МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Комп'ютерна графіка і Web-дизайн»

Група: 4КГ-08

Дипломний проєкт

здобувача освіти денної форми навчання
КГ.08.22.000.ДП

СЛИНЯВЧУК
КАТЕРИНИ РУСЛАНІВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Комп'ютерна графіка і Web-дизайн»**

Група: **4КГ-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка гри "Змійка" з додаванням модифікацій у геймплей

Проєктний матеріал складається з пояснювальної записки на 83 сторінках та графічного (презентаційного) матеріалу на 16 аркушах (слайдах)

Дипломник _____ (Слинявчук К.Р.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «20» червня 2025 р. Протокол ЕК № 1

Оцінка ЕК 4 (добре) / 80 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна графіка і Web-дизайн»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проєкт

Здобувачеві освіти *Слинявчук Катерині Русланівні*

(прізвище, ім'я, по батькові)

1. Тема проєкту *Розробка гри "Змійка" з додаванням модифікацій у геймплей*

затверджена наказом по коледжу від "14" листопада 2024р. № 246

2. Термін здачі закінченого проєкту

3. Вихідні данні до проєкту *Використання програмного рушія Unity; Використання мови програмування C# та основних бібліотек рушія Unity; Реалізація основних ігрових механік гри «Змійка»; Реалізація системи управління; Реалізація модифікацій геймплею; Реалізація ігрового інтерфейсу гри*

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

Проведення аналізу існуючих ігрових рішень; Вибір програмного забезпечення для розробки гри; Розробка концепції ігрового процесу для гри «Змійка»; Проектування основних модулів гри; Проектування модифікацій геймплею; Реалізація основних елементів гри; Реалізація модифікацій геймплею гри; Реалізація інтерфейсу; Тестування гри

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Огляд існуючих ігрових рішень; Обрання програмного забезпечення для розробки; Концепція ігрового процесу розробляємої гри; Структура основних модулів гри; Описання модифікацій геймплею гри; Реалізація основних модулів; Реалізація модифікацій геймплею; Кінцева структура проекту; Тестування та скріншоти реалізованої гри

6. Консультанти по проєкту, із зазначенням розділів проєкту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийн
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 02.06.2025

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Слинявчук К.Р.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проєкту	Термін виконання етапів дипломного проєкту (роботи)	Відмітка виконан
1	*Вступ. Постановка мети та задач проектування*	*14.05.2025*	
2	*Аналіз існуючих ігрових рішень*	*16.05.2025*	
3	*Вибір та налаштування засобів розробки*	*17.05.2025*	
4	*Проектування концепції ігрового процесу для гри*	*20.05.2025*	
5	*Проектування основних елементів гри*	*22.05.2025*	
6	*Проектування модифікацій геймплею гри*	*28.05.2025*	
7	*Реалізація основних елементів та модифікацій*	*01.06.2025*	
8	*Реалізація ігрового інтерфейсу*	*03.06.2025*	
9	*Відлагодження елементів гри*	*06.06.2025*	
10	*Тестування працездатності елементів гри*	*10.06.2025*	
11	*Виправлення виявлених помилок*	*11.06.2025*	
12	*Аналіз результатів, підготовка слайдів презентації*	*12.06.2025*	
13	*Економічні розрахунки та питання з охорони праці*	*13.06.2025*	
14	*Підготовка графічної частини проекту*	*14.06.2025*	
15	*Підготовка проекту до захисту та тестування гри*	*16.06.2025*	

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КГ 08. 22 000. 00 ДП	Дипломний проєкт		
А4			КГ 08. 22 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КГ 08. 22 000. 00 ДП		
Розробив		Слинявчук К.Р.		16.06	Розробка гри "Змійка" з додаванням модифікацій у геймплей	Літ.	Аркуш 4 / Аркушів 83
Перевірив		Джабраілов Д.В.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06.25		ВСП "ОТФК ОНТУ" гр.4КГ-08	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

В наш час комп’ютерні ігри займають значне місце у IT-ринку світу. З кожним роком цифрові розваги отримують все більше уваги від користувачів, проникаючи у все більшу кількість пристроїв та програмних платформ. Через це штат робітників цієї індустрії розширюється зі збільшенням ринку. Водночас з тим, залишаються популярними класичні прості ігри як змійка, арканоїд та інші.

Розробка такого рівня ігор може вестись як невеликими командами, так і зовсім однією людиною. Сучасні технології дають змогу відносно просто створити гру, а доступність цифрових магазинів, розмістити такі ігри на загальному ринку. Для створення подібного проекту можна використовувати умовно безкоштовні ігрові програмні рушії, прості графічні редактори, а також ассети з вільним доступом та ліцензуванням.

В рамках даного дипломного проекту передбачається проектування та реалізація гри «Змійка». Вона є класикою ігрової індустрії та відтворювалась від ігрових приставок та переносних ігрових пристроїв до перших мобільних телефонів. Досить простий ігровий процес гри дає змогу легко відтворити його на слабких пристроях.

У даному дипломному проекті гра буде розроблятись для персонального комп’ютера, із додаванням модифікацій у геймплей. Як основа обрана класична версія гри із деяким оновленням візуальної складової. Правила ігрового процесу будуть реалізовані згідно класичних ігор цього жанру. Модифікації геймплею дадуть змогу гравцям відчувати унікальний ігровий досвід, який буде змінюватись від одного проходження до іншого. Реалізація механізмів модифікацій геймплею надає проекту свіжий погляд на класику ігрової індустрії.

Гра буде розроблятись таким чином, щоби поділити її на зручні модулі для розробки. Цей підхід дасть змогу у подальшому додавати нові модифікації та зміни у ігровий процес, від візуальних до геймплейних. Використання сучасного ігрового рушія в розробці дасть змогу підтримувати проект на різних платформах та версіях операційної системи Windows.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Проведення аналізу існуючих ігрових рішень

1.1.1 Загальні відомості про ігри жанру змійка

Для реалізації теми дипломного проекту, потрібно провести аналіз такого явища як ігри жанру змійка. Це необхідно зробити з метою визначення та конкретизації основних рис ігор цього типу. Взагалі ігрова індустрія має свою жанрову градацію, як наприклад музика, книги, або скоріше кінематограф. В усіх цих випадках жанри диктують те, як буде представлений продукт а також те, яким чином ми будемо взаємодіяти із цим продуктом. Жанри ігрової індустрії мають декілька способів їх формування. Вони вперше були започатковані у 1984 році Крісом Кроуфордом та розділяли ігри за тим, які навички потрібно буде використовувати гравцю під час ігрового процесу. У подальшому класифікація ігор по жанрам еволюціонувала до поточної системи по Орланду, Стейнбергу та Томасу.

Гра, яку потрібно розробити не має чіткого відношення до будь-якої з класифікацій, навпаки, будучи однією з перших ігор, вона сама утворює цілий жанр, так само як, наприклад, тетрис, або арканоїд. Ці ігри будучи на початку розвитку ігрової індустрії дали основу для подальшого розвитку ігрового процесу – геймплею – та утворенню жанрів. Втім, не дивлячись на те, що ігрові особливості цього жанру відомі майже всім, в наш час потрібно реалізувати якісь додаткові механіки, яким б додали до ігрового процесу якусь новизну. Для цього слід виконати аналіз існуючих ігрових рішень та синтезувати в них загальне та виділити цікаві рішення для розробляємого проекту.

1.1.2 Аналіз гри «Snake.io»

Першою грою для аналізу є «Snake.io». Ця гра є кросплатформною та може виконуватись як на смартфонах с ОС Android або IOS, так и на персональних комп’ютерах та у вікнах веб-браузера. На рисунку 1.1 можна побачити скріншот ігрового процесу «Snake.io». Даний проект є мультиплеєрним, гравці обирають зовнішній вигляд для своєї змії та натискають грати, після чого автоматично

знаходиться ігровий сервер до якого підключається гравець. У кожній ігровій сесії приймають участь двадцять гравців. Сесія триває доки в ній є гравці, якщо є вільне місце, то сервер автоматично підключає гравця, який шукає вільну ігрову сесію. Головна мета гри полягає у тому, щоби збирати кульки, які дають ігровий рахунок. Разом з ігровим рахунком, змійка гравця збільшується у довжину та товщину.

Рисунок 1.1. Скріншот ігрового процесу гри «Snake.io»

Ще одною відмінною рисою гри є змінені правила взаємодії змійки зі своїм тілом та тілами інших гравців. Гра дозволяє вільно переміщуватись крізь своє тіло, але при контакті з тілом іншої змійки, гравець програє. На місці тіла змійки гравця, який програв залишаються кульки з ігровим рахунком у еквіваленті його рахунку.

Також можна відмітити, що в цій грі дещо змінена система переміщення, розробники відійшли від руху по клітинкам, та дозволили гравцям вільно рухатись у просторі, в залежності від їх розмірів. Це в купі зі зміненими правилами гри створюють дуже динамічний ігровий процес, який хоч і простий та здається одноманітним, але сильно затягує.

1.1.3 Аналіз гри «Snake Rivals»

Гра «Snake Rivals» також доступна для гри на смартфонах обох ОС, а також на комп’ютері, як у версії додатку, так і через веб-браузер. Так само, як і

попередній проект, цей є мультиплеєрним. Він бере за основу ігровий процес класичної змійки, але при цьому додає ряд нових ігрових режимів, які корінним чином змінюють геймплей, додаючи до нього дух змагання, або мирний процес побудови власного притулку. На рисунку 1.2 можна побачити скріншот розглядаємої гри.

Рисунок 1.2. Скріншот ігрового процесу «Snake Rivals»

Головною рисою цієї гри є візуальна складова, оскільки розробники реалізовали гру у повному 3D з трьох вимірним оточенням та анімаціями. Такий підхід збільшує привабливість гри для молодих гравців та дає змогу для додаткової монетизації проекту.

1.1.4 Аналіз гри Snake, snake, snake!

Останнім проектом, який був розглянутий стала гра «Snake, snake, snake!». Цей проект розроблений для персонального комп’ютеру та доступний у цифровому магазині Steam. Скріншот гри «Snake, snake, snake!» зображено на рисунку 1.3. Як можна бачити, гра намагається слідкувати духу оригіналу, намагаючись повторювати той самий зовнішній вигляд, ігровий процес та правила гри. Спочатку гра представляла лише режим гри для одного гравця, але з оновленнями додали гру на двох. Як і у класичній грі, якщо «з’їсти» квадратик

цільовий квадратик, гравець отримає ігровий рахунок, змійка збільшить свою швидкість, а також отримає додаткові клітинки у свою довжину. Навпаки, якщо натрапити на клітинку тіла змійки, то, в залежності від режиму, гравець або втрачає життя, або програє. В грі доступні різні варіанти гри, від класичної змійки до різних варіацій, наприклад з рівнем, якій рухається, з ворогами, з можливістю робити постріли з квадратиків тіла змійки.

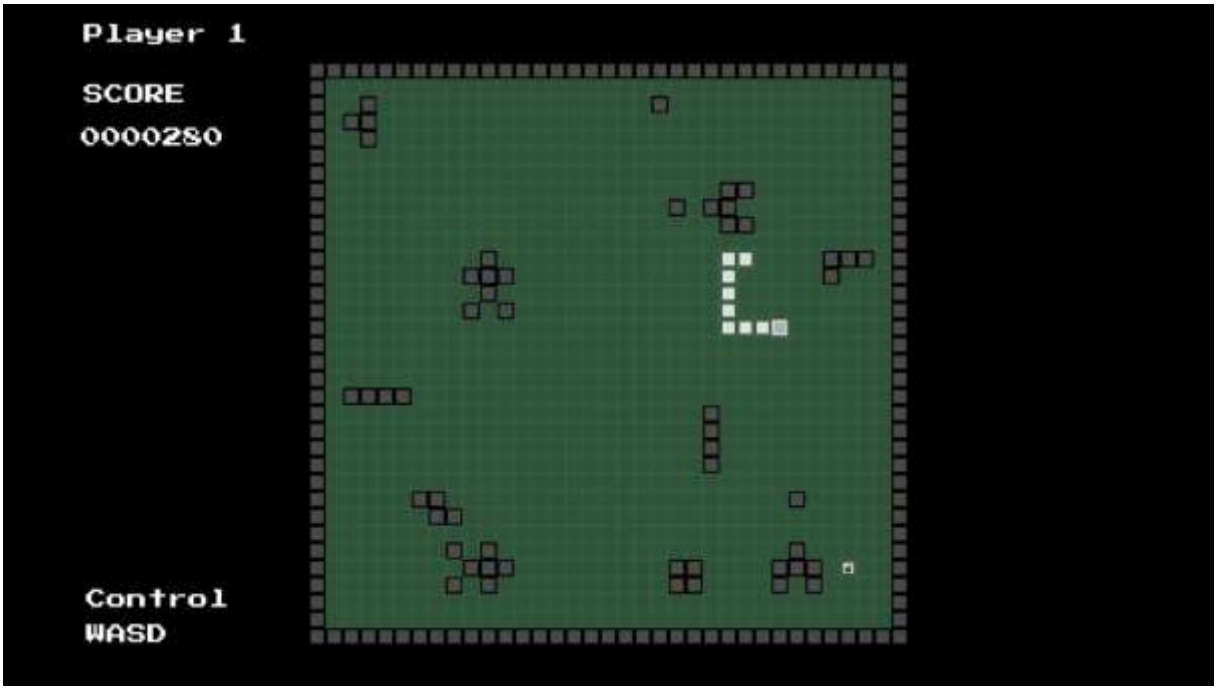


Рисунок 1.3. Скріншот ігрового процесу гри «Snake, snake, snake!»

З точки зору зовнішнього вигляду проект тримається стилізованого під старі приставки дизайну, вона повністю реалізована у 2D вигляді та має класичний підхід до управління змійкою.

1.1.5 Результати аналізу існуючих ігрових аналогів

В ході проведеного аналізу, було розглянуто три гри, які виконані у жанрі «Змійка», але мають різні підходи до реалізації. Можна зробити висновок, що існує два основних напрями в таких проектах. Перший – це напрям на онлайн складову та реалізація соціальних механік, правил гри направлених на взаємодію з іншими гравцями. Другий напрям – створення гри у класичному стилі із дотриманням старих правил гри, але з додаванням нових геймплейних елементів. Втім, видно, що в наш час створювати копію гри жанру «Змійка» є сенс лише для

тих, хто тільки починає пробувати сили у розробці ігор.

Проведений аналіз підтверджує актуальність теми у частині додавання модифікацій у геймплей. З іншого боку слід заключити, що реалізація онлайн складової не є обов'язковою, оскільки переводить гру у категорію онлайн ігор з обмеженими геймплейними особливостями. Тому при розробці концепції гри було вирішено притримувалась підходу реалізованого у грі «Snake, snake, snake!», а саме: мінімалістичний дизайн, класичне управління, стандартні правила.

1.2 Вибір програмного забезпечення для розробки гри

1.2.1 Вибір ігрового програмного рушія

Темою мого дипломного проекту є розробка гри, отже потрібно було обрати ігровий програмний рушій для цих цілей. На поточний час існує декілька рушіїв, які б мали безкоштовний варіант ліцензії, але вони відрізняють з точки зори функціоналу, способів розробки та інструментів. Перед тим, як обрати рушій, було виконано їх аналіз.

Unreal Engine – це потужний ігровий рушій, розроблений компанією Epic Games, який використовується для створення відеоігор, інтерактивних візуалізацій, фільмів, віртуальної та доповненої реальності. Він підтримує високоякісну графіку, фізику, анімацію та має вбудовану систему візуального програмування Blueprints, що дозволяє розробляти ігри без необхідності глибокого знання коду. Unreal Engine активно використовується як інді-розробниками, так і великими студіями по всьому світу. На цьому рушії створено такі відомі ігри як Fortnite, серія ігор Borderlands, Hellblade: Senua's Sacrifice та інші. Перераховані проекти ілюструють потужність рушія та його гнучкість з точки зору стилістики реалізованих на ньому ігор.

Також ігровий програмний рушій Unreal Engine має деякі власні ключові особливості. Nanite – це віртуалізована геометрія, яка дозволяє використовувати мільйони полігонів у сцені без втрати продуктивності, яка дає змогу імпортувати фотореалістичні 3D-моделі без спрощення. Lumen – система глобального освітлення та відображень у реальному часі. World Partition – автоматично

розділяє великі ігрові світи на частини й підвантажує лише потрібні зони.

Іншим ігровим програмним рушієм, який розглянула, був Unity. Це універсальний ігровий рушій для кросплатформної розробки, розроблений компанією Unity Technologies, який дозволяє створювати 2D та 3D ігри, інтерактивні додатки, VR/AR-проєкти для понад 20 платформ – ПК, мобільних пристроїв, консолей та браузерів. Unity відомий своєю доступністю, великою документацією та активною спільнотою. Він підходить як для інді-розробників, так і для великих студій, та широко використовується в індустрії ігор, архітектурної візуалізації, освіти, кіновиробництва й симуляцій. Серед ігор, які розроблені на цьому рушії можна виділити Hollow Knight, Ori and the Blind Forest, Cuphead.

Серед ключових особливостей ігрового рушія можна виділити об’єктно-орієнтовану модель, адже у Unity все побудовано на GameObject – базових об'єктах сцени, до яких додаються компоненти (Component): скрипти, фізика, колайдери, рендери тощо. Відповідно компонентна система, що дозволяє легко створювати складну поведінку, комбінуючи прості частини. Також Unity використовує C# API та його великий набір бібліотек і функцій доступний через C# – популярну, безпечну та добре документовану мову.

Для реалізації проекту було обрано ігровий рушій Unity через декілька причин. Цей рушій має обширну документацію та велику кількість навчальних матеріалів, як від компанії розробника рушія, так і інших користувачів. Завдяки цьому завжди можна знайти рішення проблемних моментів під час розробки. Також, як можна побачити, з переліку розроблених на рушіях ігор, Unreal Engine більше підходить для професійної розробки крупних проектів, в той час як Unity обирають для реалізації простих або інді проектів.

1.2.2 Вибір іншого програмного забезпечення для розробки гри

Також для розробки гри потрібно обрати ряд інших інструментів. Оскільки створити гру не написавши коду неможливо, потрібно обрати середу розробки. Мова програмування продиктована рушієм, тому потрібна така система, яка б підтримувала C#. Документація ігрового рушія рекомендує для роботи

використовувати Visual studio, який є потужним інтегрованим середовищем розробки (IDE), створеним компанією Microsoft, яке використовується для написання, налагодження, тестування та розгортання програмного забезпечення. Воно підтримує багато мов програмування, включно з C#, C++, Visual Basic, Python, JavaScript, TypeScript, і чудово інтегрується з .NET-платформою. Серед ключових особливостей підсвічування коду, автоматичне інтелектуальне доповнення, підтримка різноманітних плагінів.

Важливою частиною розробки є й робота з графікою. У розробляємому проекті буде реалізована 2D-графіка, тому знадобиться використовувати графічний редактор. Серед великої кількості варіантів, виходячи з досвіду використання, зручності та інструментарію, було обрано Adobe Photoshop. Він є потужною програмою для редагування растрової графіки, розроблена компанією Adobe Inc.. Photoshop використовується професіоналами по всьому світу для ретуші фотографій, створення цифрових ілюстрацій, дизайну інтерфейсів, підготовки матеріалів до друку та навіть створення анімацій. Серед ключового функціоналу, який знадобиться під час розробки є широкий вибір інструментів для ретуші, та редагування зображень.

1.3 Розробка концепції ігрового процесу для гри «Змійка»

1.3.1 Загальна ігрова концепція розробляємої гри

Для початку розробки необхідно розробити концепцію гри. Концепція гри – це короткий, узагальнений опис ідеї майбутньої гри. Вона визначає ключову ідею, стиль, геймплей, а також основну мету гри, на основі чого формується вся подальша розробка. Це своєрідна «візитівка» гри, з якої починається будь-який ігровий проєкт. Концепція розглядає такі складові гри як її жанр, цільова платформа, основна ідея, ігровий процес, сетінг та атмосфера, цільова аудиторія та інтерфейс. Створення концепції гри є дуже важливою частиною розробки, оскільки дозволяє не тільки провести початковий аналіз проекту, але й оцінити обсяг роботи та створити курс розробки, який не дозволить розробнику втрачати фокус.

Розробляєма гра буде мати ігровий жанр «Змійка» з видом зверху, оскільки це прописано у темі дипломного проекту. Цільовою платформою є персональні комп'ютери та системи працюючі на ОС Windows. Основною концепцією є відтворення ігрового процесу класичної змійки. Візуальний стиль розробляємої гри є досить простим, виконаним у 2D-стилі з використанням спрайтів, тайлів, та рухомих об'єктів. Допускається використання сторонніх ассетів, без порушення авторських прав, або з використанням безкоштовної ліцензії.

1.3.2 Концепція ігрового процесу та механік

Для подальшого проектування гри потрібно більш детально зосередитись на концепції ігрового процесу розробляємої гри. Чітко сформовані правила гри дадуть змогу більш ефективно проектувати та реалізовувати ігрові модулі.

Фон ігрового поля буде незмінним, але положення предметів на ньому буде відрізнятись від однієї ігрової сесії до іншої. Концептуальне зображення ігрового поля можна побачити на рисунку 1.4. Всього на полі буде знаходитись декілька видів об'єктів, а саме:

- Голова змійки – це ігровий об'єкт, котрим буде управляти гравець. Цей об'єкт буде переміщуватись по полю та взаємодіяти з оточуючими предметами;
- Куші, або перешкоди – це об'єкти, які можуть мати різний вигляд та відображають елементи ігрового поля, які не є прохідними і при взаємодії з ними гравець програє. Перешкоди можуть додаватись на рівні випадковим чином;
- Їжа – це об'єкт з яким має взаємодіяти гравець. При взаємодії, гравець збільшує свій ігровий рахунок, а також збільшує змійку у розмірі. Їжа генерується на рівні випадковим чином;
- Бонуси – це об'єкти з якими гравець також може взаємодіяти. Якщо їх «з'їсти» вони будуть давати випадкові бонуси. Що буде модифікувати ігровий процес. Бонуси генеруються на рівні випадковим чином та з випадковими властивостями;
- Кордони ігрового поля – це межа за яку гравець неповинен виходити.

При взаємодії з нею, гравець програє. У подальшому можна реалізувати ігрові режими з наскрізними кордонами ігрового поля.

Всі ці об’єкти випадковим чином будуть генеруватись кожний раз, коли гравець буде запускати рівень.

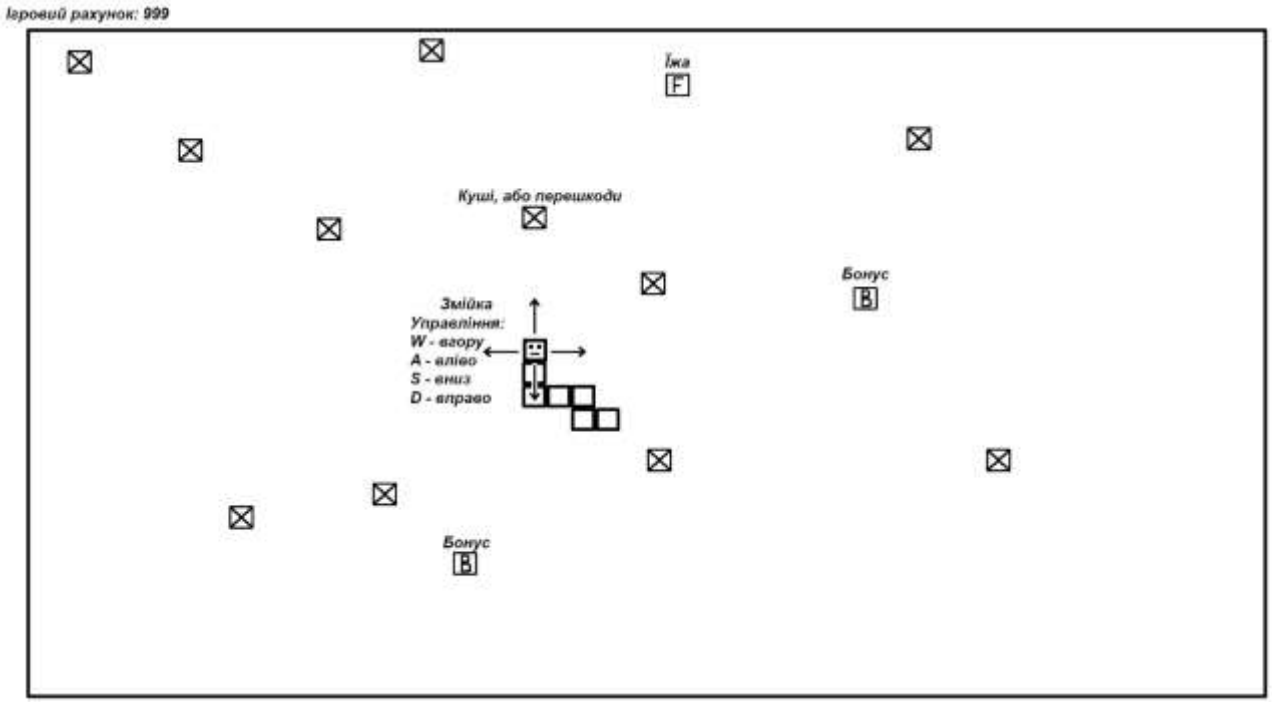


Рисунок 1.4. Концепція відображення ігрового процесу розробляємої гри

Ігрове поле розбито на клітинки, по яким гравець може рухатись. Управління змійкою виконується за допомогою клавіш клавіатури WASD, або стрілок – ці клавіші відповідають за напрями вгору, вліво, вниз, вправо відповідно.

Механізм збільшення швидкості проходить дещо особливим чином. Всього планується десять рівнів швидкості, які збільшуються в залежності від кількості ігрового рахунку. Кількість їжі, яку потрібно «з’їсти» для підвищення швидкості різне. Це зроблено для того, щоби можна було на початкових етапах, коли у змійки невелика швидкість, частіше збільшувати її, і вище, коли швидкість стає комфортною, частота її зміни буде зменшуватись.

Розміщення їжі генерується випадковим чином по всьому ігровому полю. Грає повинна відстежувати, чи можна розміщувати їжу у клітинці, чи немає в цій клітинці іншого об’єкту, або гравця чи частини змійки.

Розміщення бонусів також генерується випадковим чином. Окрім того,

бонуси дають випадковий ефект, наприклад одні можуть збільшити ігровий рахунок, а інші швидкість. Тип бонусу обирається грою при створені його на рівні. Кожний бонус при генерації має перевіряти чи є для нього вільне місце на ігровому полі. Пріоритет у розміщенні гра повинна давати саме їжі, а не бонусам.

1.4 Проектування основних модулів гри

Подальшим етапом розробки будь-якої гри є проектування модулів гри. Це виконується з метою уточнення технічних деталей реалізуємого функціоналу, та дозволяє більш детально проробити їх особливості. Створення проекту узагальнює ігровий концепт та формалізує його з точки зору технічної реалізації. За допомогою проектування, загальні ігрові елементи становляться більш точними та конкретизованими, з'являється розуміння що та як виконувати під час розробки. Окрім того проектування дозволяє знайти можливі проблемні місця у ігровому концепті та скорегувати його за необхідністю.

Почати проектування потрібно з загальної структури розробляємої гри, та закласти в неї перераховані в ігровому концепті механіки. На рисунку 1.5 можна побачити блок-схему структури коду модулів розробляємого проекту.

Рисунок 1.5. Загальна блок-схема структури коду модулів розробляємого проекту

Всього код проекту можна поділити на три частини, або блока. Як видно з рисунку 1.5, це блоки коду гравця, оточення та окремо код роботи інтерфейсу.

Блок коду гравця повинен реалізовувати ігровий функціонал для гравця та забезпечувати функціонал який притаманний лише гравцю. Таким функціоналом має бути управління змійкою, її логіка та логіка частинок тіла змійки. Сутність цих наступна:

- Модуль управління гравцем повинен реалізовувати обробку введення управління змійкою, а саме натискання на кнопки управління WASD та стрілки. Окрім того саме цей код має відповідати за переміщення голови змійки по ігровій локації. Також цей модуль має зберігати в собі всі дані, які відповідають за швидкість та її обчислення;
- Модуль логіки змійки повинен обробляти всю основну логіки гри, яка пов’язана зі взаємодією гравця та оточення та інших механік геймплею. Цей код повинен у правильний момент додавати нові частинки змійки, рухати їх, проводити перевірку на перетин ігрових об’єктів на рівні та коректним чином реагувати на це, виконувати операції які пов’язані зі збільшенням ігрового рахунку. Можна сказати, що цей модуль є важливим вузлом у роботі гри;
- Модуль логіки частин змійки відповідає за зміну графіки частинок змійки за потреби.

Блок коду оточення має реалізовувати ігрову складову пов’язану з усім оточенням на ігровому полі – це кущі, їжа та бонуси. Ці елементи мають деякі схожі етапи функціонування, але виконують різну роль у ігровому процесі. Сутність модулів наступна:

- Модуль параметрів їжі відповідає за призначення кожному новому ігровому об’єкту їжі ідентифікаційного номеру. Це потрібно буде роботи з метою контролю розміщення ігрових об’єктів їжі на локації та для спрощення пошуку цих ігрових об’єктів іншими елементами гри;
- Модуль генерації їжі відповідає за ведення підрахунку ігрових об’єктів їжі, за її розміщення на локації та відповідає за генерацію випадкових

координат для генерації нових ігрових об’єктів їжі. Слід зазначити, що цей код має відслідковувати ігрові об’єкти гравця, бонусів та кущів, щоби уникнути ситуацій генерації їжі в середині інших ігрових об’єктів на локації;

- Модуль параметрів бонусів відповідає за процес надання бонусів гравцю. Також він є унікальним для будь-якого бонусу, який буде знаходитись на ігровому полі та зберігає значення бонусів. Також, цей бонус зберігає в собі ідентифікаційний номер бонусів;
- Модуль генерації бонусів реалізовує функціонал роботи з бонусами, а саме шанс генерації бонусу, безпосередньо генерацію самого бонусу, його розміщення на ігровому полі. Також отримання випадкових координат для ігрового об’єкту бонусу та відстеження коректного його розміщення на полі, для уникання розміщення бонусів всередині інших ігрових об’єктів. Менеджмент списку бонусів;
- Модуль генерації кущів відповідає за перше розміщення кущів на рівні, або додавання ще одного куща при необхідності, а також генерацію випадкових координат для кущів та відстеження перетину позицій кущів з іншими об’єктами на ігровому полі.

Нарешті модуль процесору інтерфейсу відповідає за обробку даних для інтерфейсу гравця та їх коректне виведення на інтерфейс гравця.

Потрібно також зазначити про зв’язки між модулями гри, оскільки майже кожен з них повинен взаємодіяти з іншим, передаючи, або отримуючи дані. На рисунку 1.6 зображена схема зв’язків між модулями розробляємого проекту. Як можна побачити кількість передач даних дуже велика. Це пов’язано з тим, що гра на початку рівня майже немає заздалегідь створених ігрових об’єктів. Більшість ігрових об’єктів генерується грою самостійно, через що неможливо створити зв’язки вручну під час розробки. Це потрібно робити із коду гри, реалізуючи код таким чином, щоби він міг знаходити ігрові об’єкти, проводити їх аналіз, виконувати дії з ними, а також генерувати нові об’єкти.

Такий підхід досить сильно ускладнює проектування та реалізацію коду,

тому в процесі реалізації спроектованих модулів, можуть бути внесені деякі зміни, які не передбачені даними схемами.

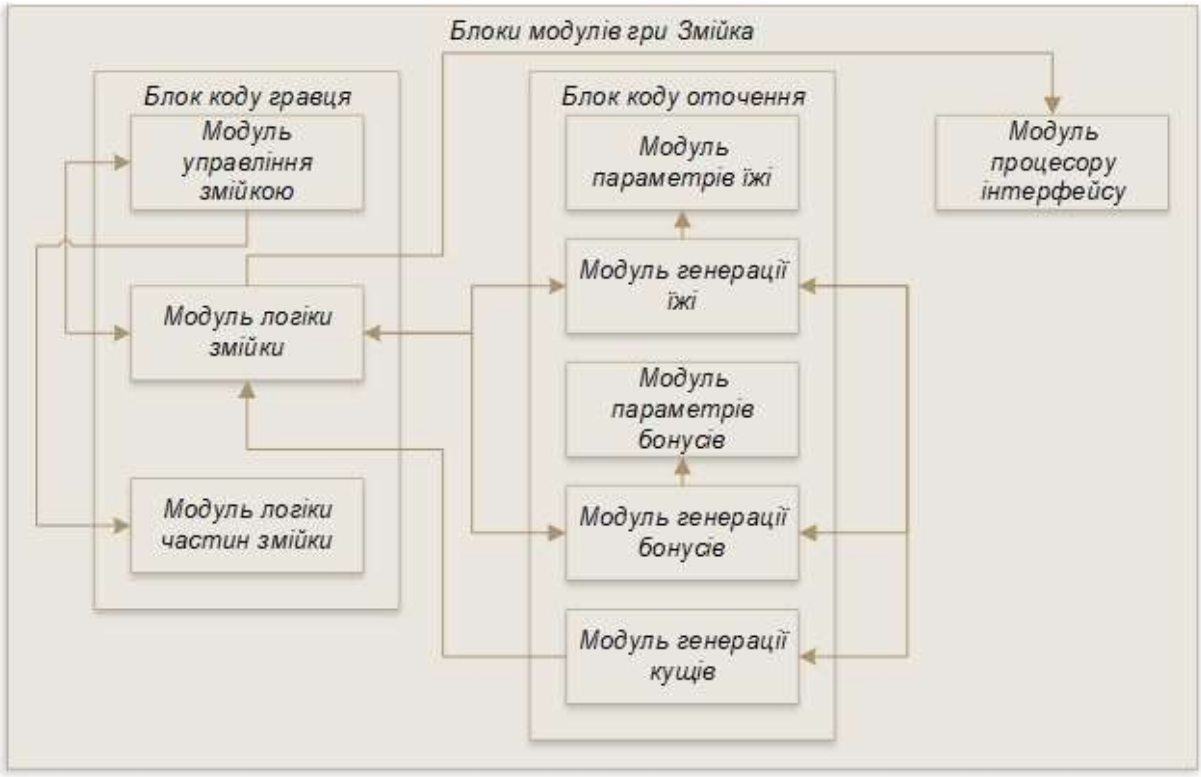


Рисунок 1.6. Схема зв’язків між модулями розробляємого проекту

Ключовою ідеєю, яка врахована під час проектування – модульність гри. Тобто під час проектування гра сформована таким чином, що реалізовані модулі можуть бути змінені окремо один від одного, при цьому вплив на інші модулі мінімальний. Це дає змогу у подальшому вносити зміни у гру, або додавати нові модулі. Навіть якщо нові механіки потребують взаємодії зі старими, це не буде проблемою, оскільки в спроектованих модулях будуть реалізовані відкриті ресурси та методи для зовнішнього доступу.

1.5 Проектування модифікацій геймплею

Окремо потрібно зупинитись на проектуванні модифікацій геймплею, що є темою даного дипломного проекту. Модифікувати геймплей змійки досить складно, оскільки оригінальний ігровий задум має досить стислі правила. Додавання значних змін будуть вносити зміни і в самий жанр, перетворюючи гру жанру Змійка в жанр Екшн – якщо це пов’язано з активним геймплеєм – або в

жанр стратегії – якщо зміни вносяться в економічно-рахункову сферу. Натомість, для збереження ігрового жанру потрібно створити такі модифікації, які б не сильно змінювали оригінальний жанр, але вносили деякі зміни у геймплей.

Модифікації геймплею будуть виконуватись у вигляді бонусів, що будуть збиратись гравцем. Бонуси будуть мати декілька особливостей, що будуть урізноманітнювати кожну ігрову сесію, а саме:

- Бонуси будуть генеруватись на рівні випадковим чином, коли гравець буде «з'їдати» їжу;
- Позиція бонусів буде завжди випадковою;
- Для активації бонусу його потрібно буде також «з'їсти»;
- Кількість бонусів не є сталою, тобто гравець може отримувати доступ до більше ніж одного бонусу у одиницю часу.

Всього для поточної версії гри було передбачено шість видів бонусів, кожний з котрих дає різні модифікації для гравця. Кожний бонус буде мати різний зовнішній вигляд, щоби гравець розумів, яка модифікація його очікує. Нижче перераховані бонуси за видом:

- Бонуси, що збільшують швидкість;
- Бонуси, що зменшують швидкість;
- Бонуси, що додають 10 очок до ігрового рахунку;
- Бонуси, що додають 50 очок до ігрового рахунку;
- Бонуси, що додають 100 очок до ігрового рахунку;
- Бонуси, що додають кущ на ігровому полі.

Слід зазначити декілька додаткових особливостей. Бонус, який додає до ігрового поля кущ виконується миттєво, тому він не має свого ігрового об'єкту і не генерується на ігровому полі. Ще однією особливістю є різна ймовірність випадіння деяких бонусів. Наприклад бонус, який додає 100 очок до ігрового рахунку має шанс випадіння 4%. Використання такого підходу буде гарантувати цікавий та неповторний ігровий процес від однієї ігрової сесії, до іншої. Нижче у таблиця 1.1 приведено шанси випадіння бонусів.

Таблиця 1.1. Шанси випадіння бонусів гри

№ з/п	*Тип бонусу*	*Шанс випадіння*	*Тип активації*
1.	Збільшення швидкості	9%	При взаємодії з гравцем
2.	Зменшення швидкості	4%	При взаємодії з гравцем
3.	Додати 10 очок ігрового рахунку	14%	При взаємодії з гравцем
4.	Додати 50 очок ігрового рахунку	9%	При взаємодії з гравцем
5.	Додати 100 очок ігрового рахунку	4%	При взаємодії з гравцем
6.	Додати 1 кущ на ігрове поле	7%	Автоматично

Як видно з таблиці 1.1, шанси випадіння бонусів досить невеликі. Це визвано намаганням створити баланс у ігровому процесі, щоби бонуси не заважали гравцю отримувати класичний ігровий досвід Змійки. Втім, у разі необхідності величини шансів випадіння тих чи інших бонусів можна досить просто змінити у коді гри. У майбутньому можливо додавати й інші модифікації ігрового процесу у вигляді бонусів, оскільки модулі які відповідають за бонуси досить гнучкі до змін.

1.6 Реалізація основних елементів гри

1.6.1 Створення та налаштування проекту

Для початку роботи по розробці ігрового проекту необхідно створити проект. У різних ігрових програмних рушіях процес створення проектів ігор має різний механізм. Є версії ігрових програмних рушіїв, де проекти створюються з шаблону файлів, в інших робота з проектом починається відразу у рушії. У ігрового програмного рушія Unity для цього є спеціальне програмне забезпечення, яке дає змогу створювати проекти та централізовано виконувати їх менеджмент, налаштування, інсталювати різні версії рушія та інше. Функціонал встановлення різних версій ігрового рушія потрібен для роботи у різних проектах з різними версіями, що є актуально для компаній, які проводять довготривалу підтримку своїх ігрових проектів. Налаштування проектів складається з параметрів використання хмарного сервісу Unity та обрання системи контролю версій. Для

поточного проекту обидві функції будуть вимкнені, оскільки розробка гри буде проходити на одному пристрої. Програмне забезпечення для створення проектів називається Unity Hub і саме в ньому буде створено новий проект. Процес створення проекту можна побачити на рисунку 1.7.

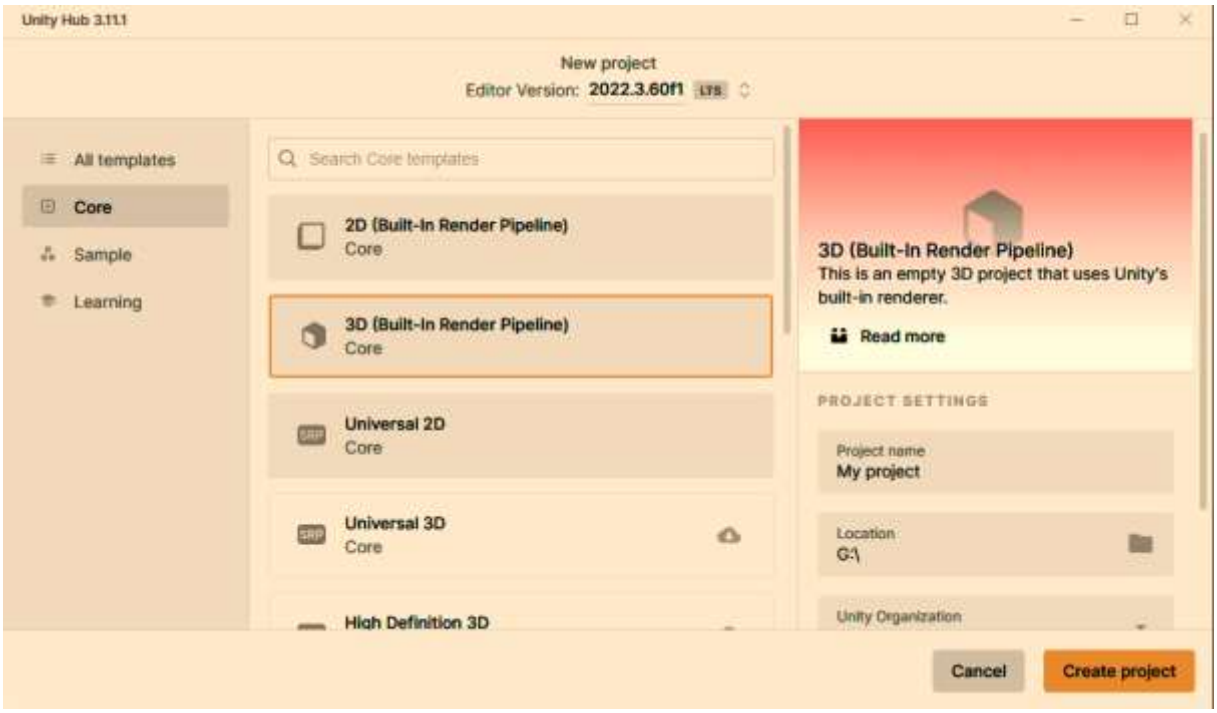


Рисунок 1.7. Вікно створення проекту у ігровому програмному рушії Unity

При створені проектів потрібно визначити із тим, який шаблон проекту використовувати. Ігровий програмний рушій дає деякий набір початкових шаблонів. Ці шаблони мають вже завантажені моделі, ігрові об’єкти, а також налаштування, які більш підходять для тієї чи іншої гри. Використання шаблонів є корисним, оскільки дає змогу використати вже створений базис та не витрачати час на його створення. Додаткові шаблони завжди можна знайти на офіційному сайті ігрового програмного рушія Unity.

Після введення даних проекту та натискання кнопки Create Project, Unity Hub власноруч виконає побудову проекту, створить всі необхідні папки та пакети, а також завантажить всі ассети, які були обрані початковим шаблоном. Слід зазначити, що ігровий програмний рушій Unity, як проект складається з двох основних каталогів. Перший каталог це Packages, якій зберігає в собі дані рушія, файли та бібліотеки, допоміжні файли та тимчасові файли, які є продуктом роботи

рушія під час редагування проекту. Другий каталог це Assets, який є основним робочім каталогом розробника, оскільки саме в нього завантажуються всі файли графіки, матеріалів, шейдерів, спрайтів, скриптів та інше. Середа розробки Unity у якій буде виконуватись робота має вигляд, як зображено на рисунку 1.8.

Рисунок 1.8. Загальний вигляд середи розробки Unity

Налаштування проекту полягають у обрані параметрів управління гравця, середі розробки для скриптів, а також створенню каталогів для ассетів розробляємої гри.

1.6.2 Реалізація графічного матеріалу

Реалізація графічного матеріалу також є важливою частиною процесу розробки, оскільки надає проекту ідентичність. З іншого боку, на початкових етапах роботи допустимо обходитись примітивами 3D-об’єктів, якщо такі використовуються, або спрайтами-заглушками, які будуть слугувати тимчасовими технічним рішенням для відпрацювання роботи коду. Ще одним важливим аспектом створення графічного матеріалу перед розробкою основних елементів гри є бачення проекту на його ранніх етапах, коли реалізовується код та можна бачити, як проект буде виглядати, або працювати в кінці.

Для розробляємої гри потрібно було створити досить велику кількість графічного матеріалу, який можна поділити на два види: Матеріали оточення та

Матеріали ігрових об’єктів.

До матеріалів оточення увійшли спрайти та тайли для відображення рівня, а також неактивних ігрових об’єктів, як кущі. Для побудови рівня, потрібно було створити тайли. Тайли – це невеличкі рисунки, які можуть безшовно поєднуватись один з одним створюючи органічний зовнішній вигляд. Наприклад, поєднувати клітинки з землею та клітинки з водою. Частіше за все тайли використовуються для малювання рівнів. Спочатку створюється великий spritemap, в якому реалізуються всі ймовірні клітинки поверхні локації, після чого частинки цього файлу використовуються у палітрі тайлів. На рисунку 1.9 можна побачити використаний для проекту набір тайлів.

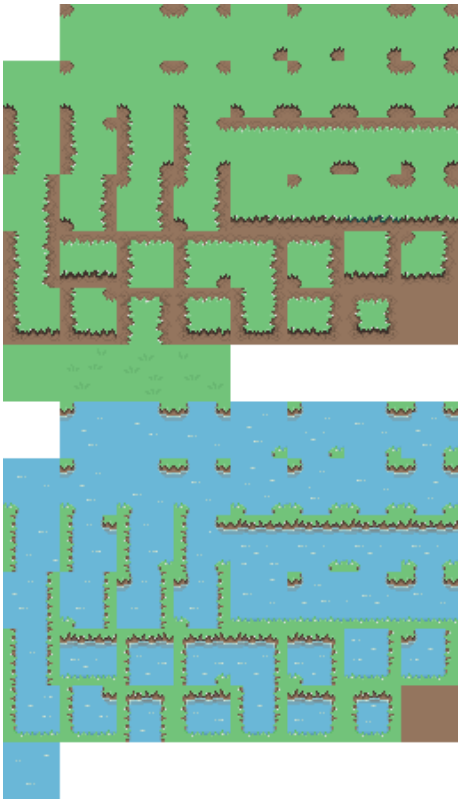


Рисунок 1.9. Набір тайлів, які були використані для гри

З зображеного вище набору тайлів було обрано клітинки поверхні, які будуть використовуватись у проекті. Для поточної версії проекту, рівень буде генеруватись з тайлу трави, узбережжя та води. У подальшому гру можна модифікувати додавши до неї інші тайли, створивши таким чином більш різноманітні поверхні для гри. Разом з тим було створено спрайти кущів для їх розміщення на рівні. Ці спрайти були створені різного зовнішнього вигляду для

того, щоби урізноманітнити картинку для гравця під час ігрової сесії та щоби кущі більш гармонійно вписувались у сцену. На рисунку 1.10 зображено всі створені матеріали оточення для розробляємої гри.

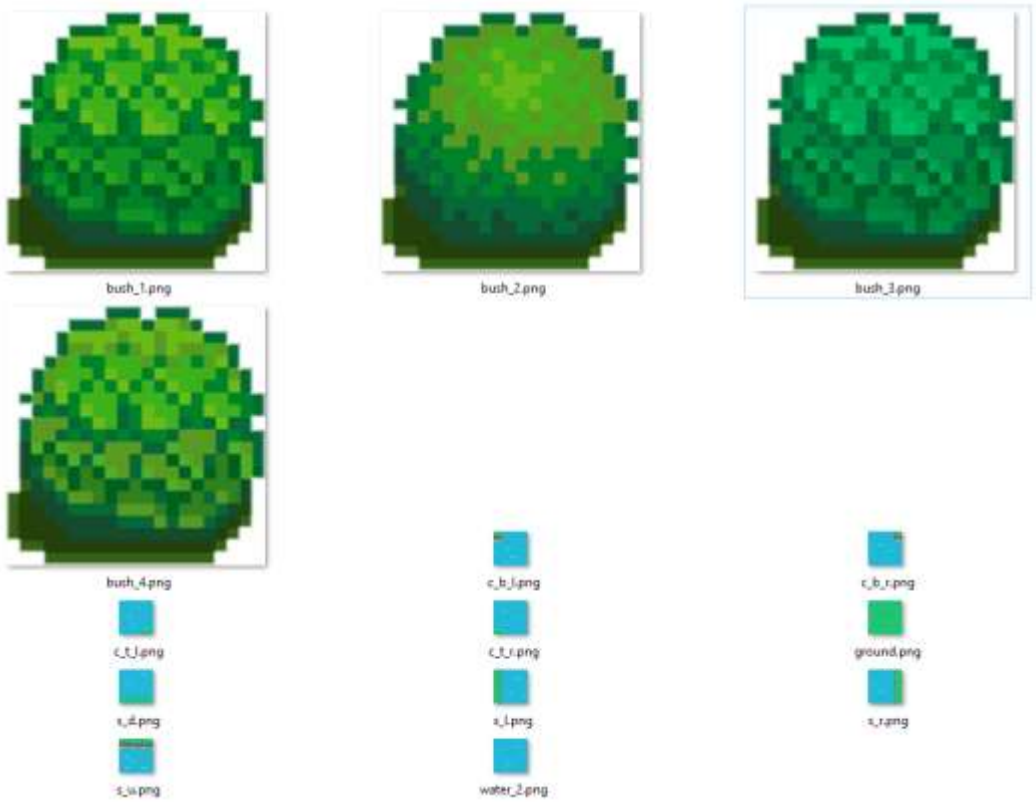


Рисунок 1.10. Створені матеріали оточення для розробляємої гри

На рисунку вище можна побачити, що матеріали мають різний розмір. Це зроблено не випадково. Більш прості поверхні створені у невеликому розмірі, оскільки їх деталізація не грає великої ролі. В той же час кущі, мають досить велику кількість деталей, тому для кращої їх передачі, спрайти кущів створені у великому розмірі. Під час використання гри, спрайти кущів будуть зменшені до розміру тайлів силами ігрового рушія без втрати якості деталізації на відміну від прямого зменшення розміру у графічному редакторі.

До матеріалів ігрових об’єктів входять спрайти, що відображають активні ігрові об’єкти, а саме: голова змійки, частинки її тіла, їжа, бонуси. Їжа була реалізована у декількох варіантах з тією ж ціллю, що і кущі – для урізноманітнення зовнішнього вигляду гри. Бонуси мають власну стилістику, але для кожного виду бонусу створено окремий спрайт, який би передавав суть бонусу. Голова змійки створена у декількох варіантах для того, щоби гравець міг

бачити напрямок руху змійки. Тіло змійки виконано у одному варіанті. Як і з тайлами поверхні, у подальшому можна змінювати та доповнювати спрайти ігрових об’єктів різними варіантами, щоби від одного рівня до іншого стиль гри трохи змінювався. На рисунку 1.11 зображено всі матеріали ігрових об’єктів для розробляємої гри.

Рисунок 1.11. Створені матеріали ігрових об’єктів для розробляємої гри

Також на першому етапі розробки проекту потрібно реалізувати ігрове поле по якому гравець буде переміщуватись під час гри. Це буде легше за все зробити за допомогою панелі інструментів Tile Palette який використовує тайли для їх швидкого розміщення на сцені – альтернативою використання цього інструментарю було б вручну розміщення тайлів на сцені, що займало би дуже багато часу і мало б невелику якість.

У Tile Palette було завантажено створені раніше тайли поверхні землі, налаштовано сітку, згідно з розмірами тайлів, після чого, за допомогою інструменту Brush та режиму роботи Default Brush з’являється можливість «малювати» тайлами на сцені. Висота та площина розміщення тайлів залежить від налаштувань сітки. У разі, якщо потрібно буде додавати другий шар тайлів поверх першого, достатньо буде створити ще один ігровий об’єкт з сіткою, налаштувати її на іншу висоту, після чого за допомогою того ж інструмента Default Brush у Tile

Palette можна наносити шар поверх першого. У випадку із поточною версією проекту це непотрібно, але може знадобитись при додаванні нових рівнів або режимів гри. Важливою рисою тайлів є можливість їх налаштовувати в залежності від необхідності розробника. На рисунку 1.12 зображено створений за допомогою Tile Palette рівень розробляємої гри.

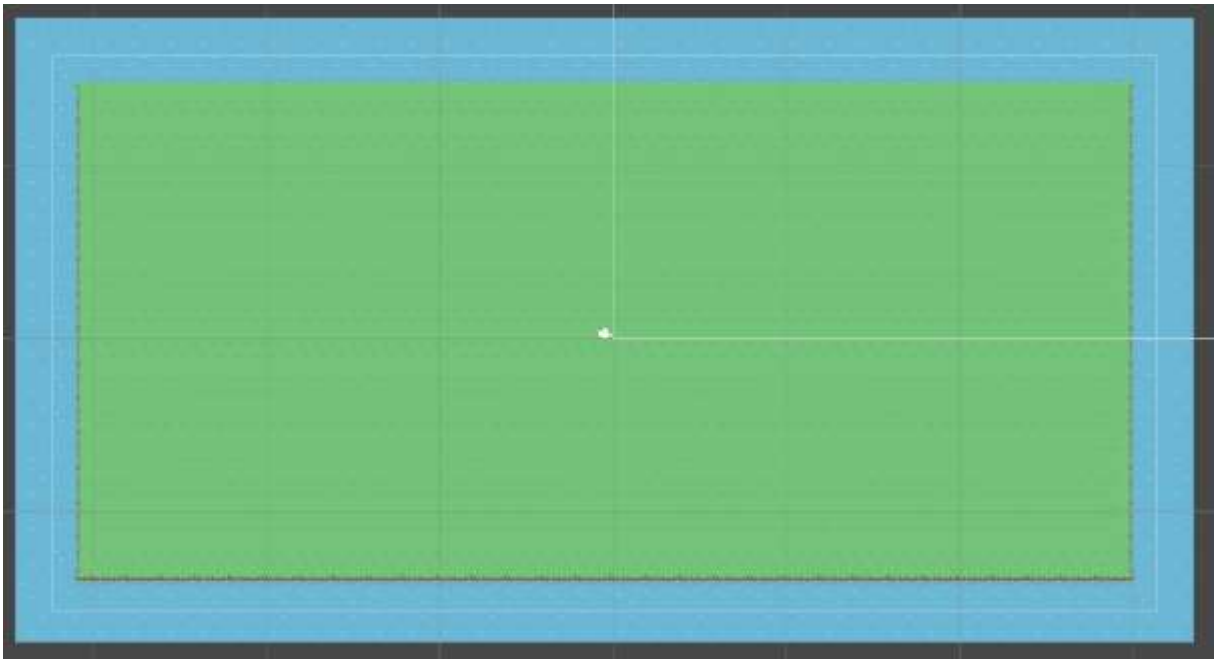


Рисунок 1.12. Створене за допомогою Tile Palette ігрове поле розробляємого проекту

Як можна побачити, ріень буде представляти собою огороджене водою поле, на якому гравцю потрібно буде переміщуватись змійкою та збирати ігровий рахунок.

1.6.3 Реалізація Блоку коду гравця

Наступним кроком у реалізації теми дипломного проекту є створення коду для гравця. Створення основних ігрових елементів перед геймплейними модифікаціями викликано необхідністю створити механіки на які будуть впливати ці модифікації. Першим етапом буде реалізація модулів Блоку коду гравця. Оскільки це основна ігрова механіка, то код гравця потрібно реалізовувати в першу чергу. Окрім того, без реалізованого коду гравця, неможливо буде тестувати роботу інших модулів гри, тому перш за все необхідно виконати його реалізацію. На рисунку 1.6 була зображена загальна структура модулів гри, яка

диктує реалізацію блоку коду гравця. Тому, виходячи з цієї структури реалізовано три модулі.

Одним з основних геймплейних модулів гри є Модуль управління змійкою. Як вже було вказано вище, цей модуль відповідає за зчитування натискання на клавіші управління та реакцію на них. На рисунку 1.13 зображено структуру цього модулю.

Рисунок 1.13. Структура Модуля управління змійкою розробляємої гри

Розробляємі модулі гри будуть мати деякі елементи, які повторюються, оскільки є стандартними у реалізації кодів у ігровому програмному рушії Unity. Одним з таких елементів є блок Ініціалізація змінних та посилань. Рушій Unity використовую систему скриптів, тобто розробник реалізує функційний код, який виконує якісь дії у активного ігрового об’єкту. Самі по собі ці скрипти не виконують ніякої дії, доки не стануть компонентами ігрових об’єктів. Оскільки ці скрипти не існують з моменту створення сцени гри, потрібно виконувати ініціалізацію змін та посилань, особливо, якщо вони посилаються на інші ігрові об’єкти. Окрім того, майже завжди, для виконання дій з ігровими об’єктами потрібно отримувати їх компоненти. Це виконуються за допомогою спеціального методу GetComponent(), який є частиною бібліотеки кодів ігрового рушія Unity.

Нижче проілюстровано приклад такого коду, який отримує посилання на скрипти логіки змійки:

```
Snake_logic _snake_logic;
_snake_logic = GetComponent<Snake_logic>();
```

Компоненти – це складові ігрових об’єктів, саме вони роблять з пустого ігрового об’єкту якийсь функційний ігровий об’єкт. Наприклад якщо до пустого ігрового об’єкту додати компоненти Cube (Mesh Filter), MeshRenderer та Box Collider, то цей ігровий об’єкт стане 3D-кубом. Зміна параметрів компонентів може впливати на ігрові об’єкти. Самим простим прикладом може слугувати зміна кутів нахилу об’єктів у компоненті Transform.

Отже, на рисунку 1.13 у блоці Ініціалізація змінних та посилань виконується першочергова ініціалізація змін для функціонування модулю таких як напрям руху, таймер, швидкість, та встановлення їх значень. Тут же ініціалізація посилань на коди Логіки змійки (Snake_logic) та Логіки частин змійки (Part_logic), а також отримання цих посилань.

Код Update також є стандартним для скриптів Unity. Цей метод виконується кожний кадр, який гра відображає. Через цю ключову властивість, цей метод можна використовувати у разі якщо потрібно перехоплювати натискання на кнопки управління, або виконувати дуже чутливі перевірки. У випадку розглядаємого модулю, цей метод буде отримувати клавіші вводу, та виставляти напрям майбутнього руху змійки. Нижче приведено код, який ілюструє цей процес:

```
if (Input.GetKeyDown(KeyCode.W) && movement_direction != Vector3.down)
    movement_direction = Vector3.up;
if (Input.GetKeyDown(KeyCode.S) && movement_direction != Vector3.up)
    movement_direction = Vector3.down;
if (Input.GetKeyDown(KeyCode.A) && movement_direction != Vector3.right)
    movement_direction = Vector3.left;
if (Input.GetKeyDown(KeyCode.D) && movement_direction != Vector3.left)
    movement_direction = Vector3.right;
```

Тут потрібно відмітити подвійну перевірку. Суть її лежить у тому, що змійка не може змінювати напрямок руху строго на протилежний поточному. Тобто, якщо змійка рухається вверх, то її напрям руху не може змінитись на рух вниз.

Наступний код FixedUpdate відрізняється від попереднього в тому, що ігровий рушій намагається виконати його шістдесят разів у секунду. Саме через цю ключову особливість рух потрібно реалізовувати тільки в цьому методі тому, що розробник може бути впевнений, що цей код буде виконуватись певну кількість разів. В той час як Update виконується не сталу кількість разів, може й шістдесят а може й дві тисячі разів. Через це ігрові об'єкти можуть рухатись від шестидесяти разів до двох тисяч разів на секунду. Особливістю руху змійки в розробляємій грі, є виконання його виконання один раз у визначений час. Нижче приведено цей код:

```
if(_snake_logic.is_snake_alive()){
   _timer += Time.fixedDeltaTime;
   if (_timer > (1f - snake_speed)){
      head_old_position = transform.position;
      transform.position = transform.position + (movement_direction);
      _timer = 0f;
      _part_logic.Change_snake_head_sprite(movement_direction);
      _snake_logic.parts_moving();}
}
```

Спочатку гра перевіряє, чи жива змійка, після чого виконується збільшення таймеру на час, який пройшов між кадрами. Для того щоби змійка рухалась один раз в секунду реального часу, потрібно було б написати перевірку на чи більше значення таймеру за 1f. Втім у реалізовуємій грі швидкість руху змійки залежить не від відстані, яку вона проходить, а від кількості виконання переміщень у секунду. Тому потрібно добавити до перевірки коефіцієнт швидкості змійки (snake_speed), який насправді є коефіцієнтом зменшення порогового часу для виконання переміщення. Початкове значення snake_speed є 0f, тому спочатку змійка рухається на одну клітинку один раз в секунду. Але, якщо значення

snake_speed буде, припустимо 0.5f, тоді за секунду змійка буде рухатись два рази на секунду.

Отже, якщо значення таймеру більше за значення часу, то гра запам’ятовує поточну позицію голови змійки. Після чого змінює позицію змійку на значення напрямку руху. Таймер скидається в нуль. У модуль логіки частин змійки передається команда на зміну спрайту голови змійки, оскільки вона має у поточний кадр відобразити правильний спрайт голови змійки. Після цього виконується рух всіх частинок змійки.

Наступні два блоки коду, Встановлення швидкості змійки та Отримання швидкості змійки, є цілком службовими для роботи зі значенням швидкості змійки.

Модуль логіки змійки структурно більш складний ніш попередній. Це викликано тим, що цей модуль виконує більшість роботи та розрахунків під час гри. Він має зв’язки майже з усіма іншими модулями, та обробляє події роботи з іншими ігровими об’єктами. Структура Модулю логіки змійки зображена на рисунку 1.14.

Рисунок 1.14. Структура Модуля логіки змійки у розробляємій грі

Його робота також починається з блоку ініціалізації. В ньому отримується доступ до інших основних модулів та їх значень, а також встановлюються параметри самої змійки. Серед таких параметрів є статус чи жива змійка, значення кількості життів, ігровий рахунок а також необхідна для перемоги кількість очок. Головним є список ігрових об’єктів частин змійки snake_body_parts. Він потрібний для того, щоби в ході гри отримувати доступ до кожної частинки змійки та виконувати дії з ними, зчитувати їх параметри, тощо. На момент створення, список не має елементів, але оскільки на сцені вже є голова змійки, її потрібно додати до snake_body_parts. Тому в блоці коду ініціалізації виконується додавання до snake_body_parts ігровий об’єкт її голови.

Код add_part() виконує задачу додавання нових частинок змійки за викликом цього методу. Процес додавання нових частинок змійки використовує метод Instantiate, який створює клон ігрового об’єкту. Таким ігровим об’єктом може бути як будь-який ігровий об’єкт на сцені, так і заздалегідь вказаний префаб ігрового об’єкту. Префаби це такі ігрові об’єкти, які розробник може зберігати у каталозі ассетів. Це зручно, коли потрібно динамічно створювати об’єкти. Розробник створює такий об’єкт, налаштовує його, після чого переміщує його до каталогу ассетів. Коли це буде зроблено, цей префаб можна розміщувати на сцені безліч разів, окрім того, кожній такий об’єкт буде мати власті параметри компонентів, томи їх редагування не буде призводити до змін у всіх префабах. Натомість, якщо розмістити префаби на сцені та змінити його компоненти через префаб у каталозі ассетів, тоді ці зміни автоматично будуть внесені до об’єктів на сцені, що є дуже зручним. Тому, для частинок тіла змійки був створений відповідний префаб. Після його генерації, цей префаб відразу записується до snake_body_parts. Після цього цій частинці передаються координати поза зони бачимості для гравця. Це зроблено для того, щоби випадково не спрацьовував тригери перетинів. Нижче приведено розглянутий код:

```
GameObject part = Instantiate(part_prefab);
snake_body_parts.Add(part);
snake_body_parts[snake_body_parts.Count - 1].transform.position = new
```

Vector3(-50, 0, 0);

Код parts_moving() виконує задачу руху частинок змійки. Він викликається після руху голови змійки. Якщо змійка жива, тоді у змінну prev_position записується стара позиція голови змійки. Після цього запускається цикл, який перебирає список snake_body_parts. Для кожного елементу списку, виконується збереження поточної позиції у temp_position, потім цій частинці буде передано значення змінної prev_postion, після чого prev_position буде перезаписано значенням temp_position. Нижче приведено цей код:

```
if (is_alive){
    Vector3 prev_position = _control.head_old_position;
    for (int i = 1; i < snake_body_parts.Count; ++i){
        Vector3 temp_position = snake_body_parts[i].transform.position;
        snake_body_parts[i].transform.position = prev_position;
        prev_position = temp_position;}
}
```

Код тригеру методу OnTriggerEnter2D() виконує завдання перевірки того, з яким ігровим об’єктом взаємодіє гравець. Вид взаємодії у грі може бути тільки один – гравець направив змійку на клітинку з іншим ігровим об’єктом. Створений метод є одним з вбудованих методів у ігровому програмному рушії Unity та викликається в той момент, коли колайдер ігрового об’єкту, який виконує цей код, перетинає колайдер тригерового типу. Collider – це компонент, який утворює границі фізичного тіла ігрового об’єкта до якого він прикріплений. Вони бувають різного типу та форми. Також вони можуть працювати як фізична границя тіла, так і як тригер, утворюючи зону, яка при потраплянні у неї іншого колайдера призведе до виклику методу OnTriggerEnter().

Для розглядаємого методу OnTriggerEnter2D() сутність коду, який був в ньому проста. По суті, цей метод буде отримувати колайдер з яким трапився перетин, далі код буде виконувати перевірки тегів ігрових об’єктів колайдерів з якими був перетин. Теги ігрових об’єктів – це створені розробником «бірки», які можна за необхідністю переглянути. Теги є простим та зрозумілим інструментом

у процесі ідентифікації об’єктів. Кожний з бонусів та активних ігрових об’єктів має свій тег, наприклад їжа – food. В залежності від того, який тег буде збігатись, буде виконуватись різний код. Якщо це їжа, тоді буде отримано ігровий об’єкт їжі, викликано знищення об’єкту їжі яку гравець зібрав. Після цього виконається метод Add_part(). Також буде викликано метод для збільшення ігрового рахунку increase_player_score() з параметром коефіцієнту отриманого рахунку який буде розглянутий нижче. Після цього виконано команду з генерації їжі та виконано кидок на виклик бонусу на рівні.

Метод increase_player_score() виконує роль збільшення ігрового рахунку гравця за допомогою параметра цього метода, якій повинен отримувати коефіцієнт збільшення ігрового рахунку. Після виклику та отримання значення, метод додає до ігрового рахунку це значення. Далі метод буде виконувати перевірку кількості ігрового рахунку у гравця. У разі, якщо ігрового рахунку достатньо для виконання умови, тоді цей метод збільшує швидкість гравця чи його кількість життів. Тут же виконується код, у разі якщо гравець зібрав необхідну для перемоги суму ігрового рахунку, буде виведено повідомлення та гра буде питати перезавантаження. Якщо ця умова не виконана, тоді код передає значення ігрового рахунку у інтерфейс гравця через метод UI_processor.

Останні два методи виконують прості службові ролі, як перевірка на статус, чи є в змійки ще життя та метод для підрахунку кількості рахунку необхідного для перемоги.

На рисунку 1.15 можна побачити структуру модуля Логіки частин змійки Part_logic. Як видно з рисунку нижче, цей модуль дуже невеликий, але він має великий потенціал у майбутньому.

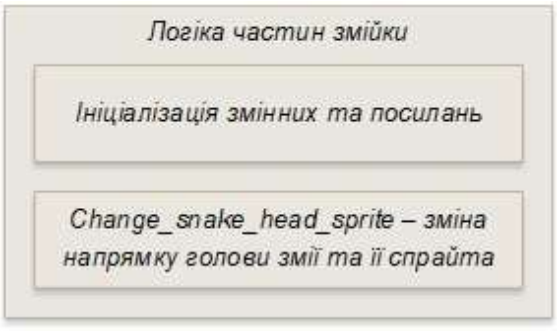


Рисунок 1.15. Структура Модуля логіка частин змійки у розробляємій грі

Головною метою цього методу є виконання розрахунків, які мають відношення до частинок змійки. Потенціал цього метода лежить у можливості в майбутньому додавати різні варіації тіл змійки, що допоможе розвивати та підтримувати проект. Ініціалізація змінних складається з отримання посилань на варіації голови змійки у вигляді спрайтів, отримання посилання на компонент Sprite_Renderer(). Також тут виконується ініціалізація напрямку голови змійки при завантажені гри.

Єдиний розглядаємий метод є Change_snake_head_sprite(), який отримує як параметр напрямок руху голови змійки. Після цього виконується перевірка цього параметра на те, в якому напрямку вона має рухатись та до неї виконується зміна спрайту ігрового об’єкта. Оновлення спрайту виконується кожний раз, коли змійка має виконати переміщення, що гарантує коректну зміну відображення голови змійки в залежності від команд гравця. Після реалізації цих кодів можна спробувати запустити гру та виконати управління змійкою. На донному етапі вже можна отримати загальне уявлення про ігровий процес.

Створені компоненти додаються до ігрового об’єкту голови змійки, яка буде грати роль центрального хабу, який буде керувати всією логікою змійки в грі.

1.6.4 Реалізація коду для об’єктів їжі та кущів

Для подальшої реалізації основних ігрових елементів, потрібно створити код, який буде забезпечувати генерацію та розміщення ігрових об’єктів оточення, як активного, з яким можна взаємодіяти, так і пасивного, яке статично знаходиться на рівні. Це забезпечить грі елемент випадковості, що позитивно сприятиме на враження гравця від ігрового процесу, оскілки такий підхід створить реіграбельність не тільки за рахунок простого ігрового процесу, але й за рахунок створення нових унікальних ігрових ситуацій. Серед ігрових об’єктів що можна назвати активними, виділяються об’єкти їжі та кущів. З їжею гравець може взаємодіяти, направляючи змійку на клітинку з нею. Оскільки ігровим процесом продиктовано необхідність розміщення їжі на рівні порційно по одній клітинці у різних місцях, то необхідно реалізувати такий код, що буде забезпечувати цей механізм. На рисунку 1.16 зображено структуру модулю Параметрів їжі.

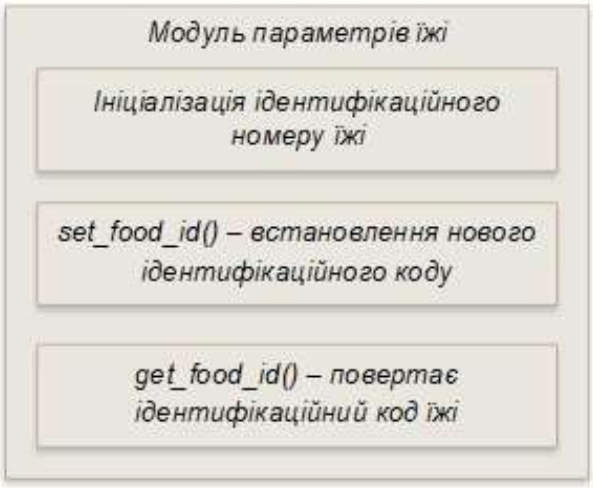


Рисунок 1.16. Структура Модуля параметри їжі для розробляємої гри

Для створених клітинок їжі, потрібно реалізувати її параметри, оскільки кожна нова клітинка буде мати свій ідентифікатор – порядковий номер. Це потрібно для того, щоби мати змогу точно видаляти ігровий об’єкт їжі зі сцени. Також, наявність ідентифікатору їжі дає змогу використовувати списки об’єктів їжі. У поточній версії гри кількість їжі буде сталою – одна клітинка, але у майбутньому, реалізований модуль дасть змогу створювати умови для розміщення більшої кількості клітинок з їжею. Розглянемо структуру модуля.

Блок ініціалізації створює змінну ідентифікаційного коду їжі та встановлює його значення у нуль. Метод set_food_id() з публічним модифікатором доступу дає змогу ззовні редагувати це значення для клітинки з їжею. Метод get_food_id() з публічним доступом дозволяє отримувати ідентифікаційний код клітинки їжі ззовні.

Наступним кодом для реалізації є безпосередньо Модуль генерації їжі. На рисунку 1.17 можна побачити структуру цього модуля. Він відповідає за створення клонів від префабів їжі, розміщення клонів на ігровому рівні, генерацію випадкових координат на ігровому рівні для розміщення їжі, а також за її знищення з ігрового поля у той момент, коли гравець стане на клітинку з їжею головою змійки. Цей модуль на відміну від попереднього має більш складну структуру та велику кількість зв’язків ззовні, які забезпечують коректну роботу інших модулів, що генерують об’єкти на рівні. Частіше за все цей зв’язок потрібен для перевірки існування об’єктів на ігровому рівні та їх позиції.

Рисунок 1.17. Структура Модуля генерації їжі для розробляємої гри

Як і будь-який модуль, цей починається зі створення змінних та посилань а також їх ініціалізації. Серед змінних і посилань можна виділити такі змінні, як координати клітинок їжі (r_x та r_y), кількість їжі на сцені (food_on_map), ідентифікаційний код їжі (food_ids), список клітинок їжі (_food_list) та посилання на компоненти параметрів їжі, генераторів кущів, бонусів, а також спрайти типів їжі. Після створення цих змінних та посилань, гра проводить отримання посилань а також запускає першу генерацію клітинок їжі.

Метод spawn_food() виконує завдання генерації клітинок їжі. Його структура виконання взагалі досить проста при викликанні цього методу, починає працювати цикл, довжина якого дорівнює кількості їжі, яка має бути на сцені. Кожну ітерацію цей цикл генерує клон префабу їжі та додає створений ігровий об’єкт у список клітинок їжі food_list. Після цього створеній клітинці їжі надається ідентифікаційний порядковий номер. Далі виконується завантаження спрайту для клітинки їжі. Оскільки концептом гри було продиктовано генерація різних варіантів зовнішнього вигляду їжі, реалізовано систему випадкового – або псевдовипадкового – обрання типу зовнішнього вигляду. Ця система основана на методи Range() класу Rand, який генерує випадкові значення у вибраних діапазонах. Оскільки всього реалізовано 4 спрайти їжі, то виконується вибірка між значеннями від 0 до 40. У разі випадіння згенерованого числа у межах між 0 та 10,

то обирається перший варіант спрайту їжі. Якщо між 10 та 20, то обирається другий і так далі. Приклад цього коду зображено нижче:

```
int r = Random.Range(0, 40);
if (r > 0 && r < 10)
   _food_sprite_renderer.sprite = food_type_1;
else if (r > 10 && r < 20)
   _food_sprite_renderer.sprite = food_type_2;
else if (r > 20 && r < 30)
   _food_sprite_renderer.sprite = food_type_3;
else if (r > 30 && r < 40)
   _food_sprite_renderer.sprite = food_type_4;
```

Отримавши випадковий зовнішній вигляд, код викликає метод генерації випадкових координат для клітинки їжі, який буде розглянутий пізніше. Після отримання цих координат, гра передає їх до компоненту Transform клітинки їжі. Ідентифікаційний код для клітинок їжі збільшується на одиницю.

Метод rand_food_position() виконує роботу генерації випадкових координат для клітинки їжі. Структура цього коду досить складна, оскільки під час виконання процесу отримання координат, гра повинна слідкувати, щоби нові координати не були вже зайняті іншими ігровими об’єктами на ігровому полі. Саме для цього, кожний ігровий об’єкт знаходиться у відповідних списках List<>. Це механізм мови програмування C#, який дозволяє створювати списки об’єктів різного типу, від значень, до класів та структур. Завдяки цьому, можна створити список ігрових об’єктів та отримувати доступ до їх параметрів за порядковим номером у списку.

Також, код має виконувати пошук позиції до тих пір, поки не буде отримано успіх у цьому процесі, а якщо такий процес не вдалось виконати більш ніж 5 разів, тоді координати їжі будуть встановлені у значення за межами ігрового поля та виведено повідомлення про перемогу, оскільки не залишилось вільних місць. Координати обираються по принципу випадковості як і у випадку із вибором зовнішнього вигляду клітинки їжі, методом Range() класу Rand, але діапазон для

X та Y координат різний. Для X це від -30 до 29, а для Y це від -13 до 14.

Після отримання цих координат гра повинна виконати перевірку чи вільні ці координати. Це виконується за допомогою перебору списків ігрових об'єктів циклом for. У разі співпадіння тригер виходу з циклу отримання координат встановлюється у значення false, що призводить до перезапуску циклую Нижче приведено приклад коду такого циклу:

```
r_x = Random.Range(-30, 29);
r_y = Random.Range(-13, 14);
for (int i = 0; i < _snake_logic.snake_body_parts.Count; ++i){
    if (r_x == _snake_logic.snake_body_parts[i].transform.position.x && r_y
== _snake_logic.snake_body_parts[i].transform.position.y)
        switcher = false;}
```

У прикладі вище, гра перевіряє отримані координати по X та Y з координатами частинок тіла змійки.

Метод destroy_food(int food_id_to_destroy) виконує знищення клітинок з їжею для чого отримає як параметр ідентифікаційний код клітинки їжі на яку натрапив гравець. В тілі цього методу виконується пошук по списку клітинки їжі з таким номером, вона знищується та метод закривається. Як вже було зазначено вище, у поточній версії гри клітинок з їжею тільки одна, але реалізований код дозволяє збільшувати їх кількість.

Ще одним генератором, який потрібно реалізувати є Модуль генерації кущів. Його структура зображена на рисунку 1.18. Цей генератор використовується не часто. Він завжди запускається при створені ігрової сесії та у випадку, якщо гравцю випаде модифікатор геймплею який збільшить кількість кушів. Принцип роботи модулю простий:

- При завантаженні рівня викликається Модуль генерації кушів;
- Під час генерації обираються випадкові координати для кущів;
- Нові кущі розміщуються на рівні та зберігаються у списках;
- При взаємодії з модифікатором на додавання куща знову генерується нова випадкова позиція;

- Розміщення об’єкту на сцені та його збереження у списку кущів.

Рисунок 1.18. Структура Модуля генерації кушів для розробляємої гри

В основному, цей модуль структурно схожий з Модулем генерації їжі. Так само тут виконується створення та ініціалізація змінних для генерації кущів та посилань на зовнішні компоненти або об’єкти. Можна виділити такі змінні як кількість кущів на рівні (bush_on_map), тригер виконується ініціалізація (is_init_spawn), а серед посилань, посилання на варіанти зовнішнього вигляду кушів, посилання на компонент логіки змійки та інші генератори, які зберігають списки відповідних ігрових об’єктів.

Метод spawn_bush() виконує роботу по генерації кущів на рівні. Принцип та послідовність роботи схожа на роботу генератора їжі, але має невеличкі особливості. Так із кожним додаванням куща в ігровий процес, кількість ігрового рахунку для перемоги зменшується, оскільки кущі займають клітинки на яких може знаходитись тіло змійки. Також у цьому методі стоїть перемикач тригера is_init_spawn який виставить його значення у false. Після цього даний метод ніколи не буде викликаний.

Метод add_one_bush() по суті є деякою копією попереднього методу, але особливістю його роботи складається в тому, що він генерує лише один кущ під час свого виконання.

Метод rand_bush_position() схожий на відповідний метод у Модулі генерації їжі, але має корінну відмінність, оскільки виконує різні перевірки у різних ситуаціях. Якщо цей метод викликається при першій генерації кущів, то він перевіряє лише співпадіння з координатами клітинок гравця через те, що при старті гри на рівні будуть існувати лице голова змійки, але списків їжі чи бонусів ще не існує. Тому, інша частина коду перевірок виконується при створення одного куща. Нижче приведений код цих перевірок які також виконуються зі списками відповідних ігрових об'єктів:

```
for (int i = 0; i < _snake_logic.snake_body_parts.Count; ++i){
    if (r_x == _snake_logic.snake_body_parts[i].transform.position.x && r_y == _snake_logic.snake_body_parts[i].transform.position.y)
        switcher = false;}
if(!is_init_spawn){
    for (int i = 0; i < _food_spawner._food_list.Count; ++i){
        if (r_x == _food_spawner._food_list[i].transform.position.x && r_y == _food_spawner._food_list[i].transform.position.y)
            switcher = false;}
    for (int i = 0; i < _bonus_spawner._bonus_list.Count; ++i){
        if (r_x == _bonus_spawner._bonus_list[i].transform.position.x && r_y == _bonus_spawner._bonus_list[i].transform.position.y)
            switcher = false;}
}
```

Перед тестуванням потрібно розмістити ці компоненти на сцені. Для цього потрібно створити пустий ігровий об'єкт та додати до нього реалізований код. Код, який створюється під час розробки ігрових проектів на ігровому програмному рушії Unity, існує лише як файл у ассетах гри. Тому для його роботи потрібно передавати код, як компонент для ігрових об'єктів. Це можна робити перетягуючи код у вікно компонентів ігрових об'єктів, або додаючи код через спеціальну команду у панелі компонентів ігрових об'єктів, чи пункт у меню середи розробки. Після цього передати всі посилання та налаштування префабів і

голови змійки. На рискну 1.19 зображено налаштування ігрового об’єкту GameManager.

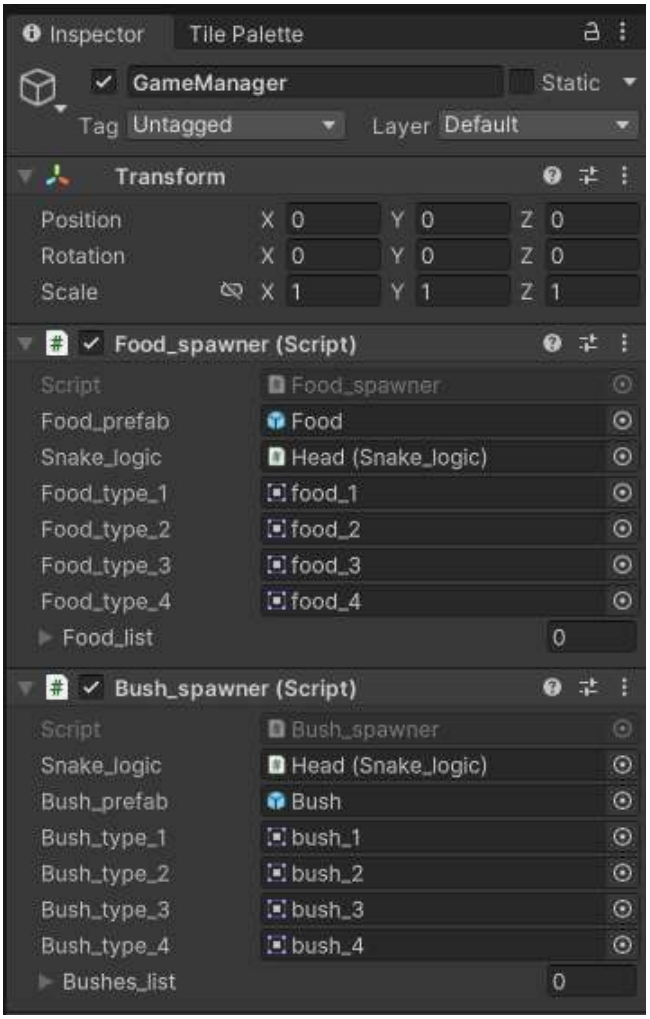


Рисунок 1.19. Налаштування ігрового об’єкту GameManager

Як можна бачити з рисунку 1.19, компоненти після додавання до ігрового об’єкту GameManager мають відповідні поля, які в коді мали публічний доступ, або модифікатор [SerializeField], який не робить змінну чи посилання публічним, але робить його як окреме поле у інтерфейсі ігрового програмного рушія Unity. Ці поля можна редагувати відповідним чином, наприклад перетаскуючи ігрові об’єкти, які мають ті, чи інші компоненти, або обираючи варіанти спрайтів. Після реалізації цих модулів, гра буде мати ігровий стан та можна буде збирати їжу, уникати кущі та потрохи заповняти рівень новими клітинками змійки. Це допоможе у подальшій реалізації проекту а також тестування її працездатності на даному етапі.

1.7 Реалізація модифікацій геймплею гри

1.7.1 Реалізація модуля параметрів бонусів

В концепті ігрового процесу вказано, що модифікаціями геймплею будуть виступати бонуси, які гравець буде отримувати при виконанні тих, чи інших умов. Якщо розглянути це питання з технічної точки зору, то реалізація цих бонусів буде мати схожі риси з механіками генерації їжі або кущів, та більш за все бонуси схожі на механіку клітинок з їжею. Єдиною відмінною рисою буде моменти генерації бонусів та спосіб їх впливу на геймплей. Окрім того концептом продиктовано використання декількох видів бонусів, які потрібно реалізувати у грі. Виходячи з усього вище зазначеного система бонусів реалізується по схожій структурі модулів – Модуль параметрів бонусів та Модуль генерації бонусів.

Модуль параметрів бонусів виконує роль компоненту, який буде зберігати у ігрового об’єкта бонусу його параметри, а саме величини, що можуть буди змінені у параметрах змійки. На рисунку 1.20 зображено структуру Модулю параметрів бонусів.

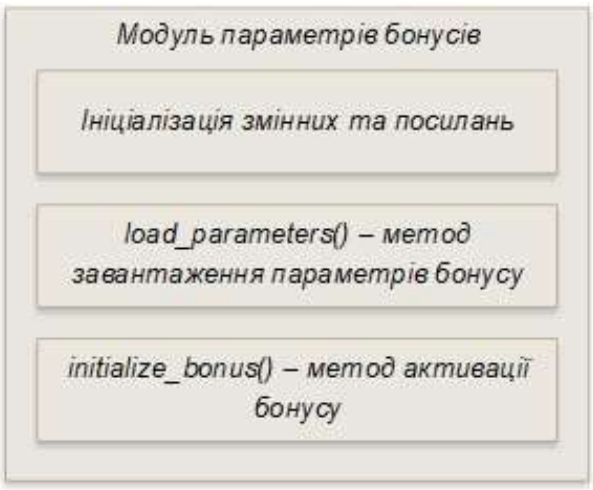


Рисунок 1.20. Структура Модулю параметрів бонусів у розробляємій грі

У частині ініціалізації модуль створює змінні модифікацій таких як кількість очок для гравця, значення збільшення, або зменшення швидкості гравця. Окремим і важливим є змінна ідентифікаційного коду бонусу. Всі ці змінні за замовченням мають нульове значення. Вони відіграють важливу роль у ігровому процесі, тому їх створення є важливою умовою, тому вони мають бути

проініціалізовані нулями, щоби середа виконання не видавала помилки під час ігрового процесу.

Метод load_parameters(string bonus_type) виконує роль запису даних у бонуси. Цей метод викликається у разі якщо випало створити бонус на рівні. Тоді, в залежності від рядка записаного у bonus_type, створені раніше змінні будуть перезаписані відповідним чином. Цей процес реалізований за допомогою оператора switch, який переглядає значення рядка bonus_type та перерахованих у тілі перемикача варіантів.

Метод initialize_bonus(GameObject snake_go) виконується у разі взаємодії голови змійки з ігровим об’єктом бонусу. Тоді змійка зчитує ігровий об’єкт з колайдера, звертається до компоненту параметрів бонусу та викликає цей метод при цьому передаючи свій ігровий об’єкт як аргумент. Метод послідовно виконує отримання посилань на логіку змійки та код, який відповідає за переміщення, після чого викликає методи для зміни показників змійки. В кінці виконання цього методу, ігровий об’єкт бонусу знищується. Нижче приведений код цього методу:

```
public void initialize_bonus(GameObject snake_go){
    Snake_logic _snake_logic = snake_go.GetComponent<Snake_logic>();
    Control _control = snake_go.GetComponent<Control>();
    _snake_logic.increase_player_score(score_to_add);
    _control.set_snake_speed(speed_to_increase);
    _control.set_snake_speed(speed_to_decrease);
    Destroy(gameObject);}
```

1.7.2 Реалізація модуля генерації бонусів

Модуль генерації бонусів є досить складним, оскільки включає в собі переліки бонусів, а також складну систему відстежування бонусів на ігровому полі. Ця система є важливою частиною, оскільки кількість бонусів на ігровому полі не є сталою і може бути різною, приймаючи великі значення у залежності від частоти «успішних» викликів бонусів. По причині того, що бонуси мають різні параметри, потрібно точно знати, який бонус має активуватись, які параметри йому потрібно передавати, а головне, перелік бонусів завжди повинен бути

доступний для гри, оскільки вони розміщуються на ігровому полі, тому на їх місце може бути розміщений кущ або їжа, якщо не реалізувати цей перелік, по якому будуть виконувати свої перевірки інші генератори об’єктів на ігровому полі. На рисунку 1.21 можна побачити структуру Модуля генерації бонусів.

Рисунок 1.21. Структура Модулю генерації бонусів для розробляємої гри

З рисунку 1.21 можна побачити, що цей модуль має найбільшу кількість методів, серед інших модулів гри, яка може бути порівняною з Модулем логіки змійки. Так відбувається через те, що бонуси впливають на геймплей, а також на інші ігрові об’єкти, тому цей модуль має велику кількість перевірок та процесів встановлення значень в залежності від перевірок.

У частині ініціалізації цей модуль створює посилання на всі генератори гри, на логіку змійки, префаб бонусу, а також перелік спрайтів для бонусів у кількості 5 штук. Також тут виконується ініціалізація списку ігрових об’єктів бонусів з публічним доступом для реалізації перевірок на розміщення бонусів на ігровому полі. З боку змінних в цій частині створюються змінні для координат бонусів які

генерується а також змінна rand_throw. При створенні цього компоненту виконується отримання всіх необхідних посилань.

Метод throw_random() відіграє роль віртуального кубика, який викликається у момент, коли гравець з'їдає їжу. Цей метод отримує випадкове значення у діапазоні від 0 до 100 та реалізує випадковість генерації бонусів на рівні за шансами, які описані в таблиці 1.1. Нижче приведений частина коду цього методу:

```
rand_throw = Random.Range(0, 100);
if (rand_throw > 95 && rand_throw < 100){
   spawn_bonus("speeddown");}
else if (rand_throw > 0 && rand_throw < 5){
   spawn_bonus("speedup");}
```

І так далі до кінця переліку ймовірних ігрових бонусів. Якщо випавше у змінній rand_throw значення підпадає під один з діапазонів, тоді оператор перевірки викликає наступний метод.

Метод spawn_bonus(string bonus_type) виконує генерацію бонусів та їх розміщення на ігровому полі. Як аргумент він отримує рядок типу бонусу, який потрібно створити. Тому в тілі цього методу викликається switch, який перебирає свою бібліотеку бонусів. В залежності від того, який бонус був викликаний, цей метод виконує різні дії, спрямовані на створення ігрового об'єкту бонусу через префаб, додавання його у список бонусів, надання бонусу ідентифікаційного номеру, а також налаштування потрібного спрайту бонусу в залежності від його типу. Також цей модуль звертається до методу bonus_placer(bonus) розміщує його на ігровому полі у випадковому місці та передає нові параметри для бонусу. На цьому робота методу завершується. Нижче представлений код однієї з варіацій створення бонусу:

```
case "10_points":
{
      GameObject bonus = Instantiate(bonus_prefab);
      _bonus_list.Add(bonus);
      for(int i=0; i< _bonus_list.Count; ++i){
```

```
Bonus_parameters temp_bonus_parameters = _bonus_list[i].
GetComponent<Bonus_parameters>();
temp_bonus_parameters.bonus_id = i;}
_sprite_renderer = bonus.GetComponent<SpriteRenderer>();
_sprite_renderer.sprite = _10_points;
bonus_placer(bonus);
Bonus_parameters _bonus_parameters = bonus.GetComponent
<Bonus_parameters>();
_bonus_parameters.load_parameters("10_points");
break;}
```

Метод bonus_placer(GameObject bonus_go) виконує розміщення ігрового об’єкту bonus_go який передається до методу як аргумент. Це є ігровий об’єкт генеруємого бонусу. В першу чергу цей методи викликає метод rand_bonus_position() для оновлення координат, після чого встановлює для компонента Transform бонусу нову позицію.

Метод rand_bonus_position() виконує вже відому задачу генерації випадкових координат для ігрового об’єкту бонусу. Ці координати отримуються через клас Rand та перевіряються по всім відомим спискам ігрових об’єктів через цикли, а саме по спискам частинок змійки, кущів та клітинок з їжею. У разі, якщо позиції отримані за допомогою Rand вже зайняті, процес повторюється деяку кількість разів, після чого, якщо не вдається отримати вільне місце, бонус буде розміщено за ігровим полем.

Метод remove_bonus_from_list(int bonus_id_to_destroy) виконує роль видалення активованих бонусів. Для цього використовується параметр bonus_id_to_destroy завдяки якому виконується пошук ідентифікаційного номера бонусу, який був активований у списку бонусів. Знайдений бонус видаляється командою RemoveAt(bonus_id_to_destroy) до якої і передається параметр методу. Таким чином виконується видалення зі списку активованих бонусів, список зсувається після чого ігровий об’єкт бонусу знищується у іншому методі.

Створені два компоненти потрібно додати до вже існуючого ігрового

об’єкту GameManager та налаштувати відповідним чином. Після цього цей ігровий об’єкт буде контролювати ймовірність генерації бонусів у разі взаємодії гравця з їжею на ігровому полі. Оскільки цей ігровий об’єкт є окремим та не має ніякого фізичного втілення, а також вміщує в собі лише компоненти генераторів об’єктів на ігровому полі, він не підлягає знищенню та існує весь цикл ігрової сесії. Використання таких об’єктів є досить зручним прийомом, який використовується під час розробки ігор на ігровому програмному рушії Unity.

1.8 Реалізація ігрового інтерфейсу

Ігровий інтерфейс для гри повинен мати в собі лише декілька полів, а саме ігровий рахунок гравця та кількість його життів. Робота з інтерфейсом у ігровому програмному рушії Unity виконується через ігровий об’єкт типу Canvas, або полотна. Воно існує окремо від ігрової сцени і знаходиться як би понад зображенням, яке транслюється на камеру гравця. На рисунку 1.22 зображено Canvas без елементів інтерфейсу.

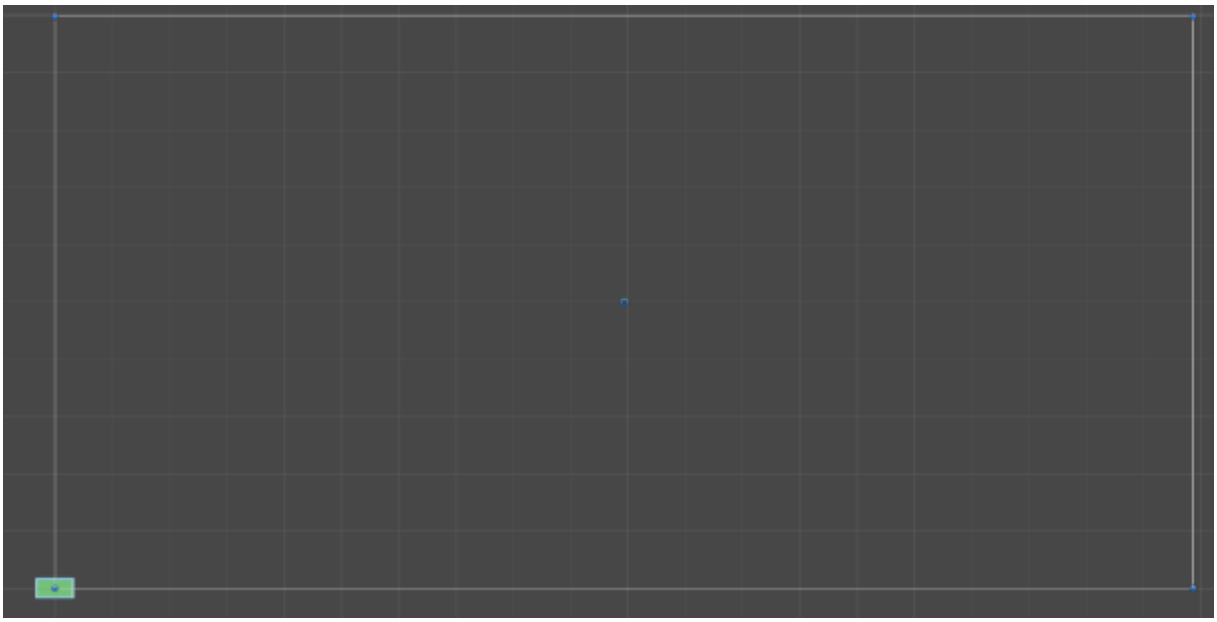


Рисунок 1.22. Ігровий об’єкт Canvas без елементів інтерфейсу

На рисунку 1.22 можна побачити рамку, вона демонструє кордони крану гравця, а також невеликий прямокутник знизу-зліва. Це є ігрове полотно, яка на фоні інтерфейсу дуже мале. Втім, ця різниця в розмірах не впливає на відображення під час гри, оскільки розмір полотна буде підстраюватись під

роздільну здатність екрана гравця.

Для розміщення елементів інтерфейсу потрібно всередині полотна створити елементи інтерфейсу. Для виконання концепту гри, достатньо буде використати елементи UI Text. Слід зазначити, що є два види текстів інтерфейсів у ігровому програмному рушії Unity. Актуальним типом текстів є Text-TextMeshPro. Цей вид текстів має широкі можливості з точки зору редагування представлення текстів та їх деформації. Інший тип є Legacy Text – він має досить прості можливості з редагування і не може деформуватись без використання додаткових інструментів. Перший тип текстів здається більш кращім вибором, але разом з тим, його використання приводить до завантаження до проекту додаткових матеріалів, які роблять гру важчою, навіть якщо ці матеріали не використовуються. Тому у поточному проекті використовуються Legacy Text.

Створивши компоненти інтерфейсу, їх потрібно розмістити на полотні та виконати прив’язку до верхнього лівого кута екрану гравця з метою, якщо гравець буде змінювати роздільну здатність монітора, щоби елементи інтерфейсу коректно відображались на екрані. Після позиціонування, можна набрати текст для строк інтерфейсу. Також для роботи з цими елементами потрібно створити код UI_processor, який буде додано до полотна проекту. Цей код буде мати в собі посилання на текстові елементи інтерфейсу а також два методи:

- change_player_score(int player_score) змінює напис текстового елементу та відображає ігровий рахунок, який передається як аргумент цього методу;
- change_player_lifes(int player_lifes) змінює напис текстового елементу та відображає кількість життів гравця, яка передається як аргумент цього методу.

Код для зміни текстових елементів представлений нижче на прикладі методу для зміни кількості очок гравця:

```
[SerializeField] Text player_score_text;
public void change_player_score(int player_score){
    player_score_text.text = "Рахунок гравця: " + player_score;}
```

Після реалізації інтерфейсу гра повністю задовільнює ігровому концепту розробленому раніше в цьому дипломному проекті. Також, в результаті розробки було сформовано проект, структуру якого можна побачити на рисунку 1.23.

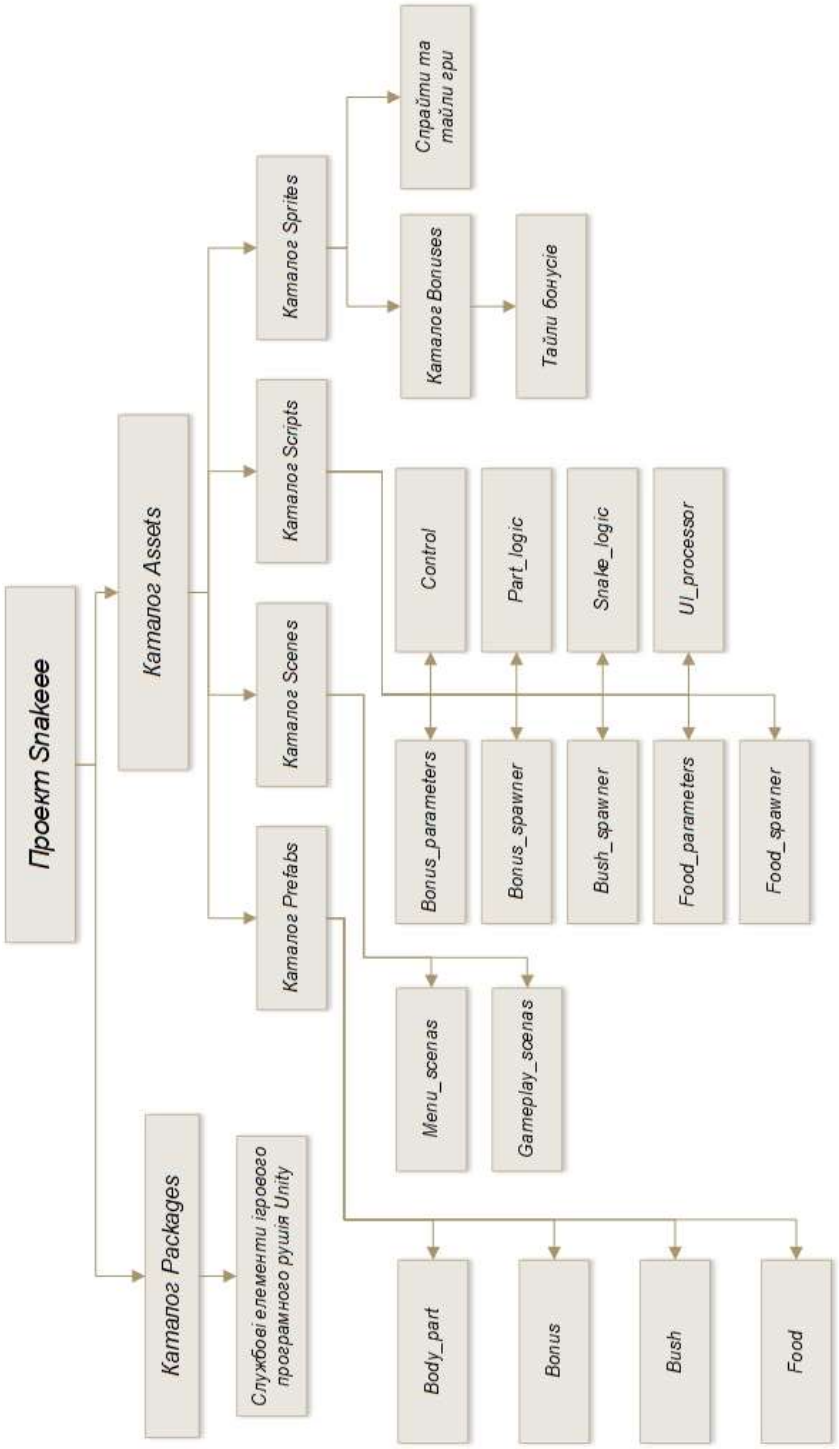


Рисунок 1.23. Структурна схема проекту Snakeee

Як можна побачити з рисунку 1.23 проект вийшов досить великим, тому розділений на відповідні каталоги, кожний з яких відповідає за свою частину проекту гри.

1.9 Тестування розробленої гри

Для якісної реалізації будь-якої гри, її необхідно тестувати та відлагоджувати не тільки під час розробки, а ще й після закінчення кожного етапу розробки. Багато ігрових продуктів продовжують тестувати навіть тоді, коли гра у фінальній версії вже відправилась до печаті, або у магазини. Це робиться для того, щоби якомога швидше виправляти помилки, які не були знайдені під час розробки.

У випадку з реалізацією ігор на ігровому програмному рушії Unity, цей процес виконується безпосередньо в самому редакторі. Для тестування та відладки гри під час розробки, потрібно встановлювати ігрові об’єкти та сцени у такий стан, який потрібно протестувати та натиснути кнопку «Play». Таким же чином виконується тестування і відладка після закінчення розробки, достатньо виставити гру у її початковий стан и натиснути ту ж саму кнопку. Це дуже потужний інструмент, що облегшує роботу та дає змогу тестувати гру не виконуючи її побудову кожний раз, коли потрібно виконати ту, чи іншу перевірку коду на працездатність. На рисунку 1.24, 1.25, 1.26 можна побачити процес тестування гри.

Рисунок 1.24. Процес тестування розробленої гри

Рисунок 1.25. Процес тестування розробленої гри

В ході тестування гри було знайдено помилки з відсутніми об’єктами у посиланнях. Це відносилось до посилань на код генерації бонусів у модулях генерації кущів та їжі. Помилка швидка виправлялась додаванням коду отримання відповідного компоненту.

Була отримана помилка з відсутнім спрайтом. Як було встановлено, була допущена помилка в імені спрайту після його заміни зі старого варіанта на новий, що призводило до втрати посилання на спрайт. Ім’я спрайту було встановлено, як потрібно та помилка зникла.

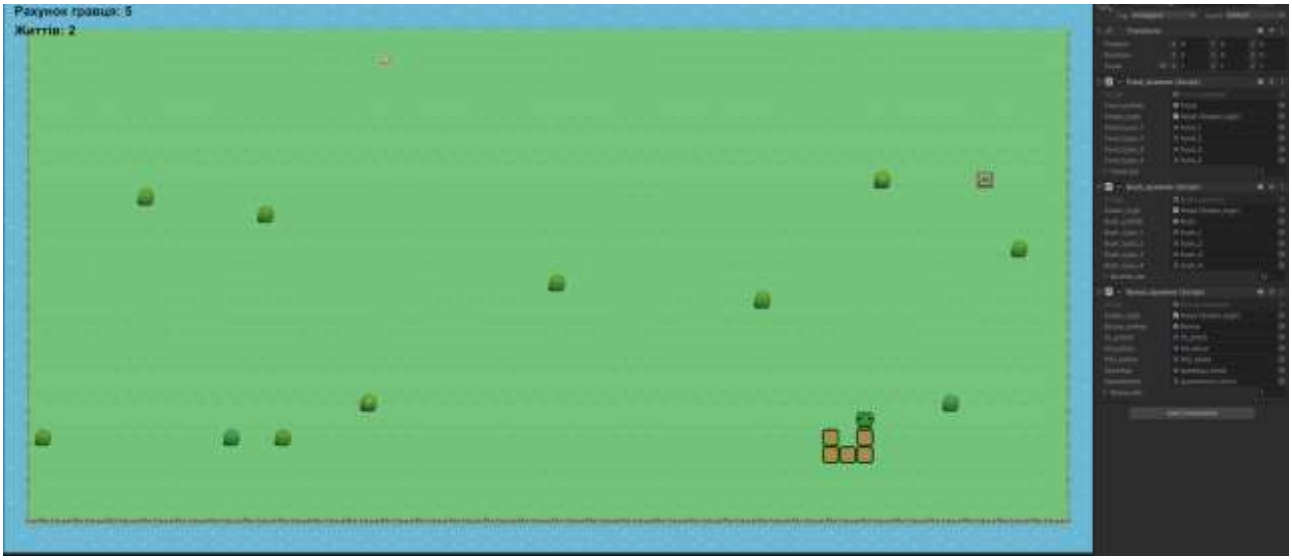


Рисунок 1.26. Процес тестування розробленої гри

Також, результатом тестування було знаходження помилки пов’язаної з невірною реакцією на бонусні очки, оскілки у коді впливу на ігровий процес рахунку було початково прописано збільшення швидкості та життів в залежності від поточного значення ігрового рахунку. Тобто, отримавши значення три, гравець отримував збільшення швидкості. Але, якщо до значення три отримати бонус на десять очок, це збільшення швидкості пропускалось. Ця помилка ігрової логіки була виправлення шляхом зміни логіки розрахунку умов на отримання модифікатора швидкості.

Серед іншого, під час розробки було отримано низьку інших помилок, більшість з яких стосується саме посилання на об’єкти. Помилки цього типу викликані через те, що ініціалізація виконується в один і той самий момент у різних скриптах. Але оскільки паралельне виконання неможливе, деякі змінні намагаються отримати доступ до тих змінних, які ще не створені. Тому для уникнення цієї помилки потрібно було створити послідовність ініціалізації змінних у спеціальній панелі рушія Unity, яка дає змогу змінювати порядок ініціалізації скриптів гри.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

Темою мого дипломного проекту була «Розробка гри «Змійка» з додаванням модифікацій у геймплей». В результаті було розроблену варіацію класичної гри Змійка з оригінальними правилами гри, але з модифікаціями до геймплею. Гравець в ході ігрового процесу може отримувати бонуси, які напряму впливають на геймплей. Завдяки використанню випадковій генерації позицій а також шансів випадіння бонусів отримано гру що дає гравцю отримати різний ігровий досвід. У цьому розділі здійснюється аналіз вартості створеного програмного забезпечення.

Ефективність програмного забезпечення формується як його якістю, так і результативністю процесу його створення. Якість ПЗ оцінюється за кількома критеріями, зокрема з точки зору користувача, ефективності використання ресурсів і відповідності визначеним вимогам. Для користувача важливо враховувати витрати на розробку програмного продукту, зокрема обсяг трудових ресурсів і загальну вартість. У цьому розділі здійснюється аналіз вартості створеного програмного забезпечення.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість створення програмного продукту визначається низкою чинників, серед яких – обсяг робіт, рівень трудомісткості, кваліфікація розробників, а також строки, що задаються ринковими умовами. Для оцінки обсягу програмного забезпечення використовується метод структурної аналогії, згідно з яким на основі спеціалізованих каталогів аналогів визначається кількість умовних машинних команд для подібного програмного продукту (у тисячах команд).

Таблиця 2.1. Каталог аналогів

Найменування ПП	*Обсяг функції ПП – V_o, усл. машинних командах.*
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП імітаційного моделювання	1300 – 4200

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПЗ, що містить Vo в умовних машинних командах, трудомісткість визначаємо на основі табл.2.2.

Таблиця.2.2. Таблиця трудомісткості

Обсяг ПЗ, тис.умов.машинних команд	*Норма часу, люд/год*
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначаємо укрупнену норму часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПЗ, тобто в умовах комп'ютера, Кк=0,7÷0,8), для нашого варіанта виділено сірим кольором:

Тар = 229 х 0,8 = 183,2 (люд/годин).

Трудомісткість програмного продукту визначаємо по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.3.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5.).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	*Ступінь новизни*		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	*Ступінь новизни*	*Значення $K_н$*
А	Принципово нове ПЗ	1,75 – 1,2
Б	ПЗ – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПЗ маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПЗ типовими програмами, %	*Значення $K_т$*
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором. Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз}=Ta*L_1*K_н=183,2 *0,12*0,7= 15,38 \text{ (люд/годин)} \quad (2.1)$$

Трудомісткість розробки технічного проекту

$$T_{тп}=Ta*L_2*K_н=183,2 *0,11*0,7 = 14,11 \text{ (люд/годин)} \quad (2.2)$$

Трудомісткість розробки робочого проекту

$$T_{рп}=Ta*L_3*K_н*K_т=183,2 *0,61*0,7*0,6= 46,94 \text{ (люд/годин)} \quad (2.3)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$=2 (стр), розробка ТП $N_{тп}$=28 (стр), розробка

робочого проекту $N_{рп}$=7 (стр), пояснювальна записка відповідно $N_{пз}$ 20 (стр). Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	*Розрахунок, годин.*		
1.ТЗ	$T_{Ртз}$=15,38	$T_{кк}=0,7*N_{тз}$= 0,7*3=2,1	$T_{нк}=0,15*N_{тз}$=0,15*3=0,45
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}=0,7*N_{тп}$=0,7*26=18,2	$T_{нк}=0,15*N_{тп}$=0,15*26=3,9
3.Розробка РП	$T_{Ррп}$= 46,94	$T_{кк}=0,7*N_{рп}$=0,7*6=4,2	$T_{нк}=0,15*N_{рп}$=0,15*6=0,9
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}$= 1,5*21=31,5	$T_{кк}=0,7*N_{тз}$=0,7*21=14,7	$T_{нк}=0,15*N_{пз}$=0,15*21=3,15
Усього, в т.ч.:	155,53		
- на розробку	$\Sigma T_р$=107,93		
- контроль керівника		$\Sigma T_{кк}$=39,2	
-нормоконтроль			$\Sigma T_{нк}$=8,4

2.3 Розрахунок ціни програмного продукту

Для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, загальні витрати на розробку ПЗ. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2025» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року – 8000 гривень; мінімальну погодинну тарифну ставку – 48,00 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	*Трудомісткість робіт, години*	*Погодинна тарифна ставка, грн.*	*Розрахунок, грн.*
1.Розробка ПП	107,93	48,00	5180,64
2.Контроль керівника	39,2	100,00	3920,00
3.Нормоконтроль	8,4	100,00	840,00
Усього	-	-	ΣЗо= 9940,64

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8:

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	*Тип, модель*	*Кількість*	*Ціна одиниці, грн.*	*Вартість, грн.*
Папір	Лист А4	56	5.0	280,00
Разом	-	-	-	$В_{м\,і}$=280,00
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1*280=28,0
Усього				$В_м$=$В_{мі}$+$В_{тр\ з=}$ 308,00

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	*Значення, грн.*	*Формула розрахунку*
1. Матеріали	308,00	$В_м$ (див. табл. 2.8.)
2. Основна заробітна плата	9940,64	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	994,06	$Зд = 0{,}1 \times Зо$ = 9940,64*0,1
4.Відрахування до єдиного фонду соціального внеску	2405,63	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (9940,64+994,06)
5. Накладні витрати	3976,26	$Внак. = 0{,}4 \times Зо$ =0,4 * 9940,64
6. Повна собівартість	17624,59	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 308+9940,64+994,06+2405,63 +3976,26

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$$П=(С_{пов}*Р)/100=(17624,59*10)/100=1762,46 \text{ грн.} \quad (2.4)$$

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_{пов}+П = 17624,59+1762,46 = 19387,05 \text{ грн.} \quad (2.5)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного забезпечення становитиме:

$$Ц_р=Ц_о+ПДВ = 19387,05 + 19387,05 *0.2 = 23264,46 \text{ грн грн.} \quad (2.6)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Забезпечення здорового та безпечного робочого середовища є ключовим завданням керівництва підприємств, установ і організацій. Адміністрація несе відповідальність за впровадження сучасних заходів охорони праці, що мінімізують ризики виникнення травм і сприяють створенню комфортних санітарно-гігієнічних умов. Це, своєю чергою, допомагає запобігти професійним захворюванням і забезпечує сприятливі умови для продуктивної діяльності співробітників.

3.1 Аналіз шкідливих та ризикових факторів

При проведенні паяльних робіт співробітники піддаються впливу низки шкідливих та небезпечних чинників, що виникають при використанні спеціалізованих інструментів. Серед основних факторів ризику слід відзначити:

- роботу з комп’ютерною та електротехнічною апаратурою,
- недостатню освітленість робочої зони,
- психоемоційні навантаження,
- високий рівень шуму,
- недостатню вентиляцію приміщення,
- порушення правил пожежної безпеки тощо.

3.2 Гігієнічні вимоги до виробничого середовища

Для безперебійного, безпечного та якісного виконання паяльних робіт необхідно суворо дотримуватись правил техніки безпеки та організувати робоче місце оптимальним чином. Це означає, що всі інструменти та матеріали для паяння мають бути систематизовано розміщені, а роботи виконувати у заздалегідь підготовлених зонах, де мінімізовано вплив зовнішніх факторів.

Параметри мікроклімату робочої зони повинні відповідати вимогам санітарних норм мікроклімату виробничих приміщень (ДСН 3.3.6.042-99).

Рівень шуму має не перевищувати встановлених норм щодо виробничого шуму, ультразвуку та інфразвуку (ДСН 3.3.6.037-99).

Допустимі показники вібрації на робочих місцях зумовлені державними санітарними нормами загальної та локальної виробничої вібрації (ДСН 3.3.6.039-99).

Вимоги до рівнів електромагнітних полів визначені державними санітарними нормативами і правилами, затвердженими наказом МОЗ України від 18.12.2002 № 476.

3.3 Вимоги до організації робочого місця працівника

Згідно зі ст. 13 Закону України «Про охорону праці» (від 14.10.1992 р. № 2694-XII), роботодавець зобов'язаний забезпечити створення належних умов праці в кожному структурному підрозділі відповідно до чинних нормативно-правових актів та організувати лабораторні дослідження робочого середовища.

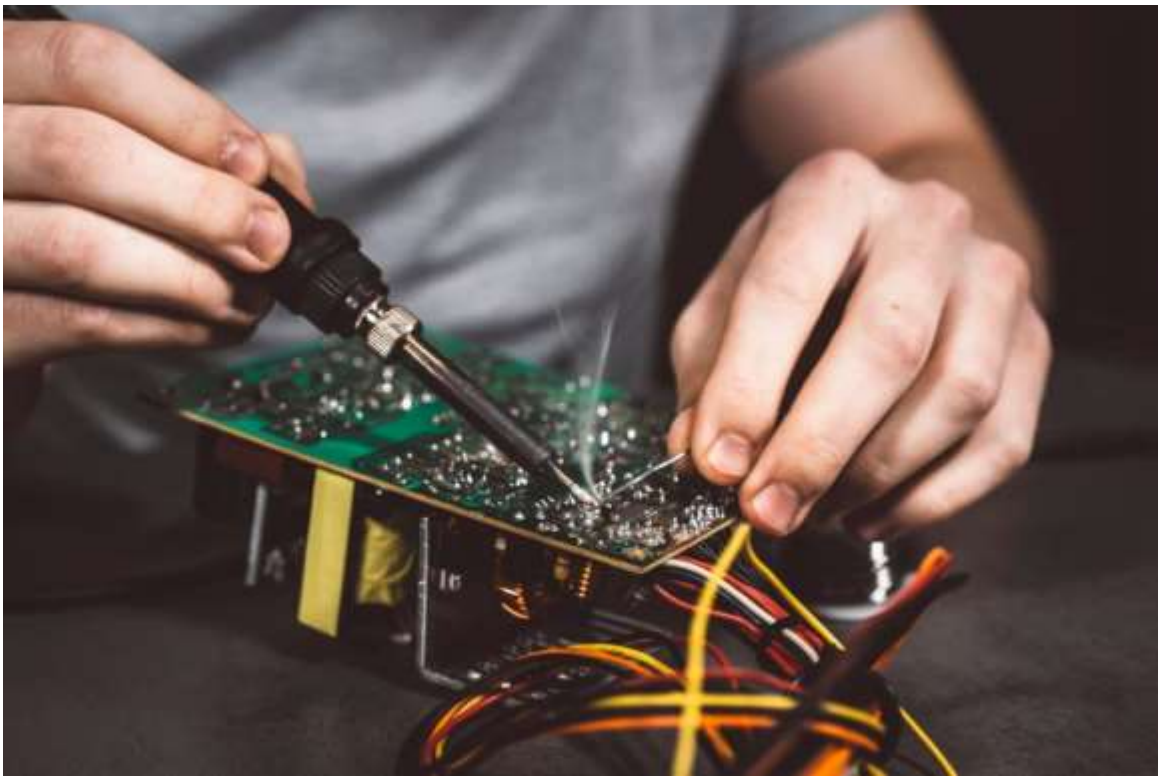


Рисунок 3.1. Процес паяння пристрою

Паяння використовується для з'єднання заготовок зі сталі, кольорових металів і їх сплавів, а також для створення з'єднань із зазначених матеріалів. Найчастіше ця технологія застосовується в електромонтажних роботах, монтажі контрольно-вимірювальних приладів, виробництві радіо- та електроприладів,

створенні теплових обмінників, а також у технологічних процесах, де використовують вироби з армованих пластин з твердих сплавів.

У виробничих приміщеннях концентрація шкідливих речовин не повинна перевищувати гранично допустимих значень, визначених відповідними стандартами (наприклад, ГОСТ 12.1.005-88 «Система стандартів безпеки праці. Загальні санітарно-гігієнічні вимоги до повітря робочої зони»).

Працівники, залучені до паяльних робіт, повинні мати забезпечення засобами індивідуального захисту, а також профілактичними засобами у вигляді захисних кремів, паст чи спеціального лікувально-профілактичного харчування.

Роботодавець повинен організувати:

Організувати проведення попередніх медичних оглядів (при прийнятті на роботу) та регулярних періодичних оглядів відповідно до затвердженого порядку МОЗ України (наказ від 21.05.2007 № 246).

Провести атестацію робочих місць за умовами праці відповідно до встановлених норм (відповідно до постанови Кабінету Міністрів України від 01.08.1992 № 442).

У разі необхідності розробити і впровадити заходи з мінімізації шкідливого впливу виробничих чинників на здоров’я співробітників.

3.4 Електробезпека

Обладнання, таке як персональні комп'ютери, периферійні пристрої, апаратура управління, контрольно-вимірювальні прилади та освітлювальні засоби, а також електропроводи і кабелі, мають відповідати класифікаційним вимогам за зоною застосування та бути обладнаними захисними елементами для запобігання коротким замиканням та іншим аварійним ситуаціям.

Лінія електропостачання для ПК і периферії повинна формувати окрему групову мережу з трьома провідниками: фазовим, робочим нульовим та захисним нульовим. При цьому нульовий захисний провід використовується виключно для заземлення апаратів, а його функціональність не може дублювати робочий нульовий провід. Він прокладається окремо від робочої лінії від групового

розподільника до електроживильних розеток, причому недопустиме підключення обох провідників до одного контактного затискача.

3.4.1 Причини травмування та способи запобігання

Основними причинами травмування електричним струмом є:

- прямий контакт з відкритими проводами,
- взаємодія з внутрішніми компонентами комп'ютера,
- використання несправного обладнання,
- відмова засобів захисту, з якими контактує користувач,
- непередбачене виникнення напруги через пошкодження ізоляції.

Для ефективного запобігання ураження струмом необхідно:

- суворо дотримуватись інструкцій з виконання робіт і правил експлуатації обладнання,
- забезпечувати недоступність частин пристроїв, що працюють під високою напругою, для оператора,
- використовувати високоякісні ізоляційні матеріали, товщина яких відповідає вимогам безпеки,
- підключати електроживлення через спеціально обладнані розетки з функцією занулення,
- розраховувати споживану потужність для запобігання перевантаженням,
- здійснювати надійне заземлення всіх металевих корпусів, доступних для оператора.

3.5 Пожежна безпека

Виробничі приміщення, технологічні установки та будівлі повинні бути обладнані першоджерельними засобами пожежогасіння, до яких належать:

- вогнегасники,
- контейнери з піском,
- негорючі покривала з теплоізоляційного матеріалу,
- високоміцні тканинні вироби тощо.

Ці засоби повинні відповідати нормативним вимогам, затвердженим документами з технологічного проектування та Правилами пожежної безпеки в Україні (НАПБ А.О1.001-2014). Вогнегасники слід встановлювати в легкодоступних, добре помітних місцях (наприклад, в коридорах, біля входів та виходів або у зонах підвищеного ризику виникнення пожежі), захищаючи їх від прямого сонячного випромінювання та впливу опалювальних приладів. Розміщення вогнегасників має забезпечувати їхнє повне відкриття, причому вони встановлюються не вище 1,5 м від підлоги та на безпечній відстані від дверей.

Рисунок 3.2. Засоби пожежогасіння

Також засоби пожежогасіння не повинні заважати евакуації персоналу. Виробничі приміщення повинні забезпечуватись запасними виходами, а двері до них мають бути позначені зрозумілими освітленими написами, наприклад, «Запасний вихід». План евакуації повинен бути розміщений у видному місці біля основного виходу.

ВИСНОВКИ

Темою мого дипломного проекту була «Розробка гри «Змійка» з додаванням модифікацій у геймплей». За мету ставилось реалізувати варіант класичної гри Змійка у відповідному ігровому жанрі. Також потрібно було спроектувати та реалізувати модифікації для геймплею.

Для розробки гри було обрано ігровий програмний рушій Unity, оскільки він більше підходив для реалізації проекту подібного рівня. Для створення та редагування коду обрано середу розробки Visual Studio, яка дає змогу ефективно створювати код. Як графічний редактор використовувалась пробна версія Adobe Photoshop.

На початкових етапах реалізації теми дипломного проекту, було розроблено ігровий концепт, який сформував правила гри та систему модифікацій у геймплей. Під час реалізації гри, в першу чергу створювався графічний матеріал, а саме спрайти та тайли для клітинок ігрового поля. Також було спроектовано та реалізовано основні ігрові механіки та безпосередньо модифікації до геймплею гри.

В результаті було розроблену варіацію класичної гри Змійка з оригінальними правилами гри, але з модифікаціями до геймплею. Гравець в ході ігрового процесу може отримувати бонуси, які напряму впливають на геймплей. Завдяки використанню випадковій генерації позицій а також шансів випадіння бонусів, отримано гру, яка дає змогу гравцю отримати різний ігровий досвід. Завдяки великому ігровому полю, довжина ігрової сесії може бути досить довгою.

Для реалізованого проекту, у майбутньому, можна розробити та реалізувати додаткові бонуси для геймплею, створити різні варіанти змійок або ігрового оточення. Також сам по собі жанр через свою простоту, дозволяю додавати нові режими, як для одного гравця, так і для гри у двох, або гри по мережі Інтернет. Це дасть змогу досить довго підтримувати проект. Його простота також може послугувати для створення порту проекту на Android чи у веб-браузери, для гри через окремий веб-додаток.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ляшенко О.М., Електронний навчальний посібник «Розробка комп'ютерних ігор за допомогою Unity 3D», Херсон: видавництво ФОП Вишемирський В.С., 2018.;
2. Джозеф Хокінг Unity в дії / Джозеф Хокінг., 2018. – 335 с.;
3. Дейбук В.Г. Віртуальний електронний практикум: Навчальний посібник – Чернівці: Чернівецький нац. ун-т, 2021. – 188 с.;
4. Трофименко О.Г. С++. Алгоритмізація та програмування: підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. – Одеса : Фенікс, 2019. – 477 с.;
5. Джон Менінг Unity для розробників. Мобільні мультиплатформені ігри / Джон Менінг, Періс Батфилд-Еддісон., 2018. – 352 с.;
6. Мосіюк О.О., Федорчук А.Л. Операційні системи та системне програмування: навчально-методичний посібник. Житомир: Вид-во ЖДУ ім. Івана Франка, 2022. – 76 с.;
7. Stroustrup B. A Tour of C++ (Second Edition). – Addison-Wesley, 2018. – 240 p. – ISBN 978-0-13-499783-4;
8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms – Boston., McGraw-Hill, 2001. – 1082p.;
9. Robert Sedgewick, Kevin Wayne. Algorithms - Addison-Wesley, 2020. – 956 p. – ISBN 978-0321573513;
10. Procedural Generation in Game Design / Тарн Адамс, 2017. – 336 с.;
11. Бібліотека MSDN [Електронний ресурс]. – Режим доступу: URL http://msdn.microsoft.com/ru-ru/library/default.aspx (дата звернення 20.05.2025);
12. Бібліотека Unity [Електронний ресурс]. – Режим доступу: URL https://docs.unity.com/ (дата звернення 20.05.2025).

ДОДАТОК А. Лістинг основних модулів гри мовою C#

Скрипт Snake_logic.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Snake_logic : MonoBehaviour
{
    [SerializeField] UI_processor _ui_processor;
    [SerializeField] Food_spawner _food_spawner;
    [SerializeField] Bonus_spawner _bonus_spawner;
    Control _control;
    public List<GameObject> snake_body_parts = new List<GameObject>();
    [SerializeField] private GameObject part_prefab;
    bool is_alive = true;
    int player_lives = 2;
    int player_score = 0;
    int score_to_win = 1593;
    void Start()
    {
        _control = GetComponent<Control>();
        _ui_processor.change_player_lifes(player_lives);
        snake_body_parts.Add(gameObject);
    }
    public void add_part()
    {
        GameObject part = Instantiate(part_prefab);
        snake_body_parts.Add(part);
        snake_body_parts[snake_body_parts.Count - 1].transform.position = new
Vector3(-50, 0, 0);
    }
    public void parts_moving()
    {
        if (is_alive)
        {
            Vector3 prev_position = _control.head_old_position;
            for (int i = 1; i < snake_body_parts.Count; ++i)
            {
                Vector3 temp_position = snake_body_parts[i].transform.position;
                snake_body_parts[i].transform.position = prev_position;
                prev_position = temp_position;
            }
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "food")
        {
            Food_parameters _food_parameters =
collision.GetComponent<Food_parameters>();
            _food_spawner.destroy_food(_food_parameters.get_food_id());
            add_part();
            increase_player_score(1);

            _food_spawner.spawn_food();
            _bonus_spawner.throw_random();
```

```
            }
            else if (collision.gameObject.tag == "bonus")
            {
                Bonus_parameters                    _bonus_parameter                    =
collision.GetComponent<Bonus_parameters>();
                _bonus_spawner.remove_bonus_from_list(_bonus_parameter.bonus_id);
                _bonus_parameter.initialize_bonus(gameObject);
            }
            else if (collision.gameObject.tag == "borders")
            {
                player_lives -= 1;
                _ui_processor.change_player_lifes(player_lives);
                if(player_lives == 0)
                    is_alive = false;
            }
            else if (collision.gameObject.tag == "bushes")
            {
                player_lives -= 1;
                _ui_processor.change_player_lifes(player_lives);
                if (player_lives == 0)
                    is_alive = false;
            }
            else if (collision.gameObject.tag == "body")
            {
                player_lives -= 1;

                _ui_processor.change_player_lifes(player_lives);
                if (player_lives == 0)
                    is_alive = false;
            }
        }
        public void increase_player_score(int amount)
        {
            player_score += amount;
            if (player_score > 3 && _control.get_snake_speed() < 0.1f)
            {
                _control.set_snake_speed(0.1f);
            }
            else if (player_score > 23 && _control.get_snake_speed() < 0.5f)
            {
                _control.set_snake_speed(0.5f);
            }
            else if (player_score > 35 && _control.get_snake_speed() < 0.6f)
            {
                _control.set_snake_speed(0.6f);
                player_lives += 1;
            }
            else if (player_score > 80 && _control.get_snake_speed() < 0.7f)
            {
                _control.set_snake_speed(0.7f);
                player_lives += 1;
            }
            else if (player_score > 90 && _control.get_snake_speed() < 0.8f)
            {
                _control.set_snake_speed(0.8f);
                player_lives += 1;
            }
            else if (player_score > 100 && _control.get_snake_speed() < 0.9f)
            {
```

```
            _control.set_snake_speed(0.9f);
            player_lives += 1;
        }
        if (player_score == score_to_win)
        _ui_processor.change_player_score(player_score);
    }
    public bool is_snake_alive()
    {
        return is_alive;
    }
    public void change_score_to_win(int amount)
    {
        score_to_win += amount;
    }
}
```

Скрипт Part_logic.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Part_logic : MonoBehaviour
{
    [SerializeField] Sprite snake_head_up;
    [SerializeField] Sprite snake_head_right;
    [SerializeField] Sprite snake_head_down;
    [SerializeField] Sprite snake_head_left;
    SpriteRenderer _sprite_renderer;
    public Vector3 part_direction;
    void Start()
    {
        _sprite_renderer = GetComponent<SpriteRenderer>();
    }
    public void Init_head_direction(Vector3 direction)
    {
        part_direction = direction;
    }
    public void Change_snake_head_sprite(Vector3 movement_direction)
    {
        if (movement_direction == Vector3.up)
            _sprite_renderer.sprite = snake_head_up;
        else if (movement_direction == Vector3.right)
            _sprite_renderer.sprite = snake_head_right;
        else if (movement_direction == Vector3.down)
            _sprite_renderer.sprite = snake_head_down;
        else if (movement_direction == Vector3.left)
            _sprite_renderer.sprite = snake_head_left;
    }
}
```

Скрипт Food_spawner.cs

```
using System.Collections;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Food_spawner : MonoBehaviour
{
    [SerializeField] GameObject food_prefab;
    [SerializeField] Snake_logic _snake_logic;
```

```
        Bush_spawner _bush_spawner;
        Bonus_spawner _bonus_spawner;
        Food_parameters _food_parameters;
        SpriteRenderer _food_sprite_renderer;
        [SerializeField] Sprite food_type_1;
        [SerializeField] Sprite food_type_2;
        [SerializeField] Sprite food_type_3;
        [SerializeField] Sprite food_type_4;
        public List<GameObject> _food_list = new List<GameObject>();
        int food_on_map = 1;
        int food_ids = 0;
        int r_x = 0;
        int r_y = 0;
        private void Start()
        {
            _bush_spawner = GetComponent<Bush_spawner>();
            _bonus_spawner = GetComponent<Bonus_spawner>();
            spawn_food();
        }
        public void spawn_food()
        {
            for (int i = 0; i < food_on_map; ++i)
            {

                GameObject food = Instantiate(food_prefab);
                _food_list.Add(food);
                _food_parameters = _food_list[i].GetComponent<Food_parameters>();
                _food_parameters.set_food_id(food_ids);
                _food_sprite_renderer = food.GetComponent<SpriteRenderer>();
                int r = Random.Range(0, 40);
                if (r > 0 && r < 10)
                    _food_sprite_renderer.sprite = food_type_1;
                else if (r > 10 && r < 20)
                    _food_sprite_renderer.sprite = food_type_2;
                else if (r > 20 && r < 30)
                    _food_sprite_renderer.sprite = food_type_3;
                else if (r > 30 && r < 40)
                    _food_sprite_renderer.sprite = food_type_4;
                rand_food_position();
                Vector3 spawn_food_position = new Vector3(r_x, r_y, -1f);
                food.transform.position = spawn_food_position;
                ++food_ids;
            }

        }
        void rand_food_position()
        {
            bool pass_trigger = false;
            short attempt_count = 0;
            do
            {
                bool switcher = true;
                r_x = Random.Range(-30, 29);
                r_y = Random.Range(-13, 14);
                for (int i = 0; i < _snake_logic.snake_body_parts.Count; ++i)
                {
                    if (r_x == _snake_logic.snake_body_parts[i].transform.position.x &&
r_y == _snake_logic.snake_body_parts[i].transform.position.y)
                        switcher = false;
```

```
                }
                for (int i = 0; i < _bush_spawner.bushes_list.Count; ++i)
                {
                    if (r_x == _bush_spawner.bushes_list[i].transform.position.x && r_y
== _bush_spawner.bushes_list[i].transform.position.y)
                        switcher = false;
                }
                if(_bonus_spawner._bonus_list.Count > 0)
                {
                    for (int i = 0; i < _bonus_spawner._bonus_list.Count; ++i)
                    {
                        if (r_x == _bonus_spawner._bonus_list[i].transform.position.x
&& r_y == _bonus_spawner._bonus_list[i].transform.position.y)
                            switcher = false;
                    }
                }

                ++attempt_count;
                if (attempt_count > 5)
                {
                    r_x = -50;
                    r_y = -50;
                    pass_trigger = true;
                }
                if (switcher)
                    pass_trigger = true;
            } while (pass_trigger != true);
        }
        public void destroy_food(int food_id_to_destroy)
        {
            for (int i = 0; i < _food_list.Count; ++i)
            {
                _food_parameters = _food_list[i].GetComponent<Food_parameters>();
                if(_food_parameters.get_food_id() == food_id_to_destroy)
                {
                    _food_list.RemoveAt(i);
                    Destroy(_food_parameters.gameObject);
                    break;
                }
            }
        }
    }
```

Скрипт Bonus_spawner.cs

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using UnityEngine.XR;
    public class Bonus_spawner : MonoBehaviour
    {
        Bush_spawner _bush_spawner;
        Food_spawner _food_spawner;
        [SerializeField] Snake_logic _snake_logic;
        [SerializeField] GameObject bonus_prefab;
        SpriteRenderer _sprite_renderer;
        [SerializeField] Sprite _10_points;
        [SerializeField] Sprite _50_points;
        [SerializeField] Sprite _100_points;
```

```
        [SerializeField] Sprite _speedup;
        [SerializeField] Sprite _speeddown;
        public List<GameObject> _bonus_list = new List<GameObject>();
        int rand_throw = 0;
        int r_x = 0;
        int r_y = 0;
        void Start()
        {
            _sprite_renderer = GetComponent<SpriteRenderer>();
            _bush_spawner = GetComponent<Bush_spawner>();
            _food_spawner = GetComponent<Food_spawner>();
        }
        public void throw_random()
        {
            rand_throw = Random.Range(0, 100);
            if (rand_throw > 95 && rand_throw < 100)
            {
                spawn_bonus("speeddown");
            }
            else if (rand_throw > 0 && rand_throw < 5)
            {
                spawn_bonus("speedup");
            }
            else if(rand_throw > 5 && rand_throw < 20)
            {
                spawn_bonus("10_points");
            }
            else if(rand_throw > 80 && rand_throw < 90)
            {
                spawn_bonus("50_points");
            }
            else if(rand_throw > 25 && rand_throw < 30)
            {
                spawn_bonus("100_points");
            }
            else if(rand_throw > 70 && rand_throw < 78)
            {
                spawn_bonus("add_bush");
            }
        }
        void spawn_bonus(string bonus_type)
        {
            switch (bonus_type)
            {
                case "10_points":
                    {
                        GameObject bonus = Instantiate(bonus_prefab);
                        _bonus_list.Add(bonus);
                        for(int i=0; i< _bonus_list.Count; ++i)
                        {
                            Bonus_parameters  temp_bonus_parameters  =  _bonus_list[i].
GetComponent<Bonus_parameters>();
                            temp_bonus_parameters.bonus_id = i;
                        }
                        _sprite_renderer = bonus.GetComponent<SpriteRenderer>();
                        _sprite_renderer.sprite = _10_points;
                        bonus_placer(bonus);
                        Bonus_parameters    _bonus_parameters    =    bonus.GetComponent
<Bonus_parameters>();
```

```
                        _bonus_parameters.load_parameters("10_points");
                        break;
                    }
                case "50_points":
                    {
                        GameObject bonus = Instantiate(bonus_prefab);
                        _bonus_list.Add(bonus);
                        for (int i = 0; i < _bonus_list.Count; ++i)
                        {
                            Bonus_parameters  temp_bonus_parameters  =  _bonus_list[i].
GetComponent<Bonus_parameters>();
                            temp_bonus_parameters.bonus_id = i;
                        }
                        _sprite_renderer = bonus.GetComponent<SpriteRenderer>();
                        _sprite_renderer.sprite = _50_points;
                        bonus_placer(bonus);
                        Bonus_parameters    _bonus_parameters    =    bonus.GetComponent
<Bonus_parameters>();
                        _bonus_parameters.load_parameters("50_points");
                        break;
                    }
                case "100_points":
                    {
                        GameObject bonus = Instantiate(bonus_prefab);
                        _bonus_list.Add(bonus);
                        for (int i = 0; i < _bonus_list.Count; ++i)
                        {
                            Bonus_parameters  temp_bonus_parameters  =  _bonus_list[i].
GetComponent<Bonus_parameters>();
                            temp_bonus_parameters.bonus_id = i;
                        }
                        _sprite_renderer = bonus.GetComponent<SpriteRenderer>();
                        _sprite_renderer.sprite = _100_points;
                        bonus_placer(bonus);
                        Bonus_parameters        _bonus_parameters        =        bonus.
GetComponent<Bonus_parameters>();
                        _bonus_parameters.load_parameters("100_points");
                        break;
                    }
                case "speedup":
                    {
                        GameObject bonus = Instantiate(bonus_prefab);
                        _bonus_list.Add(bonus);
                        for (int i = 0; i < _bonus_list.Count; ++i)
                        {
                            Bonus_parameters  temp_bonus_parameters  =  _bonus_list[i].
GetComponent<Bonus_parameters>();
                            temp_bonus_parameters.bonus_id = i;
                        }
                        _sprite_renderer = bonus.GetComponent<SpriteRenderer>();
                        _sprite_renderer.sprite = _speedup;
                        bonus_placer(bonus);
                        Bonus_parameters        _bonus_parameters        =        bonus.
GetComponent<Bonus_parameters>();
                        _bonus_parameters.load_parameters("speedup");
                        break;
                    }
                case "speeddown":
                    {
```

```
                        GameObject bonus = Instantiate(bonus_prefab);
                        _bonus_list.Add(bonus);
                        for (int i = 0; i < _bonus_list.Count; ++i)
                        {
                            Bonus_parameters  temp_bonus_parameters  =  _bonus_list[i].
GetComponent<Bonus_parameters>();
                           temp_bonus_parameters.bonus_id = i;
                        }
                        _sprite_renderer = bonus.GetComponent<SpriteRenderer>();
                        _sprite_renderer.sprite = _speeddown;
                        bonus_placer(bonus);
                        Bonus_parameters    _bonus_parameters    =    bonus.GetComponent
<Bonus_parameters>();
                        _bonus_parameters.load_parameters("speeddown");
                        break;
                    }
                case "add_bush":
                    {
                        _bush_spawner.add_one_bush();
                        break;
                    }
            }
        }

        void bonus_placer(GameObject bonus_go)
        {
            rand_bonus_position();
            Vector3 spawn_bonus_position = new Vector3(r_x, r_y, -1f);
            bonus_go.transform.position = spawn_bonus_position;
        }
        void rand_bonus_position()
        {
            bool pass_trigger = false;
            short attempt_count = 0;
            do
            {
                bool switcher = true;
                r_x = Random.Range(-30, 29);
                r_y = Random.Range(-13, 14);
                for (int i = 0; i < _snake_logic.snake_body_parts.Count; ++i)
                {
                    if (r_x == _snake_logic.snake_body_parts[i].transform.position.x &&
r_y == _snake_logic.snake_body_parts[i].transform.position.y)
                        switcher = false;
                }
                for (int i = 0; i < _bush_spawner.bushes_list.Count; ++i)
                {
                    if (r_x == _bush_spawner.bushes_list[i].transform.position.x && r_y
== _bush_spawner.bushes_list[i].transform.position.y)
                        switcher = false;
                }
                for (int i = 0; i < _food_spawner._food_list.Count; ++i)
                {
                    if (r_x == _food_spawner._food_list[i].transform.position.x && r_y
== _food_spawner._food_list[i].transform.position.y)
                        switcher = false;
                }
                ++attempt_count;
                if (attempt_count > 5)
```

```
                {
                    r_x = -50;
                    r_y = -50;
                    pass_trigger = true;
                }
                if (switcher)
                    pass_trigger = true;
            } while (pass_trigger != true);
        }
        public void remove_bonus_from_list(int bonus_id_to_destroy)
        {
            _bonus_list.RemoveAt(bonus_id_to_destroy);
            for (int i = 0; i < _bonus_list.Count; ++i)
            {
                Bonus_parameters                    _bonus_parameters                    =
_bonus_list[i].GetComponent<Bonus_parameters>();
                _bonus_parameters.bonus_id = i;
            }
        }
    }
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка гри "Змійка" з додаванням модифікацій у геймплей

Дипломний проект студента групи 4КГ-08: Слинявчук Катерини Русланівні

Керівник: Джабраілов Д.В.

Огляд існуючих ігрових рішень

Скріншот ігрового процесу гри «Snake.io»

Скріншот ігрового процесу «Snake Rivals»

Гра-основа з якої формувався ігровий концепт

Скріншот ігрового процесу гри «Snake, snake, snake!»

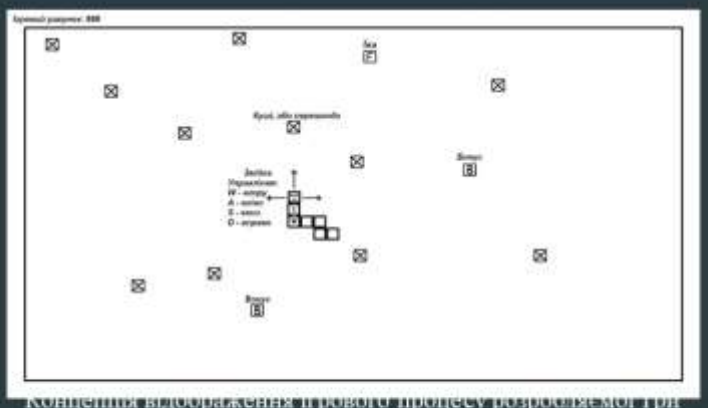
Обрання програмного забезпечення для розробки

Visual Studio

Ps

Unity

Концепція ігрового процесу розробляємої гри

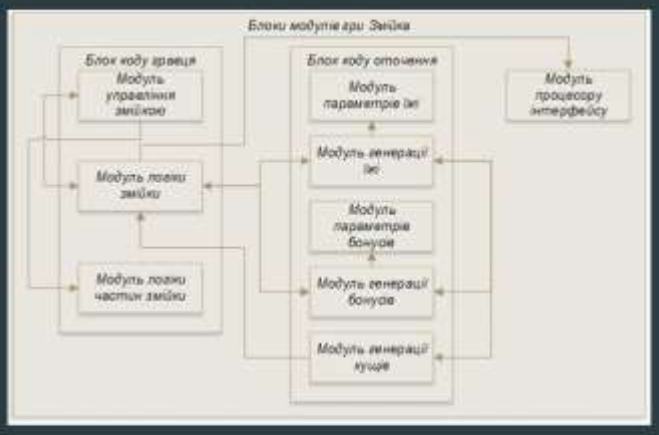


Концепція відображення ігрового процесу розробляємої гри

Структура основних модулів гри

Загальна блок схема структури коду модулів розробляємого проекту

Схема зв’язків між модулями розробляємого проекту

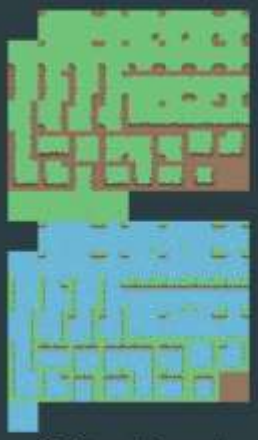


Описання модифікацій геймплею гри

№ з/п	Тип бонусу	Шанс випадіння	Тип активації
1.	Збільшення швидкості	9%	При взаємодії з гравцем
2.	Зменшення швидкості	4%	При взаємодії з гравцем
3.	Додати 10 очок ігрового рахунку	14%	При взаємодії з гравцем
4.	Додати 50 очок ігрового рахунку	9%	При взаємодії з гравцем
5.	Додати 100 очок ігрового рахунку	4%	При взаємодії з гравцем
6	Додати 1 куш на ігрове поле	7%	Автоматично

Шанси випадіння бонусів гри

Реалізація графічної складової гри

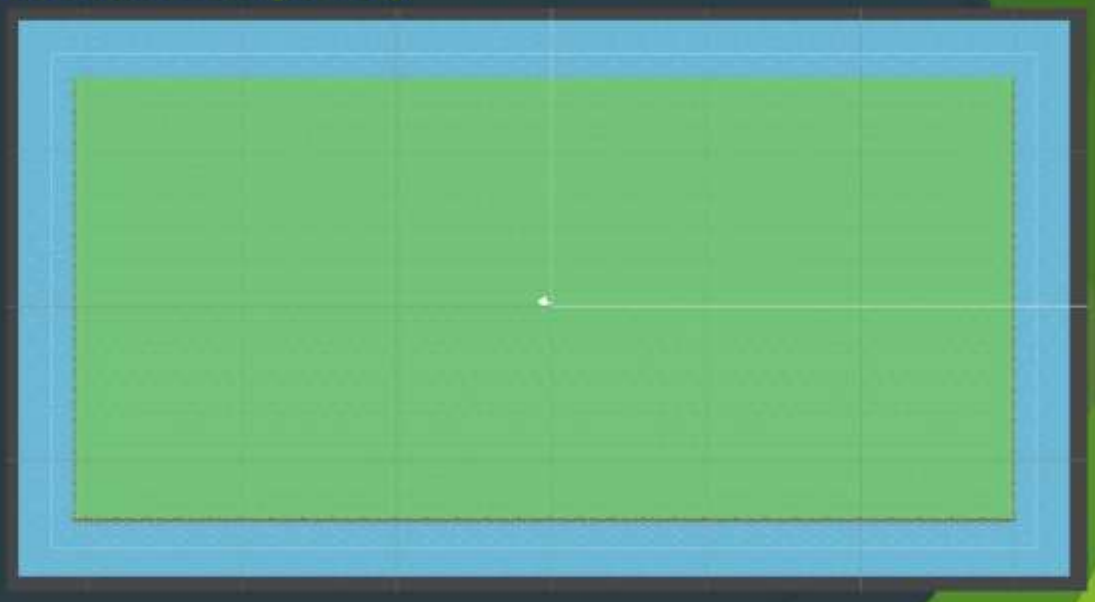


Набір тайлів, які були використані для гри

Створені матеріали оточення для розробляємої гри

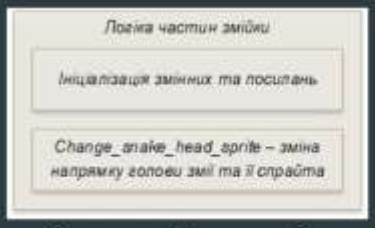
Створені матеріали ігрових об'єктів для розробляємої гри

Створене за допомогою Tile Palette ігрове поле розробляємого проекту



Реалізація основних модулів

Структура Модуля управління змійкою розробляємої гри



Структура Модуля логіка частин змійки у розробляємій грі

Структура Модуля логіки змійки у розробляємій грі

Реалізація модулів пов'язаних з генерацією

Структура Модуля параметри їжі для розробляємої гри

Структура Модуля генерації їжі для розробляємої гри

Структура Модуля генерації кущів для розробляємої гри

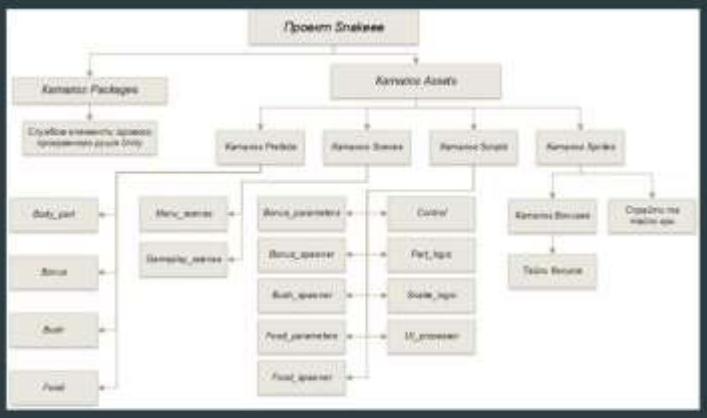
Налаштування ігрового об’єкту GameManager

Реалізація модифікацій геймплею

Структура Модулю параметрів бонусів у розробляємій грі

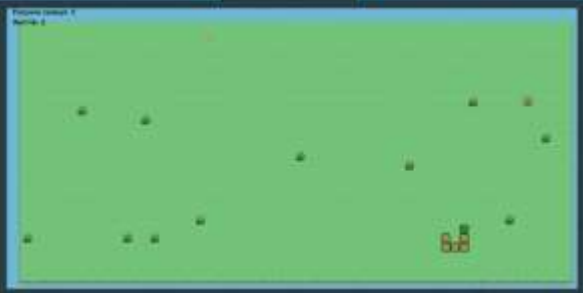
Структура Модулю генерації бонусів для розробляємої гри

Кінцева структура проекту



Структурна схема проекту Snakeee

Тестування та скріншоти реалізованої гри



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Слинявчук Катерина Русланівна

(прізвище, ім`я та по батькові)

Спеціальність 123 " Комп'ютерна інженерія "

Освітня програма «Комп'ютерна графіка і Web-дизайн»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка гри "Змійка" з додаванням модифікацій у геймплей*

Обсяг розрахунково-пояснювальної записки 83 сторінок

Обсяг графічної (презентаційної) частини 16 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity зокрема.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано концепцію гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені роботи з ігровим процесом.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано процес виконання розробки гри;
2. Виконано проектування елементів гри із поясненнями на схемах та за допомогою коду;
3. Класичний дизайн гри оновлено приємними графічними елементами.

д) основні недоліки дипломного проекту (роботи)

1. Прогресія збільшення швидкості досить повільна;
2. Випадковість реалізована через Random є досить передбачуваною.
3. Текст пояснювальної записки недостатньо структурований і не повністю відповідає етапам розробки програмного застосунку, не всі етапи проілюстровано блок-схемами і діаграмами

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Рудніченко Микола Дмитрович

Місце роботи і посада рецензента Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій

Підпис: ICS

«20» червня 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Слинявчук Катерина Русланівна

(прізвище, ім`я та по батькові)

Спеціальність: *123 "Комп'ютерна інженерія"*

Освітньо-професійна програма: *«Комп'ютерна графіка і Web-дизайн»*

Тема дипломного проєкту: *Розробка гри "Змійка" з додаванням модифікацій у геймплей*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проєкт виконано відповідно технічному завданню. Пояснювальна записка містить 83 сторінки. У пояснювальній записці виконано опис етапів розробки комп'ютерної гри, виконано аналіз проектів жанру змійка, виконано концептуалізацію та проектування гри, її графічної частини. Графічна частина складається з 16 слайдів мультимедійної презентації, які також містять графічні матеріали, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини відмінна, розробку виконано в повному обсязі.*

б) самостійність роботи над проєктом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувачка освіти Слинявчук К.Р. поступово та послідовно виконувала всі етапи розробки. Всі роботи здобувачка освіти виконувала самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувачка освіти Слинявчук К.Р. під час роботи над дипломним проєктом вивчила достатню кількість літературних джерел та матеріалів за даною тематикою.*

Вважаю, що теоретична підготовка дипломника добра і вона готова до захисту дипломного проєкту

г) вміння розв'язувати виробничі та конструкторські питання _____

Під час дипломного проєктування здобувачка освіти Слинявчук К.Р. мала змогу самостійно приймати рішення з реалізації елементів і модулів гри, та показала вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проєкту _____

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проєкту _____

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії

Підпис _____

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Слинявчук К.Р.
здобувач освіти гр. 4КГ-08, та

Джабраілов Д.В.,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка гри "Змійка" з додаванням модифікацій у геймплей» (автор роботи – Слинявчук К.Р., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Слинявчук К.Р. /

Керівник _____ / Джабраілов Д.В. /

«16» червня 2025 р.

Д О В І Д К А

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КГ-08

Слинявчук Катерини Русланівни

на тему *Розробка гри "Змійка" з додаванням модифікацій у геймплей*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з несуттєвими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 15.06.2025 р. значення коефіцієнту подібності в роботі становить 7,71%, коефіцієнт цитування – 1,13%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Слинявчук К.Р.* (П.І.Б.)

проведена « *16* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка гри "Змійка" з додаванням модифікацій у геймплей

Автор
Слинявчук Катерина Русланівна
Науковий керівник / Експерт
Джабраілов Дмитро Володимирович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1

КЦ

25	15376	116677
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		15
Інтервали		0
Мікропробіли		1
Білі знаки		0
Парафрази (SmartMarks)		44

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	83 0.54 %
2	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	75 0.49 %
3	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	64 0.42 %
4	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	40 0.26 %
5	https://card-file.ontu.edu.ua/bitstreams/158e44b0-583e-4b2d-b758-6b86979e33bb/download	40 0.26 %

6	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	40 0.26 %
7	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	34 0.22 %
8	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	31 0.20 %
9	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	31 0.20 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	29 0.19 %

з домашньої бази даних (0.49 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка web-застосунку для генерації повідомлень із використанням технологій штучного інтелекту [redacted] 6/14/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	33 (3) 0.21 %
2	Розробка веб-застосунку інтелектуального пошуку та сумісного перегляду аніме-контенту [redacted] 6/14/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	22 (2) 0.14 %
3	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	21 (2) 0.14 %

з програми обміну базами даних (0.11 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка технологічного процесу плазмового наплавлення бандажної полки лопатки авіадвигуна [redacted] 6/9/2023 National University "Zaporizhzhia Polytechnic" (Кафедра "Інтегровані технології зварювання та моделювання конструкцій")	12 (1) 0.08 %
2	Вебресурс ДНЗ "Почаївське вище професійне училище" [redacted] 6/14/2024 Rivne Professional college of National University of Life and Environmental sciences of Ukraine (ВСП ,,Рівненський фаховий коледж НУБіП України")	5 (1) 0.03 %

з Інтернету (7.11 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content**	**205 (7) 1.33 %**
2	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**160 (5) 1.04 %**
3	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**131 (4) 0.85 %**
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	100 (6) 0.65 %
5	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**71 (4) 0.46 %**

6	**https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download**	**50 (2) 0.33 %**
7	**https://card-file.ontu.edu.ua/bitstreams/158e44b0-583e-4b2d-b758-6b86979e33bb/download**	**49 (2) 0.32 %**
8	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	46 (3) 0.30 %
9	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	42 (3) 0.27 %
10	https://www.essuir.sumdu.edu.ua/bitstream/123456789/91932/1/Veretelnikov_bak_rob.pdf	36 (3) 0.23 %
11	**http://www.matmodel.puet.edu.ua/files/lic2020/rp-ossp-24.pdf**	**25 (1) 0.16 %**
12	https://card-file.ontu.edu.ua/server/api/core/bitstreams/f5042058-9544-4ac7-bd33-42bd733c8e6b/content	22 (3) 0.14 %
13	http://dspace.zsmu.edu.ua/bitstream/123456789/3123/1/Seval'njevAI15_Navch_mpdpz.doc	22 (3) 0.14 %
14	https://card-file.ontu.edu.ua/server/api/core/bitstreams/21ac499a-a9e9-4137-810c-5f21a0318048/content	16 (1) 0.10 %
15	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	16 (2) 0.10 %
16	https://en.wikipedia.org/wiki/Set_(mathematics)	16 (1) 0.10 %
17	http://oppb.com.ua/news/ohorona-praci-ofisnyh-pracivnykiv-organizaciya-ta-obladnannya-robochogo-miscya	15 (2) 0.10 %
18	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	15 (1) 0.10 %
19	https://card-file.ontu.edu.ua/server/api/core/bitstreams/4bb7255e-46d4-4349-9726-9698476da02d/content	12 (1) 0.08 %
20	https://ena.lpnu.ua/bitstreams/96104f0b-a0f4-4e6b-a9b5-066fa0a6e39e/download	12 (1) 0.08 %
21	https://iq.vntu.edu.ua/repository/getfile.php/4834.pdf	12 (1) 0.08 %
22	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	9 (1) 0.06 %
23	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	6 (1) 0.04 %
24	https://gi.edu.ua/koledzh/pidrozdily/biblioteka/repozytorii/%D1%84%D0%B0%D0%B9%D0%BB/%D0%B7%D0%B0%D0%B2%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D0%B8%D1%82%D0%B8/51c51db83db4752d1e928dba86d35f15	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітньо-професійна програма: «Комп'ютерна графіка і Web-дизайн» Група: 4КГ- 08

Дипломний проєкт здобувача освіти денної форми навчання КГ. 08.22.000.ДП

СЛИНЯВЧУК
КАТЕРИНИ РУСЛАНІВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»