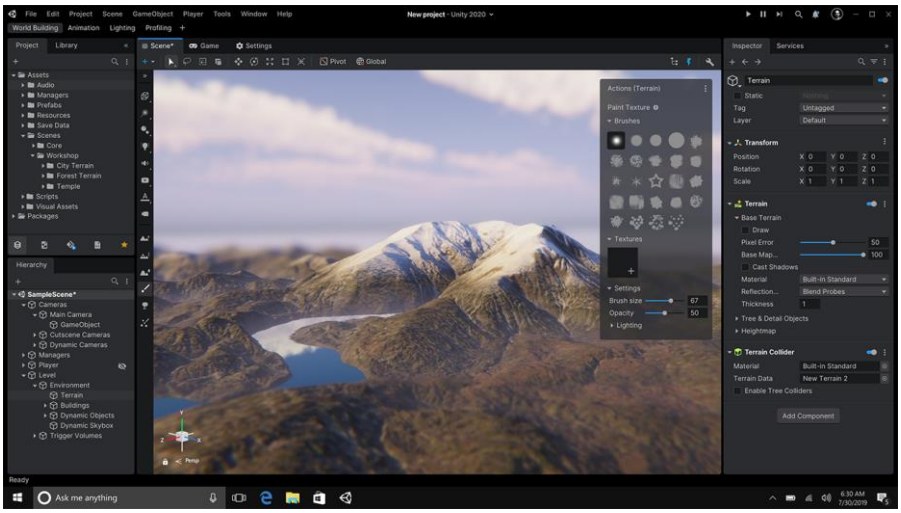


Рисунок 1.1 – Інтерфейс рушія Unity

Однак у ході практичного аналізу було встановлено, що така універсальність не завжди є перевагою. У випадку невеликого або навчального проєкту значна частина функціоналу Unity залишається невикористаною, але при цьому впливає на загальну складність роботи. Розробнику доводиться взаємодіяти з великою кількістю інструментів, навіть якщо фактично використовуються лише базові можливості рушія. Це створює додаткове навантаження та ускладнює процес розробки.

Особливу увагу слід приділити структурі проєкту в Unity. Вона базується на використанні об’єктів сцени, які складаються з набору компонентів. Такий підхід забезпечує гнучкість, але водночас потребує розуміння того, як саме ці компоненти взаємодіють між собою. У процесі роботи це може призводити до ситуацій, коли навіть прості зміни вимагають додаткового налаштування та перевірки, що уповільнює розробку. Інтерфейс редактора Unity також є досить складним і насиченим великою кількістю елементів. Він включає різні вікна для роботи зі сценою, ресурсами, властивостями об’єктів і налагодженням. Хоча це надає широкі можливості, для ефективного використання необхідно витратити значний час на освоєння інструменту. Для студентського проєкту це може бути недоцільним, оскільки основна увага повинна приділятися створенню гри, а не вивченню складного середовища розробки.

Крім того, Unity є досить вимогливим до апаратних ресурсів. Його використання передбачає наявність достатньо продуктивного комп’ютера, що може бути проблемою у навчальних умовах. Повільна робота редактора або тривале завантаження проєкту негативно впливають на ефективність розробки та знижують швидкість виконання завдань[3].

Ще одним важливим аспектом є використання сторонніх ресурсів. Хоча Unity надає доступ до великої кількості готових матеріалів, їх інтеграція може ускладнювати структуру проєкту. У деяких випадках виникають конфлікти між різними компонентами, що ускладнює процес налагодження та оптимізації. У результаті це може призводити до зниження стабільності роботи гри. Таким чином, незважаючи на широкий функціонал і популярність, Unity у контексті даного проєкту виявився занадто складним і перевантаженим інструментом, використання якого не є повністю доцільним.

Другим було розглянуто Cocos2d як легке та спеціалізоване рішення для створення двовимірних ігор. Даний рушій історично орієнтований на мобільну розробку та проєкти середнього і малого масштабу, де важливими є продуктивність, стабільність роботи та мінімальне навантаження на апаратні ресурси. Завдяки своїй архітектурі та відносно простій структурі виконання,

Cocos2d забезпечує високу швидкість роботи і передбачувану поведінку ігрових сцен навіть на пристроях із обмеженими характеристиками.

Архітектурно Cocos2d базується на концепції сцен і вузлів, що дозволяє будувати ігрові об’єкти у вигляді ієрархічної структури. Такий підхід спрощує організацію графічних елементів та логіки взаємодії між ними, оскільки кожен елемент сцени є вузлом із власними властивостями та поведінкою. Це створює зрозумілу модель побудови гри, яка добре підходить для 2D-проєктів і простих ігрових механік.

Водночас при більш глибокому аналізі стає очевидно, що можливості Cocos2d поступаються сучаснішим універсальним рушіям. Основний акцент у класичній реалізації Cocos2d-x робиться на програмному керуванні логікою гри, що означає значну залежність від ручної реалізації більшості систем. Це підвищує вимоги до розробника та збільшує час, необхідний для створення навіть базового функціоналу[4]. На відміну від сучасних візуальних середовищ, Cocos2d має обмежений графічний інтерфейс, що орієнтує розробника переважно на написання коду для керування ігровими об’єктами (рис. 1.2).

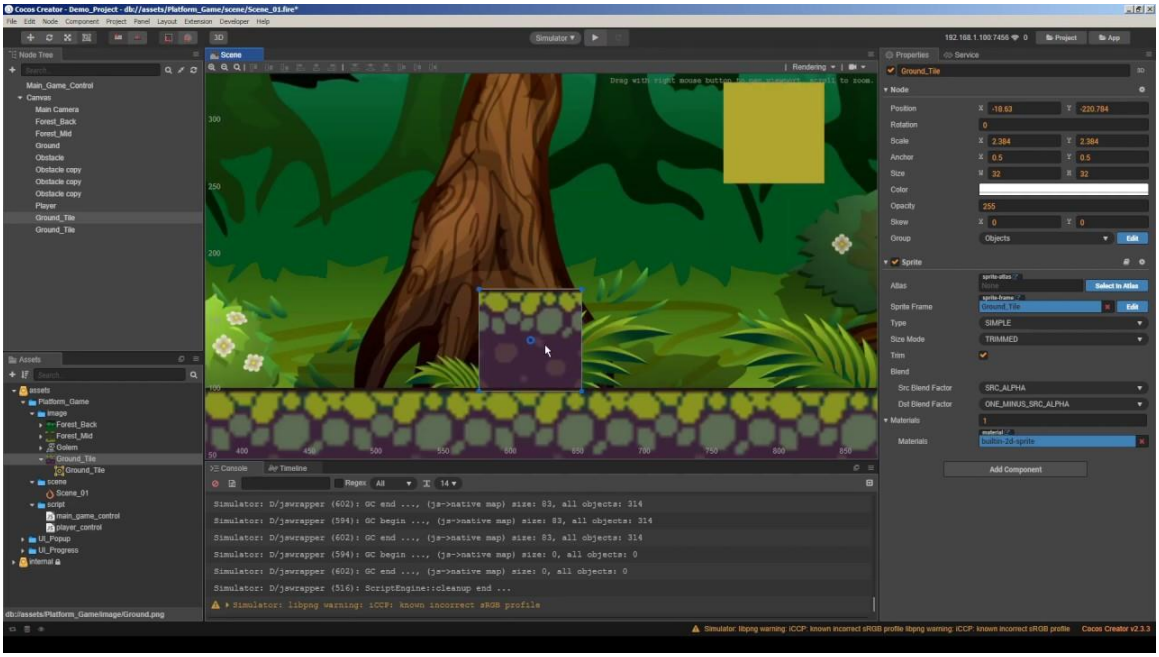


Рисунок 1.2 – Інтерфейс рушія Cocos Creator

Слід також враховувати, що сучасна екосистема Cocos включає платформу Cocos Creator, яка надає графічний редактор та більш зручне середовище для
розробки. Однак навіть у цьому випадку рівень інтеграції інструментів та загальна зручність робочого процесу в окремих сценаріях можуть поступатися більш розвиненим рушіям, таким як Unity або Godot, які пропонують більш цілісне середовище розробки «з коробки».

Однією з практичних особливостей використання Cocos є те, що значна частина процесів, пов’язаних із налаштуванням сцени, розміщенням об’єктів та організацією ігрової логіки, може вимагати додаткових ручних дій. Це підвищує складність розробки та зменшує швидкість створення прототипів, особливо у випадках, коли необхідно швидко тестувати ігрові механіки або змінювати структуру проєкту.

Крім того, у порівнянні з більш сучасними рушіями, Cocos2d має менш розвинену кількість вбудованих інструментів та готових рішень. Це означає, що розробнику часто доводиться самостійно реалізовувати функціональність, яка в інших середовищах вже інтегрована на рівні рушія. У результаті це впливає на загальну швидкість розробки та підвищує навантаження на процес проєктування системи. Незважаючи на свої переваги у вигляді продуктивності, легкості та орієнтації на 2D-ігри, Cocos2d виявляється менш зручним інструментом у контексті швидкої розробки та навчальних проєктів, де важливо мінімізувати час на налаштування середовища та зосередитися на реалізації ігрової логіки.

У результаті проведеного аналізу також було розглянуто рушій Godot як альтернативне рішення. Даний рушій поєднує достатній рівень функціональності з простотою використання, має інтуїтивно зрозумілу структуру та інтегроване середовище розробки, що дозволяє швидко створювати та тестувати ігрові прототипи. У порівнянні з Unity він є менш ресурсомістким, а у порівнянні з Cocos2d – більш зручним завдяки наявності вбудованих інструментів та цілісного робочого середовища.

Таким чином, на основі проведеного аналізу було прийнято рішення відмовитися від використання Unity та Cocos2d і обрати Godot як основний інструмент для реалізації проєкту, оскільки він найбільш повно відповідає вимогам навчальної розробки та забезпечує оптимальний баланс між простотою, функціональністю та швидкістю створення проєкту.

1.2.2 Ігровий рушій Godot

Godot – це сучасний відкритий ігровий рушій, який використовується для створення 2D та 3D ігор (рис. 1.3). Його головною особливістю є повна безкоштовність та відкритий вихідний код, що означає можливість вільного використання, модифікації та розповсюдження без ліцензійних обмежень. Це робить рушій особливо популярним серед інді-розробників, студентів та невеликих команд.



Рисунок 1.3 – Рушій Godot

Історія Godot починається приблизно у 2007 році, коли два розробники з Аргентини, Juan Linietsky та Ariel Manzur, почали створювати власний ігровий рушій для внутрішнього використання. Спочатку він не був публічним продуктом і використовувався як інструмент для невеликих комерційних

проєктів. Лише у 2014 році рушій був офіційно оприлюднений як відкритий проєкт під ліцензією MIT, що стало важливим моментом у його розвитку, оскільки дозволило спільноті активно долучатися до його вдосконалення.

З моменту відкриття коду Godot почав швидко розвиватися завдяки активній підтримці спільноти. Рушій отримував регулярні оновлення, які покращували продуктивність, стабільність та розширювали функціональність. Особливо значним кроком став перехід до версії Godot 4, де було суттєво покращено 2D та 3D рендеринг, а також оптимізовано роботу з графікою. Надалі розвиток продовжився у напрямку більш сучасної архітектури, і з виходом Godot 4 рушій отримав новий рендерер, покращену продуктивність та розширені можливості для 3D-графіки.

Однією з ключових концепцій Godot є система сцен і вузлів. Усі елементи гри, включаючи персонажів, об’єкти, інтерфейс та ефекти, представляються у вигляді вузлів, які об’єднуються у сцени. Такий підхід дозволяє будувати проєкт модульно, тобто розділяти його на незалежні частини, які легко перевикористовуються. Наприклад, персонаж може бути окремою сценою, яку можна додавати в різні рівні гри без необхідності повторного налаштування.

Ця система інтегрована у зручний графічний інтерфейс редактора, що дозволяє наочно маніпулювати вузлами та миттєво бачити результат налаштувань (рис. 1.4).

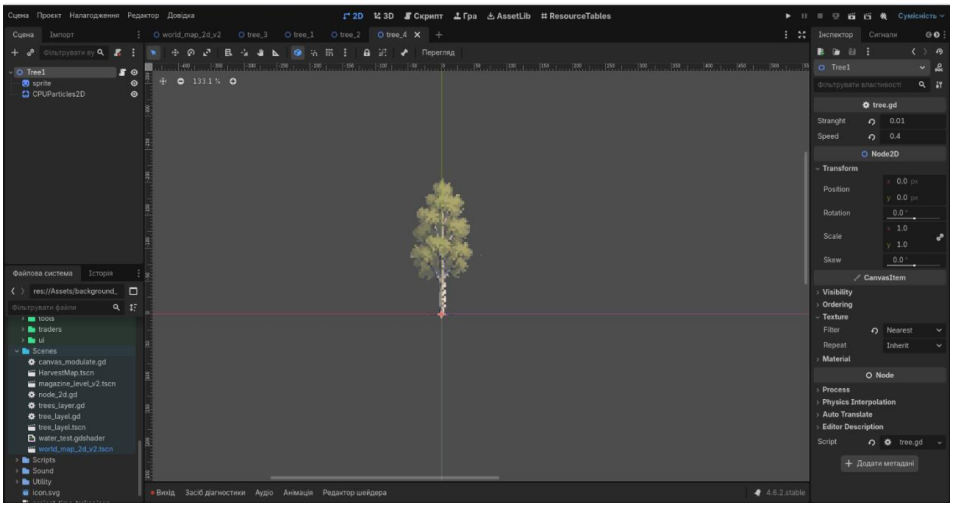


Рисунок 1.4 – Інтерфейс рушія Godot

Godot підтримує повноцінну двовимірну та тривимірну розробку, однак особливо сильним він є саме у 2D-сегменті.

Ще однією важливою особливістю є вбудована мова програмування GDScript. Вона створена спеціально для Godot і має простий синтаксис, що нагадує Python. Це дозволяє швидко освоїти основи розробки і зосередитися на створенні ігрової логіки, а не на складних технічних деталях. Крім того, рушій підтримує й інші підходи до програмування, що робить його більш гнучким.

З технічної точки зору Godot є досить легким рушієм. Він не потребує великої кількості системних ресурсів і може працювати навіть на середніх або слабких комп'ютерах. Це особливо важливо для студентських проєктів, де не завжди є доступ до потужного обладнання. Крім того, швидкість запуску редактора та тестування гри є досить високою, що позитивно впливає на процес розробки.

До переваг Godot можна віднести також повністю інтегроване середовище розробки. У ньому вже є все необхідне: редактор сцен, інструменти для анімації, робота з ресурсами, налагодження та тестування гри. Це дозволяє працювати в одному середовищі без необхідності використовувати додаткові програми, що значно спрощує процес створення гри.

Серед основних переваг Godot можна виділити:

– відкритий вихідний код та безкоштовну ліцензію;
– легкість та швидкість роботи;
– зручну систему сцен і вузлів;
– сильну підтримку 2D-розробки;
– просту та зрозумілу мову програмування;
– низькі системні вимоги;
– швидке створення прототипів.

Попри значну кількість переваг, Godot також має певні обмеження. Зокрема, його рівень поширеності у великих комерційних студіях все ще поступається більш відомим і давно закріпленим на ринку рушіям. Це безпосередньо впливає на кількість готових професійних рішень, бібліотек та

комерційних прикладів використання, які доступні для розробників. У результаті деякі специфічні задачі можуть вимагати більшої самостійної реалізації порівняно з більш популярними платформами.

Окремо варто відзначити, що хоча можливості тривимірної розробки в Godot активно розвиваються і з кожною версією стають більш досконалими, у деяких складних графічних сценаріях вони все ще можуть поступатися більш спеціалізованим рушіям, які мають довшу історію розвитку у сфері 3D-розробки. Водночас для 2D-проєктів ці обмеження практично не є критичними, оскільки саме двовимірний напрямок є одним із найсильніших аспектів цього рушія.

Також слід зазначити, що екосистема готових плагінів та інтеграцій у Godot є менш масштабною у порівнянні з більш комерційно орієнтованими рушіями. Це означає, що деякі додаткові функції можуть вимагати власної реалізації або адаптації існуючих рішень. Проте цей недолік поступово зменшується завдяки активному розвитку спільноти та регулярним оновленням рушія, які розширюють його функціональність і підвищують стабільність екосистеми.

У цілому Godot можна охарактеризувати як сучасний та збалансований ігровий рушій, який поєднує в собі простоту використання, гнучкість архітектури та достатній набір інструментів для реалізації як навчальних, так і невеликих комерційних проєктів. Його відкритий вихідний код, легкість у освоєнні та орієнтація на швидкий процес розробки роблять його особливо зручним для індивідуальної розробки, де важливими є ефективність роботи, швидкість створення прототипів та зрозуміла структура проєкту.

1.3 Обґрунтування вибору мови програмування

У процесі розробки ігрового проєкту важливим етапом є вибір мови програмування, оскільки саме вона визначає підхід до реалізації ігрової логіки,

взаємодії об’єктів та роботи з функціоналом рушія. Для реалізації проєкту на базі Godot було розглянуто декілька варіантів, зокрема вбудовану мову GDScript та альтернативну підтримку мови C#. Основна увага приділялася простоті використання, швидкості розробки та інтеграції з архітектурою рушія.

1.3.1 Огляд мови GDScript та її інтеграція з вузловою системою

Основною мовою програмування, яка використовується в Godot, є GDScript. Це високорівнева скриптова мова, яка була спеціально розроблена для даного рушія з метою забезпечення максимально зручної та швидкої взаємодії з його внутрішньою архітектурою. На відміну від універсальних мов програмування, GDScript створювався саме з урахуванням особливостей Godot, що робить його тісно інтегрованим у всі процеси розробки гри.

Однією з ключових особливостей GDScript є його безпосередня інтеграція з вузловою node-based системою рушія. У Godot будь-який елемент гри представлений у вигляді вузла, а сцена являє собою ієрархічну структуру таких вузлів. GDScript дозволяє напряму працювати з цією структурою, отримувати доступ до вузлів, змінювати їх властивості, викликати їх методи та керувати їх поведінкою без необхідності використання додаткових складних механізмів або проміжних шарів абстракції. Це робить процес розробки більш інтуїтивно зрозумілим та логічним.

Синтаксис GDScript є простим і максимально наближеним до природної логіки програмування. Його структура нагадує Python, що проявляється у використанні відступів замість фігурних дужок, простих конструкцій умов та циклів, а також відсутності великої кількості службового коду. Завдяки цьому зменшується кількість “шуму” в коді, і розробник може зосередитися безпосередньо на реалізації ігрової логіки. Це особливо важливо в умовах навчальних або інді-проєктів, де швидкість розробки має велике значення.

Ще однією важливою перевагою GDScript є його глибока інтеграція з внутрішнім API рушія Godot. Це означає, що більшість функцій рушія доступні

безпосередньо з коду без необхідності додаткових налаштувань або складних підключень. Розробник може легко керувати сценою, взаємодіяти з об’єктами, обробляти події та реалізовувати ігрову логіку в межах одного середовища. Такий підхід значно прискорює процес створення гри та спрощує її структуру.

GDScript є високорівневим, об’єктно-орієнтованим, імперативним і поступово введена мова програмування, створена для Godot. Він використовує синтаксис на основі відступів, схожий на такі мови, як Python. Його метою є оптимізація та тісна інтеграція з Godot Engine, що забезпечує велику гнучкість для створення та інтеграції вмісту. GDScript повністю не залежить від Python і не базується на ньому. [5]. GDScript має свою документацію (рис. 1.5).

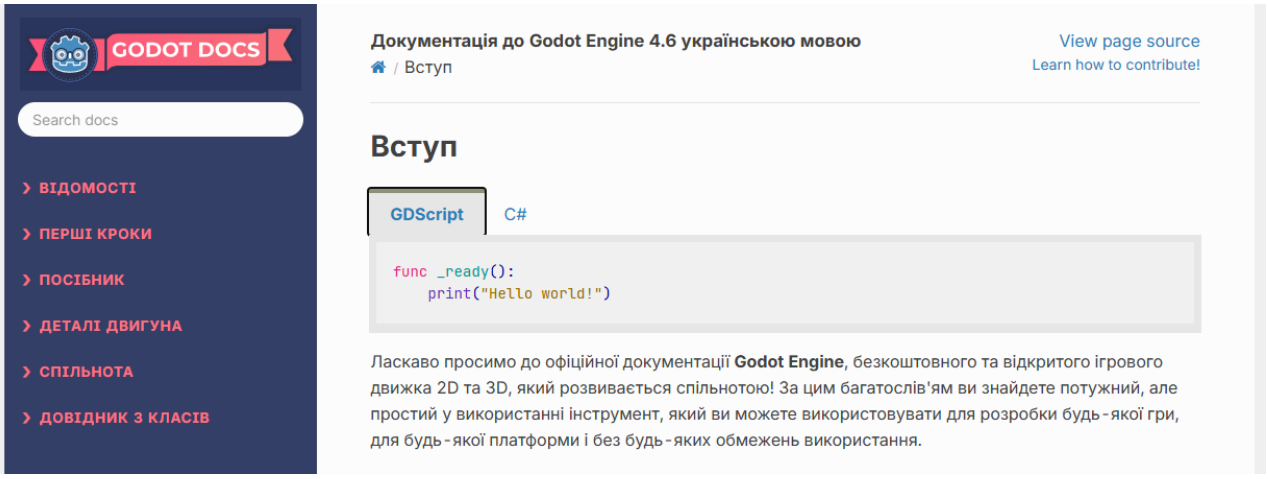


Рисунок 1.5 – HTML сторінка документації GDScript

GDScript має система сигналів, яка є однією з ключових концепцій взаємодії між об’єктами в Godot. Сигнали дозволяють реалізовувати обмін подіями між різними вузлами без прямої залежності між ними. Це означає, що один об’єкт може реагувати на події іншого, не знаючи його внутрішньої реалізації. Такий підхід зменшує зв’язаність коду, робить архітектуру більш гнучкою та полегшує подальшу підтримку і масштабування проєкту.

Наприклад, у ігровій логіці це може виглядати як ситуація, коли персонаж отримує сигнал про зіткнення з ворогом або зміну стану гри, і відповідно реагує на нього. При цьому самі об’єкти не повинні безпосередньо

викликати методи один одного, що значно спрощує структуру взаємодії та зменшує ризик виникнення помилок у складних системах.

GDScript підтримує швидкий цикл розробки. Зміни в коді можуть бути протестовані практично одразу, без тривалих етапів компіляції. Це дозволяє швидко перевіряти ідеї, вносити корективи та оптимізувати поведінку гри в режимі реального часу. Така особливість є особливо корисною під час створення прототипів або навчальних проєктів, де важливо швидко отримати результат.

Завдяки поєднанню простого синтаксису, тісної інтеграції з рушієм, гнучкої системи сигналів та високої швидкості розробки, GDScript є оптимальним інструментом для роботи в Godot. Він дозволяє ефективно реалізовувати ігрову логіку, зменшує складність проєкту та забезпечує зручність у процесі розробки, що особливо важливо при створенні невеликих та середніх ігор.

1.3.2 Порівняльний аналіз GDScript та C#

Вибір мови програмування для реалізації логіки ігрового проєкту є критично важливим етапом, оскільки він визначає не лише швидкість розробки, а й архітектурну стійкість та продуктивність фінальної системи. Рушій Godot пропонує розробнику мультимовне середовище, де основними інструментами виступають GDScript (спеціалізована високорівнева мова) та C# (універсальна об’єктно-орієнтована мова на базі платформи .NET). Проведений порівняльний аналіз дозволив оцінити доцільність використання кожного з цих інструментів у межах системного проектування ігрових механік.

Мова C# представляє собою потужний інструмент промислового рівня, що характеризується строгою типізацією та розвиненою системою управління типами даних. Її основною перевагою є висока швидкість виконання обчислювально складних операцій, що забезпечується за рахунок Just-In-Timeкомпіляції. Застосування C# дозволяє будувати складні ієрархічні

структури та використовувати переваги розгалуженої екосистеми бібліотек .NET. Це робить її привабливою для масштабних комерційних проєктів, де пріоритетом є масштабованість систем та інтеграція зі стороннім програмним забезпеченням.

Проте при глибинному аналізі інтеграції C# у середовище Godot виявляється ряд системних ускладнень. По-перше, взаємодія між керованим кодом .NET та нативним ядром рушія вимагає створення додаткових маршалінг-прошарків. Кожен виклик функції рушія з боку C# супроводжується конвертацією типів даних, що при високій інтенсивності викликів (наприклад, у циклах оновлення фізики fixed_process може нівелювати переваги у продуктивності самої мови. По-друге, робота з C# потребує використання зовнішніх середовищ розробки IDE, таких як Visual Studio, що ускладнює загальну структуру проєкту та вимагає додаткових ресурсів для синхронізації метаданих між рушієм та кодом[6].

На відміну від універсальних рішень, GDScript був спроектований з урахуванням специфіки вузлової архітектури Godot. Це високорівнева мова з динамічною типізацією, хоча версія 4.x підтримує і статичну типізацію для оптимізації, синтаксис якої максимально наближений до Python. Головна перевага GDScript полягає у його «нативності»: інтерпретатор мови вбудований безпосередньо у ядро рушія. Це забезпечує безшовну інтеграцію з типами даних Godot такими як Vector2, Array, Dictionary без необхідності їх додаткової трансформації.

Важливим аспектом у контексті індивідуальної розробки є швидкість ітераційного циклу. GDScript підтримує миттєве оновлення коду без необхідності повної перекомпіляції проєкту. У C# будь-яка зміна у програмній логіці вимагає перезапуску збірки через MSBuild, що призводить до кумулятивних часових витрат. Для проєктів, що потребують частих експериментів з ігровим балансом та механіками (наприклад, налаштування параметрів процедурної генерації або алхімічних реакцій), такий ітеративний підхід у GDScript є значно ефективнішим.

Також слід врахувати поріг входження та читабельність коду. Лаконічний синтаксис GDScript дозволяє реалізовувати складну логіку меншою кількістю рядків коду у порівнянні з C#, де велика кількість синтаксичного «шуму» типу дужки, крапки з комою, оголошення просторів імен, може ускладнювати візуальний аналіз структури. В умовах розробки диплому, де акцент робиться на реалізації алгоритмічної частини, а не на побудові промислової архітектури підприємства, простота та надійність GDScript стають вирішальними факторами.

Підсумовуючи результати порівняльного аналізу, можна стверджувати, що вибір GDScript для даного проєкту є технічно обґрунтованим рішенням. Він забезпечує максимальну синергію з вбудованими інструментами Godot, мінімізує накладні витрати на взаємодію між підсистемами та дозволяє зосередити ресурси розробника на проектуванні ігрових механік, а не на обслуговуванні інфраструктури коду. Використання C# у даному випадку було б надлишковим, оскільки специфічні переваги цієї мови не перекривають переваг нативної інтеграції GDScript у межах обраного технологічного стеку.

1.4 Аналіз допоміжного програмного забезпечення

1.4.1 Створення графічний ресурсів Aseprite

Для реалізації візуальної складової ігрового проєкту було обрано стилістику pixel-art, яка є однією з найпоширеніших у сучасних інді-іграх. Даний стиль характеризується використанням обмеженої кількості пікселів для формування зображення, що дозволяє створювати чіткі, стилізовані та легко читабельні ігрові об’єкти. Однією з ключових причин вибору саме цього стилю є його універсальність, а також здатність зберігати візуальну якість незалежно від роздільної здатності екрана, що особливо важливо для невеликих 2D проєктів.

Основним інструментом для створення та редагування графічних ресурсів було обрано спеціалізований редактор Aseprite. Дане програмне забезпечення орієнтоване саме на роботу з піксельною графікою та надає широкий набір інструментів для створення як статичних зображень, так і анімацій. Його використання дозволило значно спростити процес розробки візуальних елементів гри та забезпечити єдиний стилістичний підхід до всіх ігрових об’єктів. Цьому сприяє також специфічний інтерфейс редактора, виконаний у піксельному стилі, що забезпечує швидкий доступ до палітри кольорів та інструментів малювання без перевантаження робочого простору (рис. 1.6).

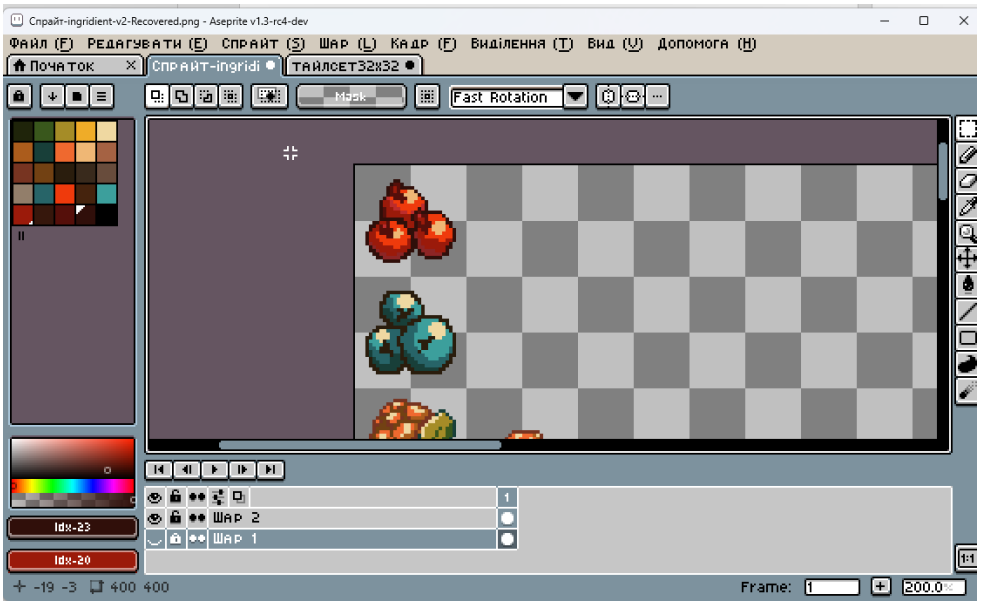


Рисунок 1.6 – Інтерфейс Aseprite

Однією з ключових переваг Aseprite є наявність зручної часової шкали timeline, яка використовується для створення покадрової анімації. Такий підхід дозволяє розробнику працювати одночасно з окремими кадрами та шарами, що значно полегшує процес створення анімацій руху персонажів, взаємодії з об’єктами та різноманітних візуальних ефектів. Завдяки цьому забезпечується більш природна та плавна поведінка ігрових елементів.

Також важливою особливістю Aseprite є інструменти точного піксельного малювання, які дозволяють уникати помилок, характерних для ручного

редагування зображень. Наприклад, функції автоматичного вирівнювання мазків та корекції ліній забезпечують чистоту контурів і запобігають появі зайвих пікселів. Це є особливо важливим у pixel-art стилі, де навіть незначні відхилення можуть впливати на загальне сприйняття об’єкта.

Ще однією перевагою є можливість автоматичного формування спрайт-листів, що дозволяє об’єднувати окремі кадри анімації в один файл. Такий підхід є зручним для подальшої інтеграції в Godot, оскільки зменшує кількість окремих текстур і оптимізує процес завантаження та обробки графіки. Це позитивно впливає на продуктивність гри та зменшує навантаження на систему.

Окрім створення власних графічних ресурсів, у процесі розробки також використовувалися готові асети, отримані з платформи itch.io [7] (рис. 1.7). Дана платформа є одним із найбільших ресурсів для незалежних розробників, де можна знайти велику кількість безкоштовних та платних графічних пакетів, що розповсюджуються за різними ліцензіями. Використання таких ресурсів дозволило значно пришвидшити процес розробки та зосередитися на реалізації унікальних елементів гри.



Рисунок 1.7 – Платформа itch.io

Застосування готових графічних пакетів дало можливість забезпечити базову стилістичну цілісність проєкту на ранніх етапах розробки. Зокрема, використовувалися набори тайлів та елементів оточення, які відповідають обраному піксельному стилю та мають узгоджену кольорову палітру. Це дозволило швидко сформувати ігрове середовище без необхідності створення кожного елемента з нуля.

Водночас завантажені ресурси часто піддавалися подальшій обробці та адаптації відповідно до потреб проєкту. Для цього вони імпортувалися до Aseprite, де проводилося редагування кольорів, зміна деталей, а також

створення додаткових варіацій об’єктів. Такий підхід дозволив поєднати переваги готових рішень із індивідуальними елементами дизайну, що позитивно вплинуло на унікальність візуального стилю гри.

Таким чином, використання Aseprite у поєднанні з ресурсами платформи itch.io дозволило сформувати ефективний робочий процес створення графіки. Це забезпечило баланс між швидкістю розробки, якістю візуальних елементів та можливістю їх подальшої адаптації під потреби проєкту. У результаті було досягнуто цілісного та узгодженого візуального стилю, який відповідає загальній концепції гри та вимогам до інді-проєкту.

1.5 Аналіз допоміжного програмного забезпечення

Процедурна генерація контенту, далі ПГ, є одним із ключових підходів сучасної розробки ігрових систем, що дозволяє автоматизувати створення ігрового середовища на основі математичних моделей та алгоритмів. У контексті двовимірних 2D ігор застосування ПГ особливо актуальне через необхідність формування великих, варіативних ігрових просторів при обмежених ресурсах розробки.

Основною перевагою ПГ є зменшення залежності від ручної роботи художників і дизайнерів на етапі побудови кожного окремого елемента сцени. Натомість створюється система правил і алгоритмів, яка самостійно формує контент під час виконання програми або на етапі попередньої генерації.

З точки зору системного аналізу, методи ПГ доцільно класифікувати за способом організації даних і логікою побудови ігрового простору. У цьому контексті виділяють спектр підходів — від жорстко структурованих моделей, де світ складається з визначених блоків, до повністю стохастичних систем, заснованих на шумових функціях. Така класифікація дозволяє оцінити компроміси між продуктивністю, керованістю та візуальною різноманітністю.

1.5.1 Створення графічний ресурсів Aseprite

У 2D іграх з орієнтацією типу side-scroller ключовим завданням є генерація безперервного ігрового простору вздовж однієї координатної осі – зазвичай осі X. Такий підхід передбачає динамічне формування світу в міру просування гравця, що дозволяє створювати практично нескінченні рівні.

У сучасній практиці розробки можна виділити кілька основних підходів до реалізації лінійної генерації:

1.5.1.1 Послідовна лінійна генерація

Послідовна лінійна генерація є одним із базових підходів до формування 2D-світу вздовж осі X, який широко застосовується в іграх типу side-scroller завдяки своїй простоті та високій продуктивності. Сутність методу полягає у покроковому обчисленні стану ігрового простору для кожного нового значення координати X, що з’являється у зоні видимості гравця.

У рамках цього підходу ігровий світ умовно поділяється на вертикальні елементи – «стовпці». Кожен стовпець являє собою набір значень уздовж осі Y, які визначають наявність або відсутність об’єктів, наприклад, платформи, перешкоди, декорації. Генерація відбувається в одному ітераційному циклі, де для кожного нового значення X виконується обчислення параметрів відповідного стовпця.

Алгоритмічно це може реалізовуватись як функція виду:

$$f(x) \to \{y_1, y_2, \dots y_n\},$$

де $f(x)$ визначає конфігурацію об’єктів у вертикальному перерізі світу.

Основною перевагою такого підходу є його лінійна обчислювальна складність O(n), що робить його ефективним для генерації великих або нескінченних рівнів у реальному часі. Крім того, він добре підходить для

використання з генераторами псевдовипадкових чисел, що забезпечує відтворюваність через seed або іншими словами «зерно».

Водночас, без додаткових механізмів варіативності наприклад, шумових функцій або шаблонів, цей метод може призводити до надмірної повторюваності структури рівня. Тому на практиці його часто комбінують з іншими підходами.

1.5.1.2 Генерація на основі сегментів

Метод генерації на основі сегментів передбачає використання попередньо створених логічних блоків рівня, які комбінуються у певній послідовності для формування ігрового простору. Кожен сегмент, або їх ще називають «чанками», є завершеним елементом дизайну, що містить визначену структуру об’єктів, геометрію рівня та, за необхідності, поведінку ігрових елементів.

Зазвичай такі сегменти зберігаються у вигляді шаблонів або префабів, що значно спрощує їх повторне використання. Генератор у цьому випадку виконує задачу вибору наступного сегмента на основі заданих правил або ймовірностей.

Формально процес можна описати як побудову послідовності:

,

де c_i – окремий сегмент світу.

Перевагою цього підходу є високий рівень контролю над якістю контенту. Оскільки кожен сегмент розробляється вручну, можна гарантувати відповідність ігровим вимогам, баланс складності та коректність розташування об’єктів.

Крім того, можлива реалізація складніших механік, таких як:

- обмеження повторюваності однакових сегментів;
- адаптація складності залежно від прогресу гравця;
- контекстно-залежний вибір сегментів.

Основним недоліком є обмежена варіативність. При невеликій кількості сегментів гравець може швидко помітити повторення, що знижує відчуття новизни. Також зростає обсяг підготовчої роботи на етапі дизайну.

1.5.1.3 Зональна сегментація з динамічним кроком

Зональна сегментація є більш складним і гнучким підходом до ПГ, що поєднує елементи детермінованості та випадковості. В її основі лежить поділ ігрового простору на послідовність зон (біомів), кожна з яких характеризується унікальними параметрами генерації.

Кожна зона визначається інтервалом:

$$Z_i = \{Z_{start}, Z_{end}\},$$

де довжина зони може змінюватися залежно від випадкових або заданих параметрів.

Для кожного типу зони задається набір характеристик:

– каталог об’єктів;
– ймовірності появи об’єктів;
– правила генерації геометрії;
– візуальні та стилістичні параметри.

Алгоритм генерації працює у два етапи:

– визначення послідовності зон та їх довжин.
– генерація вмісту всередині кожної зони відповідно до її правил.

Перевагою цього підходу є високий рівень керованості та контекстної узгодженості. Наприклад, у «лісовій» зоні генеруються лише відповідні об’єкти.

Додатковою перевагою є можливість реалізації ігрових сценаріїв:

– поступове ускладнення середовища;
– контроль рідкісних ресурсів;

– створення логічних переходів між зонами.

Недоліком є необхідність реалізації механізмів згладжування меж між зонами, оскільки різкі переходи можуть виглядати неприродньо. Для цього часто використовуються додаткові техніки, такі як інтерполяція параметрів або накладання шумових функцій.

1.5.2 Порівняльний аналіз шумових функцій та зональної сегментації

Після розгляду основних підходів до ПГ доцільно провести їх порівняльний аналіз з точки зору застосування у 2D іграх із лінійною структурою світу. Основна відмінність між шумовими функціями та зональною сегментацією полягає у способі організації простору: безперервному та дискретному відповідно.

Шумові функції забезпечують формування середовища на основі безперервної математичної залежності, що дозволяє досягти високого рівня візуальної плавності. Значення параметрів змінюються поступово, що створює ефект природності та органічності ландшафту. Проте така безперервність ускладнює реалізацію чітко визначених ігрових зон, оскільки відсутні явні межі між різними типами середовища.

На відміну від цього, зональна сегментація базується на дискретному поділі простору на окремі інтервали, кожен із яких має фіксовані характеристики. Це забезпечує високий рівень контролю над генерацією, дозволяє точно регулювати розподіл ресурсів і реалізовувати різні ігрові сценарії. Водночас дискретність призводить до появи різких переходів між зонами, що може потребувати додаткової обробки для покращення візуального сприйняття.

Ключовою відмінністю також є підхід до керування ігровим балансом. У випадку шумових функцій розподіл об’єктів має стохастичний характер, що ускладнює прогнозування та контроль. Зональна сегментація, навпаки, дозволяє задавати точні параметри генерації в межах кожної зони, що значно спрощує

балансування гри. Результати зіставлення обох методів за ключовими параметрами наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця шумових функцій та зональної сегментації

Критерій	Шумові функції	Зональна сегментація
Тип структури	Безперервна	Дискретна
Плавність переходів	Висока	Низька
Контроль генерації	Обмежений	Високий
Керування ресурсами	Складне	Передбачуване
Контроль біомів	Непрямий	Прямий
Візуальна природність	Висока	Середня
Гнучкість геймдизайну	Обмежена	Висока
Складність балансування	Висока	Низька
Обчислювальна складність	O(n)	O(n)

Порівняльний аналіз показує, що шумові функції та зональна сегментація орієнтовані на вирішення різних задач у процедурній генерації. Перші забезпечують природність і плавність середовища, тоді як другий підхід надає повний контроль над структурою ігрового простору.

З огляду на вимоги до керування ігровими ресурсами, логіки появи об’єктів та балансування ігрового процесу, більш доцільним є використання зональної сегментації. Вона дозволяє формувати передбачувану та керовану систему генерації, що є критично важливим для ігор із елементами збору ресурсів та крафту.

Разом із тим, оптимальним рішенням у практичних реалізаціях є комбінування обох підходів, де зональна сегментація визначає структуру світу, а шумові функції використовуються для деталізації та згладжування всередині зон.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Робота в Aseprite

На початковому етапі було визначено загальний художній стиль гри, який відповідає ізометричній 2D-проекції. Сформовано обмеженоу кольорову палітру, що дозволило досягти стилістичної єдності та покращити сприйняття об’єктів у межах ігрового середовища.

У процесі роботи було створено графічні спрайти інгредієнтів (рис. 2.1), які є ключовими елементами ігрової механіки. Кожен інгредієнт розроблявся як окремий об’єкт із унікальними формою, кольором та деталізацією, що забезпечує їхню швидку ідентифікацію гравцем під час збору та використання. При цьому враховувалась необхідність подальшого використання цих об’єктів у системі алхімії, що вплинуло на їх візуальне кодування.

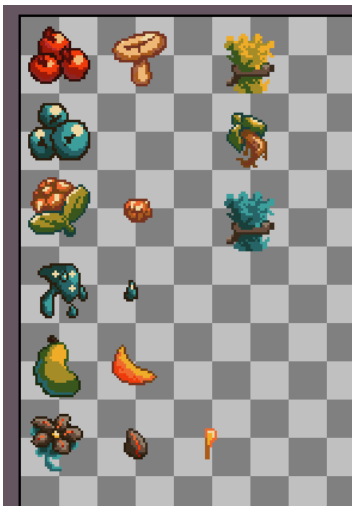


Рисунок 2.1 – Спрайт інгредієнтів

Наступним етапом стало створення тайлової графіки для формування ігрового світу. Було розроблено набір тайлів (рис. 2.2), які використовуються для побудови ландшафту та навколишнього середовища. При їх створенні враховувалась можливість безшовного поєднання, що є критично важливим для

процедурної генерації рівнів. Тайли тестувалися на сумісність між собою для уникнення візуальних артефактів при автоматичному генеруванні карти.

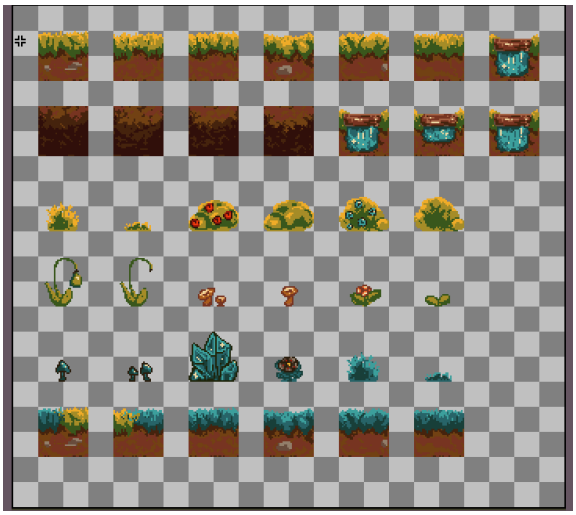


Рисунок 2.2 – Набір тайлів для побудови ігрової карти

Окрему увагу було приділено розробці графічного оформлення сцен гри (рис. 2.3). Зокрема, було створено візуальні елементи для сцени приготування зілля, яка є центральною з точки зору ігрової механіки. У даній сцені деталізовано такі об’єкти, як казан, робоча поверхня, інгредієнти та візуальні ефекти змішування. Це дозволяє підсилити занурення гравця в процес алхімії та зробити взаємодію більш інтуїтивною.



Рисунок 2.3 – Сцена приготування зілля та продажу

Під час створення графіки активно використовувалися можливості роботи з шарами, що дозволило розділити окремі компоненти спрайтів та

значно спростити процес редагування. Це також дало змогу швидко вносити зміни без необхідності повного перероблення зображення.

Важливим аспектом розробки була оптимізація графічних ресурсів. Усі спрайти створювалися з урахуванням обмежень ігрового рушія, таких як розмір текстур, кількість кадрів анімації та обсяг використовуваної пам’яті. Це забезпечило ефективну інтеграцію графіки в ігровий рушій без втрати продуктивності.

Таким чином, використання Aseprite дозволило створити повноцінний набір графічних ресурсів, які відповідають як художнім, так і технічним вимогам проєкту. Розроблені елементи забезпечують цілісність візуального стилю гри та ефективно підтримують реалізацію ігрових механік.

2.2 Проєктування сцен

У процесі розробки ігрового прототипу було сформовано узагальнену архітектуру сцен, яка відображає основні етапи ігрового процесу та логіку взаємодії гравця з системою.

У рушії Godot сцени виступають основними структурними одиницями, що дозволяють організувати проєкт у вигляді набору незалежних, але взаємопов’язаних компонентів. Кожна сцена відповідає за реалізацію окремої функціональної частини гри.

У межах даного проєкту було виділено такі основні логічні сцени:

– сцена збору ресурсів;
– сцена обробки інгредієнтів та алхімії;
– підсистема замовлень;
– підсистема торгівлі.

Незважаючи на те, що деякі з цих елементів реалізовані в межах однієї сцени рушія, з точки зору системного аналізу вони розглядаються як окремі функціональні підсистеми.

Ігровий процес організовано у вигляді циклічної моделі, де кожен етап логічно пов’язаний з попереднім. Гравець починає з дослідження світу, після чого переходить до обробки ресурсів, створення предметів і взаємодії з економічною системою гри. Загальна схема взаємодії сцен вказана в Додатку А.

Такий підхід дозволяє:

– структурувати ігрову логіку;
– розділити відповідальність між модулями;
– спростити подальше розширення проєкту.

2.2.1 Реалізація сцени процедурної генерації ігрового світу

У процесі розробки ігрового прототипу було реалізовано сцену процедурної генерації двовимірного ігрового світу, яка є ключовим елементом системи збору ресурсів та взаємодії гравця з навколишнім середовищем. Основним завданням даної сцени є динамічне формування ігрового простору з використанням стохастичних методів генерації, що дозволяє забезпечити варіативність ігрового процесу при збереженні керованості та балансування.

Реалізація виконана з використанням рушія Godot, у якому сцени виступають базовими структурними одиницями проєкту. У межах даної сцени використовується компонент TileMap для побудови базового шару поверхні (ґрунту), а також система інстанціювання об’єктів для створення інтерактивних елементів, таких як рослини, гриби та інші ресурси. Такий підхід дозволяє поєднати тайлову генерацію середовища з об’єктно-орієнтованою моделлю взаємодії. Схема створення світу вказана в Додатку Б.

ПГ реалізована за принципом лінійного формування світу вздовж горизонтальної осі X, що є типовим підходом для ігор із боковим виглядом side-scroller. Світ представлений у вигляді послідовності вертикальних колонок, кожна з яких генерується незалежно на основі поточного контексту. Така організація дозволяє значно спростити алгоритм генерації та забезпечити його високу продуктивність.

Ігровий простір не створюється повністю наперед, а формується зонами у межах заданої ширини світу. Кожна зона має випадкову довжину, що визначається в межах наперед заданого діапазону, і характеризується певним типом середовища – біомом. Це дозволяє уникнути одноманітності та створити відчуття природної змінності ландшафту.

Вибір біома здійснюється за допомогою генератора псевдовипадкових чисел із використанням порогових значень. Кожному типу біома відповідає певний інтервал значень, що фактично задає ймовірність його появи. Наприклад, базові біоми можуть мати вищу ймовірність генерації, тоді як рідкісні або спеціалізовані зони з'являються значно рідше. Такий підхід дозволяє досягти балансу між випадковістю та контрольованістю структури світу.

Після визначення біома для поточного сегмента виконується генерація відповідних колонок. Для кожного значення координати X формується вертикальна структура, яка включає базовий тайл поверхні та, за необхідності, декоративні або інтерактивні об'єкти.

Стохастичний характер генерації проявляється у використанні випадкових величин для визначення:

– типу об'єкта, що з'являється;
– факту появи об'єкта;
– варіацій його позиції;
– довжини зон.

Для кожної колонки обчислюється випадкове значення (spawn roll), яке порівнюється з наперед заданими порогами. Залежно від результату цього порівняння визначається, чи буде створено об'єкт і який саме. Така схема дозволяє реалізувати гнучку систему розподілу ресурсів у світі.

Кожен біом має власний набір допустимих об'єктів, що формує його унікальні характеристики (рис. 2.4). Наприклад, у межах лісового біома генеруються кущі з ягодами, гриби та декоративні елементи, тоді як у містичних зонах можуть з'являтися рідкісні або спеціалізовані ресурси.

Рівнинні ділянки, у свою чергу, містять базову рослинність і мають іншу щільність об’єктів.

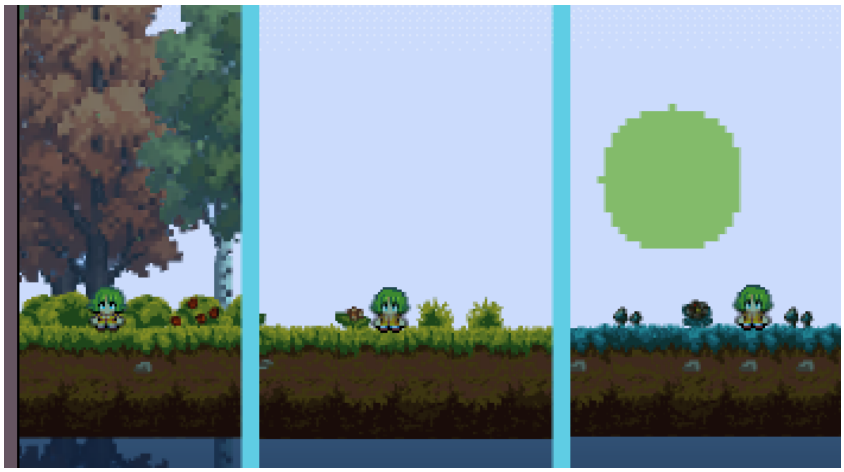


Рисунок 2.4 – Візуальні відмінності біомів

Такий підхід дозволяє реалізувати контекстну генерацію, при якій ігрове середовище залежить від типу зони. Це є важливим для ігрових механік, пов’язаних зі збором ресурсів, оскільки гравець отримує можливість цілеспрямовано шукати необхідні матеріали у відповідних біомах.

Для організації генерації використовується централізований каталог об’єктів, який містить посилання на сцени ресурсів. Це дозволяє відокремити логіку генерації від конкретної реалізації об’єктів і значно спростити розширення системи. Додавання нового типу ресурсу вимагає лише його включення до каталогу та налаштування відповідних умов появи.

Об’єкти, що генеруються, є інтерактивними та підтримують механіку збору. Для цього реалізовано систему відстеження активних елементів, у якій кожному об’єкту відповідає його позиція на мапі. При взаємодії гравця з об’єктом виконується перевірка його стану, після чого відбувається збір ресурсу та оновлення інвентаря

Важливою особливістю реалізації є поєднання стохастичної генерації з набором обмежень і правил. Повністю випадкова генерація може призводити до небажаних результатів, таких як нерівномірний розподіл ресурсів або

порушення ігрового балансу. Тому система включає ряд обмежень, які регулюють:

– максимальну частоту появи об’єктів;
– прив’язку ресурсів до конкретних біомів;
– мінімальні відстані між елементами;
– імовірність появи рідкісних ресурсів.

Це дозволяє зберегти контроль над ігровим процесом і забезпечити передбачуваність поведінки системи при збереженні ефекту випадковості.

З точки зору продуктивності, обрана модель генерації є ефективною, оскільки базується на простих ітераційних обчисленнях без необхідності складних математичних функцій. Кожна колонка генерується незалежно, що дозволяє легко масштабувати систему та адаптувати її до різних параметрів світу.

Додатково реалізовано допоміжні елементи середовища, такі як фонові об’єкти, вода та обмеження камери. Це створює цілісне ігрове середовище та покращує візуальне сприйняття сцени.

На етапі проєктування також було розглянуто альтернативний підхід до візуалізації – ізометричне представлення світу (рис. 2.5). Ізометрія дозволяє створити більш глибоку та деталізовану сцену, однак значно ускладнює процес розробки. Зокрема, виникає необхідність у створенні більшої кількості графічних ресурсів, ускладнюється система позиціонування об’єктів, а також зростає складність реалізації процедурної генерації.

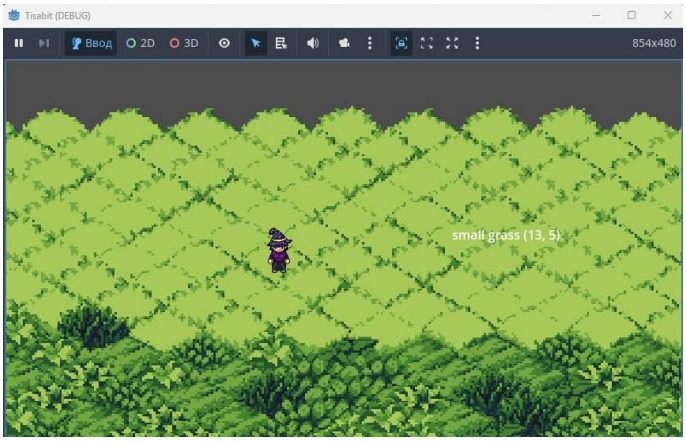


Рисунок 2.5 – Альтернативний ізометричний варіант

Натомість двовимірний вигляд (рис. 2.6) збоку забезпечує значно простішу інтеграцію з обраною моделлю генерації. Він дозволяє використовувати стандартні інструменти рушія, спрощує створення спрайтів та зменшує загальні витрати часу на розробку. Крім того, такий підхід є більш прозорим з точки зору реалізації алгоритмів.



Рисунок 2.6 – Обраний варіант візуалізації

У результаті було обрано саме двовимірне представлення світу, що дозволило зосередитися на реалізації ігрових механік і логіки генерації без надмірного ускладнення графічної частини.

Отже, реалізована сцена процедурної стохастичної генерації забезпечує створення динамічного, варіативного та керованого ігрового середовища. Поєднання випадковості з набором правил дозволяє досягти балансу між непередбачуваністю та стабільністю, що є важливим для ігор із елементами дослідження та збору ресурсів. Запропонований підхід є гнучким та масштабованим, що дозволяє розширювати систему шляхом додавання нових біомів, ресурсів і механік без суттєвої зміни її архітектури.

2.2.2 Система обробки інгредієнтів та створення зілля

У процесі подальшої реалізації ігрового прототипу було приділено значну увагу формуванню цілісної архітектури сцени обробки інгредієнтів та створення зілля, оскільки дана частина ігрової системи виступає ключовим елементом, який забезпечує безпосередній перехід від накопичення ресурсів до їх трансформації у функціональні ігрові об’єкти. Важливість цієї сцени полягає у тому, що вона фактично об’єднує декілька різних підсистем гри в єдиний логічний процес, який включає збір, обробку, модифікацію та комбінування інгредієнтів. З точки зору проєктування ігрових систем, подібна інтеграція є одним із найскладніших завдань, оскільки вимагає узгодженої роботи підсистем інвентаризації, фізичної взаємодії, параметричного моделювання та візуального зворотного зв’язку.

У межах загальної концепції реалізації було прийнято рішення організувати сцену у вигляді умовної алхімічної лабораторії, що дозволяє створити інтуїтивно зрозуміле середовище для взаємодії гравця з ігровими механіками. Подібний підхід також сприяє підвищенню рівня занурення у ігровий процес, оскільки візуальне оформлення безпосередньо підтримує логіку дій, які виконує користувач. Концептуально алхімічна лабораторія є добре відомим архетипом у жанрі RPG та пригодницьких ігор, що забезпечує гравцю мінімальний поріг входження та дозволяє зосередитися на освоєнні унікальних механік, а не на розумінні базового контексту.

Головна сцена (рис. 2.7) демонструє загальну компоновку сцени, включаючи розташування основних інтерактивних елементів, робочої поверхні, а також допоміжних зон, які використовуються для взаємодії з інгредієнтами. Візуальна структура сцени була побудована таким чином, щоб забезпечити максимальну читабельність ігрового простору та уникнути перевантаження користувача надмірною кількістю елементів. При проєктуванні інтерфейсу дотримувався принцип мінімалістичного UI: елементи керування відображаються лише тоді, коли вони є контекстуально релевантними, що зменшує когнітивне навантаження на гравця та підвищує загальну зручність використання.

Слід зазначити, що сцена умовно поділена на декілька функціональних зон, кожна з яких виконує окрему роль у загальному процесі створення зілля. Зокрема, можна виділити зону зберігання інгредієнтів, яка відповідає за доступ до наявних ресурсів, зону первинної обробки, у межах якої здійснюється зміна фізичних та параметричних властивостей інгредієнтів, а також основну робочу зону, де відбувається їх комбінування та формування кінцевого продукту. Кожна із зазначених зон має власний набір інтерактивних елементів та правил взаємодії, що робить процес гри структурованим та передбачуваним з точки зору користувацького досвіду. Перехід між зонами реалізований як плавний та природний – гравець може переміщати інгредієнти між зонами, що є стандартом для подібного типу ігрових сцен.



Рисунок 2.7– Загальний вигляд сцени

Отримано логічну структуру взаємодії між різними частинами сцени та послідовність переходу інгредієнтів від одного стану до іншого (рис. 2.8). З точки зору програмної реалізації, кожна зона представлена окремим контролером, який відповідає за управління об’єктами у межах своєї відповідальності та взаємодіє з іншими контролерами через єдину систему подій.

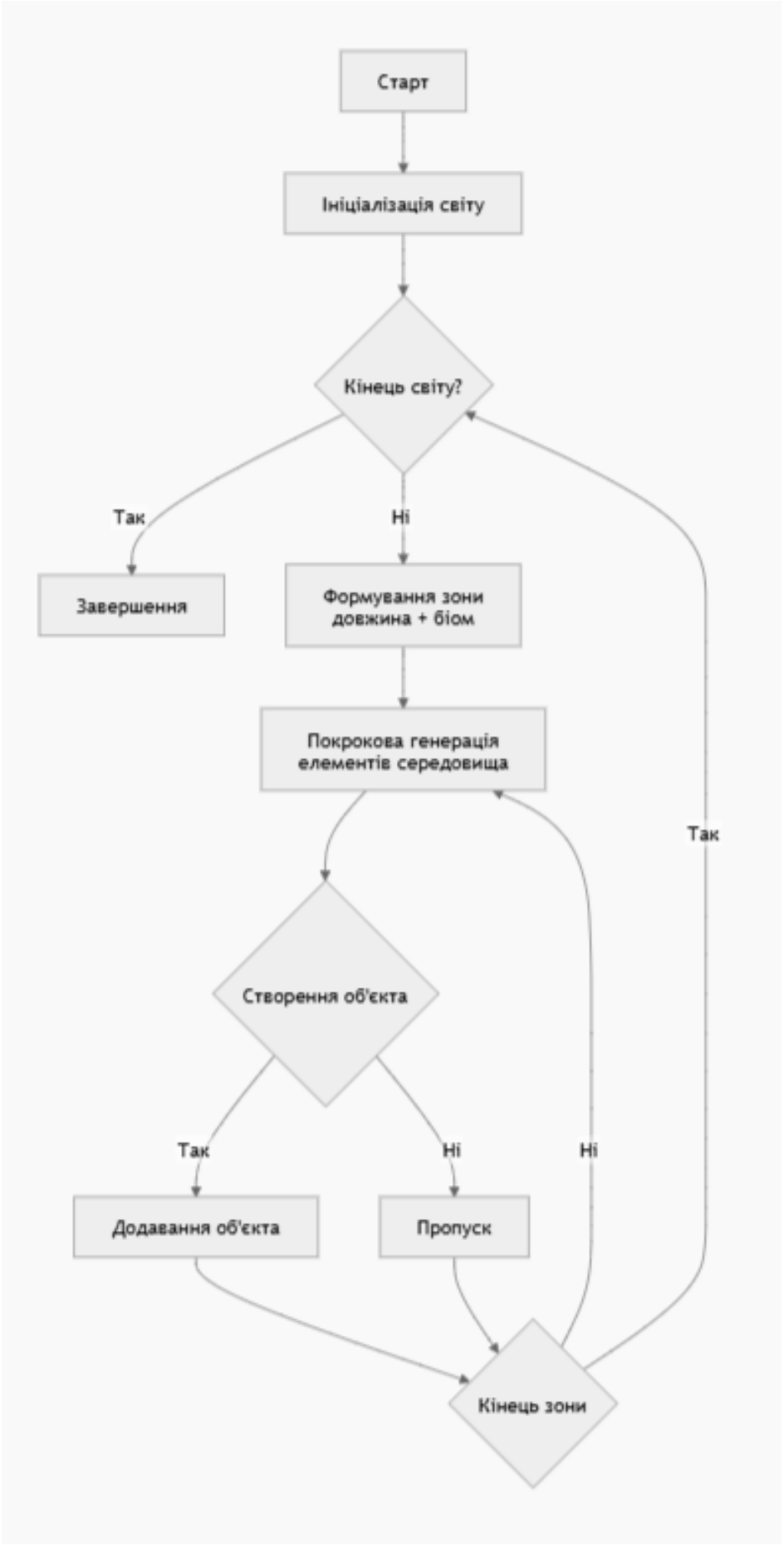


Рисунок 2.8 – Діаграма створення зілля

Особливу роль у реалізації сцени відіграє робоча поверхня (рис. 2.9), яка є центральним елементом взаємодії гравця з інгредієнтами. На відміну від класичних інвентарних систем, де об’єкти існують у вигляді абстрактних структур даних, у даному випадку використовується просторове представлення кожного інгредієнта. Це означає, що кожен об’єкт має власну фізичну позицію у межах сцени та може взаємодіяти з іншими об’єктами відповідно до заданих правил системи. Такий підхід наближає ігрову механіку до реалістичної симуляції та посилює відчуття тактильності й фізичності взаємодії, що є важливим фактором залученості гравця.



Рисунок 2.9 – Робочий стіл

Важливо підкреслити, що робоча поверхня має обмежені розміри, що накладає додаткові умови на процес взаємодії. У випадку перевищення допустимого навантаження система переходить у стан часткового перевантаження, при якому об’єкти можуть втрачати стабільність розташування, зміщуватися або взаємодіяти між собою неконтрольованим чином. Такий механізм виконує не лише технічну, але й геймплейну функцію, оскільки стимулює гравця до більш раціонального використання простору. З технічної точки зору, контроль навантаження реалізований через лічильник активних об’єктів на поверхні: при перевищенні заданого порогу активується спеціальний стан сцени, що змінює поведінкові параметри фізичного рушія для об’єктів на поверхні.

Додатково система інгредієнтів побудована на параметричному підході, що дозволяє описувати кожен об’єкт через набір числових та логічних характеристик. До таких характеристик належать базові елементи, які відображають фундаментальну природу інгредієнта, а також модифікатори, що визначають його поведінкові властивості у процесі обробки та комбінування. Параметричний підхід обрано з метою забезпечення максимальної гнучкості системи: замість жорстко закодованих результатів взаємодії, кінцевий результат обчислюється динамічно на основі поточних значень параметрів, що дозволяє легко розширювати контент без внесення змін до програмного коду.

Базові елементи включають умовні категорії (рис. 2.10), такі як вода, вогонь, земля, повітря, життя та смерть, при цьому кожен інгредієнт має власне унікальне співвідношення цих складових. Саме ця структура дозволяє формувати варіативні результати навіть при використанні однакових типів ресурсів, але з різними характеристиками. Наприклад, два інгредієнти категорії «вода» з різним вмістом елемента «життя» можуть давати принципово різні результати при комбінуванні – один підсилюватиме лікувальні властивості зілля, інший може нейтралізувати їх. Така варіативність забезпечує глибину ігрового процесу та стимулює гравця до експериментування.



Рисунок 2.10 – Приклад категорій та модифікаторів інгредієнта

Модифікатори, у свою чергу, відповідають за додаткові властивості інгредієнта, такі як стабільність, чистота, тривалість ефекту та інші параметри, що впливають на якість кінцевого зілля. Важливо відзначити, що ці характеристики можуть змінюватися у процесі обробки, що створює динамічну систему трансформацій. Кожен інгредієнт має свою параметри(рис. 2.11). Зокрема, параметр реактивності може знижуватися при надмірному подрібненні або підвищуватися при застосуванні певних фільтрувальних інструментів. Такі залежності формують систему компромісів, де гравець змушений приймати свідомі рішення щодо способу обробки кожного конкретного інгредієнта.



Рисунок 2.11 – Приклад параметрів інгредієнтів

Особливістю системи є також можливість багаторівневого розкладу інгредієнтів, що дозволяє отримувати додаткові компоненти з одного вихідного об’єкта. Наприклад, квітка може бути розділена на пелюстки, тичинки та інші складові частини, кожна з яких має власні параметри та може бути використана окремо у процесі створення зілля. Такий підхід значно розширює варіативність системи та дозволяє створювати складні ланцюжки взаємодій між

інгредієнтами. З точки зору реалізації, кожен інгредієнт зберігає у своїй структурі даних список можливих компонентів розкладу разом із відповідними інструментами, необхідними для їх отримання. Це дозволяє системі динамічно визначати доступні дії залежно від поточного стану об’єкта та наявних у гравця інструментів.

Процес обробки реалізований через використання різних інструментів, кожен з яких має специфічний вплив на стан інгредієнта. У процесі використання ступки (рис. 2.12) відбувається активна зміна внутрішніх параметрів об’єкта, що виражається у підвищенні його реактивності. Під реактивністю розуміється здатність інгредієнта до більш швидкої та інтенсивної взаємодії з іншими компонентами системи, а також його схильність до більш виражених змін під час подальшого використання у процесі створення зілля. Таким чином, ступка фактично виконує функцію активації інгредієнта, підсилюючи його потенціал та збільшуючи силу впливу на кінцевий результат. Водночас підвищення реактивності може супроводжуватися зниженням стабільності, що формує баланс між потужністю ефекту та рівнем його контрольованості.



Рисунок 2.12 – Ступка

На відміну від ступки, пічка (рис. 2.13) виконує протилежну функцію та спрямована на зниження інтенсивності властивостей інгредієнтів. У процесі обробки в пічці відбувається поступова стабілізація структури об’єкта, що призводить до зменшення вираженості його ефектів. Іншими словами, пічка знижує загальну силу інгредієнта, роблячи його властивості більш м’якими,

стабільними та передбачуваними. Такий підхід дозволяє контролювати надмірно сильні або нестабільні інгредієнти та приводити їх до більш збалансованого стану, який є зручнішим для подальшого використання у системі крафту.



Рисунок 2.13 – Пічка

Особливу увагу було приділену котлу (рис. 2.14). Котел реалізований як інтерактивний об’єкт, у який гравець може додавати інгредієнти у довільній кількості, що створює додаткову варіативність процесу створення зілля. Водночас система має чітко визначене обмеження, яке полягає у неможливості перевищення сумарного значення певних параметрів понад 100%. Кожен інгредієнт вносить свій вклад у загальну структуру зілля через набір базових характеристик, і якщо сумарне значення будь-якого з параметрів перевищує встановлений поріг, процес створення зілля вважається нестабільним і кінцевий результат не формується. Такий механізм вводить елемент балансу та змушує гравця уважно підбирати комбінації інгредієнтів, враховуючи їх вплив на загальну систему параметрів.

Окремою особливістю даної механіки є залежність візуального результату зілля від складу використаних інгредієнтів. Зокрема, колір готового зілля формується на основі сумарного впливу всіх компонентів, що були додані до котла. Кожен інгредієнт має власний набір характеристик, які умовно

впливають на колірну складову кінцевого продукту, і в результаті їх поєднання утворюється унікальний візуальний відтінок.



Рисунок 2.14 – Котел та зілля

Таким чином, колір зілля виступає не лише декоративним елементом, але й непрямим індикатором складу та пропорцій використаних інгредієнтів. Це дозволяє гравцю візуально оцінювати результат своїх дій та робити висновки щодо ефективності обраної комбінації ресурсів.

2.2.3 Система обробки замовлень

Система обробки замовлень є одним із ключових елементів ігрового процесу, який забезпечує практичне застосування механіки створення зілля та формує логічний зв’язок між ігровими діями гравця і вимогами умовних замовників. Вона виконує функцію постановки завдань, визначаючи конкретні параметри кінцевого продукту, які необхідно досягти в процесі крафту, а також задає загальну структуру взаємодії між гравцем та ігровою системою.

У межах інтерфейсу (рис. 2.15) замовлення відображається опис необхідного зілля, який включає набір числових та умовних параметрів. Дані параметри формують вимоги до кінцевого продукту і можуть містити як

мінімальні значення певних ефектів, так і максимальні обмеження на інші характеристики. Наприклад, може бути задано вимогу досягти ефекту повітря на рівні не нижче 50 одиниць, при цьому одночасно встановлюється обмеження на реактивність, яка не повинна перевищувати значення –30. Така система параметрів змушує гравця враховувати взаємодію різних властивостей інгредієнтів та їх вплив один на одного.



Рисунок 2.15 – Віконце системи замовлень

Подібний підхід формує багатошарову систему обмежень, у якій результат залежить не лише від вибору інгредієнтів, але й від їх комбінації, обробки та внутрішнього балансу характеристик. Це значно підвищує складність прийняття рішень і робить процес створення зілля більш аналітичним, оскільки гравець повинен прогнозувати кінцевий результат ще до завершення крафту.

Окрему роль у системі відіграє візуальний індикатор у вигляді руки (рис. 2.16), яка з’являється у вікні замовлення та символізує очікування клієнта. Даний елемент виконує функцію не лише декоративного відображення, але й важливого інструмента зворотного зв’язку, який інформує гравця про активний стан замовлення та необхідність його виконання. Таким чином формується додатковий рівень взаємодії, де візуальні сигнали замінюють текстові підказки.



Рисунок 2.16 – Індикатор очікування клієнта

На виконання замовлення відводиться обмежений проміжок часу, який варіюється в межах від однієї до трьох хвилин. Такий часовий діапазон обрано з метою забезпечення балансу між можливістю експериментування та необхідністю швидкого прийняття рішень у процесі взаємодії з ігровими механіками. У результаті гравець змушений одночасно контролювати процес створення зілля, стежити за станом інгредієнтів та враховувати часові обмеження, що формує додатковий рівень напруги та динаміки в ігровому процесі. Подібний підхід дозволяє уникнути надмірно повільного темпу гри та підтримує постійну залученість гравця до виконання завдань.

У процесі наближення до завершення відведеного часу індикатор у вигляді руки змінює свою поведінку, переходячи у стан тремтіння. Даний візуальний ефект є індикатором критичного рівня очікування і фактично сигналізує про те, що залишився мінімальний проміжок часу для завершення завдання. Використання такого підходу дозволяє передавати інформацію про стан системи без застосування текстових повідомлень, що робить інтерфейс більш інтуїтивним та зменшує когнітивне навантаження на гравця під час активної взаємодії.

Якщо гравець не встигає передати готове зілля у встановлений час, система фіксує провал замовлення. У такому випадку клієнт залишає ігрову сцену, а поточне завдання вважається невиконаним. Після цього гравцю необхідно очікувати появи нового замовлення та повторювати цикл створення зілля з урахуванням нових вимог та умов. Подібний механізм формує безперервний ігровий цикл, у якому кожна помилка або затримка безпосередньо впливає на загальну ефективність проходження та темп розвитку ігрового процесу

Додатково варто зазначити, що система замовлень виконує не лише функцію постановки завдань, але й формує загальну структуру ігрового прогресу та взаємодії. Вона визначає напрямок використання доступних ресурсів, задає критерії успішності дій гравця та забезпечує постійне оновлення ігрових викликів, що поступово ускладнюються або змінюються залежно від умов. Завдяки цьому кожне нове замовлення сприймається як окремий сценарій взаємодії, який потребує аналізу, планування та адаптації до змінних умов ігрового середовища.

У результаті система обробки замовлень інтегрується у загальну архітектуру гри як динамічний механізм, що поєднує часові обмеження, параметричні вимоги та логіку прийняття рішень. Це забезпечує безперервний ігровий цикл, підтримує мотивацію гравця до взаємодії з системою та сприяє формуванню більш глибокого і варіативного ігрового досвіду.

2.2.4 Збереження даних у форматі JSON

Окремим важливим елементом ігрового прототипу є система збереження даних, яка забезпечує можливість фіксації стану ігрового процесу та його подальшого відновлення. Реалізація цієї системи є необхідною умовою для коректного функціонування гри, оскільки дозволяє зберігати прогрес гравця між ігровими сесіями, а також забезпечує стабільність роботи всіх пов’язаних

механік, включаючи систему крафту, обробку інгредієнтів та виконання замовлень.

У якості основного формату збереження даних використовується JSON, який є легким текстовим форматом для структурованого представлення інформації. Вибір даного формату обумовлений його простотою, зручністю для серіалізації та десеріалізації даних, а також широкою підтримкою у середовищі розробки Godot. Завдяки цьому забезпечується швидка обробка даних без значного навантаження на систему та можливість легкої модифікації структури збережень у процесі розробки. Структура даних вказана діаграмою в Додатку В.

Логіка збереження побудована на принципі серіалізації ключових ігрових параметрів у єдину структуровану модель даних. Такий підхід дозволяє об’єднати всі основні елементи ігрового стану в одному логічному контейнері, який надалі може бути легко записаний у файл та відновлений без втрати інформації. До таких параметрів належать стан інвентарю гравця, наявні інгредієнти та їх характеристики, прогрес у виконанні замовлень, а також поточний стан ігрової сцени. Кожен із цих елементів представлений у вигляді окремого об’єкта всередині JSON-структури, що забезпечує чітку ієрархічну організацію даних та спрощує процес їх подальшого відновлення

Подібна структура збереження дозволяє уникнути змішування різних типів даних, оскільки кожен блок відповідає за окрему підсистему гри. Це особливо важливо для складних ігрових механік, де взаємодіють декілька незалежних систем, таких як крафт, система замовлень та взаємодія зі світом. Завдяки цьому забезпечується логічна цілісність даних і зменшується ризик втрати або некоректного відновлення стану гри.

Особливу увагу при проєктуванні системи збереження приділено параметричним характеристикам інгредієнтів, оскільки саме вони є основним елементом системи крафту. Кожен інгредієнт зберігається не лише як окремий об’єкт, але й разом зі своїми базовими елементами та модифікаторами, що повністю описують його властивості. Це дозволяє відновити його стан з

максимальною точністю після завантаження гри. Такий підхід є важливим, оскільки навіть незначні зміни у параметрах можуть суттєво впливати на результат комбінування інгредієнтів та створення зілля.

Додатково зберігається інформація про активні замовлення, включаючи їх поточні параметри, залишковий час виконання та стан прогресу. Це забезпечує безперервність ігрового процесу, навіть у випадку завершення сесії. Завдяки цьому гравець може продовжити виконання завдань з того самого етапу, на якому було здійснено збереження, без втрати логіки або прогресу. Такий механізм особливо важливий для систем, які базуються на часових обмеженнях та динамічній взаємодії з ігровим середовищем.

У середовищі Godot реалізація збереження здійснюється за допомогою вбудованих засобів роботи з файлами та JSON-об’єктами. На першому етапі всі ігрові дані структуруються у вигляді словника, що дозволяє організувати їх у зручну ієрархічну форму. Після цього структура перетворюється у JSON-рядок, який записується у файл збереження. При завантаженні гри процес виконується у зворотному порядку: файл читається, JSON-рядок парситься, після чого дані відновлюються у відповідні ігрові об’єкти та системи.

Важливою перевагою даної системи є її масштабованість та гнучкість. Використання JSON-структури дозволяє легко розширювати формат збереження шляхом додавання нових полів без необхідності суттєво змінювати існуючу архітектуру. Це забезпечує можливість подальшого розвитку проєкту, зокрема інтеграції нових типів інгредієнтів, розширення системи характеристик, додавання нових ігрових механік або ускладнення логіки взаємодії між системами. Таким чином, система збереження не обмежує розвиток гри, а навпаки створює основу для її подальшого масштабування.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІГРОВИХ МЕХАНІК

3.1 Інструменти та загальна характеристика середовища розробки

Для реалізації проєкту було обрано ігровий рушій Godot Engine версії 4.x. Вибір даного рушія зумовлений його високою ефективністю при роботі з двовимірною графікою та зручною системою візуального проектування інтерфейсів користувача. Оскільки гра базується на лінійному переміщенні персонажа та активній взаємодії з різноманітними меню (магазин, інвентар, вікно замовлень), вузлова система Godot дозволила швидко інтегрувати графічні елементи та логіку їх динамічного оновлення без втрати продуктивності.

Процес розробки базувався на використанні вбудованих засобів рушія, що забезпечило безшовну інтеграцію візуальних асетів у програмне середовище. Використання об’єктно-орієнтованого підходу дозволило розділити гру на автономні сцени: персонажа, ігрове середовище та складні інтерфейсні панелі. Це значно полегшило процес налаштування торгівлі та системи замовлень, оскільки кожен елемент інтерфейсу працює як незалежний модуль, що обмінюється даними через систему сигналів. Такий підхід забезпечує масштабованість проєкту та легкість у виправленні помилок на пізніх етапах розробки. Детальний перелік та призначення інструментальних засобів, використаних у ході розробки, систематизовано в таблиці 3.1.

Таблиця 3.1 – Програмне забезпечення, використане при розробці

Категорія	Назва інструменту	Функціональне призначення
Ігровий рушій	Godot Engine 4	Створення сцен, обробка фізики та логіки інтерфейсу.
Графічний редактор	Aseprite	Розробка персонажів, іконок товарів та елементів оточення.
Контроль версій	Git	Відстеження змін.

Окрему увагу було приділено мові програмування GDScript, яка є рідною для Godot. У поєднанні з гнучким інструментарієм для створення UI, це дало змогу реалізувати вікно магазину з картками товарів, яке працює стабільно.

3.2 Графічне оформлення та візуальна складова

Візуальний стиль гри виконано в техніці Pixel Art. Такий вибір дозволив створити чіткий та зрозумілий інтерфейс, де кожен ігровий предмет має унікальну впізнавану іконку. Кожен графічний елемент було адаптовано під єдину палітру кольорів для збереження візуальної цілісності проєкту.

3.2.1 Розробка ігрових спрайтів та персонажа

Головний персонаж гри спроєктований спеціально для лінійного переміщення по горизонтальній площині. Було створено набори кадрів анімації для руху вправо та вліво, що активуються при натисканні відповідних клавіш керування дій. Використання горизонтальної орієнтації дозволило зосередити увагу гравця на самому процесі збору ресурсів та моніторингу навколишнього середовища.

Особлива увага приділялася деталізації іконок у магазині та вікні інвентарю. Для кожного ресурсу розроблено окремі спрайти з високим рівнем контрастності. Це технічне рішення було необхідним для того, щоб у насиченому вікні магазину гравець міг миттєво відрізнити один товар від іншого, орієнтуючись на візуальний силует та колір предмета.

3.2.2 Система станів персонажа та інгредієнтів

Ігровий простір у проєкті реалізований як відкрита горизонтальна локація.

Для побудови якісного 2D-середовища було використано спеціалізовані класи Godot, що дозволяють ефективно працювати з шарами графіки та фізикою. Специфікацію базових вузлів, які формують програмний фундамент і візуальну складову гри, наведено в таблиці 3.2.

Таблиця 3.2 – Основні класи Godot, використані в архітектурі гри

Клас	Призначення в проєкті	Ключові особливості
CharacterBody2D	Головний персонаж	Використовується для реалізації руху та обробки колізій з оточенням.
AnimatedSprite2D	Візуалізація персонажа	Керує перемиканням кадрів анімації.
TileMapLayer	Основа ігрового світу	Формує поверхню локації за допомогою набору тайлів.
ParallaxBackground	Задній план	Створює ефект глибини через різну швидкість руху фонових шарів.
Area2D	Зони взаємодії	Реєструє вхід гравця в зону збору ресурсу або входу в магазин.
Camera2D	Огляд ігрового поля	Слідує за персонажем, забезпечуючи плавне панорамування.

Використання TileMapLayer дозволило створити гнучуку параметричку лінійну зону пересування. Замість малювання одного великого зображення фону, світ складається з маленьких фрагментів, що значно економить ресурси на створення рівнів. Ефект ParallaxBackground додає об’єму статичному 2D-зображенню: далекі гори та хмари рухаються повільніше за передній план, що створює ілюзію простору в лінійному світі.

Персонаж на базі CharacterBody2D дозволяє реалізувати плавне керування з урахуванням прискорення. Завдяки вбудованим функціям обробки зіткнень, герой не провалюється крізь поверхню TileMapLayer.

3.3 Реалізація ігрового процесу та взаємодій

Ігровий процес розробленого проєкту базується на циклічній системі «збір-обробка-реалізація». Кожна механіка спроєктована таким чином, щоб стимулювати гравця до вивчення рецептур та оптимізації своїх дій. Основна логіка взаємодій винесена в окремі скрипти, що дозволяє легко модифікувати параметри гри без ризику порушити цілісність всього проєкту.

3.3.1 Система станів персонажа та інгредієнтів

Для підвищення глибини геймплею було впроваднено систему станів. Кожен інгредієнт під час обробки може перебувати в одному з логічних станів, що впливає на його подальше використання. Це реалізовано через спеціальний клас ItemData, який містить змінну стану та методи для її перевірки перед використанням у крафтових станціях.

Основний ігровий виклик сконцентрований не на швидкості дій, а на інтелектуальному забезпеченні точної відповідності параметрів продукту вимогах клієнта. Перелік можливих станів ігрових ресурсів та їхнє функціональне значення в межах ігрової логіки представлено в таблиці 3.3.

Таблиця 3.3 – Класифікація станів ігрових об’єктів

Стан	Характеристика	Вплив на ігровий процес
Сирий	Початковий стан ресурсу.	Має базові характеристики, обробка є опціональною.
Оброблений	Проміжний етап.	Дозволяє використовувати ресурс у складних рецептах.
Активний	Процес трансформації в котлі.	Тимчасовий стан під час виконання крафтового скрипту.
Готовий	Кінцевий продукт.	Може бути зданий замовнику або виставлений

		напродаж.

3.3.2 Модифікатори предметів та система вимог

Модифікатори у грі виступають як набір числових параметрів, які гравець повинен привести у відповідність до замовлення. Кожен предмет має 6 базових характеристик та три модифікатори – Реактивність, Стабільність та Смак. Обробка змінює значення характеристик та модифікаторів. Взаємозв'язок між типами обробки інгредієнтів та динамікою зміни їхніх показників систематизовано в таблиці 3.4.

Таблиця 3.4 – Логіка зміни параметрів через інструменти обробки

Інструмент	Процес	Вплив на характеристик	Вплив на модифікатор
Ступка	Подрібнення	зростання значень	Реактивність, Смак
Пічка	Прожарка	зниження значень	Стабільність,Смак

Основний виклик полягає у дотриманні верхньої межі. Якщо клієнт вимагає зілля з реактивністю менше 30, а після ступки вона становить 35, замовлення не буде прийняте. У такому разі гравець має використати інші інгридієнти в пічці, щоб «збити» зайву реактивність, розуміючи, що разом із нею пропорційно впаде показник інших модифікаторів. Це створює систему цікавого регулювання показників.

3.3.3 Модифікатори предметів та система вимог

Механіка обробки базується на зміні цифрових станів інгредієнта. Ступка та пічка діють як інструменти прямої та побічої дії. Обидва інструменти реалізовані як однофазні процеси: гравець запускає цикл обробки один раз, отримуючи миттєвий результат трансформації.

Важливою особливістю логіки є одноразовість процесу: обробляти конкретний інгредієнт можна лише один раз за ігровий цикл. Це означає, що гравець має заздалегідь вирішити – чи потрібно йому підняти якість у ступці ризикуючи вийти за ліміт реактивності, але підвищивши ціну через смак, чи стабілізувати предмет у пічці. Після переходу в стан Processed предмет фіксує свої показники до моменту змішування в котлі.

Пічка працює за принципом одноразового циклу, знижуючи всі параметри предмета на сталий коефіцієнт. Ця функція є ключовою для замовлень із жорстким лімітом реактивності, коли після обробки в ступці показники стають занадто високими. Оскільки повторне прожарювання неможливе, гравець має одразу зробити стратегічно правильний вибір між підвищенням якості та стабілізацією ресурс

Будь-яка помилка на етапі вибору інструмента призводить до того, що інгредієнт перестане відповідати умовам замовлення. У такому разі гравцеві доведеться шукати новий ресурс, що додає грі стратегічної глибини та відповідальності за кожну дію.

3.3.4 Проєктування системи замовлень та торгівлі

Система замовлень у грі базується на генерації контрактів з процедурними діапазонами вимог. Основна особливість цієї системи полягає в комплексному технічному контролі, який охоплює всі дев’ять параметрів предмета одночасно. Кожне нове замовлення створює нові умови гри: для одного контракту критичним може бути показник життя на рівні 50+, але зі строгим обмеженням смаку до 20-, тоді як наступний замовник може вимагати максимальну стабільність при низькій реактивності.

Алгоритм валідації замовлення виконується послідовно. Спочатку система аналізує тип наданого предмета на відповідність замовленню. Далі відбувається поглиблена перевірка всіх 6 основних модифікаторів – якщо хоча б один показник виходить за межі індивідуально визначеного діапазону [min;

max], клієнт відмовляється від угоди. Наступним етапом є технічний контроль допоміжних показників: Реактивності, Смаку та Стабільності.

Наприклад, якщо Реактивність перевищує допустимий ліміт, замовлення вважається неякісним. Фінальний етап – розрахунок вартості, де показник смаку виступає основним множником прибутку, але лише за умови повної відповідності всім попереднім технічним критеріям. Така структура змушує гравця постійно балансувати між бажанням заробити більше та необхідністю вписатися в жорсткі рамки контракту.

3.4 Організація збереження ігрового прогресу

Для забезпечення збереження стану ігрового світу та прогресу користувача було реалізовано систему експорту даних у формат JSON. Вибір цього формату зумовлений його легкістю та вбудованою підтримкою в Godot Engine через формат JSON. Система фіксує не лише координати гравця, а й повний стан інвентарю, активні замовлення та параметри об’єктів у світі.

Збереження даних у форматі JSON:

```
const SAVE_PATH := "user://save.json"
func save_game(player, inventory, orders, world):
	var save_data = {
		"player": {
			"position": [player.global_position.x, player.global_position.y],
			"progress": player.progress
		},
		"inventory": inventory_to_dict(inventory),
		"orders": orders_to_dict(orders),
		"world": {
			"scene": world.current_scene_name,
			"objects": world.get_objects_state()
```

```
            }
        }
        var json_string = JSON.stringify(save_data, "\t")
        var file = FileAccess.open(SAVE_PATH, FileAccess.WRITE)
        if file:
            file.store_string(json_string)
            file.close()
```

Функція save_game формує словник (Dictionary), який агрегує дані з різних систем гри. Використання JSON.stringify з параметром відступу робить файл придатним для читання при налагодженні. Файл зберігається в захищеній директорії user://, що є стандартом для збережень у Godot.

Завантаження даних з JSON:

```
func load_game():
    if not FileAccess.file_exists(SAVE_PATH):
        return null
    var file = FileAccess.open(SAVE_PATH, FileAccess.READ)
    var content = file.get_as_text()
    file.close()
    var result = JSON.parse_string(content)
    if result == null:
        return null
    return result
```

Процес завантаження передбачає попередню перевірку наявності файлу. У разі успішного парсингу рядка, функція повертає об’єкт, який потім використовується для ініціалізації параметрів персонажа та відновлення стану всіх 9 модифікаторів кожного предмета в інвентарі.

ВИСНОВКИ

У процесі виконання дипломного проєкту було розроблено ігровий прототип, який реалізує комплексну систему взаємодії гравця з ігровим світом через механіки збору, обробки інгредієнтів, створення зілля та виконання замовлень. Основною ідеєю проєкту стало створення гнучкої та розширюваної системи крафту, яка виходить за межі класичних рецептурних підходів і базується на параметричному описі ігрових об’єктів. Такий підхід дозволяє формувати більш варіативний ігровий процес, у якому результат залежить не лише від фіксованих комбінацій, а й від сукупності властивостей використаних компонентівя

Розроблена архітектура гри включає декілька взаємопов’язаних підсистем, кожна з яких виконує окрему функціональну роль, але при цьому інтегрується у єдиний ігровий цикл. До таких підсистем належать система обробки інгредієнтів, система створення зілля, система замовлень та система збереження ігрового стану. Їх взаємодія забезпечує безперервний процес гри, у якому кожна дія гравця має логічні наслідки та впливає на подальший розвиток подій. Завдяки цьому формується цілісна структура ігрового досвіду, що поєднує елементи планування, експериментування та прийняття рішень.

Особливу увагу в роботі було приділено реалізації системи інгредієнтів, яка базується на поділі характеристик на базові елементи та модифікатори. Такий підхід дозволяє описувати кожен інгредієнт як багатовимірний об’єкт із власним набором властивостей, що відкриває можливості для гнучкого комбінування та формування великої кількості варіантів кінцевого результату. Додатково було реалізовано механізми обробки інгредієнтів за допомогою різних інструментів, які змінюють їх параметри та впливають на подальшу взаємодію у системі крафту. Це створює додатковий рівень глибини, оскільки один і той самий інгредієнт може мати різний результат залежно від способу його обробки.

Система створення зілля побудована як інтерактивний процес, у якому гравець безпосередньо працює з інгредієнтами в межах обмеженого робочого простору. Важливим елементом даної системи є обмеження параметрів, зокрема неможливість перевищення певних критичних значень, що додає елемент балансу та змушує гравця більш уважно підходити до вибору компонентів. Окрім цього, візуальні характеристики кінцевого продукту, такі як колір та інші зовнішні ознаки, залежать від складу інгредієнтів, що підсилює зворотний зв’язок між діями гравця та результатом і робить процес більш інтуїтивно зрозумілим.

Окремим значущим елементом проєкту є система замовлень, яка формує ігрові завдання з чітко визначеними параметрами. Вона задає вимоги до кінцевого зілля, включаючи як мінімальні, так і максимальні значення окремих характеристик, а також вводить часові обмеження на виконання завдань. Це створює додатковий рівень складності та формує динамічний ігровий процес, у якому гравець змушений діяти швидко, але водночас зважено, враховуючи всі доступні ресурси та обмеження. Таким чином забезпечується баланс між експериментуванням і необхідністю ефективного прийняття рішень.

Система збереження ігрових даних реалізована на основі формату JSON, що забезпечує структуроване представлення інформації та можливість її подальшого відновлення. До збережуваних даних належать координати гравця, стан інвентарю, параметри інгредієнтів, активні замовлення та стан ігрового світу. Такий підхід дозволяє забезпечити повне та коректне відновлення ігрової сесії без втрати ключових даних та логіки взаємодії систем, що є важливим для стабільності ігрового процесу.

У результаті виконання дипломного проєкту було створено цілісний ігровий прототип, який поєднує елементи процедурної генерації, параметричного моделювання та інтерактивної взаємодії. Реалізована система демонструє можливість побудови складних ігрових механік на основі відносно простої архітектури, що є важливим з точки зору подальшого масштабування та

розвитку проєкту. Отриманий результат підтверджує доцільність використання обраних підходів до організації ігрових систем та їх взаємодії.

Отримані результати можуть бути використані як основа для подальшого вдосконалення гри, зокрема шляхом розширення кількості інгредієнтів, додавання нових інструментів обробки, ускладнення системи замовлень та впровадження додаткових ігрових механік. Таким чином, розроблений прототип має потенціал до подальшого розвитку та адаптації під більш складні ігрові сценарії, а також може бути використаний як базова структура для створення подібних проєктів у жанрі системного крафту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Godot Engine. Офіційна документація рушія Godot. Godot Documentation. URL : https://docs.godotengine.org/uk/4.x/ (дата звернення : 08.02.2026).

2. Newzoo. Global Games Market Report – Indie Segment Insights. Newzoo. URL : https://newzoo.com/resources/trend-reports/ (дата звернення : 15.04.2026).

3. Unity Technologies. Офіційна документація Unity. Unity Documentation. URL : https://docs.unity.com/ (дата звернення : 12.04.2026).

4. Cocos2d-x. Офіційна документація рушія Cocos2d. Cocos Documentation. URL : https://docs.cocos.com/ (дата звернення : 12.04.2026).

5. Godot Engine. GDScript Reference. Godot Documentation. URL : https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/ (дата звернення : 10.02.2026).

6. Microsoft. Documentation for C# programming language. Microsoft Learn. URL : https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення : 12.04.2026).

7. itch.io. Платформа асетів та спрайтів URL : https://itch.io/ (дата звернення : 12.04.2026).

8. Шелл Д. Мистецтво дизайну ігор: книга лінз. Boca Raton : CRC Press, 2019. 656 с.

9. Фуллертон Т. Дизайн ігор. Boca Raton : CRC Press, 2018. 450 с.

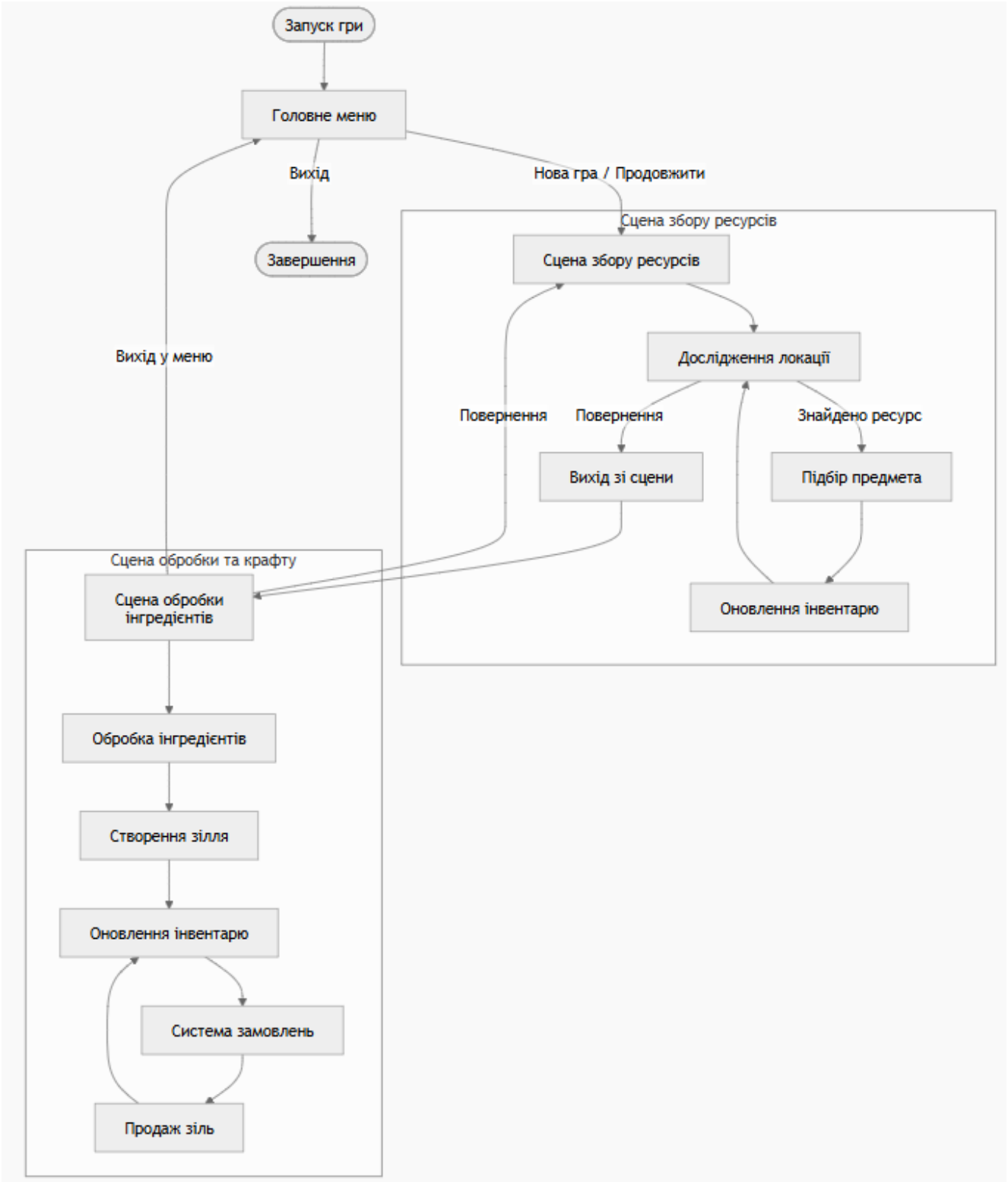
10. Роджерс С. Розробка відеоігор: посібник зі створення якісного геймдизайну. Hoboken : Wiley, 2014. 512 с.

11. Сейлен К., Циммерман Е. Основи дизайну ігор: правила гри. Cambridge : MIT Press, 2004. 688 с.

12. Найстром Р. Патерни програмування ігор. Genever Benning, 2014. 354 с.

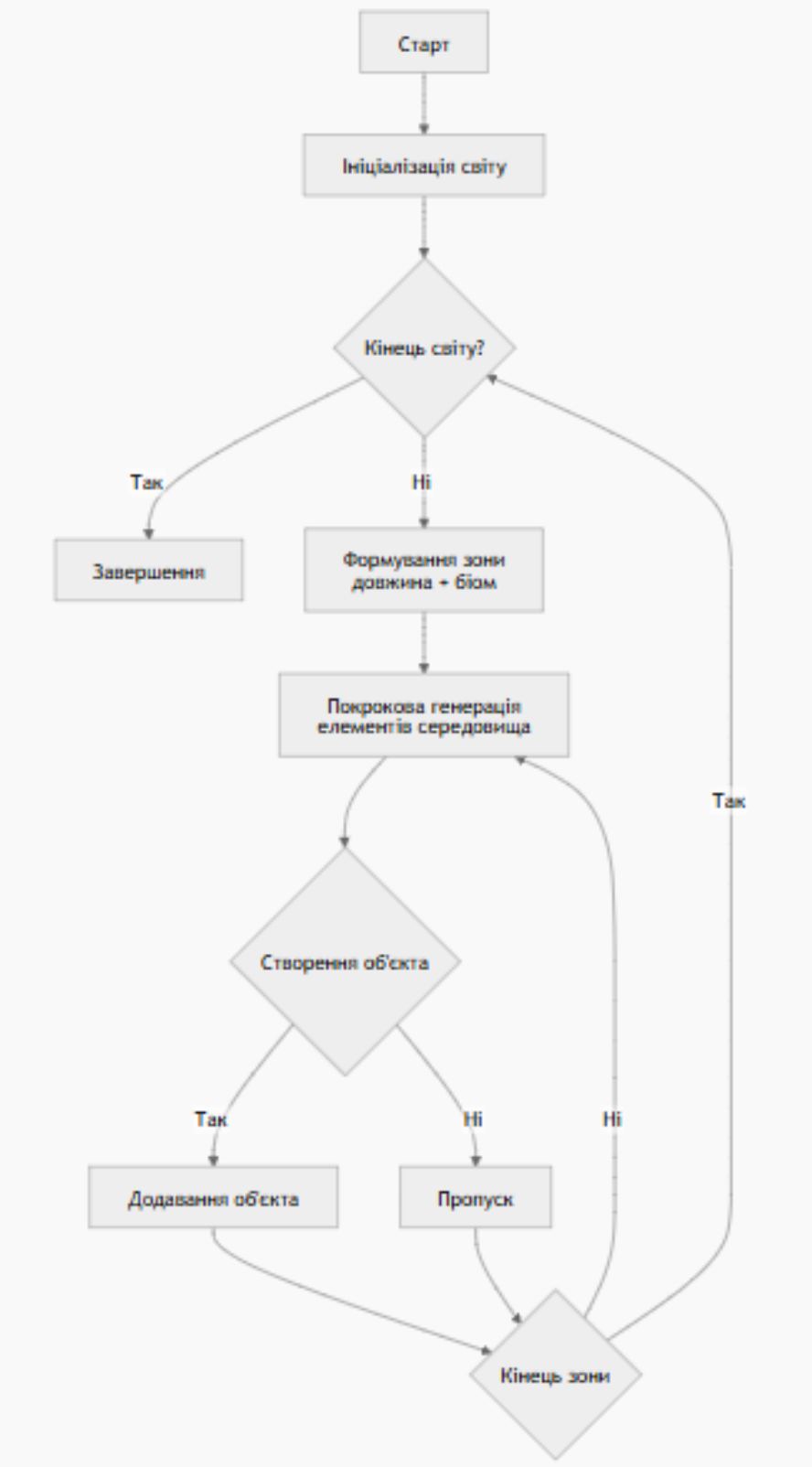
ДОДАТОК А

Діаграма загальна схема взаємодії сцен



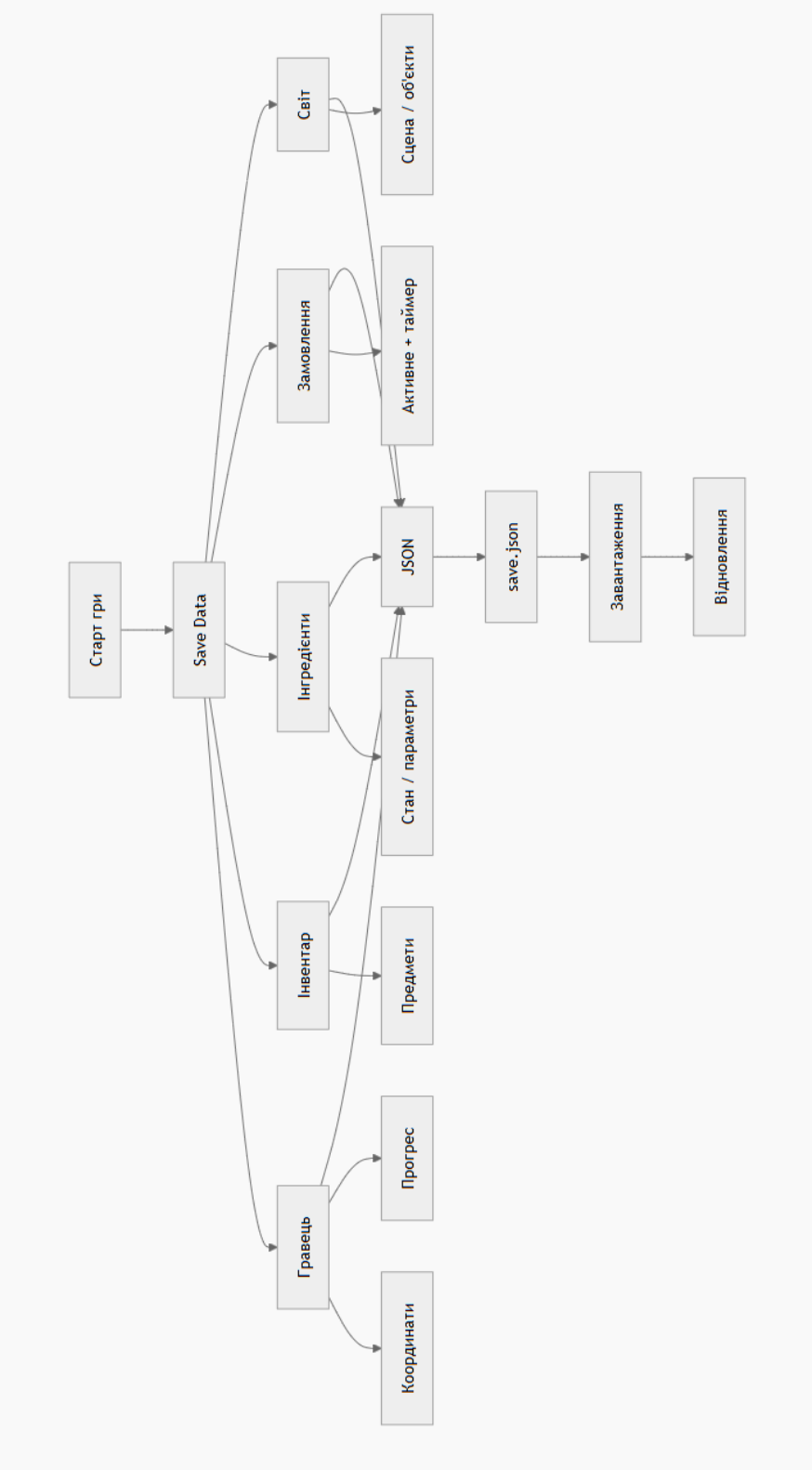
ДОДАТОК Б

Діаграма схема створення світу



ДОДАТОК В

Діаграма структура даних для збереження



Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, _____

студент(ка) ____ курсу, _____ форми навчання, математичного факультету, спеціальності _____, адреса електронної пошти _____, –

підтверджую, що написана мною кваліфікаційна робота магістра на тему «_____
_____» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Студент

_____	_____	_____
(дата)	(підпис)	(прізвище, ініціали)

Науковий керівник

_____	_____	_____
(дата)	(підпис)	(прізвище, ініціали)