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

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проєкт

здобувача освіти денної форми навчання
РП.08.13.000.ДП

ОСИПЕНКА
ВЛАДИСЛАВА ОЛЕГОВИЧА

м. Одеса
2025 р.

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

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка 2D-платформера з елементами roguelike у ICP Unity

Проектний матеріал складається з пояснювальної записки на 73 сторінках та графічного (презентаційного) матеріалу на 11 аркушах (слайдах)

Дипломник _____ (Осипенко В.О.)

Керівник _____ (Шувалова І.О.)

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 відділенням _____ (Краснокутська К.Г.)

Захист «26» 06 2025 р. Протокол ЕК № 2

Оцінка ЕК 3/60

Секретар ЕК _____

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

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*
Спеціальність *121 Інженерія програмного забезпечення*
Освітньо-професійна програма *Розробка програмного забезпечення*

ЗАТВЕРДЖУЮ:
Заст. дир. з НВР *Беркань І.В.*
"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Осипенку Владиславу Олеговичу*
(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 2D-платформера з елементами roguelike у ICP Unity*

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

2. Термін здачі закінченого проекту 16.06.2025 р.

3. Вихідні данні до проекту *1. Реалізувати мобільний застосунок у жанрі 2D-платформера з елементами бою 2. Забезпечити механіку бою з дальньою атакою. 3. Здійснити анімацію дій персонажа 4. Створити систему керування здоров'ям гравця та ворогів. 5. Реалізувати логіку появи та руху ворогів. 6. Реалізувати функціональне головне меню*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Аналіз та класифікація 2D-ігор жанру аркада; Аналітичний огляд і вибір рушія для розробки гри; Огляд засобів організації ігрової логіки; Огляд середовищ розробки для створення мобільних застосунків; Проєктування основної механіки гри; Проєктування структури даних для словника та аудіювання; Розробка та організація архітектури гри; Опис інтерфейсу та навігації у грі

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)
Діаграма станів головного персонажа Скріншоти реалізованої гри; Файлова структура проєкту у Unity; Діаграма взаємодії між об'єктами; Схема логіки перемоги/поразки Діаграма станів ворога; Скриншот головного меню

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

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

7. Дата видачі завдання 05.04.2025

Керівник *Шувалова І.О.* _____
(підпис)

Завдання прийняв до виконання *Осипенко В.О.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка задачі проєктування*	05.04.2025	Виконав
2	*Аналіз жанру 2D-платформерів та розвитку*	10.04.2025	Виконав
3	*Обґрунтування жанрів та цільової аудиторії*	14.04.2025	Виконав
4	*Порівняльний аналіз рушіїв і обґрунтування вибору*	21.04.2025	Виконав
5	*Розробка архітектури та компонентної структури*	28.04.2025	Виконав
6	*Реалізація базової механіки руху*	04.05.2025	Виконав
7	*Реалізація системи бою, стрільби та механіки смерті*	07.05.2025	Виконав
8	*Розробка системи ворогів*	10.05.2025	Виконав
9	*Впровадження бонусів, скринь і покращень гравця*	20.05.2025	Виконав
10	*Дизайн рівнів: тайлмапи, пастки, декоративні об'єкти*	31.05.2025	Виконав
11	*Розробка системи інтерфейсу*	05.06.2025	Виконав
12	*Впровадження прогресії, балансу, налаштування*	10.06.2025	Виконав
13	*Тестування функціональності*	11.06.2025	Виконав
14	*Розробка питань з охорони праці та техніки безпеки*	14.06.2025	Виконав
15	*Підготовка мультимедійної презентації проєкту*	15.06.2025	Виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 13 000. 00 ДП	Дипломний проект		
А4			РП 08. 13 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 13 000. 00 ДП		
Розробив		Осипенко В.О.		16.06.	Розробка 2D-платформера з елементами roguelike у ICP Unity	Літ.	Аркуш 4 / Аркушів 30
Перевірив		Шувалова І.О.		16.06.		НДП	
Н. контр.		Петрашова В.І.		16.06.		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.		23.06.			

ЗМІСТ

ВСТУП

У сучасній індустрії цифрових розваг ігрові проєкти набувають дедалі більшого значення як складна форма творчості, програмування та інтерактивного досвіду. Серед усіх жанрів особливе місце займають 2D-платформери, які, попри свою видиму простоту, вимагають тонкого балансування механік, візуального стилю та технічної реалізації. У поєднанні з елементами roguelike цей жанр відкриває нові можливості для формування непередбачуваного ігрового процесу, де кожна спроба є унікальним викликом.

Розробка подібного проєкту вимагає не лише технічної обізнаності з рушієм Unity, а й глибокого розуміння принципів побудови ігрових систем, створення анімацій, управління подіями та архітектури коду. Це складний багаторівневий процес, у якому важливу роль відіграють як творчий задум, так і вміння втілювати його за допомогою сучасних інструментів. Метою цієї роботи стало створення повноцінного 2D-платформера з елементами roguelike, що поєднує у собі класичну структуру рівнів і випадкові події, забезпечуючи варіативність проходження та інтерес гравця в довготривалій перспективі.

Актуальність теми зумовлена постійним зростанням популярності інді-ігор, які поєднують у собі доступність, простоту управління та глибину геймплею. Проєкти такого типу часто створюються невеликими командами або навіть однією людиною, і саме тому є чудовим майданчиком для самореалізації, навчання та демонстрації технічних і дизайнерських навичок.

Ця дипломна робота є результатом дослідження, аналізу та практичного втілення концепцій, які лежать в основі 2D-платформерів з додаванням елементів випадковості, бою, прогресії та взаємодії з динамічним середовищем. Вона відображає не лише здобуті знання у сфері комп’ютерної графіки й розробки ігор, але й демонструє здатність застосовувати їх у комплексному проєкті з повноцінною архітектурою, візуальним стилем і користувацьким інтерфейсом.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд 2D-платформерів: історія, ключові механіки та приклади

Жанр двовимірних платформерів зародився на початку 1980-х, коли компанія Nintendo представила гру Donkey Kong. У ній гравець керував Mario, що піднімався сходами й ухилявся від бочок – саме ця простота й заклала основу для подальшого розвитку жанру. Через кілька років, у 1985-му, Super Mario Bros. привнесла плавний скролінг екрана та безліч нових прийомів: гри стали не лише про стрибки, а й про дослідження світів, збір бонусів та вирішення простих головоломок. У 1990-х Кастелванія та Metroid додали платформерам нелінійний характер, відкривши гравцям великі карти з прихованими зонами й постійним пошуком апгрейдів.

До початку 2000-х інді-розробники почали повертатися до витоків, але з сучасними ідеями. Так влилися в гру емоційні історії Celeste та вишуканий дизайн Hollow Knight, де кожен рівень стає не тільки перешкодою, а частиною глибокого світу з власним характером. А наприкінці десятиліття — Spelunky та Dead Cells — стали першими успішними сплавами двовимірного платформера й roguelike: світ генерується випадково, а поразки лише підштовхують повторити спробу й навчитись новому.

Рисунок 1.1 Приклад Геймплею

У класичних платформерах керування зводиться до ходьби та стрибків. Гравець долає перешкоди — від неберучних гострих шипів до хитких рухомих плит — і взаємодіє з ворогами, які можуть просто ходити туди-сюди або переслідувати його по закладеному маршруту. Збір монет і бонусів того часу обмежувався додатковими очками чи зайвим життям, тоді як сучасні проєкти додають тимчасові силові покращення, динамічні карти та адаптивну складність. Серед класичних прикладів варто згадати Super Mario Bros., де перший рівень став еталоном: «земля», «небо», «блоки», «вороги» й «цегляні стіни» складаються у відточену арку. Sonic the Hedgehog виділився шаленою швидкістю акційного геймплею, доповненою петлями і великими падіннями. Metroid змінив ставлення до лінійності, дозволивши відкривати нові зони лише після отримання спеціальних апгрейдів. У новітніх работах, таких як Celeste, платформи стають засобом передачі емоцій: стрибки тут не просто засіб руху, а метафора внутрішніх переживань героя.

Рисунок 1.2 Приклад популярної гра платформер

Таке поєднання історичного контексту, розбору ігрових механік і конкретних прикладів допоможе глибше зрозуміти суть 2D-платформерів і правильно спланувати структуру власного roguelike-платформера на Unity.

1.2 Теоретичне обґрунтування вибору жанру

Жанрова належність гри є одним із ключових факторів, що визначають її механіку, дизайн, цільову аудиторію та навіть естетику. У межах даного дипломного проєкту було свідомо обрано поєднання жанрів 2D-платформер **і**

roguelike**,** оскільки саме ця комбінація поєднує в собі елементи динамічного геймплею та глибокого стратегічного мислення.

1.2.1 Історичні передумови розвитку жанру

Жанр roguelike бере свій початок з 1980-х років, коли з'явилася гра *Rogue* — текстовий dungeon crawler з процедурною генерацією рівнів і перманентною смертю персонажа. Незважаючи на примітивну графіку, проєкт швидко набув популярності завдяки глибокому геймплею, у якому кожна спроба була унікальною. Згодом концепцію підхопили й розвинули такі проєкти як *Nethack, ADOM* та Dungeon Crawl, що започаткували формування "канонічних" ознак жанру — випадкова генерація, незбереження прогресу в межах сесії, висока складність, глибока система предметів та вміння гравця адаптуватися.

У XXI столітті відбулося оновлення жанру через введення більшої інтерактивності, графічного інтерфейсу та гібридизації з іншими стилями. Особливу популярність набули 2D roguelike платформери**,** які поєднали вертикальний і горизонтальний рух, платформінг, бойову систему й елементи випадковості. Прикладами таких ігор *є* Spelunky, Dead Cells, Rogue Legacy — вони не тільки сформували ядро жанру, але й закріпили його успіх серед широкої геймерської аудиторії.

1.2.2 Психологія гравця і привабливість roguelike

Наукові дослідження у сфері геймдизайну, зокрема роботи Jesper Juul («The Art of Failure», 2013), доводять, що один із основних рушіїв задоволення в грі — це подолання перешкод. Roguelike-ігри забезпечують саме цей формат: повторення проходжень, які, попри поразки, дають гравцю досвід, вчать новим стратегіям і стимулюють розвиток навичок. Таким чином, кожна поразка не сприймається як кінець гри, а як старт нового виклику. Це створює ефект когнітивної прогресії*, коли* розвиток гравця відбувається не в межах механік, а в межах його власного мислення.

Крім того, вплив випадковості й перманентної смерті формує в гравця відчуття цінності кожного рішення. Кожна дія — це ризик. А отже, підвищується

емоційна залученість у процес гри. Для молодіжної цільової аудиторії (16–30 років), яка прагне короткочасних, але насичених ігрових сесій — це ідеальна комбінація.

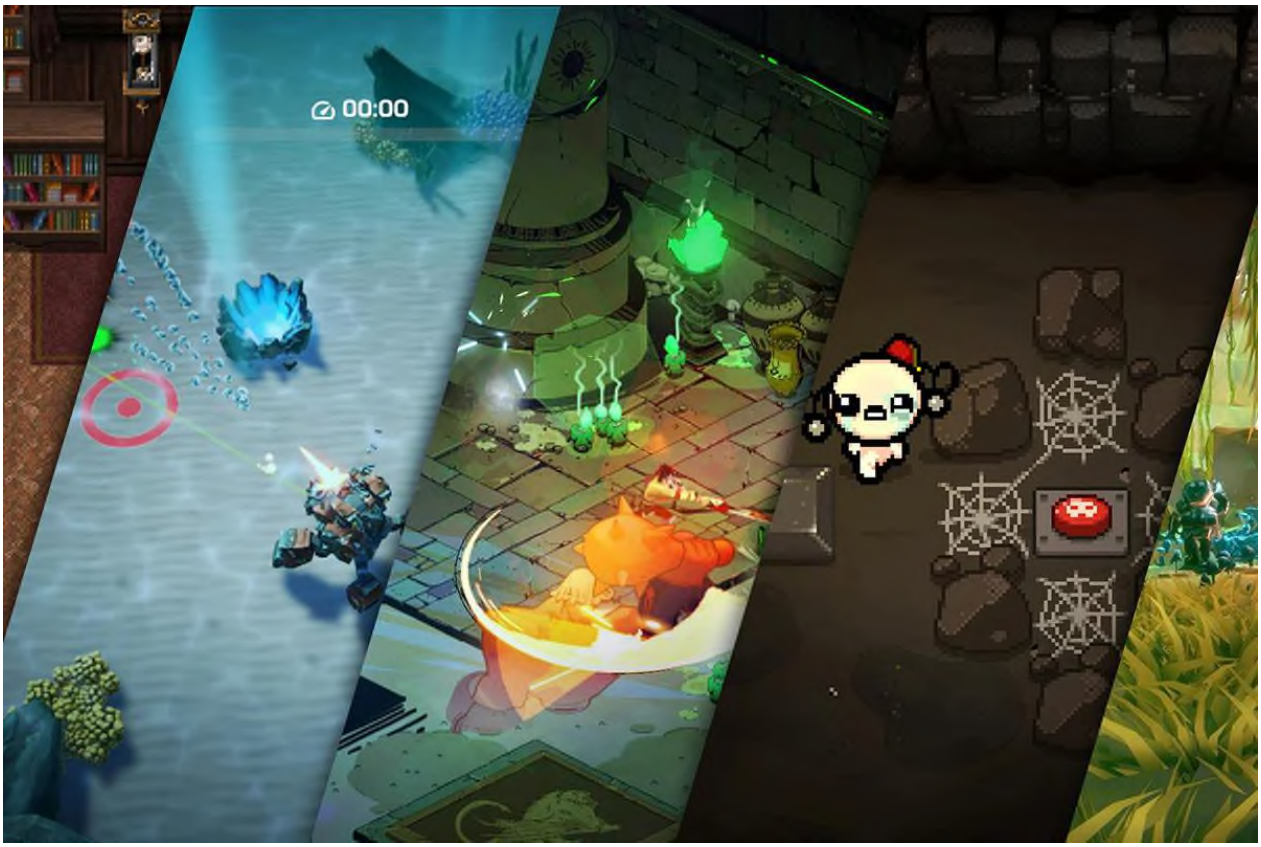


Рисунок 1.3 Представники жанру Roguelike

1.2.3 Динаміка платформера як геймплейна база

Вибір саме 2D-платформера як механічної основи зумовлений як технічними, так і ігровими причинами. По-перше, у платформерах простір ігрового світу представлений у зручному форматі: гравець миттєво розуміє фізику руху, а механіка "біг + стрибок" має інтуїтивне сприйняття. По-друге, платформери дозволяють ефективно передавати інформацію про небезпеку, бонуси та напрямок руху через візуальні символи — без потреби в додаткових інтерфейсах. Це значно полегшує адаптацію гри під нових користувачів.

Окрім того, технічно реалізувати 2D-платформер на рушії Unity є зручно завдяки підтримці Tilemap, 2D Physics, Animator Controller та інтеграції з пакунками генерації рівнів. Це зменшує поріг входу у створення складної логіки та дозволяє зосередитись на унікальності ігрової механіки.

1.2.4 Комбінація жанрів: додана цінність

Синтез roguelike і платформеру створює унікальний баланс між реакцією та стратегічністю**.** Платформер додає гравцеві постійне відчуття контролю через динамічний рух і точне позиціонування, тоді як roguelike-елементи вносять ризик і непередбачуваність, які підвищують реіграбельність. У результаті, кожна сесія гри має унікальну структуру, але зберігає знайому механічну основу, що забезпечує стійке залучення користувача**.**

Більш того, така гібридна форма дозволяє ефективно моделювати навчальні криві: прості платформи у вступі гри дають змогу гравцю ознайомитися з керуванням, а поступове ускладнення через випадкові вороги, пастки, бонуси — розвиває навички адаптації, що є ядром roguelike-досвіду.

З урахуванням психологічних, історичних і технічних чинників, поєднання жанрів 2D-платформер та roguelike є виправданим і ефективним вибором для реалізації навчального проєкту. Воно дозволяє створити гру, яка одночасно проста в освоєнні, багата на тактичні можливості, а також забезпечує високу ступінь залучення гравця. Технічна база Unity лише підсилює можливості цієї комбінації, надаючи потужні засоби реалізації генеративних та фізичних механік. Отже, обраний жанр не лише відповідає сучасним тенденціям інді-розробки, а й є придатним для демонстрації ключових навичок програмування, геймдизайну та аналітики в межах дипломного проєкту.

1.3 Особливості жанру Roguelike

Жанр Roguelike заснований на ідеї, що кожен прогін гри унікальний: рівень, розташування пасток і ворогів - все генерується заново, створюючи відчуття новизни при кожному проходженні. Випадковість поширюється як на геометрію карт, а й у появу предметів, типів ворогів і навіть параметри самого гравця (наприклад, випадкові баффи чи дебаффы початку рівня). Завдяки цьому механіці розробники домагаються того, що досвід проходження ніколи не повторюється дослівно, а гравцеві доводиться постійно адаптуватися до нових умов та виробляти гнучкі стратегії.

Перманентна смерть (permadeath) перетворює кожен крок персонажа на серйозний вибір: промах і загибель означають початок всього циклу заново, без збереження прогресу всередині рівня. Така жорстка система вирощує почуття відповідальності та змушує уважніше ставитися до ресурсів – здоров'я, боєприпаси, витратні предмети. У поєднанні з процедурною генерацією це дає потужну мотивацію вчитися на помилках: навіть якщо ви починаєте спочатку, досвід, навички та розуміння механіки залишаються з вами, формуючи неформальну «прогресію розуму»

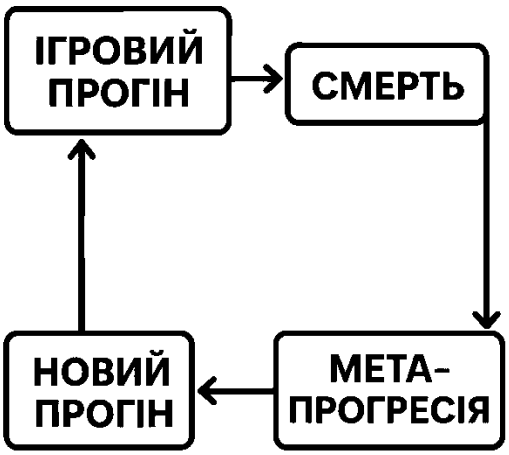


Рисунок 1.4 Принцип механіки roguelike

Щоб пом'якшити суворість permadeath і утримати гравця надовго, більшість сучасних roguelike-ігор вводять елементи мета-прогресії: накопичення валюти або окулярів досвіду, які можна витратити на постійні покращення, відкриття нових класів персонажів, унікальних артефактів або додаткових рівнів. Це дає відчуття постійного руху вперед і знижує фрустрацію багаторазових поразок. Саме баланс між випадковістю, ризиком та довгостроковими цілями забезпечує високий рівень реграбельності: кожен новий захід дарує шанс спробувати інше складання умінь, протестувати нову тактику чи вирішити інше завдання.

1.4 Аналіз існуючих 2D-платформерів з елементами roguelike

В останні роки з'явилося кілька помітних проектів, які успішно поєднали традиційний платформерний геймплей із механіками roguelike. Кожен із них привніс свої ідеї, сильні та слабкі сторони яких важливо врахувати при розробці власного проекту.

Spelunky (2008) стала піонером поджанру. Генерація рівнів гарантує, що гравець щоразу стикається з новим розташуванням платформ, пасток та ворогів, а жорстка система смерті без повернення змушує вивчити кожну дрібницю механік. Головним досягненням Spelunky було ідеальне балансування випадковості: в одній партії можна натрапити на потужний артефакт, а в іншій – програти через одну помилку. Проте різка крива навчання відштовхує новачків — відсутність мета-прогресії означає, що з умінь гравця прогрес залишається повністю у межах однієї сесії.

Dead Cells (2018) успадкувала процедурну генерацію, але додала дерево постійних покращень та систему відкриттів, що дозволяє з кожним новим заходом розблокувати нову зброю, спорядження та маршрути. Це пом'якшує ефект перманентної смерті, утримуючи інтерес новачків та даруючи відчуття поступального прогресу. Сильною стороною є плавний і чуйний бій, слабкою - відносно вузька варіативність архітектури рівнів, яку замінюють лише біоми, що змінюються.

Rogue Legacy (2013) привносить ідею «успадкування» генів: кожен новий герой отримує випадкові мутації — уповільнення, двоїння стрибка, візуальні ефекти. Така система спонукає експериментувати з різним стилем проходження. У цьому структура замків залишається загалом статичної, а відчуття новизни створюється саме рахунок мутацій персонажа, а чи не чергування самих локацій.

Vagante (2018) фокусується не лише на платформінгу, а й на глибокій рольовій системі: гравець вивчає нові навички, підбирає різноманітне спорядження та стикається з унікальними босами. Сильний елемент дослідження поєднується з процедурним геймплеєм, що забезпечує унікальний досвід у кожному проходженні. У грі реалізовано кооперативний режим, який дозволяє разом із друзями долати підземелля та комбінувати здібності персонажів. Високий рівень складності сприяє напруженій атмосфері та вимагає стратегічного підходу до бою. Проте слабкість — застаріла графіка, складний інтерфейс і відсутність чіткого навчання для новачків, що іноді відволікає від динаміки й

ускладнює занурення у гру. Незважаючи на це, Vagante отримала схвальні відгуки за свою глибину, реіграбельність і унікальний візуальний стиль у ретро-естетиці.

	Spelunky	Dead Cells	Rogue Legacy
Procedural Generation	●	●	●
Meta-Progression		●	
Character Mutation		●	●
Responsive Combat			●

Рисунок 1.5 Таблиця поріняння платформерів

Аналізуючи згадані проєкти, можна зробити висновок, що варто перейняти підхід до плавної реалізації мета-прогресії, як це зроблено в Dead Cells, оскільки це дозволяє зменшити рівень фрустрації новачків. Процедурну генерацію рівнів доцільно урізноманітнити за допомогою тематичних вставок або мікросцен, подібно до реалізації в Spelunky — це зробить кожен забіг унікальним і більш запам’ятовуваним. Впровадження нестандартних механік розвитку персонажа, на кшталт тих, що використано в Rogue Legacy, здатне урізноманітнити ігровий процес, не ускладнюючи при цьому структуру самих рівнів. Крім того, увага до точності керування та динаміки бою, як у Dead Cells, підсилює задоволення від гри, а чистота інтерфейсу та візуальний мінімалізм, характерні для Vagante, дозволяють уникнути перевантаження гравця зайвою інформацією.

1.5 Порівняння ігрових рушіїв

У сучасному світі розробки цифрових ігор наявність потужного, зручного та функціонального рушія є вирішальним фактором успішного створення ігрового продукту. Серед найпопулярніших рушіїв, які широко

використовуються як у професійному середовищі, так і в індивідуальних проєктах, найбільше уваги привертають Unity, Unreal Engine та Godot. Хоча кожен з них має свої унікальні переваги, кінцевий вибір залежить від низки практичних і стратегічних чинників, зокрема мови програмування, доступності ресурсів, продуктивності, візуальної якості, гнучкості в роботі з різними платформами та особливостей ліцензування.

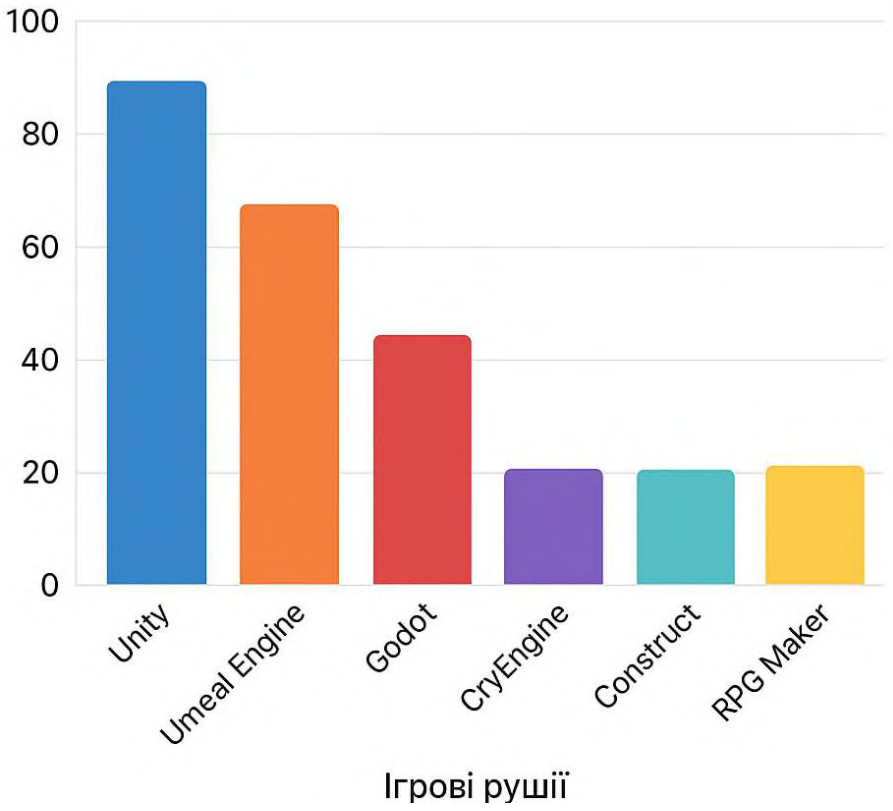


Рисунок 1.6 Порівняння ігрових рушіїв за популяністю

Unreal Engine позиціонується як інструмент професійної якості, орієнтований переважно на створення високорівневих, фотореалістичних 3D-ігор. Завдяки використанню мови C++ та унікальної системи візуального скриптингу Blueprints, він дозволяє досягти вражаючих візуальних результатів, особливо в жанрах шутерів або симуляцій. Проте складність у вивченні та потреба в більших ресурсах комп'ютера можуть створити бар'єр для новачків і ускладнити адаптацію на мобільних платформах.

Godot, навпаки, пропонує легкий, відкритий підхід до розробки ігор. Завдяки GDScript — простій мові, схожій на Python — та повністю безкоштовній моделі з відкритим кодом, цей рушій є привабливим для незалежних розробників

та освітніх проєктів. Однак його можливості у сфері 3D-графіки залишаються обмеженими, а кількість доступних навчальних матеріалів і розміри спільноти значно менші, ніж у більш поширених рушіїв.

На тлі цих варіантів Unity займає проміжну, але дуже вигідну позицію. Він поєднує гнучкість у роботі як з 2D, так і з 3D проєктами, забезпечує підтримку понад двадцяти п’яти платформ, включаючи мобільні пристрої, ПК, ігрові консолі, VR та AR середовища, і використовує мову C#, яка є зручною та широко застосовуваною. Завдяки величезній базі знань, доступній документації та активній спільноті, Unity дозволяє значно прискорити процес розробки, особливо для тих, хто тільки починає свій шлях у створенні ігор. Хоча графічні можливості Unity дещо поступаються Unreal Engine у плані фотореалізму, його технічної потужності більш ніж достатньо для реалізації інноваційних та динамічних проєктів, особливо в жанрах платформерів, казуальних ігор та навчальних симуляцій.

З огляду на всі зазначені аспекти, вибір Unity як основного рушія для реалізації ігрового проєкту є логічно обґрунтованим. Він пропонує ідеальний баланс між функціональністю, доступністю та масштабованістю, що робить його особливо придатним для навчальних, індивідуальних або малокомандних проєктів, а також дозволяє ефективно впроваджувати кросплатформені рішення без необхідності складної технічної адаптації.

1.6 Огляд ігрового рушія Unity як середовища розробки.

У Unity конвеєр роботи з ресурсами побудований так, щоб ви майже не відволікалися на рутину й могли зосередитися на творчій частині. Уявіть, що ви перетягуєте в проект папку зі спрайтами – Unity автоматично розпізнає розміри кадрів, розбере набір на окремі фрейми і підготує атлас. Після цього поділ кадрів відбувається на льоту: коли ви кладете спрайт на Tilemap, всі взаємодії між сусідніми тайлами — стикування колайдерів, вирівнювання пікселів, створення «ропних» кордонів — виконує спеціальний модуль Tilemap Renderer. Ви навіть можете задати правила, і кожен новий тайл підлаштується під свій сусідній: наприклад, автоматично підставить кутовий спрайт чи змінить візерунок.

При цьому всі налаштування зберігаються в окремих об’єктах-префабах, тому перші зміни, внесені в базовий префаб платформи, одразу відобразяться на всіх його копіях. Якщо ви вирішите змінити товщину платформи або колір фону, достатньо виправити один префаб — і ваші два десятки сцен оновляться самі, без ручного редагування кожного рівня. Анімації для персонажа і ворогів створюються у вікні Animation, де можна наочно побачити криву переходу від одного кадру до іншого, змінити швидкість програвання або додати плавне прискорення (Ease In / Ease Out) без єдиного рядка коду.

Коли ж ваш світ готовий, перейдіть у Scene View і налаштуйте освітлення та шейдери — Unity має вбудовані 2D Light- і Shader Graph, які легко зв’язуються з вашим Sprite Renderer-ом. Усе це доповнюється потужною системою фізики: Rigidbody2D та Collider2D автоматично адаптуються до форми ваших спрайтів, а налаштування матеріалів (friction, bounciness) задаються одним рухом повзунка. За потреби можна дописати власний фізичний сценарій у C#-скрипті, але навіть базові інтегровані компоненти дозволять зробити відчуття руху неймовірно природним.

Усі зміни можна моментально протестувати за допомогою Play Mode: ви редагуєте рівень, тиснете Play і відразу потрапляєте в свою гру з усім UI, фізикою та звуками. А коли потрібно зібрати фінальну збірку, кілька кліків у Build Settings — і ваш проект перетвориться на Windows-додаток, WebGL-гру або мобільний пакет. Такий конвеєр мінімізує час на технічні кроки й дає вам максимум простору для створення цікавих ігрових механік.

C#-скрипти в Unity по суті перетворюють статичні спрайти та префаби на живі об’єкти зі своєю логікою. Кожен скрипт — це клас, який наслідується від MonoBehaviour і «приклеюється» до геймоб’єкта як компонент. Ви пишете методи Start() та Update(), а Unity автоматично викликає їх під час запуску гри та на кожному кадрі. Якщо треба реагувати на зіткнення, достатньо додати OnCollisionEnter2D або OnTriggerExit2D — і движок передасть у ваш код інформацію про контакт двох Collider2D.

Щоб налаштувати поведінку без правки коду, використовують публічні поля або [SerializeField] приватні змінні. Вони одразу відображаються в Inspector, де можна задати їхні значення, змінити межі через атрибути [Range(min, max)] чи об’єднати кілька параметрів в єдину групу за допомогою [Header("Налаштування руху")]. Наприклад, якщо ви хочете бачити в інспекторі кольоровий слайдер для параметра JumpForce, достатньо дописати [Range(0, 20)] public float JumpForce = 10f; — і повзунок з’явиться автоматично, обмеживши значення від 0 до 20.

Animation Window працює з кліпами (.anim), що зберігають ключові кадри для кожного анімованого властивого. Ви перетягуєте префаб персонажа у вікно анімації, створюєте новий Animation Clip і починаєте запис — кожного разу, коли змінюєте позицію чи масштаб, Unity фіксує ключі. Потім через панель Curves ви можете вручну відкоригувати криві, додати згладжування або зворотні прискорення. Для переходів між станами використовують Animator Controller: додаєте стани (Idle, Run, Jump), малюєте стрілки з умовами (наприклад, bool isGrounded) та виставляєте часові переходи. У результаті будь-який запит на зміну стану — виклик SetBool("isRunning", true) або перехід через Trigger — відтворює складну послідовність анімацій без додаткових перевірок у вашому коді.

Завдяки цьому підходу в Unity ви отримуєте надзвичайну гнучкість: дизайнер може змінити швидкість анімації, розставити ключові кадри на нові позиції чи відредагувати криві прискорення, не торкаючись жодного рядка C#. Код відповідає лише за постановку значень параметрів Animator, а всі візуальні подробиці залишаються в руках аніматорів через інтуїтивний інтерфейс Animation Window.

Унікальність Unity – це не тільки “коробковий” набір інструментів, а й гнучка система розширень, що базується на Package Manager та Asset Store. Ось як це працює детальніше:

Unity Package Manager (UPM) є ключовим інструментом для керування залежностями в Unity-проєкті. Кожен пакет у цій системі представляє собою окремий модуль, що має власний файл package.json, де вказано версію, залежності, тип (наприклад, Experimental або Verified) та інші метадані. За

замовчуванням Unity використовує офіційний Unity Registry, але розробник може легко підключити додаткові джерела, зокрема приватні реєстри (Scoped Registries) або безпосередньо витягати пакети з GitHub або NPM, просто додавши відповідний URL у налаштуваннях проєкту в розділі Project Settings → Package Manager.

Система підтримує семантичне версіонування (SemVer), що дозволяє точно контролювати, яка саме версія пакета буде використана: мажорна, мінорна або патч. Наприклад, розробник може зафіксувати конкретну стабільну версію або вказати тег на кшталт @release, щоби проєкт автоматично слідкував за останніми релізами. Під час відкриття проєкту Unity перевіряє наявність новіших версій та, за потреби, пропонує оновлення з докладним журналом змін (Changelog), що дає змогу уникнути неочікуваних збоїв після оновлення бібліотек.

Ще однією важливою перевагою UPM є підтримка кешування. Завантажені пакети зберігаються локально, тож ви можете продовжувати роботу навіть без доступу до інтернету. Це особливо корисно в умовах обмеженої мережевої доступності або під час автоматизованого збирання проєктів у середовищах на кшталт Unity Cloud Build, де система може автоматично підтягувати потрібні пакети з кешу або репозиторію без ручного втручання. Завдяки цьому UPM забезпечує зручну, надійну та гнучку систему керування всіма залежностями вашого Unity-проєкту.

Asset Store у Unity є потужним інструментом для пошуку, придбання та інтеграції готових ресурсів, що пришвидшують розробку проєктів. Через вбудований інтерфейс редактора розробник може здійснювати пошук за категоріями, рейтингами, популярністю або ціновими діапазонами, а після придбання активи автоматично додаються до облікового запису Unity ID. Усі пакети мають чітко визначену структуру, що включає перелік файлів, залежностей і ліцензійних умов, які можуть варіюватися від повністю відкритого коду до комерційного використання.

Процес інтеграції максимально простий: після завантаження активу система самостійно розпаковує його в обраний каталог — Packages або Assets, відповідно

до налаштувань користувача. При імпорті відкривається вікно з вибором необхідних компонентів, де можна вручну визначити, які саме частини пакета будуть включені в проєкт — це можуть бути скрипти, префаби, 3D-моделі, текстури або супровідна документація. Такий підхід дає змогу уникнути зайвих файлів і зберегти контроль над структурою проєкту.

Asset Store також підтримує систему оновлень. Якщо розробник використовує вже встановлений пакет, Unity автоматично перевіряє його версію та порівнює з актуальною. У разі, якщо з'явилися зміни, редактор запропонує скористатися інструментом Merge Tool для порівняння та злиття конфліктних файлів — наприклад, якщо префаби були змінені вручну після імпорту. Для командної роботи з системами контролю версій на кшталт Git або Perforce доступна історія змін, що дозволяє відстежувати оновлення та синхронізувати їх між учасниками проєкту. Таким чином, Asset Store значно спрощує доступ до якісних ресурсів і їх підтримку на всіх етапах розробки.

У Unity існує низка ключових категорій пакетів, що охоплюють різні аспекти розробки та дозволяють значно розширити функціональність редактора. Для 2D-проєктів особливо корисними є такі спеціалізовані модулі, як Tilemap Editor, 2D Animation і 2D PSD Importer. Вони оптимізують роботу з графікою, спрайтами та анімаціями, дозволяючи створювати складні 2D-сцени з ефективною обробкою шарів, кісток і ключових кадрів.

Візуальні інтерфейси також мають потужну підтримку через UI-фреймворки. Окрім класичного Canvas із компонентами Unity UI, розробникам доступний новий UI Toolkit, який працює на декларативному підході, подібному до веброзробки — з використанням XML-структури (UXML) та стилів (USS), що нагадують CSS. Це дозволяє створювати гнучкі й легко підтримувані інтерфейси з чітким розділенням структури й стилів.

У сфері фізики та керування ресурсами важливу роль відіграють пакети Physics2D та Physics Shapes, які надають точнішу систему колайдерів і дозволяють працювати з кастомними формами об'єктів. Для оптимізації пам'яті та завантаження контенту під час виконання використовується система

Addressables, яка дає змогу завантажувати ресурси за запитом, кешувати їх локально та повторно використовувати без дублювання.

Мережеві функції реалізуються через такі інструменти, як Netcode for GameObjects, Relay і Matchmaker. Вони забезпечують синхронізацію об’єктів у реальному часі, управління сеансами гри, автоматичне з'єднання клієнтів навіть за умов NAT Traversal, що критично для багатокористувацьких проєктів.

У свою чергу, хмарні сервіси Unity значно полегшують процес розгортання та підтримки гри. Наприклад, Cloud Build дозволяє автоматизувати збірки для Android, iOS та інших платформ, а сервіси Analytics і Remote Config дають змогу в реальному часі збирати статистику поведінки гравців і змінювати внутрішні налаштування гри без потреби оновлювати клієнт.

При інтеграції пакету типу «roguelike» типовий робочий процес починається з імпорту відповідного набору ресурсів через Package Manager або Asset Store. Далі розробник налаштовує сцени, персонажів, ворогів та генератори рівнів відповідно до структури пакета. Якщо система використовує ScriptableObjects або компоненти для генерації підземель, вони вбудовуються у власну логіку гри. При потребі виконується адаптація UI або керування ресурсами через Addressables, а підключення до хмарних функцій дозволяє тестувати збірки та аналізувати геймплей у реальному часі. Такий підхід забезпечує швидкий старт і гнучке розширення проєкту на основі готового функціоналу.

Процес інтеграції пакету типу *roguelike* в Unity починається з пошуку відповідного рішення через Package Manager. Введення ключового слова “roguelike” дозволяє знайти кілька готових пакетів, зокрема офіційно підтверджених (Verified), що містять шаблони генерації рівнів, базову логіку AI та стартові сцени. На етапі оцінки розробник знайомиться з документацією пакету — як правило, у вигляді файлу README — та переглядає демонстраційні сцени, щоб зрозуміти, чи сумісний API цього пакету з поточною архітектурою гри.

Після підтвердження сумісності натискання кнопки “Install” автоматично додає пакет у структуру проєкту, зазвичай у каталог Packages/YourCompany.RoguelikeTemplate, де вже міститься вся його логіка,

ресурси та конфігураційні файли. У процесі налаштування розробник через Inspector або спеціальні редакторські вікна задає початкові параметри генерації — наприклад, seed-значення, набір ворогів, правила розміщення кімнат, дверей та предметів. Це дозволяє швидко зібрати прототип або інтегрувати шаблонну систему в реальний геймплей.

Якщо стандартна реалізація пакету не відповідає потребам проєкту, її можна легко адаптувати. Для цього розробник створює fork репозиторію (якщо він розміщений на GitHub), змінює вміст файлу package.json, наприклад — назву або шлях до залежностей, а далі підключає вже змінений варіант як локальний пакет, вказавши шлях у Project Settings. Такий підхід дозволяє зберегти всі переваги роботи з модульною структурою Unity Package Manager і водночас забезпечує повну гнучкість при кастомізації поведінки генерації рівнів, логіки AI чи будь-якої іншої складової roguelike-гри.

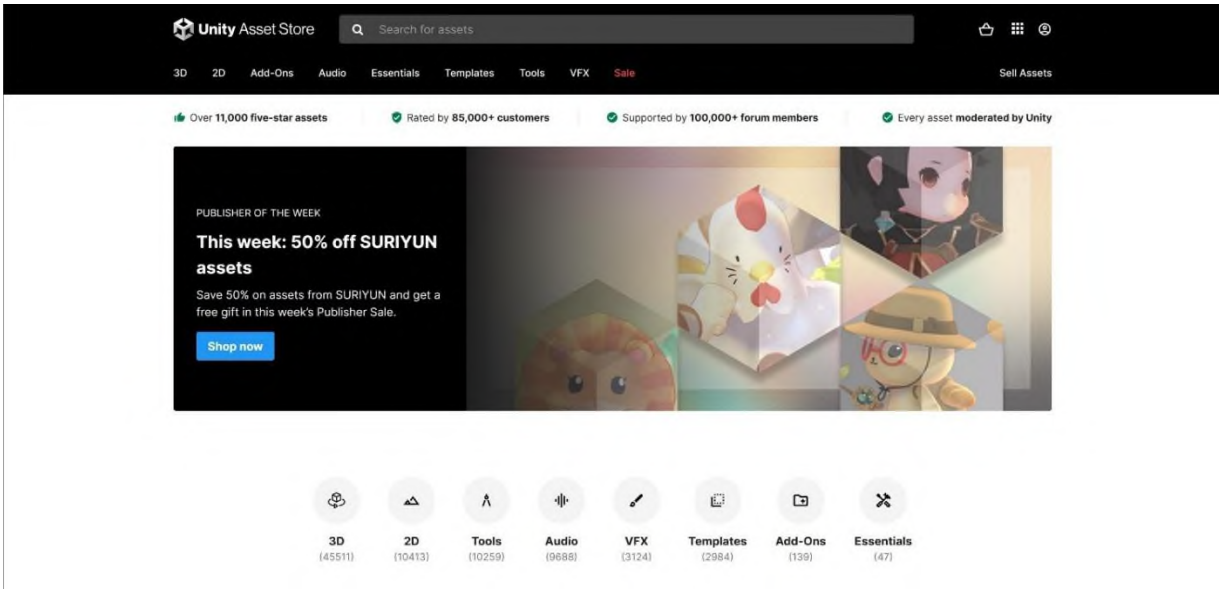


Рисунок 1.7 Інтерфейс Asset Store

Спільнота Unity — це одна з наймасштабніших екосистем у світі інді- та професійної розробки. Починаючи з офіційного форуму на сайті Unity, де кожен розділ (від 2D та UI до штучного інтелекту й мережевої синхронізації) має сотні тисяч сторінок обговорень, ви завжди знайдете відповіді на вузькі технічні питання. Паралельно працює Unity Answers — аналог StackOverflow лише для Unity, де досвідчені девелопери й інженери движка модерують контент,

гарантують високу якість відповідей і позначають «золотими» найкорисніші рішення.

Крім офіційних ресурсів, існують десятки незалежних порталів і спільнот: Unity Discord-сервери для миттєвого чату й обміну кодом, локальні MeetUp-групи (понад 400 у різних містах світу) із регулярними онлайн- та офлайн-зустрічами, а також сотні YouTube-каналів (від Brackeys до Game Dev Guide), де автори створюють серії туторіалів, лайвкодових сесій і розборів чужих проектів.

Документація Unity розділена на два ключові блоки: Manual і Scripting API. У першому описуються робочі процеси — як працювати з інтерфейсом, налаштувати освітлення, організувати роботу з AssetBundle чи Addressables. У другому детально перераховані класи, методи, події та структурні елементи C#-бібліотек Unity, доповнені підказками параметрів, прикладами коду та інформацією про сумісність із різними версіями движка. Більше того, Unity Learn пропонує понад 200 інтерактивних курсів із сертифікацією: від початківців до професіоналів, включно з офіційними навчальними планами для університетів та школярів, а також проектними «challenge»-задачами, які відтворюють реальні робочі процеси.

Коли виходить нова версія Unity, увага спільноти зосереджується на сторінці Release Notes і форумах Preview, де розробники описують важливі зміни API, покращення редактора й можливі проблемні місця при оновленні. Для автоматизації міграції Unity надає утиліти командного рядка (например, Unity –batchmode –upgradeProject), а система Assembly Definition Files (.asmdef) дозволяє ізолювати власний код від пакетів і швидко виявляти конфлікти при зміні версії..

1.7 Концепція гри

Моя гра — це динамічний 2D-платформер із базовими елементами roguelike, створений на Unity. Ідея полягає в поєднанні традиційного “бігу й стрибків” з повторюваністю спроб та випадковістю у вигляді різних наборів ворогів і бонусів на кожному рівні. З кожним новим заходом гравець стартує з чистого аркуша — без збережень усередині рівня, але з можливістю отримати

випадкові підсилення (наприклад, збільшення швидкості чи запасу здоров’я), що робить кожну спробу унікальною.

Розробка буде вестися на заздалегідь спланованих статичних рівнях: тайлмапами викладені платформи, перешкоди й місця появи ворогів. Хоча процедурна генерація поки що не реалізована, ми компенсуємо цей момент випадковими “пакетами” предметів і ворогів, які система спавну обирає з набору варіантів. Це дозволяє зберегти контроль над дизайном рівнів, але одночасно привносить елемент несподіванки й перманентної смерті.

Основні цілі гравця полягають у поступовому освоєнні механік руху й стрибків, що дозволяє ефективно долати складні конфігурації платформ і уникати небезпечних пасток. У процесі проходження гравець має навчитися розпізнавати шаблони поведінки ворогів і відповідно реагувати на їхні атаки, ухиляючись або використовуючи контратаку. Важливим елементом ігрового досвіду є збирання випадкових бонусів, які можуть покращити характеристики персонажа чи дати тактичну перевагу — уміння використовувати ці ресурси в потрібний момент значно підвищує шанси на успішне проходження. Кінцева мета полягає в подоланні всіх запланованих рівнів і досягненні фінального етапу гри без втрати всіх доступних життів.

Проєкт орієнтований на цільову аудиторію у віці від 16 до 30 років — активних гравців, які цінують динамічний ігровий процес, виклики та можливість для вдосконалення власної майстерності. Такий гравець шукає цікаву й водночас складну гру, яка вимагає уваги, реакції та стратегічного мислення.

Платформерну складність з елементами ризику й ризикованої винагороди. Це поціновувачі класики, знайомі з Super Mario чи Celeste, але й відкриті до невеликих експериментів із roguelike-механікою.

У проєкті вже реалізовано або активно впроваджуються кілька унікальних механік, які роблять ігровий процес більш динамічним і варіативним. Однією з ключових особливостей є система випадкового спавну ворогів і бонусів: хоча сам рівень залишається фіксованим за структурою, точки появи супротивників та

корисних предметів обираються з кількох підготовлених варіантів. Це створює ефект новизни при кожній спробі й підвищує реіграбельність.

Іншою важливою особливістю є механіка перманентної смерті з чекпойнтами, розташованими на початку рівня. Якщо гравець втрачає всі життя, він змушений почати рівень заново, проте зберігає у пам’яті місця розташування ворогів і пасток, що стимулює навчання на власних помилках і дозволяє краще підготуватися до повторної спроби.

Проєкт також підтримує гнучку систему налаштувань параметрів через інспектор Unity. Усі ключові значення — швидкість руху, запас здоров’я, діапазони спавну, інтервали між хвилями ворогів тощо — задані як публічні поля в C#-скриптах і можуть бути змінені без необхідності перезапуску гри або перекомпіляції коду. Це значно пришвидшує балансування та тестування рівнів.

Окрема увага приділена користувацькому інтерфейсу. За допомогою TextMeshPro реалізовано систему UI-повідомлень і логів. Контролер повідомлень виводить на екран підказки для гравця, сповіщення про критичний стан здоров’я, результати проходження рівня, а також інші важливі дані, що підтримують гравця під час гри та створюють відчуття цілісного, продуманого інтерфейсу.

Такий підхід дозволяє зберегти чітку художню концепцію рівнів і водночас надати їм процедурну варіативність, що сприяє високій реіграбельності — ключовій характеристиці roguelike-жанру. Комбінування можливостей Tilemap Editor з гнучкістю налаштовуваних скриптів значно пришвидшує цикл розробки: від створення первинного ескізу до ітеративної оптимізації дизайну. Завдяки цьому процес прототипування стає не лише ефективним, а й адаптивним до змін у геймдизайні, дозволяючи швидко тестувати нові ідеї та впроваджувати балансування.

1.8 Дизайн ігрових механік

Переміщення гравця спирається на класичну комбінацію ходьби, бігу та стрибків із додаванням взаємодії із вертикальними поверхнями. За замовчуванням персонаж ковзає по платформі з фіксованою швидкістю ходьби, яка збільшується при натисканні клавіші бігу. Сила та висота стрибка

регулюються єдиним параметром JumpForce у скрипті CharacterController, дозволяючи гравцю долати як стандартні платформи, так і вузькі щілини.

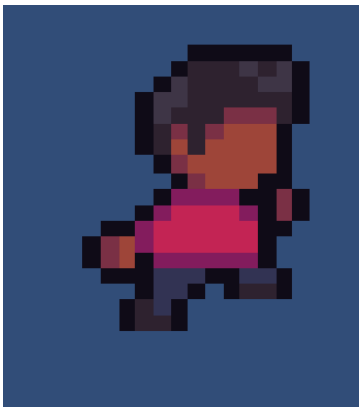


Рисунок 1.8 Головний персонаж

Якщо персонаж торкається стіни під час стрибка, активується стан “прилипання” — його падіння уповільнюється, а управління в повітрі стає обмеженим, щоб надати відчуття опору. У цьому стані гравець може відштовхнутися від стіни повторним стрибком, що відкриває можливість досягати важкодоступних ділянок та додасть глибині руху. Всі ці стани — OnGround, InAir, OnWall — контролюються через Animator Controller, де кожен перехід між ними підкріплюється плавною анімацією.

У базовому варіанті бою передбачено поєднання атаки зброєю в ближньому бою та використання здібностей із відновлюваним часом кд (cooldown). Якщо супротивник у зоні ураження — застосовується функція ApplyDamage(), яка зменшує його здоров’я й відштовхує назад, надаючи відчуття імпульсу.

Здібності (наприклад, короткий ривок уперед чи метальний снаряд) реалізовані як окремі методи з власними параметрами Range і Cooldown. Їхній стан зберігається в скрипті AbilityManager, який відстежує час до повторного використання та активує візуальні індикатори на HUD. Взаємодія з ворогом реалізується через систему перевірки колізій, яка визначає факт зіткнення з атакуючим об’єктом (наприклад, снарядом або ближнім ударом). У разі успішного влучення гравець отримує візуальний зворотний зв’язок у вигляді екранної тряски (screen shake**)** та ефекту іскор**,** що покращує читаємість бойових подій і підсилює відчуття впливу.

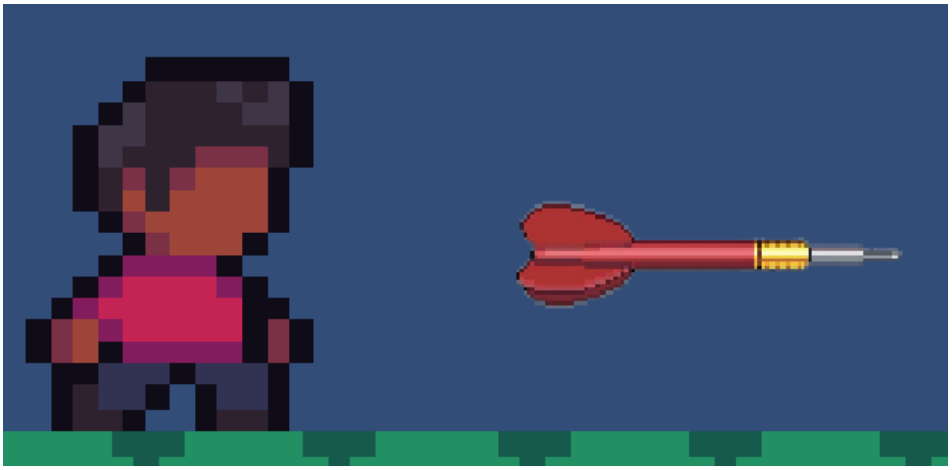


Рисунок 1.9 Атака головного персонажу

Хоча базові рівні побудовані статично, елемент випадковості забезпечується через генератор RoomSpawner, який при старті рівня обирає із заданого переліку шаблонів кімнат певну послідовність. Кожна кімната має тип (бойова, бонусна, пастка) та виходи у чотирьох напрямках. Алгоритм підбирає до трьох різних шаблонів на ходу, зберігаючи загальну структуру рівня, але змінюючи порядок і контент.

Система предметів реалізована за допомогою класу Item, який містить основні характеристики: BaseValue, Modifier та Rarity. Після проходження кімнати відбувається випадкове породження луту: скрипт LootManager звертається до таблиці зважених шансів і обирає один або два предмети. Алгоритм враховує рідкість ефектів — рідкісні модифікатори (наприклад, збільшення швидкості на 20 %) мають значно нижчу ймовірність випадіння, тоді як поширені бонуси, на кшталт додаткових очок здоров’я, трапляються частіше. Такий підхід підтримує баланс ігрового процесу й водночас мотивує гравця досліджувати більше, сподіваючись на цінні знахідки.

Перманентна смерть означає, що після втрати всіх життів прогрес у поточному рівні втрачається, але зберігається мета-прогресія: між забігами гравець накопичує ресурси (наприклад, монети), які можна витратити в головному меню на постійне підвищення базових характеристик (збільшення максимальної енергії, розширення інвентаря тощо). Ця проста система дозволяє пом’якшити відчуття жорсткої смерті й мотивує пробувати знову, застосовуючи нові комбінації навичок і предметів.

1.9 Дизайн рівнів та візуальний стиль

При створенні ігрових рівнів керування простором здійснюється так, щоб поєднувати дослідження, напругу та поступове введення нових викликів. Використовується тайлмап-система Unity: кожен елемент світу — платформа, стіна чи декоративний об’єкт — представлений квадратом стандартного розміру. Від масштабу тайлів (наприклад, 32×32 пікселів для «ретро» чи 64×64 для «мультяшного») визначається відчуття загального стилю та читабельність на екрані.

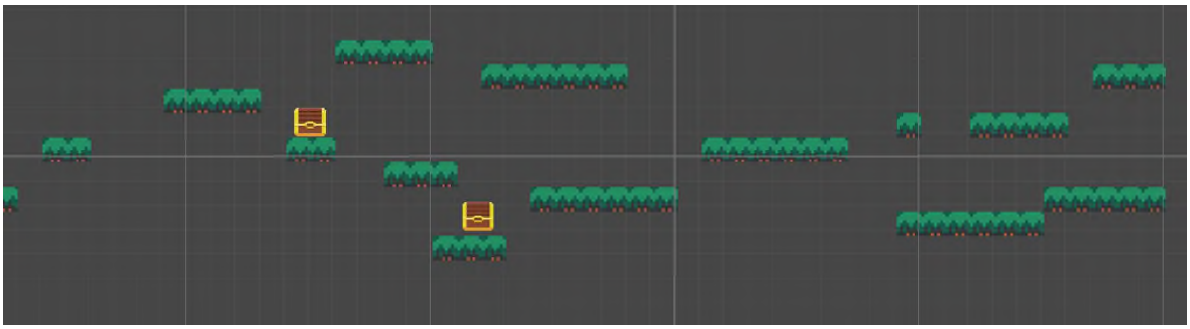


Рисунок 1.10 Приклад рівня у редакторі

Спочатку формується ескіз рівня в зовнішньому редакторі або на папері, де позначаються ключові зони: безпечні ділянки, коридори, що звужують рух, та приховані області з бонусами. Потім цей ескіз імпортується в Unity як Background Tilemap: лише силуети без деталей, щоб перевірити прохідність і масштабування. Після цього на окремому шарі розгортається основна Tilemap із Collider2D — кожний тайл із можливістю контакту гравця отримує фізичні властивості.
Декоративні елементи виводяться на другому шарі без колайдерів, щоб запобігти порушенню фізики, але надати рівню багатство деталей (наприклад, плюскіт води чи виступаючі кущі). Розділення шарів дозволяє змінювати художню складову без втручання у фізичні налаштування.

Художній стиль обраний у жанрі піксель-арт із обмеженою палітрою кольорів на рівень (8–12 кольорів на тайл, 32–64 кольори загалом). При цьому яскравіші кольори підкреслюють точки інтересу та небезпеки, а приглушені лишаються фоном. Так забезпечується чітка візуальна семантика, яка легко сприймається на будь-якому екрані.

У проєкті обрано художній стиль у жанрі піксель-арт, що відсилає до класичних ретро-ігор, але реалізований із сучасним підходом до кольору й композиції. Основна концепція стилю ґрунтується на використанні обмеженої палітри кольорів для кожного рівня — зазвичай від 8 до 12 кольорів на тайл, із загальним обсягом палітри в межах 32–64 кольорів на всю сцену. Такий підхід дозволяє зберегти візуальну цілісність та художню гармонію, а також значно полегшує оптимізацію графіки для пристроїв із різним розширенням екрана та обчислювальними можливостями.

У піксель-арті важливу роль відіграє колірна ієрархія, і саме вона використовується для формування візуальної семантики. Яскраві кольори в межах палітри навмисне застосовуються для виділення ключових ігрових елементів — точок інтересу, таких як двері, бонуси, важливі предмети або активні вороги. Водночас приглушені, менш контрастні відтінки слугують для формування фону, стін, підлоги або декоративних об’єктів, які не повинні відволікати увагу гравця. Це дозволяє не лише створити атмосферну сцену, а й інтуїтивно спрямовувати погляд користувача в потрібні зони екрана, підвищуючи читабельність простору та зрозумілість ігрових ситуацій.

Завдяки такому продуманому поєднанню стилістики та функціональності, візуальне оформлення гри не лише приваблює увагу, але й виконує практичну роль: допомагає орієнтуватися в рівні, розпізнавати небезпеки та приймати швидкі рішення під час геймплею. Стиль добре масштабується, адаптується під різні платформи та зберігає естетику навіть на невеликих екранах, що робить його оптимальним вибором для проєкту такого типу.

Окрім того, візуальні ефекти та продумана кольорова палітра відіграють важливу роль у формуванні цілісної атмосфери гри — вони акцентують динаміку подій і підсилюють настрій окремих сцен (наприклад, напруження під час боїв або спокій дослідження). Завдяки легкій стилізації графіки та інтуїтивно зрозумілому інтерфейсу, користувачі швидко орієнтуються в середовищі гри й легко занурюються в процес, не відволікаючись на зайві візуальні елементи. Такий підхід сприяє комфортному ігровому досвіду навіть для новачків

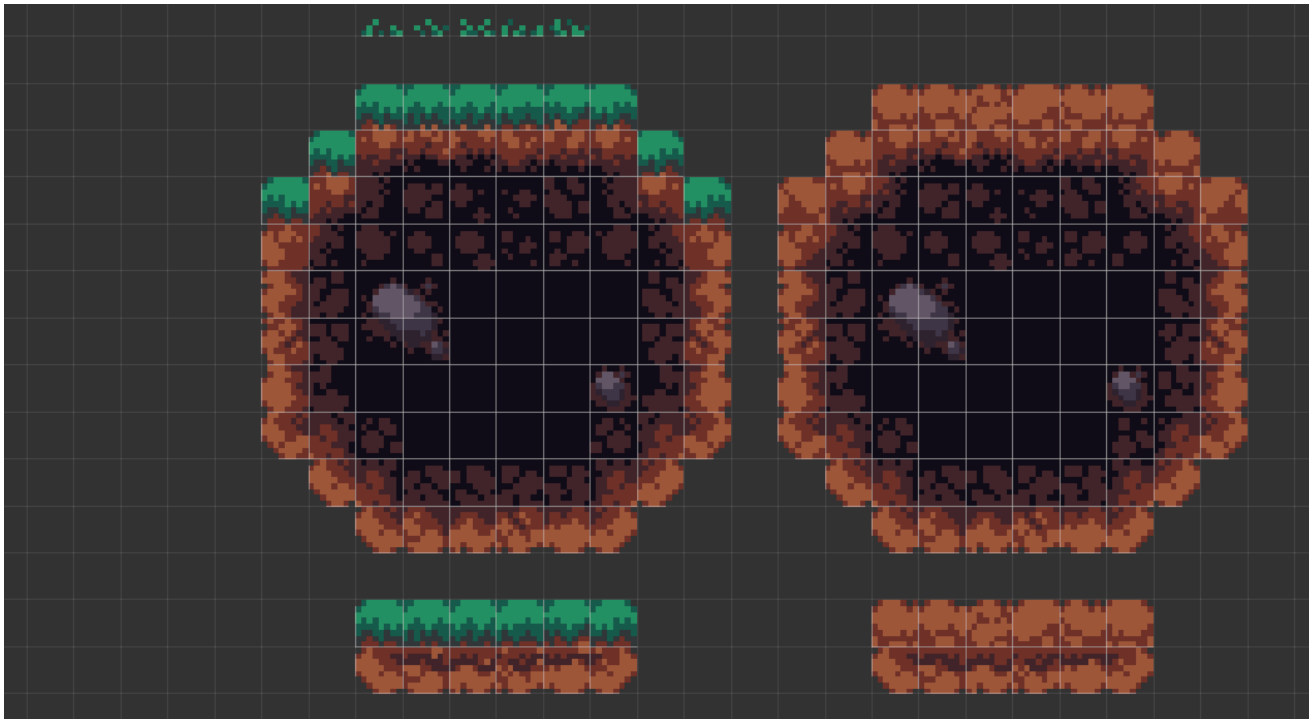


Рисунок 1.11 Використанний Tilemaps

1.10 Реалізація персонажів, ворогів та інтерактивних об’єктів

Персонаж керується за допомогою горизонтальної осі (Horizontal), що реагує на натискання клавіш керування (A/D, ←/→) або відповідні сигнали з контролера. Зчитування значення осі відбувається в методі Update(), після чого обчислюється вектор напрямку, який передається в Rigidbody2D через метод MovePosition або шляхом встановлення швидкості компонента velocity.

Швидкість руху контролюється змінною moveSpeed, яка за замовчуванням дорівнює 5 одиниць, але може бути змінена для балансування геймплею або адаптації до різних типів персонажів. Завдяки цьому гравець отримує плавне й чутливе управління, яке залишається стабільним незалежно від частоти кадрів.

Щоб забезпечити правильне відображення персонажа, напрям руху використовується також для повороту спрайта (наприклад, через transform.localScale). Таким чином, при зміні напрямку руху вліво чи вправо змінюється орієнтація спрайта, що створює більш природне враження руху.

```
private void HandleMovement()
{
    float hor = Input.GetAxisRaw("Horizontal");
```

*rb.linearVelocity = new Vector2(hor * moveSpeed, rb.linearVelocity.y);*

if (hor != 0)

transform.localScale = new Vector3(Mathf.Sign(hor), 1, 1);

anim.SetBool("isRunning", hor != 0);

anim.SetBool("isGrounded", isGrounded);

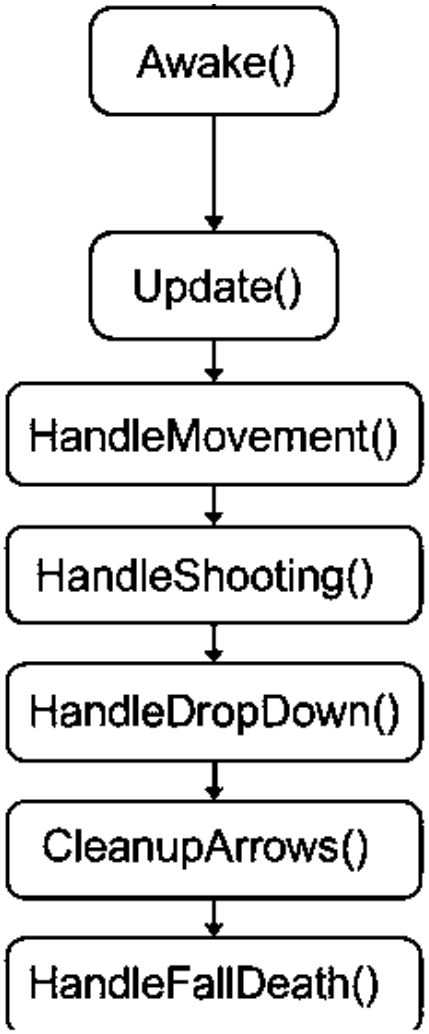


Рисунок 1.12 Діаграма послідовності виклику

- `Mathf.Sign(hor)` визначає напрямок руху та відповідає за розворот спрайта.

- Анімація ходьби (`isRunning`) активується при ненульовому значенні осі.

Сила стрибка задається параметром `jumpForce` (12 одиниць). Реалізовано подвійний стрибок через лічильник `jumpsRemaining`:

private void HandleJump()

if (Input.GetButtonDown("Jump") && jumpsRemaining > 0)

```
{
    rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
    anim.SetTrigger("jump");
    jumpsRemaining--;
```

- При натисканні пробілу (`Jump`) викликається стрибок, якщо залишилися спроби (`jumpsRemaining > 0`).

- Лічильник оновлюється при приземленні (див. `OnCollisionEnter2D`).

Можливість подвійного стрибка активується після підбирання відповідного покращення (`PowerupType.DoubleJump`):

```
public void ApplyPowerup(PowerupType type)
{
   case PowerupType.DoubleJump:
      if (!hasDoubleJump)
      {
         hasDoubleJump = true;
         jumpCount = 2; // Дозволяє два стрибки
         jumpsRemaining = 2;
      }
      break;
```

- Після активації `jumpCount` збільшується до 2, що дозволяє виконати другий стрибок у повітрі.

Стан персонажа «на землі» (isGrounded) визначається за допомогою фізичної перевірки колізій у зоні під його ногами. Для цього зазвичай використовується Raycast, BoxCast або OverlapCircle, який перевіряє наявність зіткнення з колайдером об’єкта, що має тег Ground або належить до шару OneWayPlatform. Якщо таке зіткнення виявлено — прапорець isGrounded встановлюється в true, що означає, що персонаж стоїть на поверхні та може, наприклад, здійснити стрибок. Цей підхід дозволяє точно визначати момент контакту з землею, враховуючи як стандартні платформи, так і односторонні, які можна пройти знизу.

```
private void OnCollisionEnter2D(Collision2D col)
{
    if (col.collider.CompareTag("Ground") || col.collider.gameObject.layer == LayerMask.NameToLayer(platformLayerName))
    {
        isGrounded = true;
        jumpsRemaining = jumpCount; // Відновлення стрибків
    }
```

- При зіткненні з землею `jumpsRemaining` скидається до значення `jumpCount` (1 або 2).

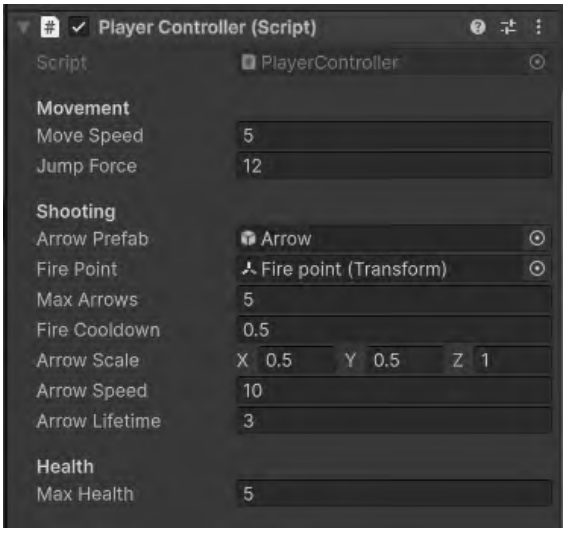


Рисунок 1.13 Налаштування скрипту

У проєкті система анімацій реалізована за допомогою компонента Animator, який керується через встановлення параметрів у коді відповідно до стану персонажа. Наприклад, якщо гравець починає рухатися, у змінну isRunning передається значення true або false залежно від того, чи є горизонтальний ввід (hor != 0), що активує відповідну анімацію бігу. Під час стрибка викликається тригер jump, який миттєво запускає анімацію стрибка без необхідності тримати стан постійно активним. Крім того, стан isGrounded сигналізує аніматору, чи персонаж перебуває на землі, що впливає на перехід між анімаціями стрибка, приземлення та простої. Завдяки цим параметрам досягається плавна і реалістична зміна анімацій залежно від дій гравця.

Особливості реалізації включають кілька ключових аспектів бойової механіки. Швидкість стрільби обмежена змінною fireCooldown, що дорівнює 0.5 секунди, завдяки чому досягається контрольована частота пострілів та підтримується ігровий баланс. Для створення снарядів використовується метод Instantiate, а максимальна кількість одночасно активних стріл у сцені обмежується значенням maxArrows = 5, що запобігає перевантаженню ігрового простору. Смерть персонажа настає в одному з двох випадків: якщо його координата у опускається нижче за −10 (падіння в прірву), або якщо показник здоров'я (HP) знижується до нуля. У таких ситуаціях викликається метод Die(), який відповідає за обробку поразки та перезапуск або завершення гри.

Код демонструє гнучке поєднання фізики, анімацій та ігрової логіки, що характерно для платформерів.

Усі публічні поля [SerializeField] було виведено в Inspector для подальшого тонкого налаштування без перекомпіляції коду.

Атрибути: відкриті поля, які налаштовуються через Inspector (moveSpeed, jumpForce, arrowPrefab, firePoint, maxArrows, fireCooldown, arrowScale, arrowSpeed, arrowLifetime, maxHealth).

Методи: ключові функції, що реалізують логіку контролю персонажа (Awake, Update, HandleMovement, HandleJump, HandleDropDown, HandleShooting, Shoot, CleanupArrows, HandleFallDeath, TakeDamage, Die, ApplyPowerup, OnCollisionEnter2D, OnCollisionExit2D).

Клас головного персонажа взаємодіє з іншими об'єктами гри через асоціації з класами Arrow та PowerupType, реалізовані у відповідних методах.

Асоціація з класом Arrow відбувається через метод Shoot, який відповідає за створення та запуск стріли. Коли гравець активує стрільбу, персонаж викликає Instantiate, створюючи новий екземпляр об'єкта типу Arrow, після чого додає його до списку activeArrows — цей список використовується для відстеження всіх стріл, які були випущені, ще знаходяться у грі, й потенційно підлягають видаленню або повторній активації.

Взаємодія з класом PowerupType реалізована у методі ApplyPowerup, який дозволяє персонажу отримувати тимчасові посилення (бафи). Метод приймає параметр типу PowerupType, визначаючи, який саме бонус було активовано — наприклад, підвищення швидкості, додаткові стрибки або посилення атаки. На основі отриманого типу відбувається відповідна зміна параметрів персонажа, наприклад: збільшення moveSpeed, модифікація jumpForce, або ж активація спеціальних ефектів.

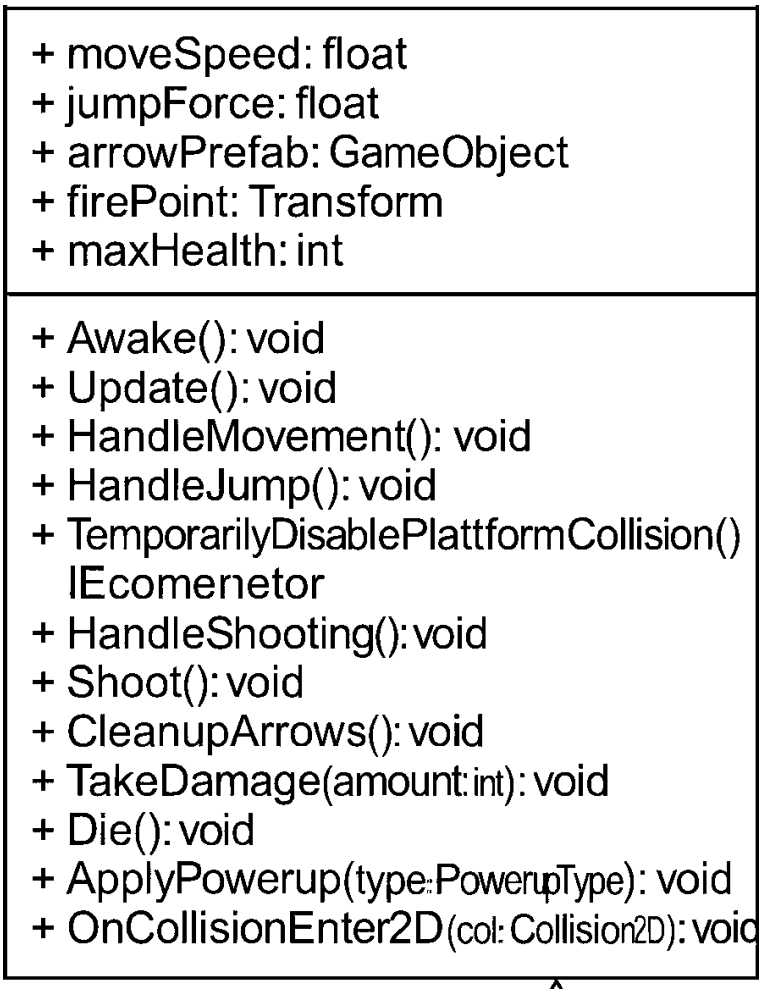


Рисунок 1.14 Структура класу PlayerController

Визначено три базових типи ворогів, кожному з яких присвоєна окрема поведінкова логіка в скрипті EnemyAI:

Тип ворога PatrolEnemy характеризується пересуванням між двома заздалегідь визначеними точками з постійною швидкістю, яка задається параметром PatrolSpeed. У разі, якщо гравець потрапляє в поле зору ворога, що визначається за допомогою перевірки через Physics2D.Raycast, ворог миттєво

переходить у стан переслідування. При досягненні контакту з гравцем ініціюється атака, під час якої завдається шкода методом ApplyDamage() та застосовується відштовхування за допомогою Knockback().

Тип ворога ChargerEnemy перебуває у стані очікування (Idle), доки гравець не з’явиться в межах визначеного радіуса агресії. Як тільки цю умову виконано, ворог миттєво здійснює потужний ривок уперед зі значно збільшеною швидкістю, що задається параметром ChargeSpeed. Після завершення ривка він переходить у стан відновлення (Cooldown), протягом якого його здатність до пересування тимчасово обмежена.

Тип ворога RangedEnemy підтримує визначену дистанцію до гравця, орієнтуючись на його положення за допомогою контролера повороту (Yaw-and-Pitch). Атака здійснюється шляхом періодичної стрільби, під час якої створюється снаряд із використанням префабу ProjectilePrefab з інтервалом, заданим параметром FireRate. Траєкторія польоту снаряда розраховується на основі напрямку до поточної позиції гравця на момент пострілу, що дозволяє здійснювати точні атаки навіть під час руху.

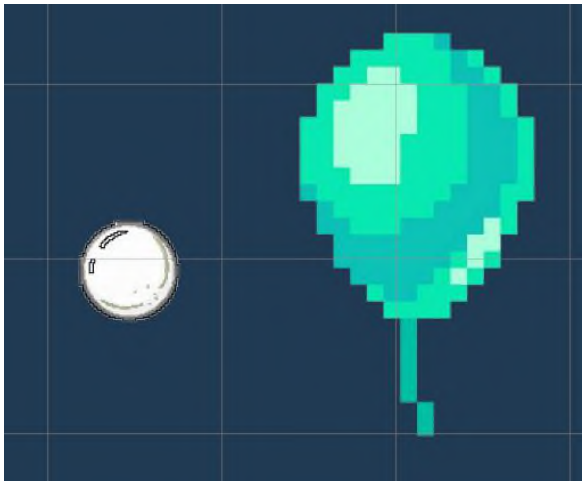


Рисунок 1.15 Дизайн ворога

Поведінка ворога прописана у коді: void Start()

```
{
   player = GameObject.FindGameObjectWithTag("Player")?.transform;
}

void Update()
{
```

```
    if (player == null) return;
    Vector2 direction = (player.position - transform.position).normalized;
    transform.position += (Vector3)direction * moveSpeed * Time.deltaTime;
    if (isVisible)
    {
        fireTimer += Time.deltaTime;
        if (fireTimer >= fireRate)
        {
            ShootAtPlayer();
            fireTimer = 0f;
        }
    }
}

void ShootAtPlayer()
{
    if (player == null) return;

    GameObject proj = Instantiate(projectilePrefab, firePoint.position,
Quaternion.identity);
    Vector2 shootDir = (player.position - firePoint.position).normalized;
    proj.GetComponent<EnemyProjectile>()?.Launch(shootDir);
}
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        other.GetComponent<PlayerController>()?.TakeDamage(1);
    }
}
```

У методі Start здійснюється пошук об’єкта з тегом "Player". Якщо такий об’єкт знаходиться, з його компонента Transform запам’ятовується позиція гравця (змінна player).

У Update спочатку перевіряється наявність посилання на гравця. Якщо воно відсутнє, подальше виконання методу припиняється. Далі обчислюється нормалізований вектор напрямку від поточної позиції ворога до позиції гравця, після чого ворог поступово переміщується в бік гравця з урахуванням швидкості moveSpeed і часу кадру (Time.deltaTime). У разі, коли прапорець isVisible встановлено в true, накопичується час у лічильнику fireTimer. Коли цей лічильник

досягає значення fireRate, викликається метод ShootAtPlayer, а лічильник скидається.

Метод ShootAtPlayer перевіряє наявність гравця, потім створює новий екземпляр снаряда (projectilePrefab) у точці firePoint. Напрямок польоту снаряда обчислюється як нормалізований вектор від firePoint до позиції гравця; запущений снаряд одержує цей вектор через виклик Launch(shootDir) у його скрипті EnemyProjectile.

У OnTriggerEnter2D відстежується зіткнення з колайдером гравця. Якщо тег у колайдера other — "Player", викликається метод TakeDamage(1) у компоненті PlayerController, внаслідок чого гравцю завдається одинична шкода

Для кожного ворога було створено окремий Animator Controller з параметрами Speed, IsCharging, IsDead, що забезпечує плавні переходи між станами і відповідні анімації.

Опис модулю сундуки

При відкритті через виклик методу OpenChest() активується анімація кришки та виконується виклик LootManager для спавну випадкового предмета. Стан об’єкта (відкритий/закритий) зберігається в ChestData для уникнення дублювання луту.

Рисунок 1.16 Дизайн інетрактивного об'єкту сундук

Коли викликається метод OpenChest(), насамперед встановлюється логічна змінна isOpened = true, що сигналізує про те, що сундук відкрито й більше його не можна повторно активувати. Після цього змінюється візуальне відображення об'єкта: до спрайту sr.sprite призначається зображення відкритого сундука (openSprite), а колір повертається до стандартного (defaultColor), що може

сигналізувати про завершення анімації підсвічування або ефекту «не відкритого» стану.

Далі викликається метод GrantRandomPowerup(), який відповідає за видачу випадкового покращення. Хоч його код не показано, очевидно, що він визначає тип бонусу (значення перерахування PowerupType) і передає його для обробки. Кожен тип PowerupType описується в окремому методі GetPowerupName(PowerupType type), який повертає текстову назву покращення українською мовою для подальшого виводу в інтерфейсі гравця або логів.

Таким чином, при відкритті сундука гравець отримує одне або кілька випадкових покращень, які можуть змінювати характеристики персонажа — наприклад, збільшувати швидкість руху (MoveSpeed), прискорювати стрільбу (FireRate), додавати можливість подвійного стрибка (DoubleJump) тощо. Усе це реалізовано через централізовану систему типів бонусів і дозволяє легко розширювати набір покращень у майбутньому.

Кожен бонус має унікальний ідентифікатор та визначену логіку застосування, що дозволяє безперешкодно інтегрувати його в загальну механіку гри. Випадковість вибору покращень забезпечується генератором псевдовипадкових чисел, а система обмежень не дозволяє дублювати несумісні або надмірно сильні ефекти. Такий підхід робить кожне відкриття сундука непередбачуваним і водночас збалансованим, що стимулює повторне проходження гри.

```
private void OpenChest()
{
    isOpened = true;
    sr.sprite = openSprite;
    sr.color = defaultColor;

    GrantRandomPowerup();
      }
```

З нього ми можемо отримати один або декілька бонусі як прописанно у классі :

```
private string GetPowerupName(PowerupType type)
```

```
{
    switch (type)
    {
        case PowerupType.MoveSpeed: return "Швидкість";
        case PowerupType.ArrowLifetime: return "Час життя стріли";
        case PowerupType.FireRate: return "Швидкість стрільби";
        case PowerupType.ArrowSpeed: return "швидкість стріли";
        case PowerupType.TripleShot: return "Тріплшот";
        case PowerupType.DoubleShot: return "Подвоєння";
        case PowerupType.DoubleJump: return "Подвійний стрибок";
        default: return "Покращення";
```

1.11 Технічна архітектура гри

Усі ключові елементи коду були організовані за модульним принципом з чітким розділенням відповідальностей і впровадженням усталених патернів проектування.

У проєкті використано кілька класичних патернів проєктування, що дозволяють зробити архітектуру гнучкою, розширюваною та легко підтримуваною. Ключову роль у керуванні ігровим процесом відіграє патерн Singleton, реалізований у класі GameManager. Цей клас існує в єдиному екземплярі протягом усієї гри та забезпечує глобальний доступ до важливих даних — таких як статус поточного рівня, статистика гравця, налаштування чи активні параметри. Статичний доступ гарантує, що будь-яка частина проєкту може звертатися до GameManager без створення зайвих об’єктів, що особливо важливо для централізованого керування ігровим циклом.

Для забезпечення взаємодії між підсистемами без прямого зв’язування між ними використано патерн Observer. В його основі лежить EventManager — простий механізм розсилання подій, до якого можуть підписатися будь-які зацікавлені компоненти. Наприклад, при початку рівня або натисканні паузи EventManager надсилає відповідну подію, на яку реагують UIManager, EnemyManager, AudioManager та інші системи, виконуючи власну логіку. Такий підхід значно зменшує залежність між об’єктами та спрощує додавання нових функцій без потреби змінювати існуючий код.

Організація ігрових станів реалізована через патерн State Machine. Всі ключові етапи гри — головне меню, активна гра, пауза та завершення — представлені як окремі стани, кожен з яких має власні методи Enter() та Exit(). Завдяки цьому переходи між станами виконуються чітко і керовано, без великої кількості умов у головному коді. Це дозволяє легко змінювати логіку конкретного стану, не порушуючи загальну структуру гри.

Для створення ворогів, предметів або бонусів використано патерн Factory. Замість створення об'єктів вручну, проєкт використовує фабричні методи, наприклад, у класах EnemyManager або LootManager. Ці методи централізують налаштування параметрів при створенні нових об'єктів, що дозволяє легко змінювати логіку ініціалізації або додавати нові типи ворогів та луту без переписування великого обсягу коду.

Загалом, завдяки цим патернам проєкт має чітку модульну структуру, що дозволяє ефективно розвивати гру, змінювати внутрішню логіку окремих систем та зменшувати ризик помилок при масштабуванні.

Структура коду

```
Assets/
├── Scripts/
│   ├── Managers/       // GameManager, UIManager, EnemyManager, AudioManager
│   ├── Controllers/    // PlayerController, EnemyController
│   ├── Systems/        // StateMachine, EventManager, InputSystem
│   ├── UI/             // компоненти HUD і меню (HUDController, MenuController)
│   └── Utilities/      // корутини, розширювальні методи, атрибути
├── Scenes/             // MainMenu, Level1, Level2, ..., GameOver
└── Prefabs/            // префаби гравця, ворогів, бонусів, інтерфейсу
```

Кожен скриптовий файл відповідає за одну чітко окреслену зону відповідальності.

Взаємодія між компонентами гри реалізована через чітку послідовність ініціалізації, подій та переходів між станами, що забезпечує узгоджену роботу всіх підсистем. Після завантаження сцени метод Awake() у класі GameManager відповідає за створення та збереження посилань на всі основні менеджери —

таких як UIManager, EnemyManager, LootManager тощо. Це дозволяє централізовано керувати всіма ігровими системами вже з початку виконання.

Коли гравець починає гру, StateMachine ініціює перехід у стан Playing, надсилаючи відповідну подію через EventManager. Ця подія (OnGameStart) передається ключовим компонентам, серед яких PlayerController, EnemyManager та UIManager. Отримавши цю подію, кожен із них активує свої підсистеми: PlayerController починає обробку вводу, UIManager оновлює інтерфейс, а EnemyManager розпочинає генерацію ворогів.

У процесі геймплею взаємодія відбувається динамічно. PlayerController слухає події введення та, наприклад, у разі смерті гравця, генерує власну подію OnPlayerDeath, на яку можуть реагувати інші компоненти. EnemyManager після старту гри використовує фабрику для створення ворогів, а також підписується на події, пов’язані з їх знищенням або перезапуском рівня. LootManager і UIManager, у свою чергу, реагують на події типу OnEnemyDefeated — один відповідає за створення трофеїв, інший — за візуальне оновлення статистики чи індикаторів.

Коли гравець досягає мети рівня, LevelController генерує подію OnLevelComplete, яка сигналізує про завершення. У відповідь StateMachine змінює стан гри на GameOver, і UIManager автоматично викликає відображення відповідного меню з результатами або варіантами продовження. Уся ця система дозволяє компонентам працювати незалежно один від одного, взаємодіючи лише через події, що значно спрощує масштабування та обслуговування проєкту.

Завдяки такому підходу всі компоненти в системі взаємодіють між собою через узагальнені інтерфейси та подієву модель, що дозволяє уникнути жорсткого зв’язування між об’єктами. Це забезпечує низьку зв’язність (loose coupling), коли кожен елемент залишається максимально автономним, не покладаючись на конкретну реалізацію інших частин проєкту. У результаті досягається висока модульність: будь-яку нову підсистему — наприклад, новий тип UI, додатковий логгер чи систему ефектів — можна легко інтегрувати в проєкт, просто підписавши її на відповідні події через EventManager або інтерфейсну обгортку.

Такий підхід значно спрощує розширення функціональності, тестування окремих компонентів і подальшу підтримку гри без ризику порушити вже існуючу логіку.

Крім того, подієва модель забезпечує ефективну обробку асинхронних подій — наприклад, одночасне оновлення статистики, збереження прогресу та запуск анімацій після завершення рівня — без блокування основного потоку гри. Завдяки інкапсуляції логіки в окремих модулях (наприклад, GameStateController, RewardSystem, AudioManager), кожна система може еволюціонувати незалежно, не впливаючи на загальну стабільність.

У випадку командної розробки це дозволяє паралельну роботу над різними частинами проєкту без ризику конфліктів: один розробник може змінювати логіку переходів між рівнями, інший — UI або зберігання даних, не порушуючи єдності структури. Така архітектура також полегшує юніт-тестування, оскільки більшість компонентів мають чітко визначені точки входу/виходу й не залежать від конкретного стану середовища.

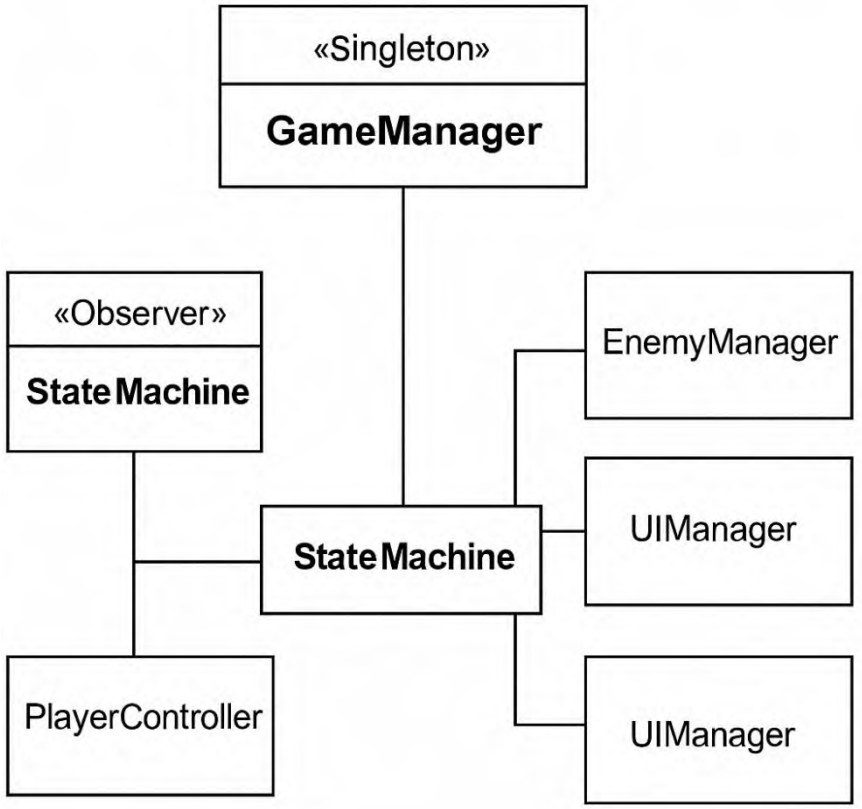


Рисунок 1.17 Схема прийняття рішень

Система апгрейдів надає гравцеві можливість вдосконалювати свого персонажа, використовуючи ігрову валюту - золото. Вся механіка реалізована через клас UpgradeManagerSimple, який керує меню покращень.

Ініціалізація системи відбувається при старті гри. Спочатку завантажуються збережені дані гравця:

```
gold = PlayerPrefs.GetInt("Gold", 0);
for (int i = 0; i < 5; i++) {
    upgradeLevels[i] = PlayerPrefs.GetFloat(statKeys[i], 0);
}
```

Інтерфейс меню складається з п'яти кнопок, кожна з яких відповідає за певний параметр. Назви та ефекти цих параметрів зберігаються в масивах:

```
private string[] statKeys = new string[5] {
    "MoveSpeed", "FireRate", "ArrowSpeed", "ArrowLifetime", "PlayerHP"
private float[] statIncrements = new float[5] {
    1f, -0.05f, 1f, 0.5f, 1f
```

Процес покращення запускається при натисканні кнопки. Система спочатку перевіряє наявність достатньої кількості золота:

```
if (gold >= cost) {
    gold -= cost;
    upgradeLevels[index] += statIncrements[index];
    PlayerPrefs.SetFloat(statKeys[index], upgradeLevels[index]);
    PlayerPrefs.SetInt("Gold", gold);
    UpdateUI();
```

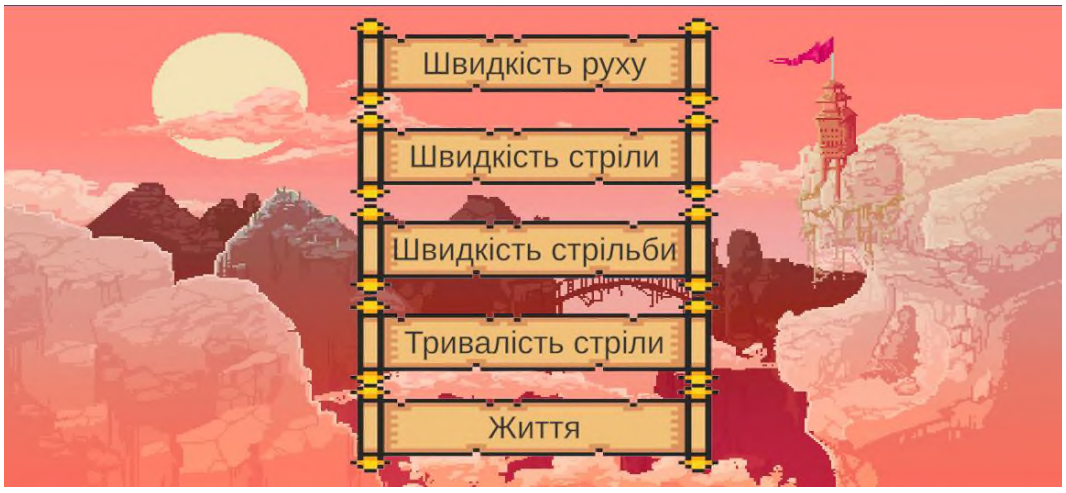


Рисунок 1.18 Меню апргрейд

Оновлення інтерфейсу відбувається через метод UpdateUI(), який формує інформаційні написи для кожної кнопки:

```
upgradeTexts[i].text = $"{name}\nРівень: {val:F1} + {inc}\nЦіна: {cost}";
```

Збереження прогресу реалізоване через систему PlayerPrefs, що дозволяє зберігати всі покращення між ігровими сесіями. Наприклад, після покращення швидкості руху значення зберігається так:

```
PlayerPrefs.SetFloat("MoveSpeed", upgradeLevels[index]);
```

Ця система формує прозору та інтуїтивно зрозумілу механіку вдосконалення персонажа, яка дозволяє гравцеві самостійно впливати на темп і напрямок прогресу. Кожне покращення не лише змінює характеристики героя, а й відкриває нові можливості взаємодії з ігровим світом — збільшує маневровість, бойову ефективність або виживання в складних ситуаціях. Завдяки цьому гравець має змогу обирати індивідуальний шлях розвитку, орієнтуючись на власні вподобання, тактичні рішення чи особливості проходження рівнів. Такий підхід підвищує залученість та створює мотивацію для повторного проходження гри з різними стилями поведінки.

Рисунок 1.19 Головне меню

Головне меню є основним інтерфейсом, з якого гравець починає взаємодію з грою. Воно реалізоване за допомогою класу MainMenu та містить чотири основні функції:

Метод PlayGame() завантажує перший рівень гри (сцену "LevelOne"), використовуючи вбудований менеджер сцен Unity. Це дозволяє гравцеві швидко розпочати нову гру.

Функція OpenUpgrades() відкриває окрему сцену "UpgradeScene", де гравець може витрачати ігрову валюту на покращення характеристик свого персонажа. Це створює додатковий шар стратегії у грі.

Метод OpenSettings() може або завантажувати окрему сцену з налаштуваннями ("SettingsScene"), або активувати відповідну панель у поточному інтерфейсі. Тут гравець може регулювати гучність, графіку та інші параметри.

Функція QuitGame() коректно завершує роботу додатка. У редакторі Unity вона також виводить повідомлення "Гра закрита" до консолі для налагодження.

1.12 Тестування створеної системи

Тестування є завершальним етапом життєвого циклу програмного забезпечення та забезпечує перевірку відповідності створеного продукту його технічним вимогам і очікуванням користувача. У процесі розробки гри тестування проводилося поступово, паралельно з реалізацією ключових модулів, а наприкінці – комплексно, у формі інтеграційного та функціонального тестування в середовищі Unity.

Основна мета тестування полягала у виявленні помилок, нестабільної поведінки, логічних суперечностей у роботі скриптів та загальної ігрової логіки. Оскільки гра реалізовувалась у стилі 2D-платформера з елементами roguelike, особлива увага приділялася як базовим механікам керування та взаємодії, так і системам випадкового спавну, перманентної смерті, генерації луту та мета-прогресії.

Тестування здійснювалося переважно в режимі Play Mode в Unity, що дозволяло миттєво аналізувати зміни в поведінці об’єктів після модифікації скриптів або конфігурації. Для відслідковування процесів використовувалися вбудовані засоби Unity: консоль логів, профайлер, інструменти анімації (Animator Debug), а також додаткові поля в інспекторі, які давали змогу вручну змінювати параметри та проводити експерименти.

Однією з перших протестованих систем була логіка руху головного персонажа, включно зі стрибками, падінням, взаємодією з платформами та стінами. Було підтверджено, що стан «на землі» визначається коректно, а активація подвійного стрибка працює лише у випадку, якщо гравець перебуває у повітрі після першого стрибка. Особливу увагу приділено механіці «прилипання» до стіни, де гравець уповільнює падіння і має можливість відштовхнутися. У кількох тестових сесіях виникали ситуації, коли при недостатньо точному колайдері платформи стан «OnWall» не активувався — проблема була вирішена за допомогою збільшення площі сенсорної зони перевірки наявності стіни.

Важливою частиною тестування була перевірка системи бою. Вона включала ближню атаку, застосування здібностей із cooldown, нанесення шкоди ворогам та відображення візуального фідбеку. Під час тестування було виявлено, що при занадто частому натисканні кнопки атаки, функція ApplyDamage() викликалася декілька разів у межах одного кадру, створюючи зайве зниження здоров’я ворога. Це було виправлено додаванням захисту через фреймову затримку між спробами влучання.

Окрема серія тестів проводилась над поведінкою ворогів**,** які мають різну логіку. Для PatrolEnemy було перевірено алгоритм патрулювання між точками та адекватне реагування на появу гравця у полі зору. ChargerEnemy виконував ривок при досягненні гравця, однак на ранньому етапі тестування іноді застрягав у стінах — це пояснювалося неправильним обрахунком нормалізованого вектора та було усунено введенням додаткової перевірки на відстань перед ривком. RangedEnemy стабільно стріляв у гравця, але в одному з випадків не відбувалася ініціалізація player.position, що призводило до помилки NullReference. Після уточнення методу Start() та введення умови на наявність гравця проблема була вирішена.

Тестування елементів roguelike — зокрема, системи випадкового спавну предметів та ворогів — проводилося з використанням багаторазових запусків рівня. Була створена спеціальна тестова сцена з прискореним стартом гри, де RoomSpawner кожного разу обирав різні шаблони кімнат. Зокрема, аналізувалася

частота випадання предметів різної рідкісності, яка відповідала ймовірностям, заданим у таблицях. У кількох випадках повторне відкриття скрині дозволяло отримати додаткові бонуси — після цього була реалізована змінна isOpened, що блокує механіку повторного спавну.

Значну увагу було приділено мета-прогресії. Було перевірено, що після смерті гравець втрачає прогрес у рівні, однак ресурси (наприклад, монети) зберігаються у GameManager для подальшого використання. Цей механізм проходив перевірку шляхом багаторазового повторення рівнів із цілеспрямованою загибеллю гравця.

У рамках тестування UI було досліджено правильність відображення індикаторів та повідомлень. Контролер TextMeshPro стабільно виводив повідомлення про завершення рівня, втрату здоров’я, отримання бонусів. Також був проведений аналіз продуктивності: навіть при наявності великої кількості ворогів, HUD та анімаційна система працювали без збоїв і в межах оптимального FPS. Unity Profiler підтвердив, що завантаження пам’яті залишається в межах допустимих значень, а піки CPU-навантаження виникають лише під час одночасного рендерингу великої кількості об’єктів, що не є критичним.

У підсумку, всі основні функціональні модулі гри були протестовані на відповідність очікуваним сценаріям поведінки. Завдяки модульній структурі коду й використанню шаблону Observer усі компоненти залишаються слабо зв’язаними, що значно спростило виявлення та усунення помилок. Попри незначні виявлені недоліки (наприклад, помилки при надмірному використанні здібностей або відсутності об’єкта гравця в окремих викликах), загальна стабільність гри підтверджена, що дозволяє розглядати її як завершений та придатний до подальшого використання ігровий продукт.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

Метою цієї роботи стало створення повноцінного 2D-платформера з елементами roguelike, що поєднує у собі класичну структуру рівнів і випадкові події, забезпечуючи варіативність проходження та інтерес гравця в довготривалій перспективі. Розробка гри стала цінним досвідом комплексного підходу до реалізації цифрового продукту.

У цьому розділі представлено розрахунок вартості розробленого програмного продукту, ефективність якого визначається не лише його якісними властивостями, а й ефективністю процесу розробки. Якість ПП оцінюється за кількома основними критеріями: з урахуванням потреб і вражень користувачів, раціональності використання ресурсів та відповідності заданим вимогам. З точки зору користувача, оцінка якості включає також аналіз витрат на розробку, зокрема обсяг трудових витрат і загальні витрати.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки залежить від ряду факторів, включаючи обсяг робіт, рівень складності, кваліфікацію команди розробників та часові обмеження, зумовлені ринковими умовами. Для оцінки масштабу програмного продукту використовується метод структурної аналогії, що передбачає звернення до спеціалізованих каталогів схожих проєктів. На їхній основі визначається кількість умовних машинних команд, необхідних для реалізації аналогічного програмного рішення (у тисячах команд).

Таблиця 2.1 Каталог аналогів

Найменування ПП	**Обсяг функції ПП – V_o, умовн. машинних командах.**
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП оптимізації розрахунків	7800 – 8200

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, трудомісткості визначати на основі табл.2.2

Таблиця.2.2 Норма часу

Обсяг ПЗ, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 306 х 0,8 = 244,8 (люд/годин).

Трудомісткість програмного продукту визначається окремо для кожного етапу розробки, виходячи з показників трудомісткості відповідного аналога. При цьому враховують рівень складності розробки, ступінь інноваційності та частку використання стандартних модулів. Розрахунки проводяться згідно з наступними формулами:

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

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

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

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 $К_н$**
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП 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 ПП типовими програмами, %	**Значення $К_т$**
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

$$Т_{тз}=Т^а * L_1 * К_н = 244,8 * 0,12 * 0,7 = 20,56 \text{ (люд/годин)} \quad (2.4)$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Т^а * L_2 * К_н = 244,8 * 0,11 * 0,7 = 18,85 \text{ (люд/годин)} \quad (2.5)$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Т^а * L_3 * К_н * К_т = 244,8 * 0,61 * 0,7 * 0,6 = 62,72 \text{ (люд/годин)} \quad (2.6)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$= 1 (стр), розробка ТП $N_{тп=}$30(стр), розробка робочого проекту $N_{рп}$=12(стр), пояснювальна записка відповідно $N_{пз}$=29(стр) Розрахунок зведений у таблицю 2.6.

Таблиця 2.6 Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, годин**		
1.ТЗ	$T_{Ртз}$=20,56	$T_{кк}=0,7*N_{тз}$= 0,7*1=0,7	$T_{нк}=0,15*N_{тз}$=0,15*1=0,15
2.Розробка ТП	$T_{Ртп}$=18,85	$T_{кк}=0,7*N_{тп}$=0,7*30=21	$T_{нк}=0,15*N_{тп}$=0,15*30=4,5
3.Розробка РП	$T_{Ррп}$= 62,72	$T_{кк}=0,7*N_{рп}$=0,7*12=8,4	$T_{нк}=0,15*N_{рп}$=0,15*12=1,8
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}$= 1,5*29=43,5	$T_{кк}=0,7*N_{тз}$=0,7*29=20,3	$T_{нк}=0,15*N_{пз}$=0,15*29 =4,25
Усього, в т.ч.:	206,73		
- на розробку	$\Sigma T_{р}$=145,63		
- контроль керівника		$\Sigma T_{кк}$=50,4	
- нормоконтроль			$\Sigma T_{нк}$=10,7

2.3 Розрахунок ціни програмного продукту

Для оцінки вартості програмного продукту розглядаються основна заробітна плата виконавців, матеріальні витрати та загальні витрати на розробку ПЗ. Детальний розрахунок основної заробітної плати наведено у таблиці 2.7. Згідно зі статтею 8 «Закону про Державний бюджет України на 2025» з 1 січня 2025 року встановлено мінімальну місячну заробітну плату у розмірі 8000 гривень, а також 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	145,63	48,00	6990,24
2.Контроль керівника	50,4	90,00	4536,00
3.Нормоконтроль	10,7	95,00	1016,5
Усього	-	-	ΣЗо= 12542,74

Зробимо розрахунок матеріальних витрат на розробку ПП (табл.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	72	5.0	360,00
Разом	-	-	-	$В_{м\,і}$ =360,00
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$ = 0,1*285=28,5
Усього				$В_м = В_{мі} + В_{тр\_з}$ = 313,50

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	396,0	$В_м$ (див. табл. 2.8.)
2. Основна заробітна плата	12542,74	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	1254,27	$Зд = 0{,}1 \times Зо$ = 12542,74*0,1
4.Відрахування до єдиного фонду соціального внеску	3035,34	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$ =0,22* (12542,74+1254,27)
5. Накладні витрати	5017,09	$Внак. = 0{,}4 \times Зо$ =0,4 * 12542,74
6. Повна собівартість	22245,44	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 396,0+12542,74+1254,27+3035,34 +5017,09

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

$П=(С_{пов}*Р)/100=(22245,44*10)/100=2224,54$ грн (2.7)

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

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_{пов}+П$ = 22245,44+2224,54 = 24469,98 грн; (2.8)

Ціна реалізації розробленого програмного забезпечення становитиме:

$Ц_р=Ц_о+ПДВ$ = 24469,98 + 24469,98 *0.2 = 29363,98 грн; (2.9)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Закон України «Про охорону праці» є одним із ключових нормативних актів, що регламентує захист життя та здоров'я громадян у процесі трудової діяльності. Він визначає основні принципи організації безпечних умов праці, регулює відносини між роботодавцем і працівником у питаннях гігієни, охорони праці та виробничого середовища, а також встановлює єдині стандарти охорони праці на території України.

У дипломному проєкті розглядається процес Розробка 2D-платформера з елементами roguelike у ICP Unity

3.1 Аналіз небезпечних і шкідливих факторів

У процесі розробки пристрою для калібрування комп'ютерних моніторів важливо враховувати фактори, що можуть впливати на безпеку працівника. До таких належать електромагнітне випромінювання, можливі перепади електричної напруги, інтенсивність освітлення робочого місця, а також тепловий вплив при проведенні пайки електронних компонентів. Робоче середовище має бути організоване таким чином, щоб мінімізувати вплив шкідливих чинників та забезпечити комфортні умови праці.

3.2 Гігієнічні вимоги до виробничого середовища

Для забезпечення ефективної роботи над калібрувальним пристроєм необхідно враховувати умови виробничого середовища. Освітлення має відповідати нормам (300–400 лк згідно з ДБН В.2.5-28:2018), а робоче приміщення повинно бути обладнане вентиляцією для регулювання температури та рівня вологості. Важливо забезпечити зручне розташування робочого місця, захист від шуму та оптимальні санітарні умови.

Створення сприятливого робочого середовища є важливим аспектом продуктивної діяльності персоналу. Гігієнічні вимоги передбачають низку умов, які мають бути забезпечені у приміщенні, де здійснюється розробка та тестування пристрою.

Освітлення Освітлення робочого місця повинно відповідати встановленим нормативам. Згідно з ДБН В.2.5-28:2018, необхідно забезпечити рівень освітленості 300–400 лк, що дозволить зменшити навантаження на зір та покращити точність роботи з дрібними компонентами пристрою.

Вентиляція та якість повітря Робоче приміщення повинно мати ефективну вентиляцію, щоб усувати шкідливі пари та забезпечувати доступ свіжого повітря. У холодну пору року рекомендована температура 18–20°C, у теплу – 22–25°C. Оптимальний рівень вологості складає 40–60%, що сприяє комфортному перебуванню у приміщенні.

Захист від шуму та вібрацій У місцях, де проводяться пайкові роботи та тестування пристрою, слід мінімізувати рівень шуму та вібрацій, які можуть негативно впливати на продуктивність працівників. Для цього застосовують шумоізолюючі матеріали та спеціальні амортизаційні конструкції.

3.3 Вимоги безпеки праці працівника

Безпека працівника під час роботи з паяльними інструментами та електронними пристроями є першочерговим завданням. Необхідно дотримуватися таких заходів:

Використання індивідуальних засобів захисту – працівник повинен працювати у захисних рукавичках, спеціальному одязі та з використанням ізоляційного покриття на робочій поверхні.

Дотримання правильного розташування робочого місця – важливо розташувати технічне обладнання таким чином, щоб уникнути ризику перекидання чи випадкового контакту з нагрітими частинами.

Контроль електробезпеки – всі пристрої, які використовуються в роботі, повинні мати заземлення та відповідати технічним стандартам безпеки.

Дотримання правил експлуатації обладнання – працівник повинен перевіряти справність інструментів перед початком роботи та уникати використання пошкодженого обладнання.

3.4 Правила безпеки праці

При виконанні паяльних робіт слід дотримуватися таких правил:

- Забороняється використання несправних інструментів.
- Не можна торкатися до нагрітих частин паяльника, щоб уникнути опіків.
- Обов’язкове використання витяжки для видалення шкідливих парів припою.
- Деталі утримувати плоскогубцями або спеціальними інструментами, щоб уникнути прямого контакту з гарячими компонентами.
- Регулярно провітрювати приміщення та дотримуватися санітарних норм після завершення роботи.

3.5 Пожежна безпека

Пожежна безпека є одним із критичних аспектів організації робочого місця, особливо при роботі з електронними пристроями, такими як система калібрування моніторів.

Основними причинами виникнення пожеж у виробничому приміщенні можуть бути:

- Несправність електрообладнання – коротке замикання, перевантаження електромережі та механічні пошкодження електрокабелів.
- Неправильне зберігання легкозаймистих матеріалів – відкриті ємності з хімічними речовинами, займисті припої та ізоляційні матеріали.
- Порушення техніки безпеки при пайці – попадання розплавленого припою на горючі матеріали, перегрів електропаяльників та залишення нагрітого обладнання без нагляду.
- Недотримання правил експлуатації електромереж – використання несправних розеток, відсутність захисного заземлення та неправильне підключення обладнання.
- Необережне поводження з вогнем – використання відкритого полум’я у

робочому приміщенні, паління та неправильне поводження з нагрітими предметами.

Для запобігання пожежам необхідно дотримуватися ряду заходів безпеки:

– Систематичний контроль електромережі – перед початком роботи слід перевірити справність розеток, проводів та електроприладів.
– Забезпечення робочого приміщення засобами пожежогасіння – кожне місце роботи повинно бути оснащене необхідними протипожежними засобами, такими як:
– Вогнегасник (порошковий або вуглекислотний, залежно від специфіки приміщення).
– Азбестове покриття для гасіння невеликих локальних займання.
– Ящик з піском об’ємом не менше 0,5 м³ для ліквідації розливів рідких займистих речовин.
– Лопати та відра для ефективного використання піску.

Рисунок 3.1. Пожежні щити

Пожежні щити мають бути розміщені на видимих місцях та містити необхідний набір засобів для оперативної ліквідації займання.

Щоб мінімізувати ризик виникнення пожеж, робоче місце має відповідати наступним вимогам:

– Запасні виходи повинні бути позначені світловими покажчиками із написом «Запасний вихід», видимими навіть при недостатньому освітленні.

– Пожежні крани повинні бути доступними на кожному поверсі, у коридорах та біля сходових клітин.

– Вогнегасники слід розміщувати на видимих місцях, на висоті не більше 1,5 м від підлоги для швидкого доступу.

– Евакуаційний план повинен бути розміщений у головному вході приміщення та містити детальний маршрут виходу при пожежі.

– Будівлі та приміщення повинні бути оснащені пожежними щитами з необхідним інструментом для ліквідації загоряння.

– Електромережа повинна відповідати нормам захисту – дроти та розетки повинні бути ізольованими та не перевантаженими.

– Забезпечення контрольованого доступу до виробничих приміщень – стороннім особам забороняється перебувати в робочій зоні без відповідного дозволу.

Додаткові заходи безпеки:

- Регулярні навчання—проведення інструктажів для працівників щодо дій у надзвичайних ситуаціях.
- Моніторинг пожежної безпеки—періодичне тестування пожежної сигналізації та огляд стану вогнегасників.
- Організація шляхів евакуації—забезпечення безперешкодного проходу до аварійних виходів без зайвих перешкод.
- Перевірка вентиляційних систем

ВИСНОВКИ

Проведена розробка продемонструвала, що навіть у межах обмеженого обсягу контенту та статичних рівнів можливо створити гру, яка забезпечує відчуття новизни й виклику за кожної нової спроби. Реалізовані механіки випадкового спавну, перманентної смерті та покрокового вдосконалення персонажа дозволили досягти ключових характеристик жанру roguelike. Технічне впровадження було побудоване з урахуванням сучасних підходів до архітектури коду, з чітким розподілом відповідальностей між модулями та використанням патернів проектування, що забезпечує розширюваність та зручність підтримки проєкту.

Unity виявився ефективним середовищем для втілення ігрових ідей завдяки гнучкості, великій кількості готових рішень та активній спільноті. Отриманий результат свідчить про те, що поєднання класичних ігрових форм з елементами випадковості та експериментальності дозволяє створювати продукти, здатні зацікавити широку аудиторію гравців і дати розробнику платформу для подальшого вдосконалення.

Практичний результат, отриманий у ході реалізації дипломного проєкту, не лише продемонстрував актуальність обраного підходу, а й дозволив закріпити знання, пов’язані з роботою у рушії Unity, застосуванням C# у контексті ігрової логіки, використанням анімації, UI-системи та взаємодії об'єктів через події. Розробка гри стала цінним досвідом комплексного підходу до реалізації цифрового продукту.. Особливу увагу приділено оптимізації продуктивності та адаптації гри під різні платформи, що є критично важливим для сучасного ігрового середовища. Отриманий досвід можна успішно застосовувати не лише в ігровій індустрії, а й у ширшому колі задач розробки інтерактивних застосунків, що підтверджує практичну цінність проєкту.

У майбутньому даний проєкт може бути легко доповнений новими рівнями, розширеною генерацією, мережею, підтримкою збережень або мобільною версією. Головне, що він закладає стабільний фундамент для розвитку — як у напрямку розширення функціоналу, так і в контексті поглибленого вивчення професійних стандартів ігрової індустрії.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ферроне, Г. Вивчаємо C# через розробку ігор на Unity – К.: Packt Publishing, 2022. – 342 с.
2. Прайс, М. C# 10 та .NET 6. Сучасна кросплатформова розробка – К.: Видавництво "Фахівець", 2023. – 820 с.
3. Салтан, Б., Нероба, Ю. Патерни програмування в Unity: практичний підхід – К.: SUITSME Press, 2023. – 288 с.
4. Гонсалес Віво, П. Книга шейдерів: графіка в Unity – Львів: Holy Water Books, 2022. – 260 с.
5. Ністрем, Б. Патерни програмування ігор – Харків: GameDev UA, 2023. – 320 с.
6. Скіт, Дж. C# in Depth: тонкощі програмування – К.: Меннінг Україна, 2024. – 528 с.
7. Троєлсен, Е., Джепікс, Ф. Pro C# 9 з .NET 5 – К.: Апрес Україна, 2023. – 1372 с.
8. Альбахарі, Дж. Кишеньковий довідник C# 8.0 – К.: О'Райлі Україна, 2022. – 240 с.
9. Стеллман, Е., Грін, Дж. Head First C# – К.: О'Райлі Україна, 2023. – 800 с.
10. Вітакер, Р. Б. Посібник гравця C# – К.: Starbound Software, 2023. – 406 с.
11. Ріхтер, Д. CLR за допомогою C# – К.: Microsoft Press Україна, 2022. – 896 с.
12. Мартін, Р. Чистий код – К.: Програміст UA, 2023. – 464 с.
13. Васильєв, О. Програмування на C# для початківців – К.: ITProger Press, 2023. – 312 с.

ДОДАТОК А. Лістинг програми

Arow.cs

```
using UnityEngine;

public class Arrow : MonoBehaviour
{
    public float speed = 10f;
    public float lifetime = 3f;

    private Rigidbody2D rb;

    public void Launch(Vector2 direction)
    {
        rb = GetComponent<Rigidbody2D>();
        rb.linearVelocity = direction.normalized * speed;

        Destroy(gameObject, lifetime);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
            return;

        if (other.CompareTag("Enemy"))
        {
            other.GetComponent<EnemyBalloon>()?.TakeDamage(1);
        }

        Destroy(gameObject);
    }
}
```

Chest.cs

```
using UnityEngine;
using UnityEngine.InputSystem.XR;

public class Chest : MonoBehaviour
{
    public Sprite closedSprite;
    public Sprite openSprite;
    public float interactionDistance = 2f;

    private SpriteRenderer sr;
    private bool isOpened = false;
    private Transform player;

    private Color defaultColor;
    public Color highlightColor = Color.yellow;

    void Start()
    {
        sr = GetComponent<SpriteRenderer>();
        sr.sprite = closedSprite;
        defaultColor = sr.color;

        player = GameObject.FindGameObjectWithTag("Player").transform;
    }

    void Update()
    {
```

```
        if (player == null || isOpened) return;

        float dist = Vector2.Distance(transform.position, player.position);

        if (dist <= interactionDistance)
        {
            sr.color = highlightColor;

            if (Input.GetKeyDown(KeyCode.U))
            {
                OpenChest();
            }
        }
        else
        {
            sr.color = defaultColor;
        }
    }

    private void OpenChest()
    {
        isOpened = true;
        sr.sprite = openSprite;
        sr.color = defaultColor;

        GrantRandomPowerup();
    }

    private void GrantRandomPowerup()
    {
        PowerupType[] allTypes =
(PowerupType[])System.Enum.GetValues(typeof(PowerupType));
        int index = Random.Range(0, allTypes.Length);
        PowerupType chosen = allTypes[index];

        PlayerController pc = player.GetComponent<PlayerController>();
        pc.ApplyPowerup(chosen);

        UIController.Instance?.ShowMessage("Получено улучшение: " +
GetPowerupName(chosen));
    }

    private string GetPowerupName(PowerupType type)
    {
        switch (type)
        {
            case PowerupType.MoveSpeed: return "Скорость";
            case PowerupType.ArrowLifetime: return "Время жизни стрелы";
            case PowerupType.FireRate: return "Скорость стрельбы";
            case PowerupType.ArrowSpeed: return "Скорость стрелы";
            case PowerupType.TripleShot: return "Тройной выстрел";
            case PowerupType.DoubleShot: return "Удвоенный выстрел";
            case PowerupType.DoubleJump: return "Двойной прыжок";
            default: return "Улучшение";
        }
    }
}
```

EnemyBalloon.cs

```
using UnityEngine;

public class EnemyBalloon : MonoBehaviour
{
```

```
    public float moveSpeed = 3f;
    public float fireRate = 2f;
    public GameObject projectilePrefab;
    public Transform firePoint;

    private Transform player;
    private float fireTimer;
    private bool isVisible = false;

    public void TakeDamage(int amount)
    {
        Destroy(gameObject);
    }

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player")?.transform;
    }

    void Update()
    {
        if (player == null) return;

        Vector2 direction = (player.position - transform.position).normalized;
        transform.position += (Vector3)direction * moveSpeed * Time.deltaTime;

        if (isVisible)
        {
            fireTimer += Time.deltaTime;
            if (fireTimer >= fireRate)
            {
                ShootAtPlayer();
                fireTimer = 0f;
            }
        }
    }

    void ShootAtPlayer()
    {
        if (player == null) return;

        GameObject proj = Instantiate(projectilePrefab, firePoint.position,
Quaternion.identity);
        Vector2 shootDir = (player.position - firePoint.position).normalized;
        proj.GetComponent<EnemyProjectile>()?.Launch(shootDir);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            other.GetComponent<PlayerController>()?.TakeDamage(1);
        }
    }

    void OnBecameVisible() { isVisible = true; }
    void OnBecameInvisible() { isVisible = false; }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
```

```
[System.Serializable]
public class UpgradeData
{
    public string statName;
    public float value;
    public float increment;
    public int cost;
}

public class UpgradeManager : MonoBehaviour
{
    public UpgradeData[] upgrades;
    public Button[] upgradeButtons;
    public TextMeshProUGUI[] upgradeTexts;
    public TextMeshProUGUI goldText;

    private int gold;

    void Start()
    {
        gold = PlayerPrefs.GetInt("Gold", 0);
        LoadUpgrades();
        UpdateUI();
    }

    public void ApplyUpgrade(int index)
    {
        if (index >= upgrades.Length) return;

        UpgradeData upgrade = upgrades[index];

        if (gold >= upgrade.cost)
        {
            gold -= upgrade.cost;
            upgrade.value += upgrade.increment;
            upgrade.cost = Mathf.CeilToInt(upgrade.cost * 1.5f);

            PlayerPrefs.SetFloat(upgrade.statName, upgrade.value);
            PlayerPrefs.SetInt($"{upgrade.statName}_Cost", upgrade.cost);
            PlayerPrefs.SetInt("Gold", gold);

            UpdateUI();
        }
    }

    void LoadUpgrades()
    {
        for (int i = 0; i < upgrades.Length; i++)
        {
            upgrades[i].value = PlayerPrefs.GetFloat(upgrades[i].statName,
upgrades[i].value);
            upgrades[i].cost = PlayerPrefs.GetInt($"{upgrades[i].statName}_Cost",
upgrades[i].cost);
        }
    }

    void UpdateUI()
    {
        goldText.text = $"Золото: {gold}";

        for (int i = 0; i < upgrades.Length; i++)
        {
```

```
            upgradeTexts[i].text = $"{upgrades[i].statName}\n" +
                                   $"Рівень: {upgrades[i].value:F1}
{upgrades[i].increment}\n" +
                                   $"Ціна: {upgrades[i].cost}";
        }
    }
}
using UnityEngine;
using TMPro;
using System.Collections;

public class UIController : MonoBehaviour
{
    public static UIController Instance;
    public TextMeshProUGUI messageTMP; using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance;
    [SerializeField] private GameObject gameOverPanel;

    void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    public void ShowGameOver()
    {
        if (gameOverPanel != null)
            gameOverPanel.SetActive(true);
    }

    public void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
    public float messageDuration = 2f;

    void Awake()
    {
        if (Instance == null) Instance = this;
        else Destroy(gameObject);
    }

    public void ShowMessage(string message)
    {
        StopAllCoroutines();
        StartCoroutine(ShowMessageRoutine(message));
    }

    private IEnumerator ShowMessageRoutine(string message)
    {
        messageTMP.text = message;
        messageTMP.enabled = true;
        yield return new WaitForSeconds(messageDuration);
        messageTMP.enabled = false;
```

Розробка 2D-платформера з елементами roguelike у Unity

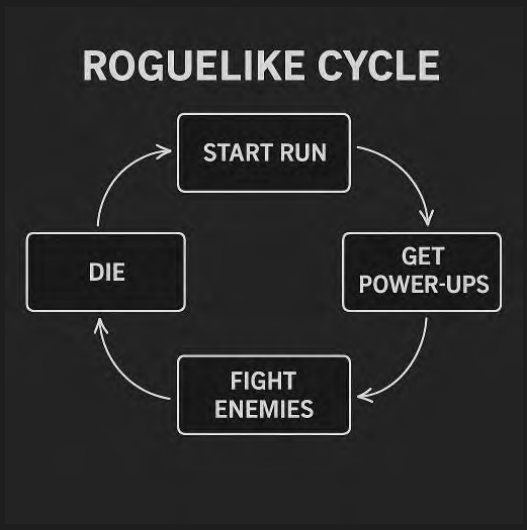
Осипенко Владислав Олегович
Керівник: Шувалова І.О.
ОТФК ОНТУ, 2025

Концепція гри

Винагорода

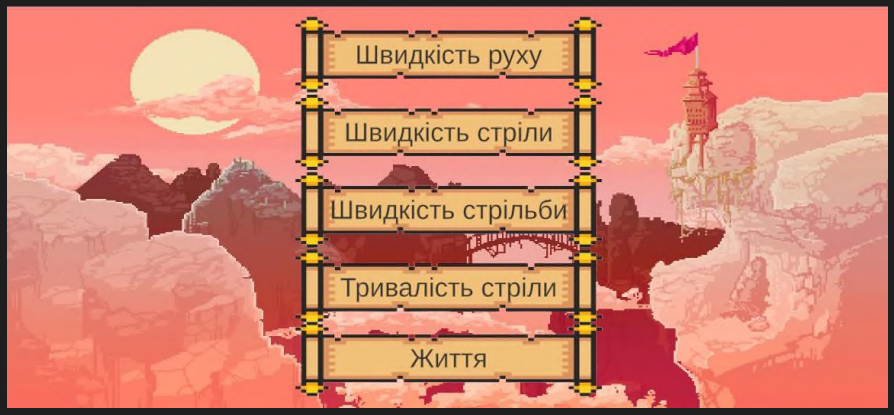
Прокачка

Битва



Ігрові механіки

Система прокачки та прогресії



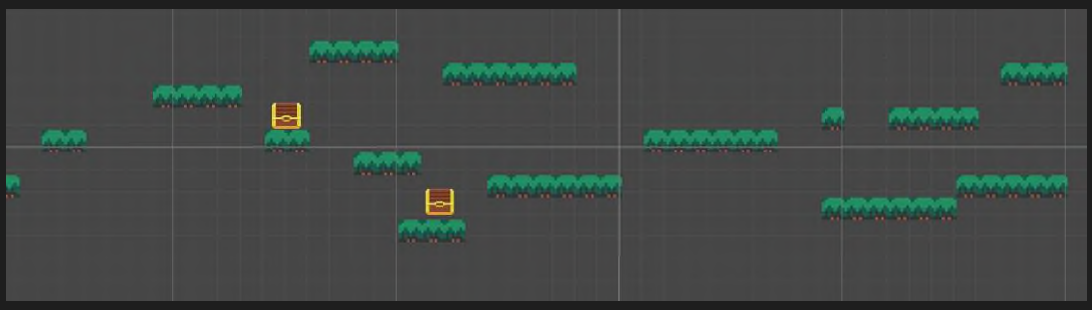
Інтерфейс та HUD

TextMeshPro

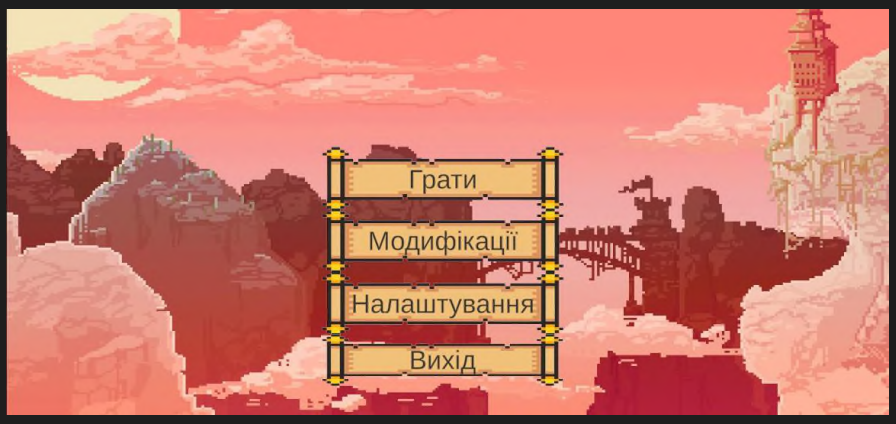
HUD

Gameover

Рівні та середовище



Меню, навігація та структура сцен



Тестування функціоналу

Висновки

У ході дипломного проєкту було реалізовано повноцінну 2D-гру з елементами roguelike на рушії Unity, що поєднує класичні механіки платформера з випадковістю та повторюваністю проходжень. Завдяки структурованому підходу вдалося досягти балансу між гнучкістю геймплею та стабільністю архітектури.

У процесі роботи:

- Розроблено і протестовано ключові ігрові механіки: рух, стрибки, бойову систему, генерацію ворогів і предметів.
- Впроваджено систему перманентної смерті та мета-прогресії, що підвищує реіграбельність.
- Застосовано стилізацію у форматі піксель-арт, яка забезпечує візуальну виразність при високій продуктивності.
- Використано можливості Tilemap, Animator Controller, ScriptableObjects та інші компоненти Unity для ефективної реалізації логіки гри.

Проєкт підтвердив здатність самостійно реалізувати складний програмний продукт, а також продемонстрував високий рівень знань у галузі геймдизайну, програмування та UX/UI. Створена гра може бути основою для подальшого розвитку, включаючи багатокористувацький режим, глибші RPG-елементи або портування на інші платформи.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Осипенка Владислава Олеговича

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Шувалова Ірина Олегівна*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка 2D-платформера з елементами roguelike у ICP Unity*

Обсяг розрахунково-пояснювальної записки *72* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений Розробка 2D-платформера з елементами roguelike у ICP Unity та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 13 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Охоплено всі етапи розробки 2D-платформера з елементами roguelike у ICP Unity.*

Забезпечено взаємодію різних об'єктів, гравця, та Ворогів у динамічному рівні

д) основні недоліки дипломного проекту *Немає звукового оформлення. Одноманітні вороги у грі. Слабка графічна частина проекту. Присутні деякі недоліки оформлення пояснювальної записки*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

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

Підпис: _____

«23» _____ 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Осипенко Владислав Олегович

(прізвище, ім`я та по батькові)

Спеціальність: *121 «Інженерія програмного забезпечення»*

Освітня програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Розробка 2D-платформера з елементами roguelike у ICP Unity*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 71 сторінок. У пояснювальній записці описано етапи Розробка 2D-платформера з елементами roguelike у ICP Unity. Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Осипенко Владислав поступово та послідовно виконував всі етапи, проявив ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Осипенко Владислав під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою.*

Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Осипенко Владислав показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування та математики, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як Microsoft Visual Studio, Qt, Microsoft PowerPoint, Microsoft Visio та ін.*

Оцінка розрахункової частини *Задовільно*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проекту *Шувалова Ірина Олегівна*

Місце роботи і посада керівника дипломного проекту *ВСП «Одеський технічний фаховий коледж ОНТУ», викладач спецдисциплін циклової комісії комп'ютерної техніки 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,

Осипенко Владислав Олегович
здобувач освіти гр. 4РП-08, та

Шувалова Ірина Олегівна,
керівник дипломного проекту,

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

«Розробка 2D-платформера з елементами roguelike у ICP Unity»
(автор роботи – Осипенко В.О., керівник роботи – Шувалова І.О.)

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 _____ / Осипенко В.О. /

Керівник _____ / Шувалова І.О. /

«16» червня 2025 р.

ДОВІДКА

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

Осипенка Владислава Олеговича

на тему *Розробка 2D-платформера з елементами roguelike у ICP Unity*

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

(підпис) *20.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 20.06.2025 р. значення коефіцієнту подібності в роботі становить 12,72%, коефіцієнт цитування – 1,30%.*

(підпис) *20.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Осипенка В.О.* (П.І.Б.)

проведена « *20* » *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.)

StrikePlagiarism.com

Дата звіту 6/20/2025
Дата редагування 6/20/2025

Документ прийнятий

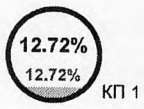
Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка 2D-платформера з елементами roguelike у ICP Unity
Автор Науковий керівник / Експерт
Осипенко Владислав ОлеговичШувалова Ірина Олегівна
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	13779	114566
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		24
Інтервали		0
Мікропробіли		47
Білі знаки		1322
Парафрази (SmartMarks)		114

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	130 0.94 %
2	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	70 0.51 %
3	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	58 0.42 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	52 0.38 %
5	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	50 0.36 %

6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	48 0.35 %
7	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	40 0.29 %
8	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	38 0.28 %
9	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	37 0.27 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	32 0.23 %

з домашньої бази даних (0.63 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка мобільного застосунку-помічника майстра манікюру [redacted] 6/18/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	87 (8) 0.63 %

з програми обміну базами даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з Інтернету (12.08 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	553 (49) 4.01 %
2	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content**	**336 (10) 2.44 %**
3	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**168 (6) 1.22 %**
4	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**121 (5) 0.88 %**
5	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**116 (11) 0.84 %**
6	**https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download**	**56 (2) 0.41 %**
7	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**46 (2) 0.33 %**
8	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	46 (4) 0.33 %
9	**https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download**	**45 (2) 0.33 %**
10	https://docs.mgu.edu.ua/docs/bibliteka/Kudritskyy.pdf	34 (4) 0.25 %
11	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	31 (4) 0.22 %
12	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	27 (2) 0.20 %
13	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	22 (1) 0.16 %
14	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	21 (1) 0.15 %
15	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	19 (1) 0.14 %
16	https://card-file.ontu.edu.ua/bitstreams/11562741-24e6-4201-bc41-a00c8013fca1/download	12 (1) 0.09 %
17	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c63b91ba-d04f-4715-890d-b16277695c7e/content	6 (1) 0.04 %

18 https://cpsm.kpi.ua/stud/bak/Gulaevich_LP_71.pdf 6 (1) 0.04 %

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: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: **«Розробка програмного забезпечення»**
Група: 4РП- 08

Дипломний проєкт
здобувача освіти денної форми навчання
РП. 08.13.000.ДП

ОСИПЕНКА
ВЛАДИСЛАВА ОЛЕГОВИЧА

м. Одеса
2025 **р.**

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

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка програмного забезпечення»
Група: 4РП- 08

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту на тему:
_____
_____
Проектний матеріал складається з пояснювальної записки на _____ сторінках та
графічного (презентаційного) матеріалу на _____ аркушах (слайдах)

Дипломник _____ (Осипенко **В.О.**) Керівник _____
(Шувалова І.О.)

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** відділенням
_____ (**Краснокутська К.Г.**)

Захист «____»_____ 2025 р. Протокол ЕК No_____
Оцінка ЕК _____

Секретар ЕК _____
Розробка 2D-платформера з елементами roguelike у ICP Unity

11
70

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

Відділення _____ Комісія _____
Спеціальність _____
Освітньо-професійна програма _____ **ЗАТВЕРДЖУЮ: Заст.**
дир. з НВР _____