ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

(рівень вищої освіти)

на тему: Використання кінцевих автоматів для моделювання поведінки віртуальних персонажів

Using finite state machines to simulate the virtual characters behavior

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія

(шифр і назва напряму підготовки, спеціальності)

Освітня програма «Комп'ютерна інженерія»

(назва освітньої програми)

Стариш Єва Георгіївна

(прізвище, ім'я, по-батькові)

Керівник к.т.н., доц. Шестопалов С.В.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент ст. викл. Трубіна Н.Ф.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від «___»____ 2024 р.

Завідувач кафедри

_____ Євгеній МАЛАХОВ

(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___

протокол № ___ від «___»____ 2024 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Алла КОБОЗЄВА

(підпис) (ім'я, прізвище)

Одеса - 2024

АНОТАЦІЯ

У кваліфікаційній роботі розробляється тема «Використання кінцевих автоматів для моделювання поведінки віртуальних персонажів».

Мета роботи – дослідження основних принципів функціонування кінцевих автоматів та їх застосування для реалізації алгоритмів керування складною поведінкою персонажів у відеоіграх жанру «2D-Platformer». Виконано аналіз предметної галузі та розглянуто існуючі аналоги з урахуванням можливих станів ворожих персонажів. Розроблено дизайнерський документ.

Реалізовані кінцеві автомати, які забезпечують моделювання поведінки персонажів. Розроблено демонстраційну версію комп'ютерної гри жанру «2D-Platformer» на основі ігрового двигуна Unity та проведено тестування змодельованої поведінки віртуальних персонажів.

Ключові слова: кінцевий автомат, відеоігри, 2D-Platformer, Unity.

ABSTRACT

The qualification work covers the topic «Using finite state machines to simulate the virtual characters behavior».

The aim of the work is to explore the fundamental principles of the functioning of finite state machines and their application in implementing algorithms for controlling complex behavior of characters in video games of the «2D-Platformer» genre. The work includes an analysis of the subject area and a review of existing analogues considering the possible states of enemy characters. A design document was developed.

Finite state machines were implemented to simulate character behavior. A demonstration version of a computer game of the «2D-Platformer» genre was developed using the Unity game engine, and the simulated behavior of the virtual characters was tested.

Keywords: finite state machine, video games, 2D-Platformer, Unity.

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

НПС – Неігровий персонаж середовища

FSM – Finite State Machine

USP – Unique Selling Points

UI – User Interface (Користувальницький інтерфейс)

FPS – Frames Per Second (Кількість кадрів в секунду)

ВСТУП

Комп'ютерні ігри стали невід'ємною частиною сучасної культури, надаючи мільйонам людей розвагу, можливість спілкування та навіть навчання. Створюються віртуальні світи, де гравці можуть поринути у різноманітні пригоди та випробувати нові емоції. Одним із ключових аспектів, що визначають захоплюючість ігрового процесу, є поведінка віртуальних персонажів, які складають навколишній світ гри та взаємодіють із гравцем.

Існує багато жанрів ігор. Один із найбільш популярних жанрів комп'ютерних ігор – це «2D-Platformer». У таких іграх гравці керують персонажами, переміщаючись різними рівнями, долаючи перешкоди і борючись з ворогами. Жанр «2D-Platformer» залишається актуальним через приваблення своєю динамічністю, цікавим геймплеєм та часто ностальгічним візуальним стилем, що асоціюється з класичними іграми минулих десятиліть. Відмінною рисою цього жанру є двомірна графіка та управління персонажем у двовимірному просторі.

Відомими представниками жанру «2D-Platformer» є ігри «Super Mario Bros» від Nintendo [1], «Super Meat Boy» від Team Meat [2], «Katana Zero» від Askiisoft [3] та «Dead Cells» від Motion Twin [4].

Дослідження та моделювання поведінки віртуальних персонажів має важливе значення для покращення ігрового досвіду. Це дозволяє створювати більш реалістичних та розумних персонажів, які здатні адаптуватися до дій гравця та навколишнього середовища. У цьому контексті використовуються кінцеві автоматі – математичні моделі із заданими станами і переходами між ними. Особливістю кінцевих автоматів є їхня простота та ефективність при моделюванні складних поведінкових систем.

Актуальність теми зумовлена складністю розробки та підтримки логіки поведінки віртуальних персонажів у комп'ютерних іграх. Розробники стикаються з необхідністю створення численних умов та ланцюжків логіки, що регулюють поведінку персонажів. Це призводить до ускладнення коду та

збільшення ймовірності виникнення помилок.

Відсутність чіткої структури управління станами персонажів робить процес налагодження та модифікації ігрової поведінки трудомістким та скрутним. Розробники змушені витрачати більше часу на пошук та виправлення помилок, а також адаптацію поведінки персонажів під зміни в ігровому процесі.

Таким чином, потреба в ефективних інструментах для управління поведінкою віртуальних персонажів стає дедалі актуальнішою.

Предметною областю даного проекту є ігри жанру «2D-Platformer».

Метою даної роботи є розробка кінцевих автоматів для моделювання поведінки віртуальних персонажів в іграх жанру «2D-Platformer».

Для досягнення мети необхідно розв’язати наступні задачі:

а) проаналізувати предметну область;

б) провести дослідження методів реалізації кінцевих автоматів для моделювання поведінки віртуальних персонажів;

в) розглянути існуючі аналоги ігор;

г) розробити проектну документацію та кінцеві автомати для віртуальних персонажів;

ґ) розробити демонстраційну версію гри жанру «2D-Platformer» з використанням розроблених кінцевих автоматів.

д) протестувати і налагодити роботу гри.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ

1.1 Аналіз предметної області

Жанр «Platformer» – «жанр відеоігор, ігровий процес в якому складається зі стрибків персонажа по різноманітним платформам (звідси й назва) та через перешкоди, збирання предметів, зазвичай необхідних для проходження рівня» [5].

Коріння жанру «Platformer» йде в аркадні ігри, в яких ігровий процес зосереджений на підйомі сходами. Ці ігри відбувалися на одному екрані і передбачали стрибки. Метою було або піднятися нагору, або перемогти всіх ворогів на екрані. Однією з таких ігор була «Space Panic», випущена 1980 року [6].

Однак широке визнання як першої платформної гри отримала «Donkey Kong», випущена компанією Nintendo у 1981 році [7]. Мета гри – досягти самої верхньої платформи на екрані, ухиляючись від бочок, що котяться, стрибаючи через них. Популярність «Donkey Kong» підштовхнула до створення безлічі ігор, в яких акцент робився на механіці стрибків. У грі також вперше з'явився один із найзнакових ігрових персонажів – Маріо, який тоді називався Джампмен (Jumpman).

На початку свого розвитку платформери були представлені як ігри з одним екраном (Single Screen Platformers). Цей формат виник з обмежень технічних можливостей ранніх ігрових систем, які не дозволяли створювати великі ігрові світи із прокручуванням екрану. Ці ігри розгортаються на одному екрані і зазвичай містять різноманітні перешкоди, які гравець повинен подолати, а також конкретну мету на кожному рівні. Прикладом такої гри є «Donkey Kong», де гравцеві потрібно дістатися до вершини екрану. Приклад геймплею гри зображено на рис. 1.1.

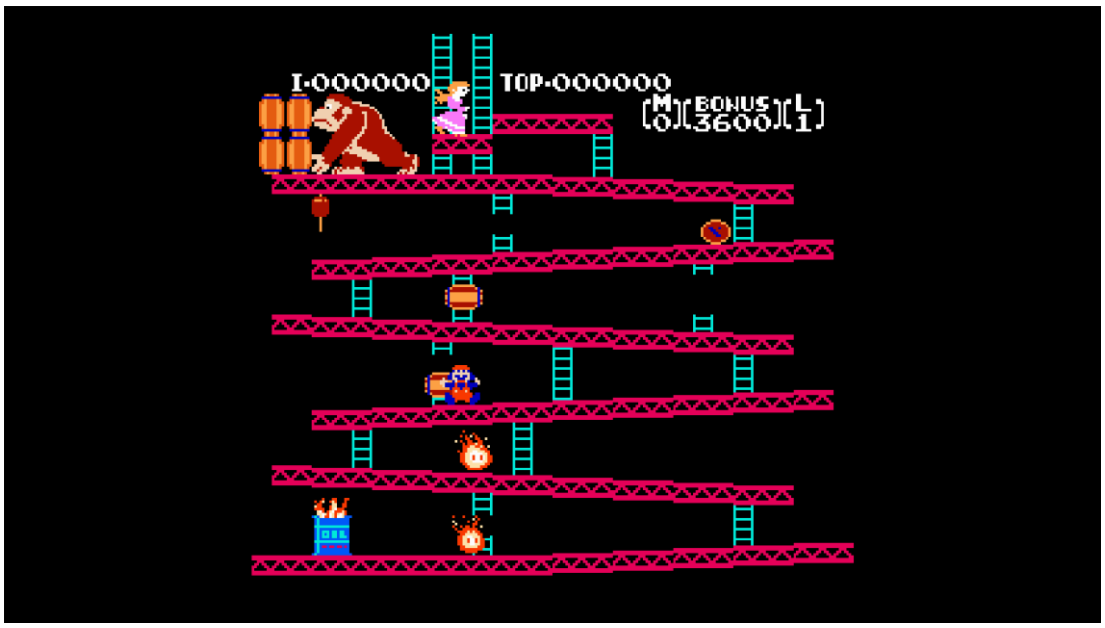


Рисунок 1.1 – Скріншот ігрового процесу гри «Donkey Kong»

З розвитком технологій та можливостей ігрових систем з'явилася можливість створення більш складних та різноманітних рівнів. Це призвело до появи платформерів з прокручуванням екрана (Side and Vertical Scrolling Platformers). У таких іграх ігровий світ представлений у вигляді великої області, що прокручується у міру просування персонажа. Прикладами таких ігор є «Super Mario Bros» та «Castlevania» [8].

З часом жанр «Platformer» розвивався і розширювався, породжуючи різні піджанри. Через п'ять місяців після виходу «Donkey Kong» було випущено ще одну впливову платформерну гра під назвою «Jump Bug» [9], де крім бігу і стрибків гравець міг стріляти в НПС. Це надихнуло створити один із ранніх піджанрів «run-and-gun platformers», який став популярним завдяки серіям ігор «Contra»[10]. Іншим піджанром стали «puzzle platformers», що поєднують елементи головоломок зі стрибками по платформах. The «Lost Vikings» [11] є гарним прикладом такої гри.

У середині 1980-х років відеоігрові консолі зазнавали спаду, і вважалося, що вони повністю згаснуть, поки не було випущено «Super Mario Bros» у 1985 році. Вона стала однією із найбільш продаваних відеоігор в історії, ініціюючи золоте століття платформерів.

Сьогодні ігри жанру «Platformer» діляться на безліч різновидів, включаючи піджанри «2D-Platformer» і «3D-Platformer». Основною відмінністю є кількість вимірів, у яких переміщається персонаж: в іграх жанру «2D-Platformer» персонаж переміщається у двох площинах, а в іграх жанру «3D-Platformer» – трьох площинах.

Популярними прикладами ігор жанру «2D-Platformer» є «Cuphead»[12], «Gris»[13], «Celesto»[14], «Spelunky 2»[15] та «Hollow Knight»[16]. Приклад геймплею гри жанру «2D-Platformer» зображено на рис. 1.2.

Рисунок 1.2 – Скріншот ігрового процесу гри жанру «2D-Platformer»

Прикладами ігор жанру «3D-Platformer» є «Crash Bandicoot 4: It`s About Time»[17], «Demon Turf»[18], «It Takes Two»[19], «Project Feline»[20] та «A Hat in Time»[21]. Приклад геймплею гри жанру «3D-Platformer» зображено на рис. 1.3.

Рисунок 1.3 – Скріншот ігрового процесу гри жанру «3D-Platformer»

Важливим аспектом ігор жанру «2D-Platformer» є моделювання поведінки персонажів. Для цього часто використовують кінцевий автомат. Кінцевий автомат визначає різні стани, в яких може перебувати персонаж, та переходи між цими станами залежно від зовнішніх умов та дій гравця. Цей інструмент дозволяє ефективно керувати діями персонажа залежно від гравця. З іншого боку, кінцевий автомат може визначати стратегії поведінки для НПС. Використання кінцевого автомата сприяє створенню чіткої та структурованої логіки поведінки, що робить процес розробки та підтримки поведінки персонажів більш простим та ефективним.

1.2 Матриця взаємодій. Кінцевий автомат

В ігровому контексті токени є дискретними елементами, якими безпосередньо чи опосередковано керує гравець [22]. Вони охоплюють широкий спектр об'єктів, включаючи персонажів, предмети, ресурси, а також абстрактні концепції, такі як стан персонажів. Токени організуються в ієрархічну структуру, де ігровий світ займає верхній рівень ієрархії, а кожен токен є частиною цього світу.

Матриця взаємодій (token interaction matrix) – це діаграма, в якій

відображені всі види взаємодій, які можуть виникнути в грі [23]. Представляючи собою таблицю, у якій перераховані всі токени як по вертикалі, так і по горизонталі, матриця взаємодій дозволяє аналізувати, які токени можуть взаємодіяти між собою, і які події можуть статися у результаті цієї взаємодії. Важливо відзначити, що токени не взаємодіють один з одним, якщо вони одного типу. Приклад реалізації матриці взаємодії зображено на рис. 1.4.

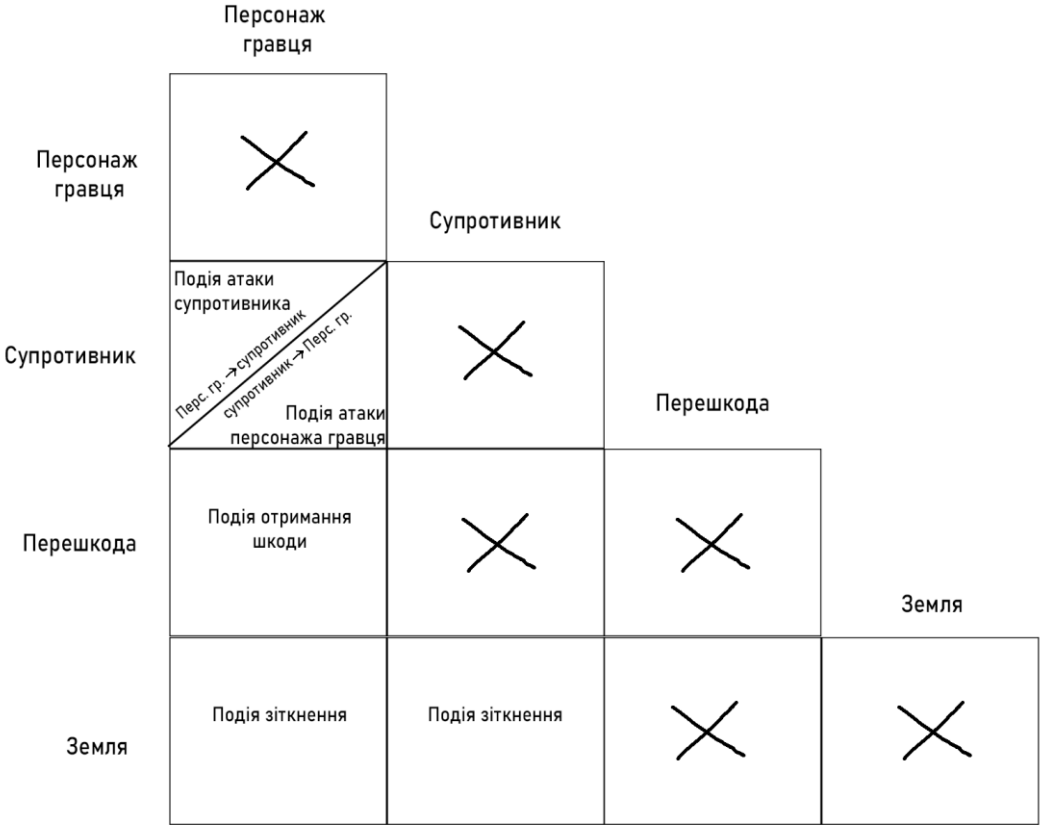


Рисунок 1.4 – Приклад матриці взаємодій

Крім взаємодій між однотипними токенами, виключають також взаємодії між перешкодою та противником і між землею та перешкодою.

Існує два види взаємодії між токенами: симетрична та асиметрична. Симетричними є взаємодії, що спрямовані однаково для обох токенів, та представлені у вигляді звичайних квадратів. Наприклад, в даній матриці симетричними взаємодіями є взаємодії між перешкодою та персонажем

гравця, землею та персонажем гравця, і землею та супротивником.

Асиметричними взаємодіями є ті, що змінюються в залежності від обраного напрямку взаємодії. Така взаємодія зображена у вигляді квадрату, поділеного на два трикутника, кожен з яких представляє різну взаємодію. Наприклад, матриця, представлена на рис. 1.4, має лише одну асиметричну взаємодію – між супротивником та персонажем гравця.

Матриця взаємодій допомагає побачити загалом систему зв'язків, що виникають між токенами, і передбачити можливі події у грі. Однак для великих ігор матриця може виявитися недостатньою, тому зазвичай розглядається кінцевий автомат кожного токена.

Кінцевий автомат або машина станів (FSM – finite state machine, state machine) – це математична модель, яка може перебувати в одному зі скінченної кількості станів у будь-який момент часу. Кінцевий автомат може переходити з одного стану в інший у відповідь на деякі вхідні дані [24]. Кінцевий автомат можна візуалізувати як орієнтований граф, де вузли є станами, а ребра, що з'єднують вузли, – переходами.

Для наочного прикладу розглядається кінцевий автомат, що моделює поведінку турнікету з монетоприймачем (рис. 1.5).

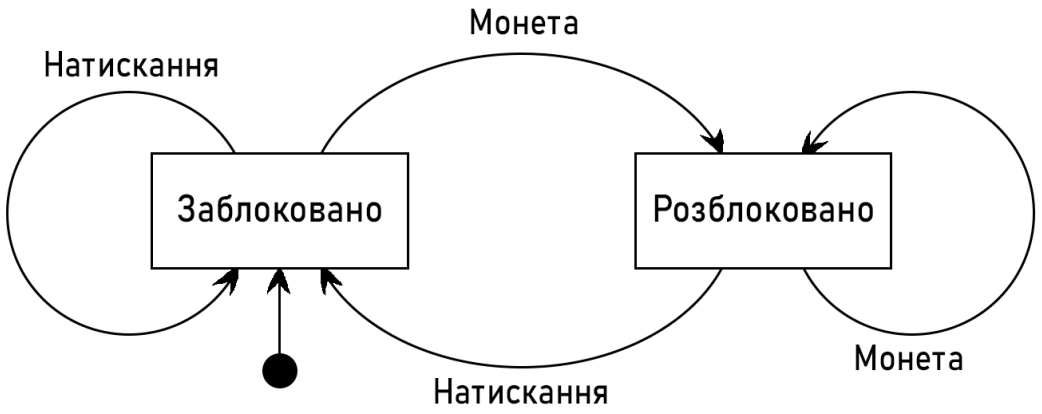


Рисунок 1.5 – Приклад кінцевого автомату турнікету з монетоприймачем

Турнікет може бути в одному з двох його станів: або в стані

«Заблоковано» – коли турнікет закритий і не пропускає нікого, або в стані «Розблоковано» – коли турнікет відкритий і можна пройти через нього. Вхідними даними вважаються монета, яка вставляється в монетоприймач, та перевірка натискання турнікету. Початковим станом цього кінцевого автомата є стан «Заблоковано».

Кінцевий автомат турнікета можна представити за допомогою таблиці переходів станів, що показує для кожного можливого стану переходи між ними (на основі вхідних даних, наданих автомату) та виходи, що є результатом кожного входу. Докладно переходи станів кінцевого автомата турнікету наведено у табл. 1.1.

Таблиця 1.1 – Переходи станів кінцевого автомата турнікету

Поточний стан	Вхідні дані	Наступний стан	Вихідні дані
Заблоковано	монета	Розблоковано	Відмикання турнікету
	натискання	Заблоковано	Ніякі
Розблоковано	монета	Розблоковано	Ніякі
	натискання	Заблоковано	Замикання турнікету

Турнікет переходить зі стану «Заблоковано» у стан «Розблоковано», коли на вхід надходить монета і на виході видається сигнал відключення турнікету. Якщо на вхід надходить натискання, турнікет залишається в стані «Заблоковано» і жодні вихідні дані не змінюються.

Якщо поточний стан турнікета «Розблоковано», то також можливий перехід у два стани. Коли на вхід надходить монета, турнікет залишається в стані «Розблоковано» і жодні вихідні дані не змінюються. У разі надходження натискання турнікет повертається у стан «Заблоковано» і блокує себе після проходу людини.

Отже, кінцевий автомат турнікету з монетоприймачем моделює його поведінку, спираючись на поточний стан та вхідні дані, щоб визначити відповідні дії у різних ситуаціях.

У контексті ігрової розробки кінцевий автомат часто застосовується для моделювання поведінки віртуальних персонажів. Він ідеально підходить для опису станів та пов'язаних з ними дій, що характеризують поведінку персонажів у грі. Наприклад, НПС можуть мати стани «іти за гравцем», «атакувати», «втікати» і т.д. Кожен з цих станів має певні дії і реакції.

В якості ще одного прикладу можна навести кінцевий автомат ворога з гри жанру «Shooter» (рис. 1.6) [25].

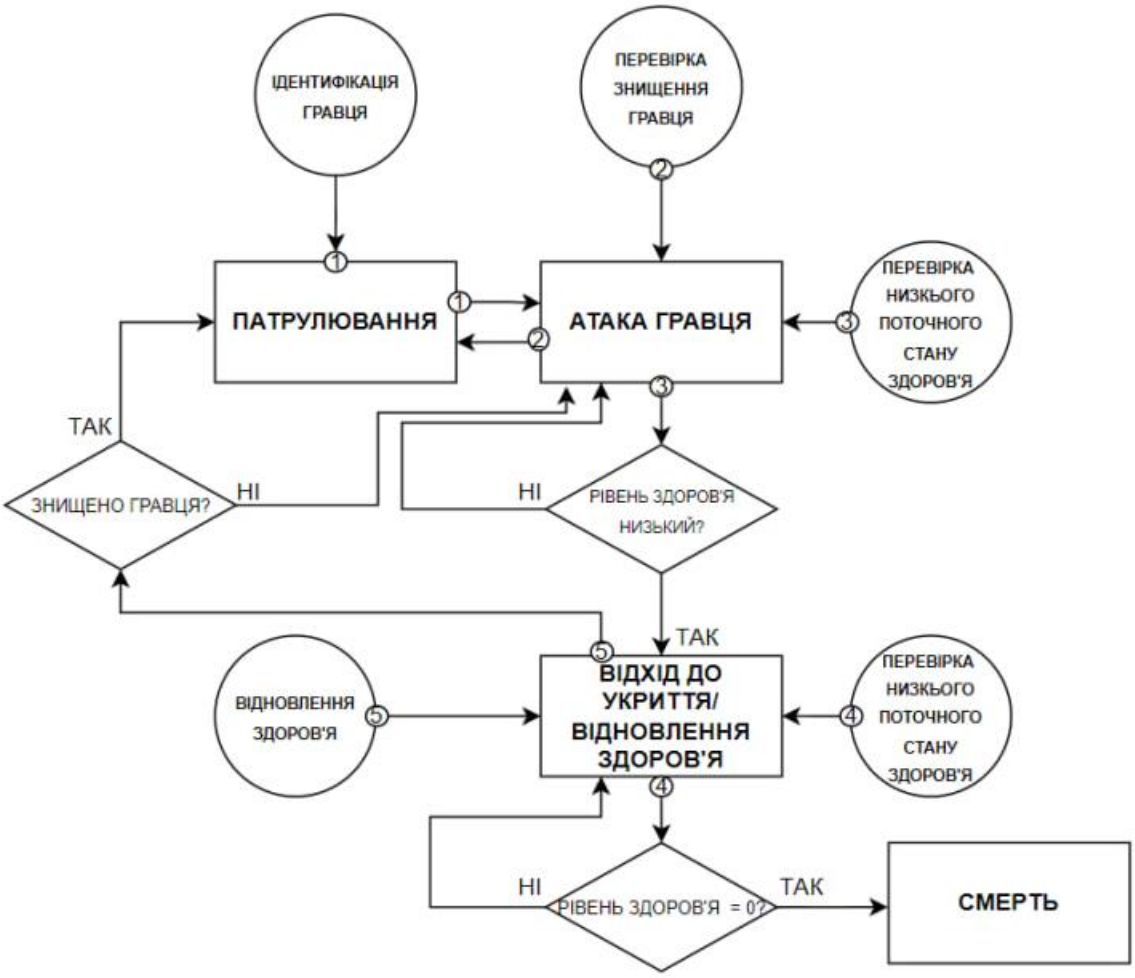


Рисунок 1.6 – Приклад кінцевого автомату ворога з гри жанру «Shooter»

Даний кінцевий автомат ворога має наступну кінцеву кількість станів: «Патрулювання», «Атака гравця», «Відхід до укриття та відновлення здоров'я» та «Смерть». Початковим станом цього кінцевого автомата є «Патрулювання».

Вхідними даними є ідентифікація гравця, перевірка знищення гравця, перевірка низького стану здоров'я ворога і перевірка відновлення здоров'я ворога.

Зі стану «Патрулювання» ворог може перейти в стан «Атака гравця», коли гравець ідентифікований. Якщо гравця знищено, то зі стану «Атака гравця» ворог переходить назад у стан «Патрулювання». Якщо у ворога виявляється низька кількість здоров'я, то зі стану «Атака гравця» він переходить у стан «Відхід до укриття та відновлення здоров'я». Якщо ж у ворога достатня кількість поточного здоров'я, то кінцевий автомат залишається у стані «Атака гравця». Кінцевий автомат залишається в стані «Відхід до укриття та відновлення здоров'я», поки ворог не відновить собі здоров'я. Якщо поточне здоров'я ворога опустилося до нуля, то кінцевий автомат переходить у кінцевий стан «Смерть», з якого не можна потрапити до інших станів.

Таким чином, кінцевий автомат відіграє важливу роль у розробці персонажів, забезпечуючи їхню реалістичну поведінку та взаємодію з ігровим світом.

1.3 Існуючі аналоги

«Super Mario Bros.» – культова відеогра, випущена Nintendo у 1985 році на NES. Гравець управляє одним із двох братів – Маріо та Луїджі. Гра складається з 8 світів, кожен з яких розділений на 4 рівні, які необхідно пройти для закінчення гри.

Основний геймплей складається з подолання перешкод, збирання монет та збивання ворогів. Гравець використовує стрибок, щоб уникати перешкоди та атакувати ворогів. Також у грі присутні різні секрети та додаткові рівні та крім того спеціальні предмети – наприклад гриби, які збільшують розмір Маріо та дають йому додаткові можливості. Скріншот ігрового процесу гри «Super Mario Bros.» зображено на рис. 1.7.

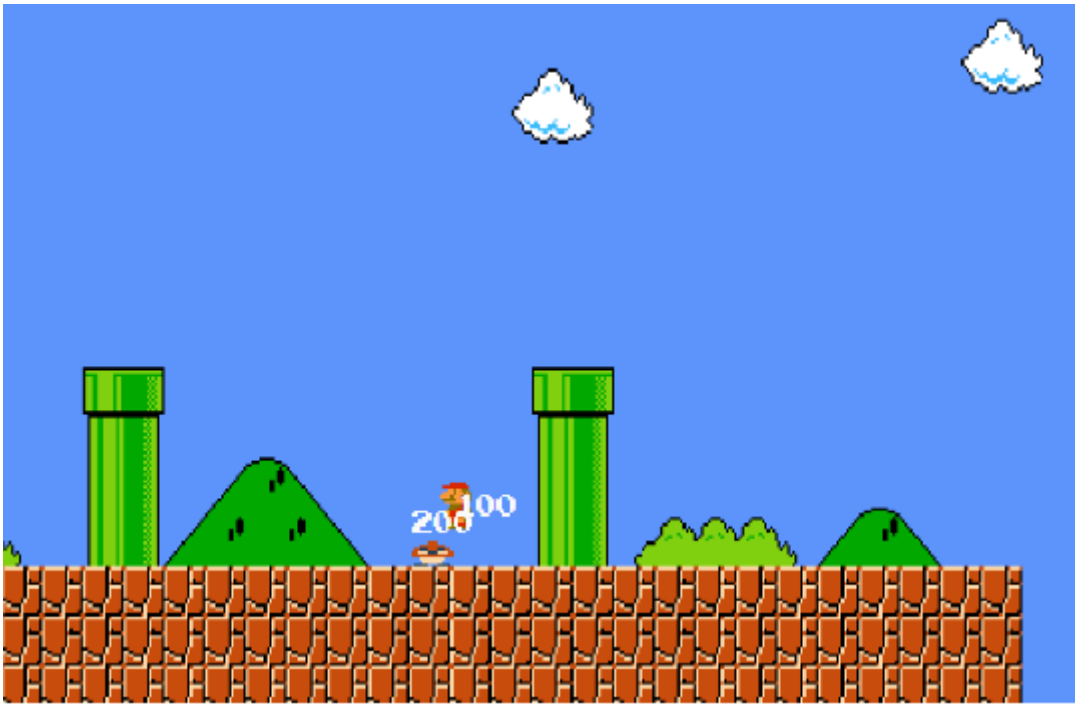


Рисунок 1.7 – Скріншот ігрового процесу гри «Super Mario Bros.»

Розглядаючи методи реалізації поведінки ворогів у грі «Super Mario Bros» можна виділити наступні недоліки.

Через технічні обмеження ресурсів і пам'яті процесора консолі противники мають прості правила руху і реагують на обмежену кількість подій. Противники мають прості алгоритми, які визначають їх рух ліворуч-праворуч, а також реакцію на гравця або на перешкоди на рівні.

Наприклад, Гумба (Goomba) рухається вліво доти, доки не зіткнеться з перешкодою, після чого повертається в протилежний бік. Купа Трупа (Koopa Troopa) змінює напрямок руху, коли досягає краю платформи.

У зв'язку з простотою моделей поведінки противники ніяк не реагують на дії гравця, не мають базового переслідування чи жодних реакцій на зміну обставин чи умов навколишнього середовища. Такі поведінки призводять до ситуацій, що постійно повторюються, через що інтерес вже досвідчених гравців втрачається.

Але ця ж простота реалізації призводить до того, що вона є і перевагою, адже такі моделі не вимагають високих навичок, швидкості

реакції чи великого досвіду в іграх цього жанру, що в сукупності з добре підібраним дизайном рівнів та динамічним геймплеєм призводить до залучення нових гравців та популяризації цього жанру.

«Super Meat Boy» – гра, розробниками якої є Едмунд Макміллер і Томмі Рефенес. Розробка почалася в 2008 як лише прототип, створений для конкурсу та безкоштовно доступний для скачування, а в 2010 після доробок гра з’явилась на Xbox 360. У зв'язку з її популярністю пізніше була портована також на PlayStaytion 4, PC і мобільні пристрої .

Сюжет починається з історії головного персонажа М'ясного Хлопця, який намагається врятувати свою кохану від зловісного Лікаря. Гра складається з п’яти основних розділів. Скріншот ігрового процесу гри «Super Meat Boy» зображено на рис. 1.8.

Рисунок 1.8 – Скріншот ігрового процесу гри «Super Meat Boy»

Протягом усієї подорожі гравець стикається з дедалі більшою кількістю пасток та деяких простих у плані реалізації супротивників, а також босів наприкінці кожного розділу. Через велику кількість пасток, основна частина геймплею зосереджена на точності дій та швидкості реакції гравця, адже на протязі всього проходження перешкод стає все більше і з'являються все нові і нові їх варіації та види, завдяки чому гравцеві доводиться шукати нові шляхи

проходження.

У зв'язку з тим, що гра більше зосереджена на уникненні пасток гравцем, вона має не надто складні моделі поведінки ворожих НПС, але чудово доповнює своєю присутністю протягом усієї гри, додаючи додаткову складність для недосвідченого гравця.

Більшість таких персонажів має лише просту логіку, як приклад рух по «стіні» (об'єкт, що є платформою вертикальної спрямованості) або звичайною платформою без виходу з цього стану, зміна вектора руху після зіткнення зі «стіною» або платформою, або переслідування гравця – використовуючи прості варіанти рухів.

«Katana Zero» – гра, створена групою розробників Askiioft і була офіційно опублікована в 18 квітня 2019 року відразу на кілька платформ, такі як Windows, MacOS, Nintendo Switch а пізніше і на Xbox One. Головний розробник Джастін Стендер розпочав розробку гри у 2013 році і для створення візуальних ефектів та музичного супроводу найняв художників та музикантів.

Це гра сайд-скроллер, в якому одне поранення призводить до смерті персонажа гравця. Кожен рівень поділено на екрани, в яких гравець повинен здолати всіх супротивників для відкриття наступної зони рівня. Якщо гравець помирає в процесі проходження рівня, то він починає його проходити заново. Але перепроходження рівнів не виглядає нудно, бо гравець швидко відроджується та кожну зону рівня можна швидко пройти. Також є можливість використовувати різні предмети, наприклад, димові гранати. Крім того, є додаткові здібності, які може використовувати гравець. Наприклад, перекати для ухилення, атака, кидки різних предметів, що підбираються, і уповільнення часу, що допомагає гравцеві легше ухилятися від куль супротивників, займаючи відповідну позицію. Скріншот ігрового процесу гри «Katana Zero» зображено на рис. 1.9.

Рисунок 1.9 – Скріншот ігрового процесу гри «Katana Zero»

На відміну від попередньої гри «Super Meat Boy» дана гра зосереджена на вбивстві всіх супротивників у зоні рівня, у зв'язку з чим тут набагато краще опрацьовані моделі поведінки кожного з ворожих НПС. Вони можуть переслідувати гравця, атакувати його в ближньому або дальньому бою в залежності від того, який це ворожий НПС. Є особливі атаки чи дії, як кидок гранати чи виклик підкріплення для того, щоб ускладнити гравцю проходження рівня.

Наприклад, Сильний Террі (Strong Terrie) – перший супротивник, з яким стикається гравець, може патрулювати кімнату або стояти на місці, намагатиметься підбігти і вдарити гравця, коли той опиниться в його полі зору. Худий Рікі (Skinny Rickie) – це ворог ближнього бою, як і Сильний Террі, але його удари можуть відбивати атаки. Бойовик (Gunman) є противником далекого бою – стріляє в гравця гвинтівкою, поки не втратить його з поля зору. Якщо гравець підійде до нього занадто близько, то супротивник відкине гравця.

Ця гра має безліч відмінних одна від одної моделей поведінки та єдиним недоліком є одноманітність супротивників, з якими можна зустрітися протягом проходження.

«Dead Cells» – гра, розроблена Motion Twin та Evil Empire і була видана Motion Twin у 2018 році. Щоб ігровий процес був не надто нудним, розробники додали близько 50 різних видів зброї та хотіли гарантувати, що кожний із видів зброї продемонструє унікальні анімацію та поведінку. Перша доступна версія була завершена на 30-40%, і використала ранній доступ Steam (Основна послуга Steam – це продаж відеоігор, виступає в ролі технічного засобу захисту авторських прав) для оцінки інтересу гравців та отримання відгуків про функції гри, що допомогло вирішити проблеми з балансом у грі. За оцінками провідного дизайнера Себастьяна Бенара, 40-50% функцій фінальної гри було взято із відгуків раннього доступу. Скріншот ігрового процесу гри «Dead Cells» зображено на рис. 1.10.

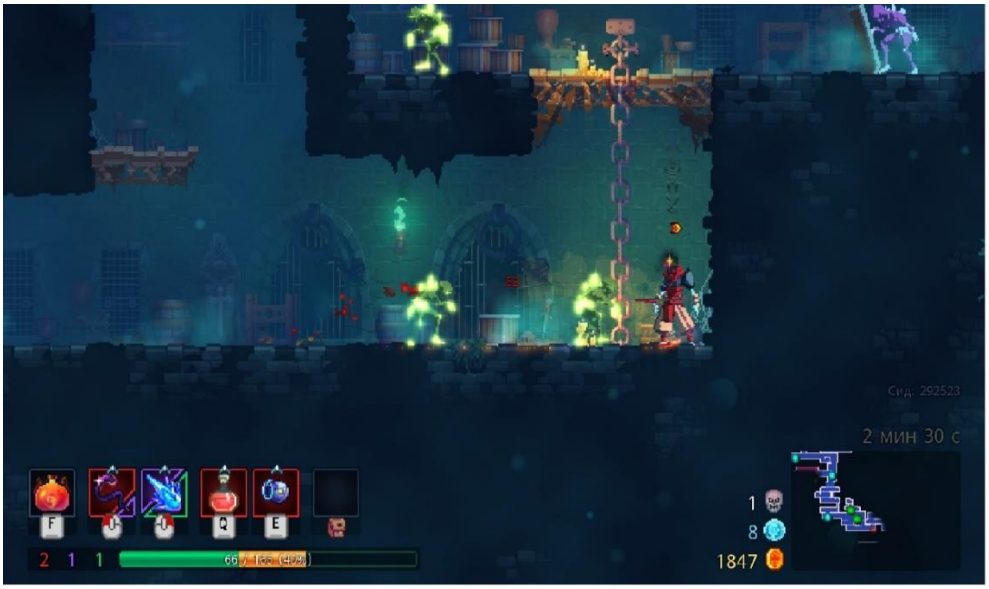


Рисунок 1.10 – Скріншот ігрового процесу гри «Dead Cells»

Основою метою гри є битви, варіанти розвитку яких можуть бути найрізноманітнішими, адже гравець стикається з рівнями, які завжди виглядають по-різному при повторному їх проходженні. У даній грі реалізовано механіку процедурної генерації – коли карта рівня генерується при кожному проходженні, заново даючи гравцеві знову проходити нову, наприклад, стартову локацію Тюремні камери. А разом з великою кількістю абсолютно різної зброї та навичок дозволяє щоразу грати по-різному, не

втрачаючи інтересу до ігрового світу. У комбінації з великою кількістю рівнів як відкритих, так і прихованих або які потребують різних предметів, щоб потрапити в них, ця гра продовжує залучати все більше нових або досвідчених гравців. Завдяки регулярним оновленням гра ще довго чаруватиме гравців своєю реалізацією.

Що ж до моделей поведінки, то їх безліч, адже в кожному новому рівні є свої відмінні від інших супротивники. Хоча все ж таки можна виділити деякі патерни. Наприклад, більшість супротивників мають патрулювання чи переслідування гравця, якщо той віддаляється на певну відстань, та інші.

Наприклад, Зомбі – поширений противник першого рівня, що має стани патрулювання та переслідування гравця, два види атаки – ближню та дальню, при якій він робить стрибок до гравця та атаку, зближуючись одночасно. Також за певних обставин може на деякий час впасти в стан оглушення або заморозки. Ожилий лучник, не менш поширений ворог, також має стан патрулювання і переслідування, якщо гравець вийшов за межі дальності атаки, але не залишив область його видимості. Цей супротивник має дальню атаку – постріл із лука. Якщо гравець підійшов до нього надто близько, то супротивник робить відскок у протилежний бік від гравця. Темний розвідник (Dark Tracker), як і попередні супротивники, має стан патрулювання, але при виявленні гравця телепортується йому за спину і робить потужну атаку. Або якщо гравець занадто близько підійшов, супротивник робить звичайну атаку ближнього бою. Інший супротивник, як Голем, також має патрулювання і переслідування, але крім них є три види атаки: перша – робить ривок у бік гравця, завдаючи шкоди, друга – удар по землі, що завдає шкоди у великій області, і третя – атака не завдає шкоди, але може змінити ситуацію на полі бою моментально – якщо гравець вийшов з області видимості Голема або перемістився на якусь із платформ, до яких він не може дістатися, в такому разі Голем телепортує гравця до себе з секундною зарядкою вміння.

У зв'язку з цим можна вважати цю гру як ту, що чудово розкриває жанр «2D-Platformer» і є одним із найкращих прикладів його реалізації, як у плані

реалізації динамічного ігрового світу, так і у створенні моделей поведінки персонажів.

1.4 Постановка задачі

Основні задачі, які потрібно вирішувати, включають наступне:

а) розробити матрицю взаємодій;

б) розробити кінцеві автомати головного героя та ворогів;

в) створити унікальний ігровий контент, такий як моделі персонажів, елементи середовища, інтерфейс користувача та інше;

г) розробити ігровий світ, включаючи рівні;

ґ) реалізувати на основі запропонованих кінцевих автоматів, поведінку головного героя та ворогів;

д) створити додаткові елементи гри;

е) провести тестування демонстраційної версії гри.

1.5 Висновки до першого розділу

а) В результаті аналізу предметної області розглянуто походження жанру «2D-Platformer» та визначено його основні характеристики та різновиди.

б) Відзначено важливість застосування кінцевого автомата для регулювання поведінки персонажів в іграх жанру «2D-Platformer». Розглянуто такі інструменти моделювання поведінки віртуальних персонажів, як матриця взаємодій та кінцевий автомат.

в) Розглянуто існуючі аналоги ігор жанру «2D-Platformer». Проаналізовано поведінку ворогів у зазначених проектах та виявлено переваги та недоліки реалізації їхньої поведінки.

г) Проведено постановку завдання з урахуванням бажаного результату, необхідного для розробки проектної документації та демонстраційної версії гри.

2 ПРОЕКТНА ДОКУМЕНТАЦІЯ

2.1 Вступ

Гра починається з кат-сцени, де головний герой, Морган Ноктфорд, у відчайдушній гонитві від переслідувачів знаходить притулок у таємничому замку. Усередині зустрічає великий лорд-вампір Вельхеор, господар цього замку і кусає головного героя. Прокинувшись в одній із темних тюремних камер замку, Морган виявляє в собі вампірські здібності і вирішує знайти лорда-вампіра, щоб помститися.

Основна мета гри полягає у проходженні рівнів шляхом використання вампірських здібностей Моргана, для розвитку яких потрібні ресурси з уражених ворогів. Ігрові рівні дають можливість проходити гру різними шляхами та наділяють гравця бонусами, як відновлення здоров'я та розблокування здібностей за дослідження.

Дизайн рівнів представлений у вигляді різних кімнат замку, наповнених ворогами, що служать лорду-вампіру. У деяких із них очікуються зіткнення з потужними босами, які потребують особливої тактики бою.

Гравцеві також належить враховувати джерела світла, які є додатковим елементом перешкоди. При попаданні в область світла гравець ризикує бути виявленим ворогами, тому йому вміло необхідно використовувати вампірські здібності, щоб ховатися в тіні та уникати прямого контакту з джерелами світла.

Графіка гри виконана в стилі піксель-арту, створюючи унікальну атмосферу похмурого вампірського замку.

2.2 Жанр та аудиторія

Гра «One Bite» відноситься до ігор жанру «2D-Platformer», в якому гравець має досліджувати замок, долаючи різні перешкоди і борючись з ворогами.

Аудиторія гри складається з людей віком понад 12 років, захоплених платформерами та вважають привабливим сеттинг вампірського світу. Для гравців, які мають не найсучаснішу конфігурацію ПК, гра також може становити додатковий інтерес.

Прикладами ігор даного жанру, які можуть зацікавити цільову аудиторію, є «Dead Cells» і «Katana Zero». Гра має інтуїтивне управління і динамічний ігровий процес, що робить її привабливою для фанатів платформерів.

2.3 Основні особливості гри

Основними особливостями гри (USP – Unique Selling Points) є:

а) Бойова система, заснована на вампірських здібностях. Гравець має можливість розвиватися у різних напрямках, використовуючи дерево умінь. Кожен напрямок пропонує унікальні здібності, які дозволяють гравцеві відповідно до переваг застосовувати різну тактику бою: мага, воїна або вбивці.

б) Піксельна графіка та атмосфера похмурого вампірського світу. Гравець занурюється в атмосферу середньовічного замку, що зачаровує, наповненого деталями оточення і підкріпленого зловісною музикою. З технічної точки зору вибраний стиль графіки дозволяє більше оптимізувати ігровий процес і запускати гру на менш потужних системах.

в) Продуманий сюжет про сім'ю вампірів та інших істот. Гра пропонує захоплюючий сюжет, що розповідає про сімейні інтриги та таємниці вампірського клану, а також про взаємини з іншими істотами в замку. Під час дослідження ігрового світу гравець може проводити діалоги з персонажами або знаходити різні ігрові елементи, такі як записки та блокноти, з текстовими записами, що дозволить зрозуміти події, які відбуваються в грі.

г) Елементи скритності. У грі передбачені елементи скритності, які дозволяють гравцеві уникати прямих зіткнень із ворогами. За допомогою

спеціальних вампірських здібностей гравець може ховатися і пройти рівень практично непоміченим, здолавши лише кілька ворогів.

Гра розрахована приблизно на 6 годин.

2.4 Хід гри

У грі збережено елементи класичних ігор жанру «2D-Platformer», де гравцю доводиться уникати різних перешкод. Основним завданням гравця є проходження рівнів, бій з різними ворогами та подолання босів, включаючи лорда-вампіра Вельхеора.

Ігровий процес полягає у використанні вампірських здібностей для ефективної протидії ворогам різних типів. Вороги можуть мати різні уразливості та тактики бою, що змушує гравця розробляти унікальні стратегії для кожного супротивника.

Перешкодами на шляху гравця є:

а) вороги;

б) ділянки локацій;

в) джерела світла;

г) обмеження ресурсів;

Опис перелічених перешкод детальніше розглянуто у табл. 2.1.

Таблиця 2.1 – Геймплейні перешкоди для гравця

Тип перешкоди	Опис перешкоди
Вороги	Вороги з різними стилями атак надають грі динаміки у битвах. Кожен тип ворога вимагає унікальної тактики та підходу з боку гравця, що робить ігровий процес більш цікавим.

Продовження таблиці 2.1

Тип перешкоди	Опис перешкоди
Ділянки локацій	Ділянки локації з перешкодами, такими як прірви, пастки та інші небезпеки, сприяють різноманітності геймплею та вимагають від гравця не лише бойових навичок, а й майстерності в управлінні персонажем.
Джерела світла	Джерела світла вносять у гру додаткову напругу. Гравець повинен уникати джерел світла, щоб не бути виявленим ворогами.
Обмеження ресурсів	Обмеження ресурсів змушує гравця приймати важливі рішення на ходу. Гравцю необхідно акуратно керувати ресурсами, такими як фрагменти душ і очки досвіду, щоб розвивати свої вампірські здібності, а також заряди крові, які необхідні для використання певних навичок.

2.5 Ігрове оточення

Ігрове оточення в грі «One Bite» представлене у вигляді середньовічного вампірського замку, оточеного таємничою атмосферою. Уся графіка виконана в піксельному стилі, що надає грі унікального ретро-візуального шарму. У замку гравця очікують різноманітні перешкоди, такі як пастки, обриви та укриті ворогами ділянки.

Освітлення ігрових локацій має важливу роль, оскільки при попаданні гравця в область світла з'являється можливість привернути увагу ворогів. У кожній локації розміщено певна кількість джерел світла, таких як настінні та підлогові магічні смолоскипи, що мають свій колір та радіус освітлення. Джерела світла розміщуються у кожній кімнаті замку з урахуванням балансу гри. Напівпохмурі кімнати, освітлені лише мерехтливим світлом смолоскипів, надають грі додаткову напруженість і змушують гравця бути пильним на кожному кроці.

Локації оформлені в різних колірних палітрах, що відображають їхній унікальний характер. Наприклад, локація «Тюремні камери» виконана в темно-синіх тонах, що підкреслюють похмурість та присутній холод місця, а локація «Бібліотека» зроблена у червоних відтінках, які надають відчуття розкоші. Концепт дизайну рівня гри зображено на рис. 2.1.

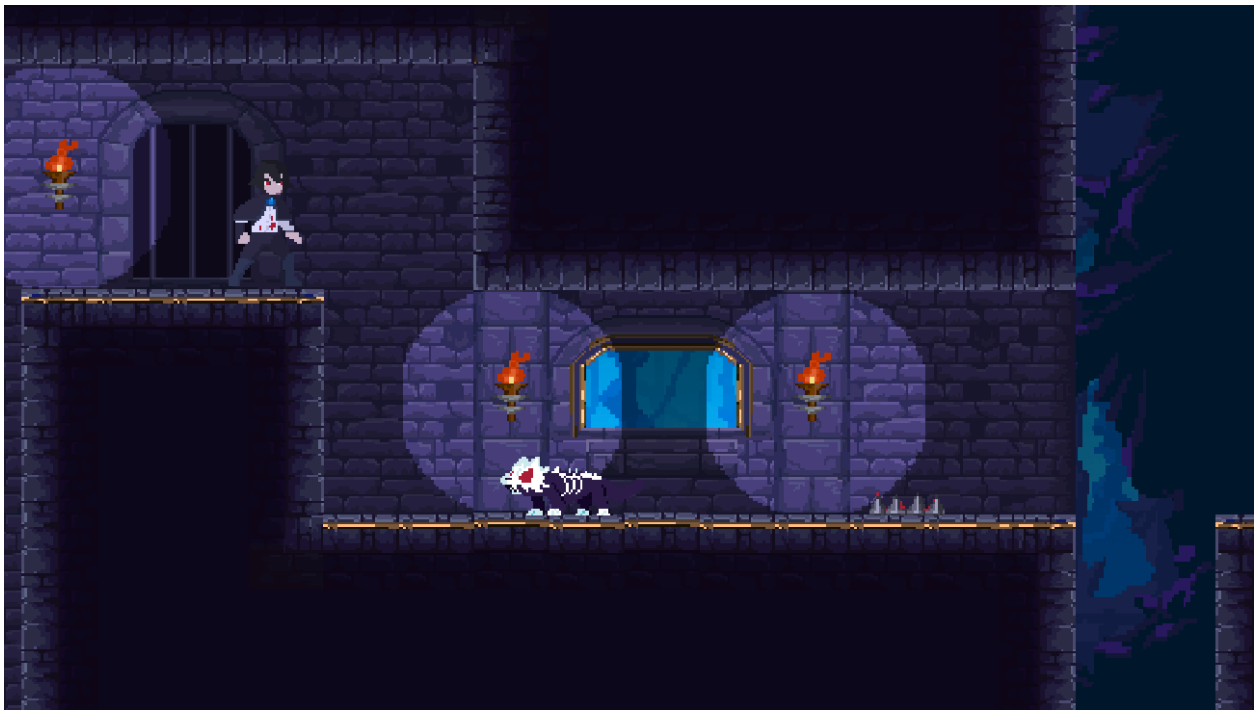


Рисунок 2.1 – Концепт дизайну рівня гри

У вампірському сеттингу найбільше переважає червоний колір, так як він символізує кров, як основне джерело сили вампірів. А головний герой, що став вампіром, володіє вміннями, зумовленими цією темною сутністю. Тому, наприклад, сліди його базової атаки кігтями залишають за собою ефекти у вигляді насиченої червоно-рожевої палітри, що підкреслюють природу його вампірських здібностей. Домінуюча червона гамма простежується й інших здібностях головного героя, характеризуючи прояв вампірської сили. Ця характерна гамма кольорів ідеально доповнює загальний стиль гри.

Всі важливі для гравця ігрові елементи також яскраво вирізняються на тлі навколишнього середовища, маючи контрастний відтінок. Таким чином, гравець завжди може помітити ворогів або предмети, що можна підібрати.

2.6 Головний герой

Насамперед головний герой повинен мати вигляд молодого дворянина часів вікторіанської епохи. Його вигляд включає темний костюм і плями крові на білій сорочці, що свідчать про укус лорда-вампіра. Зображення концепції головного героя представлено на рис. 2.2.

Рисунок 2.2 – Концепції головного героя

Враховуючи графіку в стилі піксель-арту та обраний концепт головного герою, створюється остаточний зовнішній вигляд персонажа (рис. 2.3).

Рисунок 2.3 – Головний герой

Головний герой має основні навички переміщення, типові для платформерів: він може бігати, стрибати по платформах і лазити по драбині.

Серед атакуючих умінь головний герой має базову атаку, яка є махом кігтями. За допомогою цього вміння гравець може постійно завдавати шкоди ворогам.

Вампірські здібності головного героя представлені у вигляді дерева вмінь, що має три основні гілки, якими може розвиватися гравець: «Магія крові», «Володіння кігтями» та «Приховані тіні». Ці гілки здібностей надають можливість гравцеві розвивати свого персонажа, як мага, воїна та вбивці відповідно. Однак гравець може розвиватися не тільки в одній гілці, а ще дозволяється комбінувати здібності, створюючи власний білд. Схема дерева вмінь «Вампірські здібності» зображена на рис. 2.4.

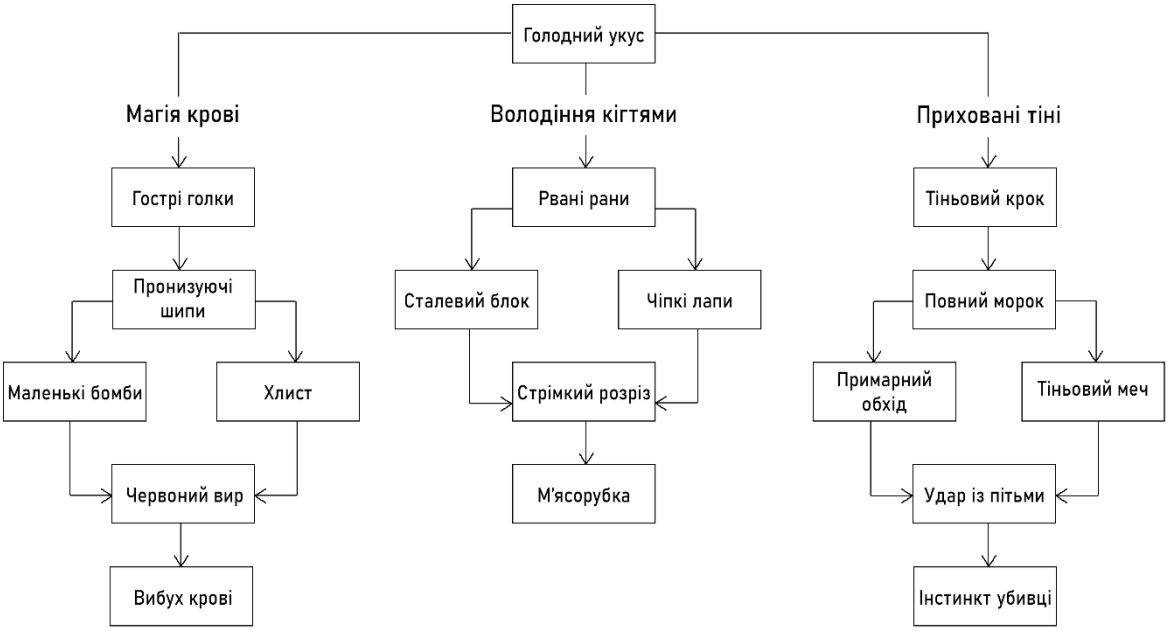


Рисунок 2.4 – Дерево вмінь «Вампірські здібності»

З самого початку ігрового процесу гравцеві доступна перша вампірська здібність «Голодний укус», з якої розгалужуються основні гілки дерева вмінь. Ця здатність дозволяє заповнювати запаси зарядів крові, за допомогою яких гравцеві можна використовувати здібності з гілки «Магія крові». Також це вміння дозволяє завдавати більше шкоди, ніж базова атака головного героя. Однак після активації деяких вампірських здібностей, у тому числі

здібності «Голодний укус», потрібен деякий час на їх відновлення, перш ніж вони знову стануть доступними для використання. Надалі такий час називається перезарядженням вміння. З точки зору балансу гри, це зроблено для того, щоб гравець не міг використовувати сильні атакуючі вміння на постійній основі. У демонстраційній версії гри гравцеві доступно, крім вампірської здібності «Голодний укус», відразу ще перші здібності «Гострі голки» та «Тіньовий крок» з основних гілок дерева.

Опис вампірських здібностей детальніше наведено в табл. 2.2.

Таблиця 2.2 – Опис вампірських здібностей

Назва	Опис
Магія крові	
Гострі голки	Персонаж кидає у противника гострі голки, завдаючи помірної шкоди. Голки проникають у ціль і залишаються в ній на деякий час, завдаючи додаткових втрат за рахунок кровотечі.
Пронизуючі шипи	Під землею виникають гострі шипи, які вириваються з-під поверхні, завдаючи шкоди всім ворогам у радіусі дії. Ця здібність завдає помірної шкоди і має високий потенціал для пошкодження групи ворогів.
Маленькі бомби	Персонаж створює і кидає сферу крові, яка вибухає при попаданні або досягненні максимальної далекобійної відстані. Вибух завдає шкоди всім ворогам навколо, а також відкидає їх убік. Ця здібність ефективна для атаки групи ворогів.
Хлист	Персонаж проводить помах кривавим хлистом, завдаючи шкоди всім ворогам у зоні ураження. Здібність може вразити кілька ворогів одночасно і має середній радіус ураження. Це вміння призначене для ефективної боротьби з групами ворогів на близькій відстані.

Продовження таблиці 2.2

Назва	Опис
Магія крові	
Червоний вир	Здібність створює навколо персонажа шар з кровавої енергії, який збільшує його захисні властивості і зменшує шкоду. Здібність також може впливати на навколишніх ворогів, наприклад, знижуючи їх швидкість пересування.
Вибух крові	Персонаж розкидає бризки крові довкола себе, створюючи зону ураження. Всі вороги в зоні отримують помірну шкоду, а також можуть бути відкинуті убік. Це особливо корисно при оточенні великої кількості ворогів.
Володіння кігтями	
Рвані рани	Здібність дозволяє персонажу завдати шкоди, використовуючи обидві руки з гострими кігтями під час стрибку. При успішному попаданні персонаж також може відкинути ворога убік.
Сталевий блок	Персонаж піднімає перед собою кігті, які дають непроникний захист. При вчасній активації вміння в момент атаки ворога відбувається парирування ворожої атаки.
Чіпкі лапи	Під час стрибка персонаж використовує кігті, щоб зачепитися за стіну чи іншу поверхню. Це дозволяє персонажу виконати вертикальне чи горизонтальне переміщення вздовж стіни, що корисно для подолання перешкод чи уникнення ворогів.
Стрімкий розріз	Персонаж робить різкий ривок вперед, завдаючи шкоди всім ворогам на своєму шляху. Це дозволяє завдавати значної шкоди цілям на середніх дистанціях.

Продовження таблиці 2.2

Назва	Опис
Володіння кігтями	
М’ясорубка	Персонаж обертається навколо своєї осі з витягнутими кігтями вперед, завдаючи шкоди всім ворогам в радіусі дії. Це вміння покликано завдавати шкоди під час сутички з великою кількістю ворогів.
Приховані тіні	
Тіньовий крок	Дозволяє персонажу різко переміститися в тінь, оминаючи джерела світла та шкоди від ворогів. Це дозволяє уникати виявлення ворогами та використовувати перевагу несподіванки.
Повний морок	Здібність створює область темряви перед персонажем, збільшуючи шкоду, що завдається в цій області, і збільшуючи швидкість пересування. Це дозволяє персонажу посилити свою атаку та маневреність у бою.
Примарний обхід	Персонаж стає невидимим на короткий час, дозволяючи йому проходити крізь ворогів і уникати шкоди. При виході з невидимості, наступна атака завдає додаткової шкоди.
Тіньовий меч	Персонаж закликає містичний меч з пітьми, який завдає шкоди ворогу і засліплює його на короткий час, зменшуючи його точність атак.
Удар із пітьми	Персонаж телепортується до ворога з тіні, завдаючи йому сильного удару і приголомшуючи його на деякий час.
Інстинкт убивці	При атаці ззаду ворога шкода збільшується. Це дозволяє персонажу ефективно використовувати тактику потайного підходу і завдавати критичної шкоди ворогам.

Характеристики головного героя включають дві основні шкали. Перша шкала відображає поточну та максимальну кількість здоров'я головного

героя. Друга шкала є кількістю зарядів крові, необхідних для використання вампірських здібностей з гілки «Магія крові».

Система зарядів крові розроблена з урахуванням потреби збалансованої гри. Вона забезпечує обмеження використання дальніх атак, що сприяє підтримці балансу між дальнім та ближнім боєм. Для гравців, які вибрали білд мага, такий підхід створює додаткові умови стратегії. У цьому випадку гравець повинен розумно розподіляти свої ресурси, іноді вдаючись до ближнього бою за допомогою здібності «Голодний укус», щоб відновити заряди крові.

2.7 Вороги

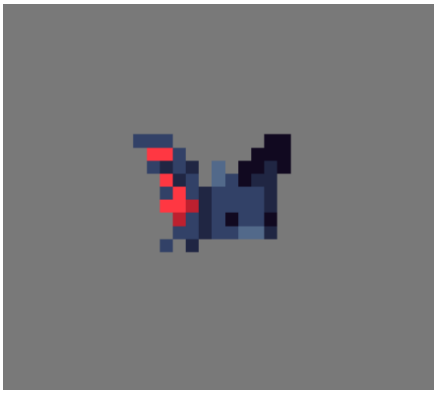
Вороги відіграють ключову роль у створенні динаміки ігрового процесу. Вони не тільки є перешкодами на шляху головного героя, а й служать важливим елементом у пожвавленні ігрового світу.

У грі присутнє різноманіття ворожих персонажів, кожен з яких має унікальні характеристики та поведінку. У демонстраційній версії гри представлено п'ять типів ворогів. Список запропонованих типів ворогів та їхні характеристики наведено у табл. 2.3.

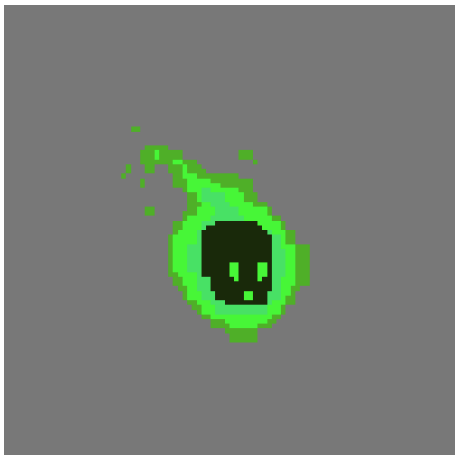
Таблиця 2.3 – Список ворогів та їхні характеристики

Тип ворога	Загальна характеристика
Кістлявий вовк  Рисунок 2.5 – Кістлявий вовк	Захищений обладунками із кісток своїх родичів. Атакує головного героя на ближній дистанції.

Продовження таблиці 2.3

Тип ворога	Загальна характеристика
Вампір-лучник  Рисунок 2.6 – Вампір-лучник	Нижчий вампір, одягнений у червоний плащ та використовує зачарований лук для бою на далекій дистанції.
Вампір-стражник  Рисунок 2.7 – Вампір-стражник	Нижчий вампір, одягнений у червону броню та використовує меч із щитом для кількох атак та захисту, відповідно, на ближній дистанції.
Кажан  Рисунок 2.8 – Кажан	Маленький житель замку, сплячий на стелях та атакуючий головного героя при його наближенні на ближній дистанції.

Продовження таблиці 2.3

Тип ворога	Загальна характеристика
Палаючий дух  Рисунок 2.9 – Палаючий дух	Одна з жертв лорда-вампіра, яка не може знайти спокою. Коли головний герой наближається до духу, то ворог починає переслідувати його та вибухає на ближній дистанції.

У пізніх локаціях гравцеві зустрічаються складніші вороги зі зміненими характеристиками: зі збільшеною кількістю здоров'я і шкоди, що наноситься. Завдяки перезарядці ворожого вміння з'являються вільні проміжки часу, в яких ворог не робить жодних дій і гравець має можливість контратакувати.

Усі вороги мають схожу поведінку – вони реагують на зустрічі з головним героєм. Для цього вони мають зону виявлення, яка може відрізнятися як за розміром, так і формою залежно від типу ворога. Наприклад, вороги з ближньою атакою можуть мати більш обмежену зону виявлення, тоді як вороги з дальньою атакою мають ширший радіус видимості.

Вороги, які мають ближню атаку, мають додаткову область ідентифікації гравця, при контакті з якою вони починають завдавати шкоди. У свою чергу, вороги з дальньою атакою починають атакувати гравця після виявлення його у своїй області видимості. Приклад взаємодії ворога, маючого ближню атаку, з головним героєм зображено на рис. 2.10.

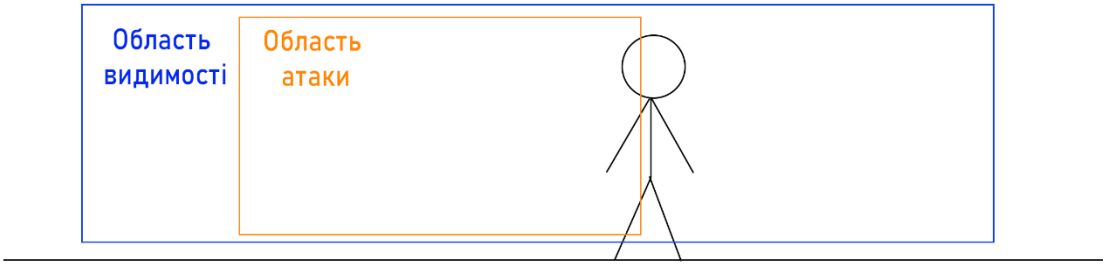


Рисунок 2.10 – Приклад взаємодії ворога з головним героєм

2.8 Ігрові бонуси

Ігрові бонуси є додатковими елементами у грі, які розширюють можливості головного героя, покращують ігровий досвід та стимулюють до дослідження ігрового світу. Вони також є способом балансування ігрового процесу та можуть змінюватися в залежності від рівня складності гри. Види ігрових бонусів наведено в табл. 2.4.

Таблиця 2.4 – Список ігрових бонусів та їхні характеристики

Вид бонусу	Загальна характеристика
Очко досвіду  Рисунок 2.11 – Очко досвіду	Випадає із знищених ворогів. Дозволяє отримати новий рівень головного героя. При підвищенні рівня характеристики також збільшуються
Фрагмент душі  Рисунок 2.12 – Фрагмент душі	Випадає із знищених ворогів. Надає доступ до нової вампірської здібності в дереві вмінь.

Продовження таблиці 2.4

Вид бонусу	Загальна характеристика
Пляшка крові  Рисунок 2.13 – Пляшка крові	Випадає із знищених ворогів. Відновлює здоров’я гравця на певний відсоток.

Гравець має доступ до ігрового інвентарю, де може зберігати зібрані ігрові бонуси. Збираючи бонуси у свій інвентар, гравець може використати їх у зручний час, щоб покращити свої вміння, підвищити шанси на виживання чи подолати складні ділянки локацій. Наприклад, «Фрагмент душ» є валютою у торговця-привида, за яку можна купити сувої, кожна з яких відповідає вампірській здібності із дерева вмінь. «Очки досвіду» дозволяють гравцеві отримати новий рівень, тим самим покращувати свої характеристики.

Ігрові бонуси надають гравцеві додаткові можливості та ресурси у процесі гри, сприяючи розкриттю потенціалу ігрового персонажа.

2.9 Кінцеві автомати віртуальних персонажів

Матриця взаємодій токенів охоплює різноманітні елементи ігрового світу. Вона є діаграмою, що відображає всі можливі взаємодії між ними. У матриці взаємодій зазначені, які дії можуть здійснювати різні токени та які наслідки ці дії можуть мати. В узагальненій матриці взаємодій токенами є головний герой, ворог, перешкода, ігровий бонус та платформа. Узагальнена матриця взаємодій токенів зображена на рис. 2.14.

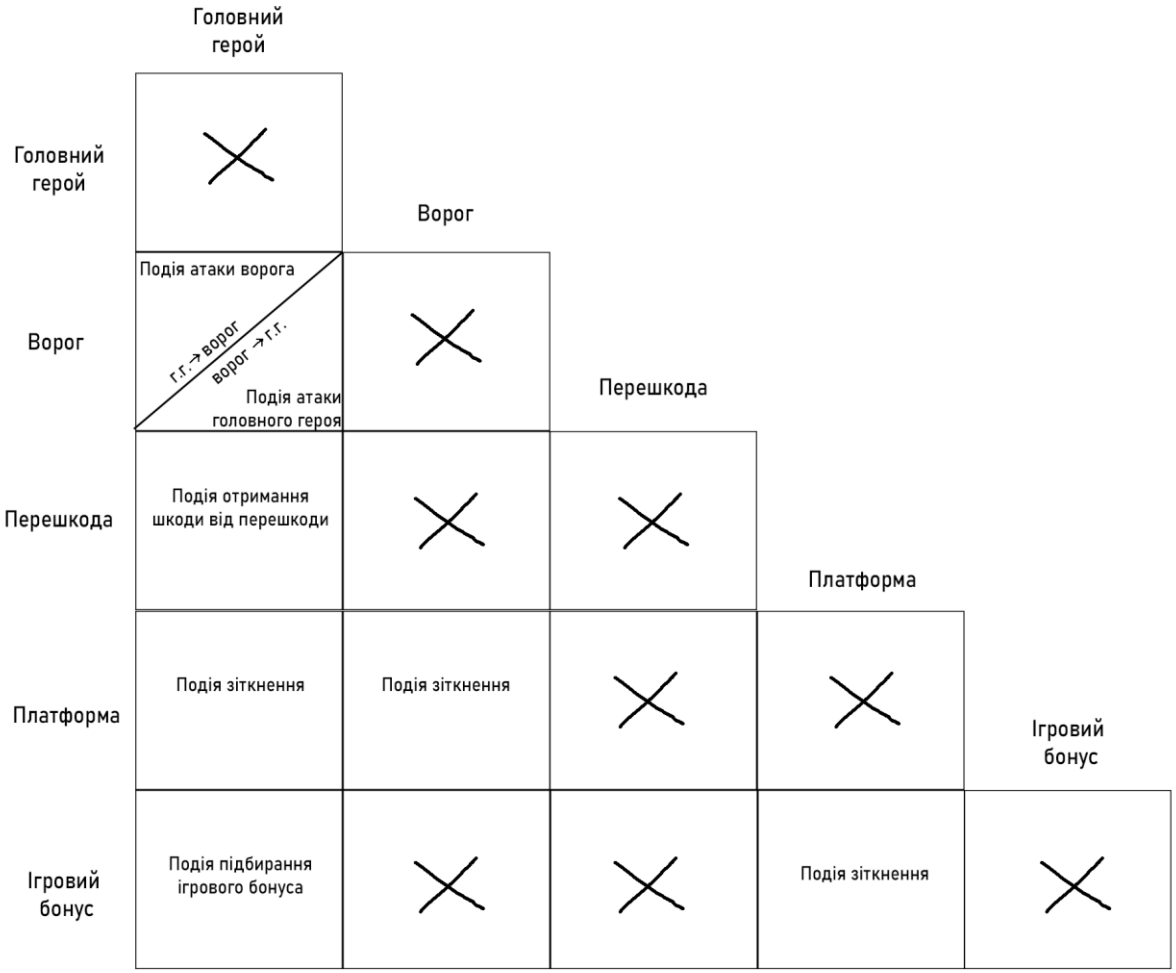


Рисунок 2.14 – Узагальнена матриця взаємодій токенів

Наприклад, в даній матриці взаємодій можна побачити, що головний герой може атакувати ворогів, збирати ігрові бонуси та отримати шкоду від перешкод. У той же час, вороги не взаємодіють з ігровими бонусами.

Кінцеві автомати віртуальних персонажів взаємодіють із цією матрицею та визначають поведінку токена залежно від його поточного стану та зовнішніх подій. Токенами, що мають чисельну кількість станів, є головний герой та ворог. Кінцеві автомати представляють собою важливий механізм управління поведінкою персонажів.

Спочатку розглядається кінцевий автомат головного героя. Оскільки повна схема цього кінцевого автомату занадто громіздка, вирішено зобразити її в більш спрощеній формі – без умов переходу між станами. Спрощену схему кінцевого автомату головного герою наведено в додатку А.

Даний кінцевий автомат описує можливу поведінку головного героя з наступною кінцевою кількістю станів: «Спокій», «Біг», «Стрибок», «Використання драбини», «Базова атака», «Голодний укус», «Гострі голки», «Тіньовий крок», «Відновлювання здоров’я» та «Смерть». Початковим станом кінцевого автомату є «Спокій», а кінцевим станом – «Смерть».

Ця схема має такі вхідні дані: «Перевірка поточного здоров’я», «Перевірка клавіші відновлення здоров’я», «Перевірка клавіші стрибку», «Перевірка клавіші базової атаки», «Перевірка зіткнення з драбиною та клавіші підйому по драбині», «Перевірка клавіші і перезаряджання здібності Голодний укус», «Перевірка клавіші та перезаряджання здібності Тіньовий крок», «Перевірка зарядів крові та клавіші здібності Гострі голки», «Перевірка завершення стрибку», «Перевірка завершення атаки на поверхні», «Перевірка завершення атаки в повітрі», «Перевірка завершення відновлення здоров’я», «Перевірка завершення здібності Голодний укус», «Перевірка завершення стану Тіньовий крок», «Перевірка завершення здібності Гострі голки».

Коли клавіша руху натиснута, головний герой переходить зі стану «Спокій» у стан «Біг». Так само й навпаки: якщо головний герой знаходиться у стані «Біг» і клавіша не є натиснутою, головний герой переходить до стану «Спокій». Якщо головний герой знаходиться у стані «Спокій» або «Біг», і натиснута клавіша, відповідальна за стрибок, – то відбувається перехід до стану «Стрибок» і повертається назад до стану «Спокій», коли стан «Стрибок» закінчується. У той момент, коли головний герой зіткнувся з драбиною і натиснув клавішу, відповідальну за підйом по драбині, відбувається перехід зі стану «Спокій» або «Біг» до стану «Використання драбини». Для виходу з цього стану необхідно натиснути клавішу стрибку і в цей момент відбувається перехід до стану «Стрибок». До стану «Базова атака» можна перейти з трьох станів, а саме: «Спокій», «Біг» і «Стрибок», натисненням клавіші, яка відповідає за базову атаку. Після закінчення цього стану головний герой повертається у попередній стан відносно стану «Базова

атака». Перехід до стану «Голодний укус» або «Тіньовий крок» відбувається у той момент, коли перезаряджання вміння скінчилося та натиснута відповідна клавіша. Такий перехід можливий зі станів «Спокій» та «Біг» та при завершенні стану «Голодний укус» головний герой повертається до минулого стану. Якщо в головного героя є в наявності необхідна кількість зарядів крові та натиснута відповідна клавіша, герой переходить до стану «Гострі голки» зі станів «Спокій» або «Біг». Якщо у головного героя кількість поточного здоров'я дорівнює нулю, відбувається перехід до стану «Смерть» з будь-якого з інших можливих станів.

Цей кінцевий автомат є найскладнішим по конструкції, тому що в ньому міститься найбільша кількість станів. Зі збільшенням числа станів розмір кінцевого автомата значно зростає, а кількість переходів між ними збільшується. Однак, саме завдяки такій деталізації можна легко розглянути логіку управління поведінкою головного героя в ігровому світі.

Далі розглядаються кінцеві автомати ворогів. Першим ворогом є Кістлявий вовк.

Кінцевий автомат ворога Кістлявий вовк описує його поведінку з наступною кінцевою кількістю станів: «Патрулювання», «Переслідування», «Спокій», «Ближня атака» та «Смерть». Початковим станом є «Патрулювання», а кінцевим – «Смерть».

Ці стани мають такі вхідні дані: «Перевірка поточного здоров'я», «Перевірка виявлення головного герою в області видимості», «Перевірка виявлення головного герою в області атаки», «Перевірка перезаряджання атаки» та «Перевірка завершення атаки».

Схему кінцевого автомату ворога Кістлявий вовк зображено на рис. 2.15.

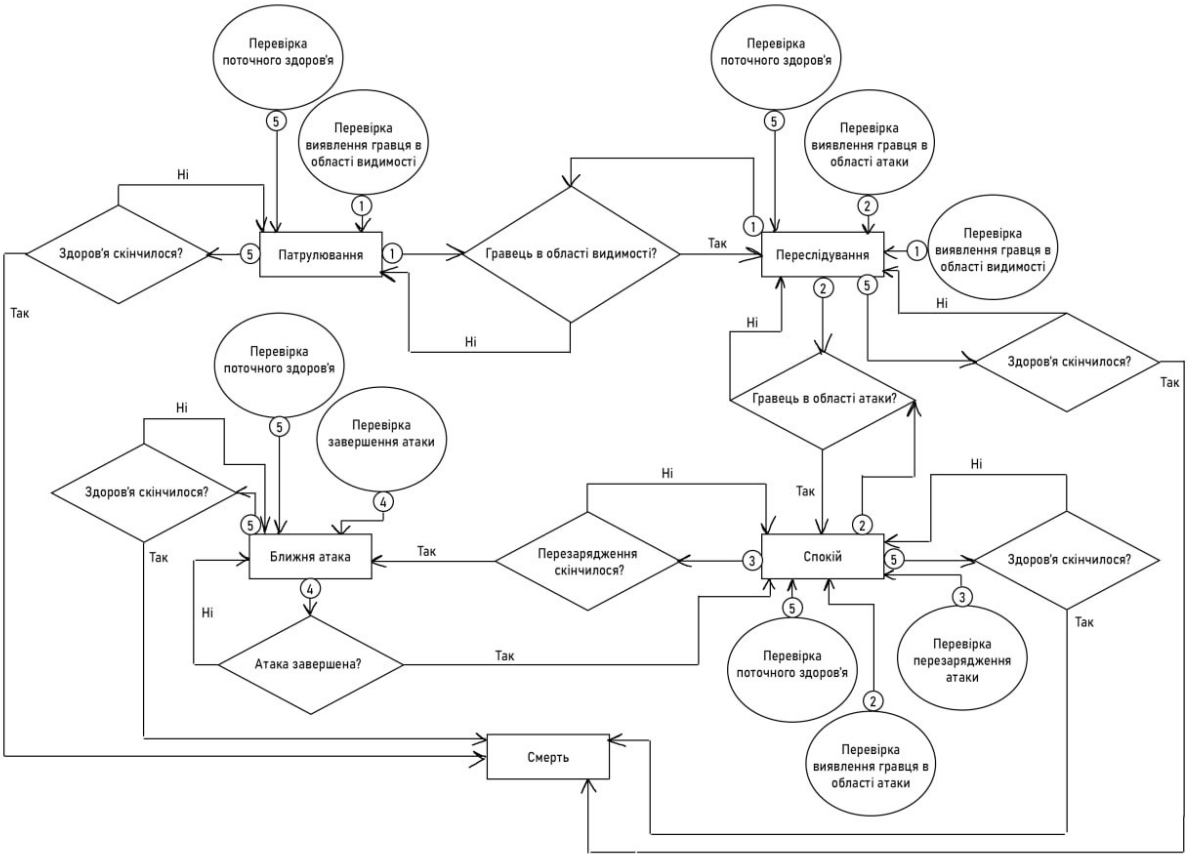


Рисунок 2.15 – Схема кінцевого автомату ворога Кістлявий вовк

Коли головний герой потрапляє в область видимості ворога, то Кістлявий вовк змінює свій стан з «Патрулювання» на «Переслідування». А якщо головний герой виходить з цієї зони, то Кістлявий вовк навпаки змінює стан з «Переслідування» на «Патрулювання». При виявленні головного герою в області атаки під час стану «Переслідування» стан змінюється на «Спокій». Якщо головний герой вийшов з області атаки, то відбувається перехід зі стану «Спокій» в стан «Переслідування». Якщо поточний стан «Спокій» та перезаряджання ворожої атаки скінчилося, то стан змінюється на «Ближню атаку». При завершенні атаки стан кінцевого автомату повертається на стан «Спокій». Якщо кількість поточного здоров'я Кістлявого вовку дорівнює нулю, то з будь-якого стану відбувається перехід до стану «Смерть».

Додавання стану «Спокій» у кінцевий автомат ворогів, включаючи ворога Кістлявий вовк, обумовлено необхідністю представлення нейтральної

поведінки ворога в очікуванні можливості атаки. Хоча ворог може бути на потрібній відстані для атаки, але його здатність атакувати може бути на перезарядці. У стані «Спокій» ворог очікує, доки його здатність атаки не буде готова до використання, надаючи гравцеві можливість контратакувати. Таким чином, цей стан дає гравцеві час на реагування, а також зрівнює можливості гравця та ворога.

Наступний кінцевий автомат, який розглядається, – є автомат ворога Вампір-лучник. Схему кінцевого автомату ворога Вампір-лучник зображено на рис. 2.16.

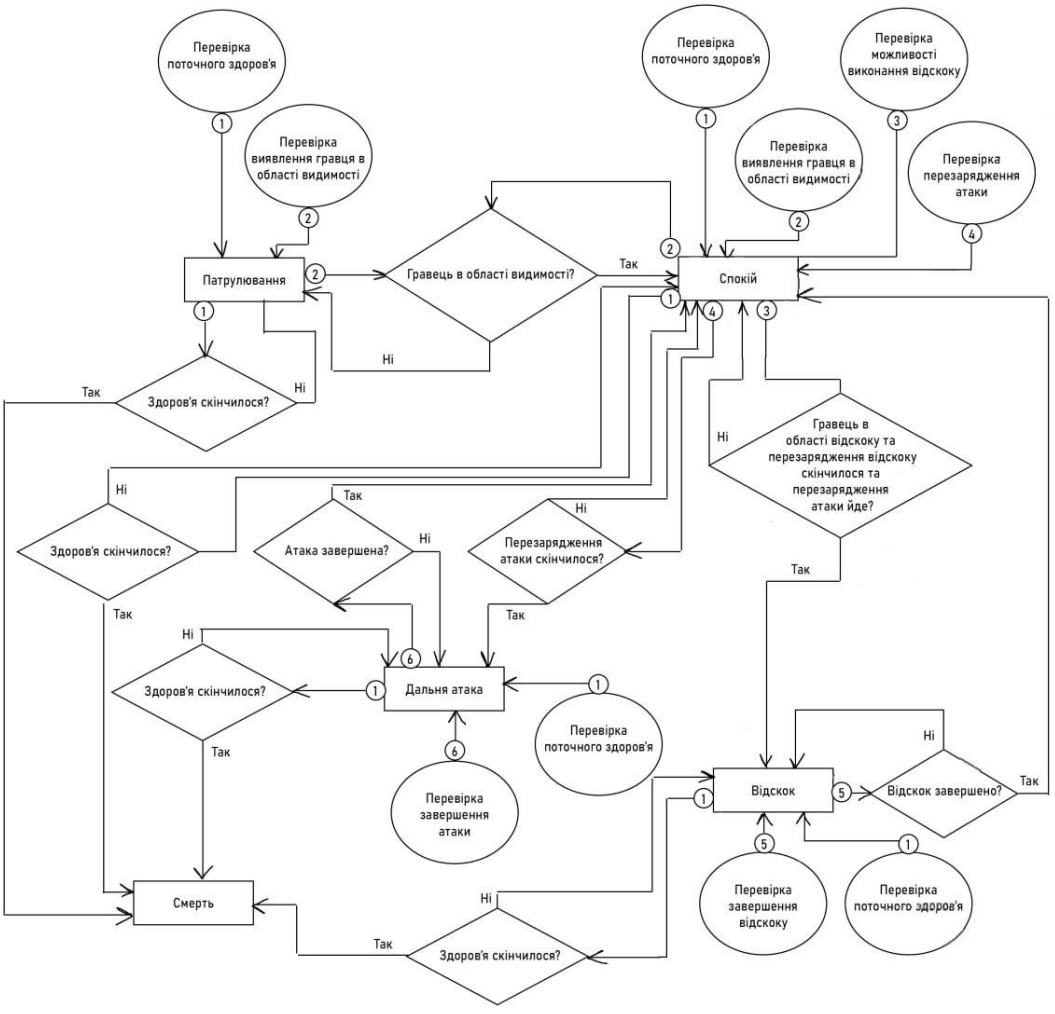


Рисунок 2.16 – Схема кінцевого автомату ворога Вампір-лучник

Даний кінцевий автомат описує поведінку ворога Вампір-лучник з наступною кінцевою кількістю станів: «Патрулювання», «Спокій»,

«Відскок», «Дальня атака», «Смерть». Початковим станом серед них є «Патрулювання», а кінцевим – «Смерть».

Ці стани мають такі вхідні дані: «Перевірка поточного здоров’я», «Перевірка виявлення головного герою в області видимості», «Перевірка можливості виконання відскоку», «Перевірка перезаряджання атаки», «Перевірка завершення відскоку» та «Перевірка завершення атаки».

Коли головний герой потрапляє в область видимості Вампіра-лучника, то відбувається перехід зі стану «Патрулювання» у стан «Спокій». Якщо головний герой вийшов з області видимості, стан змінюється зі «Спокій» в «Патрулювання». Зі стану «Спокій» Вампір-лучник може перейти до стану «Дальня атака» або стану «Відскок» за наступними умовами: в «Дальню атаку», якщо перезаряджання атаки скінчилося, а в «Відскок» – якщо головний герой увійшов у область відскоку, перезаряджання дальньої атаки йде та перезаряджання відскоку скінчилося. Назад зі станів вмінь «Дальня атака» та «Відскок» відбувається перехід, коли стан завершено. Якщо кількість поточного здоров’я Вампіра-лучника дорівнює нулю, то з будь-якого стану відбувається перехід до стану «Смерть».

Особливу увагу слід приділити умові переходу до стану «Відскок». Оскільки обидві умови переходу, пов'язані з перезарядженням атаки та відскоку, можуть завершитися одночасно, необхідно встановити пріоритет між ними. Шляхом детального визначення умов переходу забезпечується, що при одночасному закінченні перезарядки вмінь «Дальня атака» та «Відскок», першочергово виконується «Дальня атака», а не «Відскок».

Далі розглядається кінцевий автомат ворога Вампір-стражник. Схему кінцевого автомату ворога Вампір-стражник зображено на рис. 2.17.

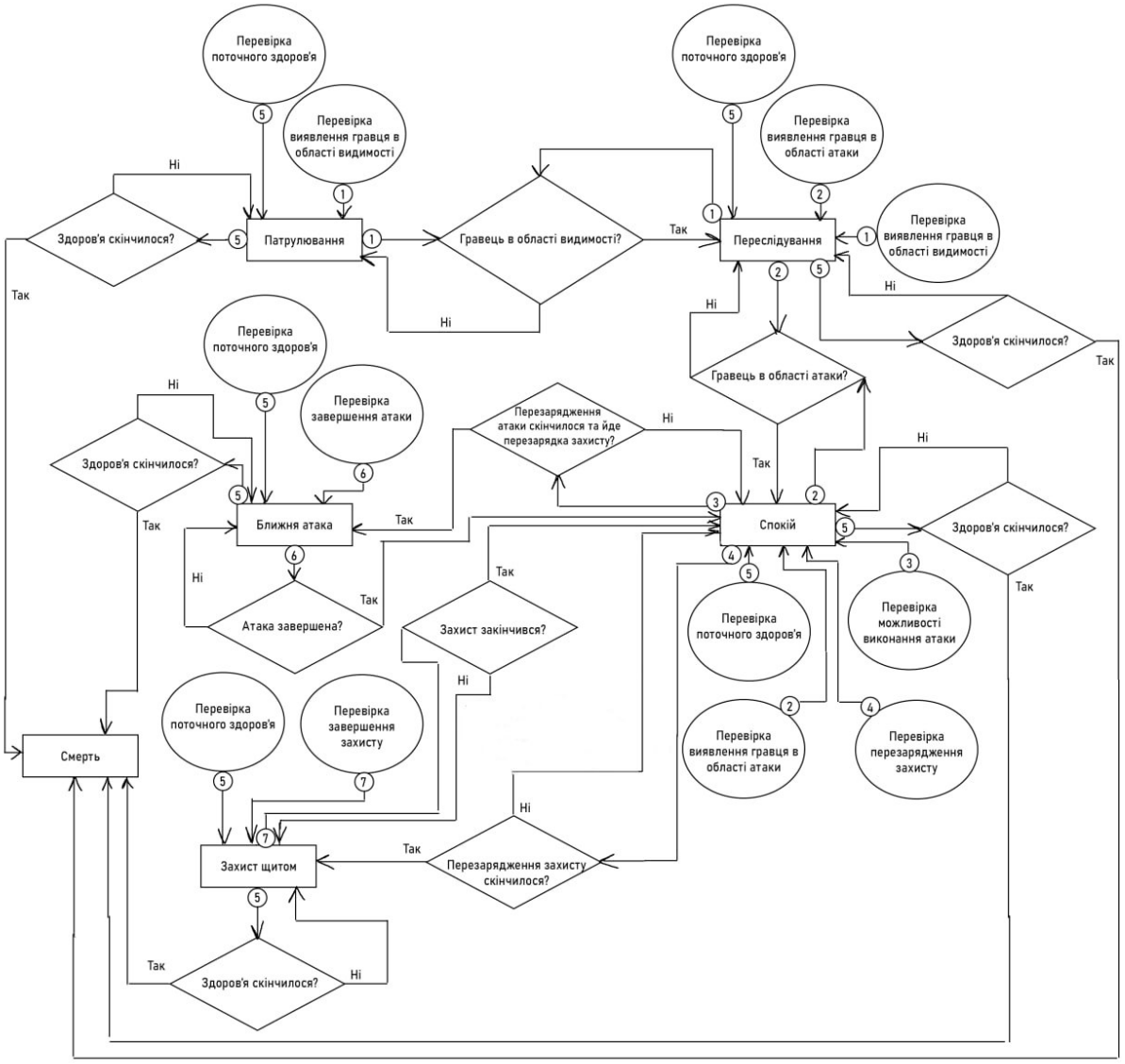


Рисунок 2.17 – Схема кінцевого автомату ворога Вампір-стражник

Даний кінцевий автомат описує поведінку ворога Вампір-стражник з наступною кінцевою кількістю станів: «Патрулювання», «Переслідування», «Спокій», «Ближня атака», «Захист щитом» та «Смерть». Початковим станом серед них є «Патрулювання», а кінцевим – «Смерть».

Ці стани мають такі вхідні дані: «Перевірка поточного здоров’я», «Перевірка наявності головного герою у області видимості», «Перевірка наявності головного герою у області атаки», «Перевірка можливості виконання атаки», «Перевірка перезаряджання захисту», «Перевірка завершення атаки» та «Перевірка завершення захисту».

Коли головний герой потрапляє в область видимості, Вампір-стражник

змінює свій стан з «Патрулювання» на «Переслідування». Якщо головний герой виходить з цієї зони, Вампір-стражник навпаки змінює стан з «Переслідування» на «Патрулювання». При виявленні головного герою в області атаки під час стану «Переслідування», стан змінюється на «Спокій». Але якщо головний герой вийшов з даної області, відбувається перехід зі стану «Спокій» у стан «Переслідування». Зі стану «Спокій» Вампір-стражник може перейти до стану «Ближня атака» або стану «Захист щитом» за такими умовами: в «Ближню атаку», якщо перезаряджання захисту щитом йде та перезаряджання ближньої атаки скінчилося, а в «Захист щитом», якщо перезаряджання захисту скінчилося. Назад зі станів вмінь «Ближня атака» та «Захист щитом» відбувається перехід до «Спокою», коли стан завершено. Якщо кількість поточного здоров'я Вампіра-стражника дорівнює нулю, то з будь-якого стану відбувається перехід до стану «Смерть».

Умови переходу в стан вмінь Вампіра-стражника побудовані аналогічним чином, як і в кінцевому автоматі Вампіра-лучника. Особлива увага приділяється ситуаціям, коли перезаряджання різних вмінь завершується одночасно. Чітко визначені умови переходу забезпечують пріоритет виконання ворожих умінь.

Розглядається наступний кінцевий автомат ворога Кажан. Даний кінцевий автомат описує поведінку ворога Кажан з наступною кінцевою кількістю станів: «Очікування», «Спокій», «Переслідування», «Ближня атака» та «Смерть». Початковим станом серед них є «Очікування», а кінцевим – «Смерть».

Ці стани мають такі вхідні дані: «Перевірка поточного здоров'я», «Перевірка виявлення головного героя у області видимості», «Перевірка виявлення головного героя у області атаки», «Перевірка перезарядження атаки», «Перевірка завершення атаки».

Схему кінцевого автомату ворога Кажан зображено на рис. 2.18.

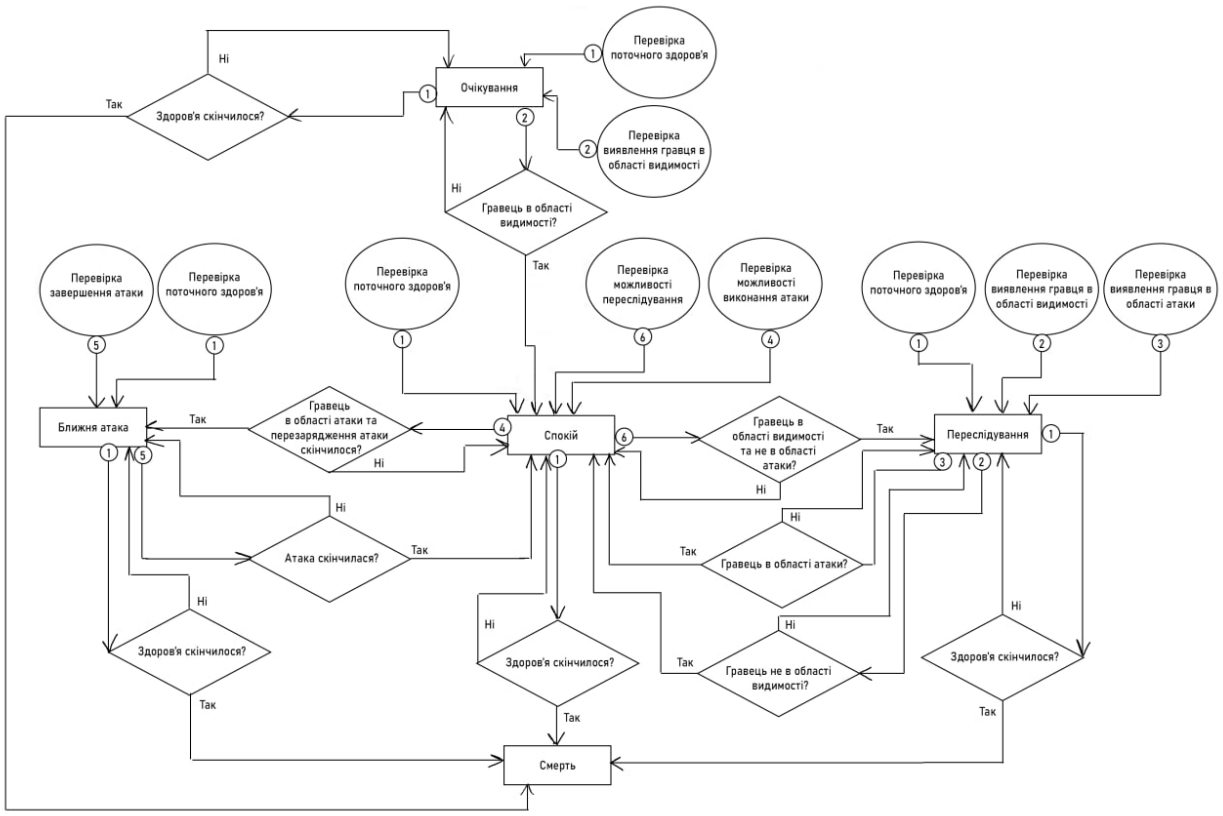


Рисунок 2.18 – Схема кінцевого автомату ворога Кажан

Коли головний герой потрапляє в область видимості ворога, поточний стан з початкового «Очікування» змінюється на «Спокій». Якщо головний герой в цей час залишається в області видимості і не знаходиться у області атаки, поточний стан змінюється на «Переслідування». Якщо головний герой виходить з цієї області, то стан змінюється на «Спокій» та залишається таким до тих пір, поки головний герой не потрапить у область видимості знов. Якщо поточним станом є «Переслідування» та головний герой потрапив у область атаки, то відбувається перехід у стан «Спокій». Перехід зі стану «Спокій» в стан «Ближня атака» відбувається, коли перезарядка атаки завершено і головний герой у області атаки. Назад до стану «Спокій» Кажан повертається коли стан «Ближня атака» скінчено. Якщо кількість поточного здоров’я ворога дорівнює нулю, то з будь-якого стану відбувається перехід до стану «Смерть».

Відмінною особливістю даного кінцевого автомата є стан «Очікування», в якому Кажан знаходиться до виявлення головного героя. У

цей час ворог прихован від поглядів гравця, висячи на стелі та готуючись до нападу. Після виявлення головного героя ворог переходить у стан «Спокій» і починає активно переслідувати ціль. Проте, якщо головний герой пропадає з поля зору, Кажан вже не повертається до стану «Очікування».

Наступним кінцевим автоматом є автомат ворога Палаючий дух. Схему кінцевого автомату ворога Палаючий дух зображено на рис. 2.19.

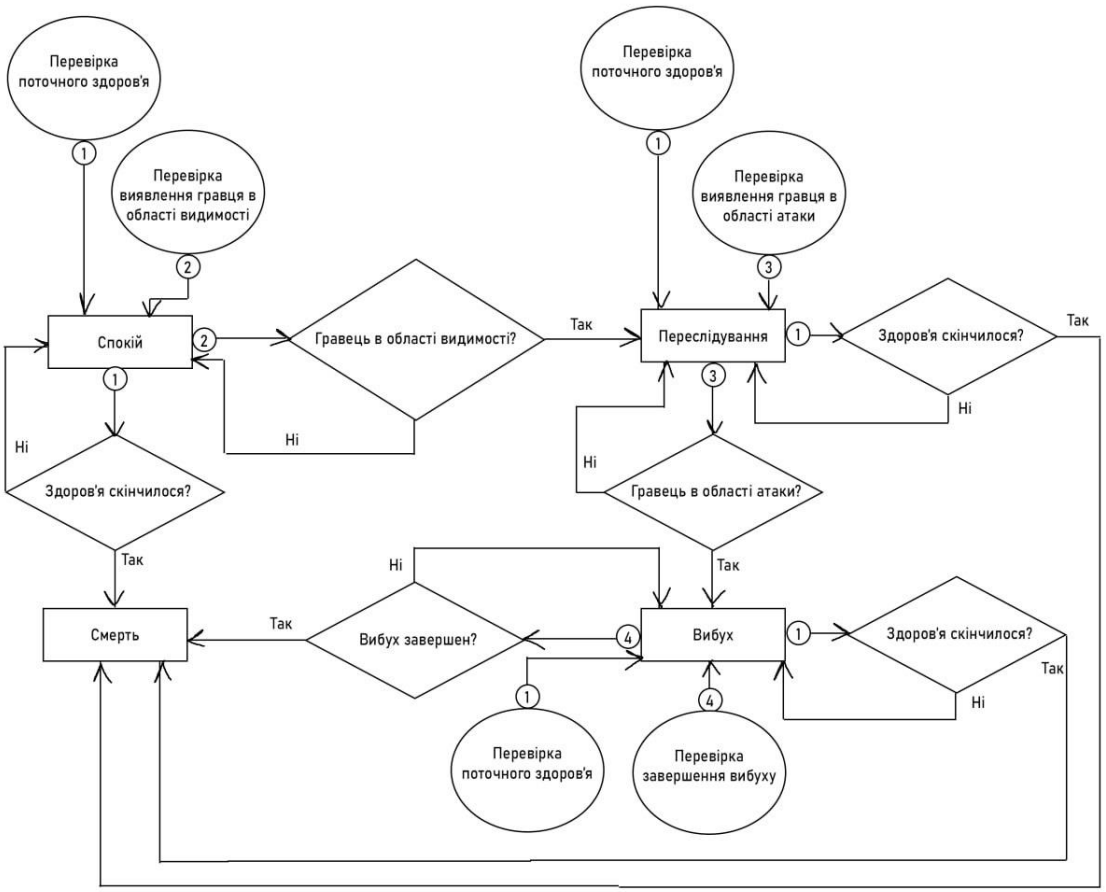


Рисунок 2.19 – Схема кінцевого автомату ворога Палаючий дух

Даний кінцевий автомат описує поведінку ворога Палаючий дух з наступною кінцевою кількістю станів: «Спокій», «Переслідування», «Вибух» та «Смерть». Початковим станом є «Спокій», а «Смерть» – кінцевим станом.

Ці стани мають такі вхідні дані: «Перевірка поточного здоров’я», «Перевірка виявлення головного героя в області видимості», «Перевірка виявлення головного героя», «Перевірка закінчення вибуху».

Коли головний герой потрапляє у область видимості ворога, то відбувається перехід зі стану «Спокій» в «Переслідування». Якщо головний герой потрапив у область атаки, відбувається перехід зі стану «Переслідування» в «Вибух». Коли вибух закінчується, відбувається перехід: в остаточний стан «Смерть». Якщо кількість поточного здоров’я дорівнює нулю, відбувається перехід з будь-якого стану у стан «Смерть».

Палаючий дух виділяється своєю унікальною поведінкою в бою. Після виявлення головного героя він неухильно переслідує його, прагнучи фінального удару – вибуху. Після вибуху ворог не повертається до попередніх станів і просто зникає. Це створює особливу динаміку бою, де гравцеві необхідно швидко реагувати та атакувати ворога до того, як той завдасть смертельного удару, оскільки уникнути його переслідування неможливо.

2.10 Платформа

Гра розробляється на ПК для операційної системи Windows. Системні вимоги для гри приведені у табл. 2.5.

Таблиця 2.5 – Системні вимоги гри

Параметр	Мінімальні вимоги	Рекомендовані вимоги
ОС	Windows 10/11	Windows 10/11
Процесор	Intel i5+	Intel i5+
Оперативна пам’ять	2 GB	4 GB
Відеокарта	Nvidia 450 GTS / Radeon HD 5750	Nvidia GTX 460 / Radeon HD 7800
Місце на диску	4 GB	4 GB

2.11 Модель гри

При запуску гри гравцеві надається головне меню, яке відображається на тлі нічного неба з місяцем. У верхній частині екрана розташовується назва гри, а під ним розміщені чотири кнопки управління: «Почати гру», «Налаштування», «Автори» та «Вихід». Схема меню зображена на рис. 2.20.

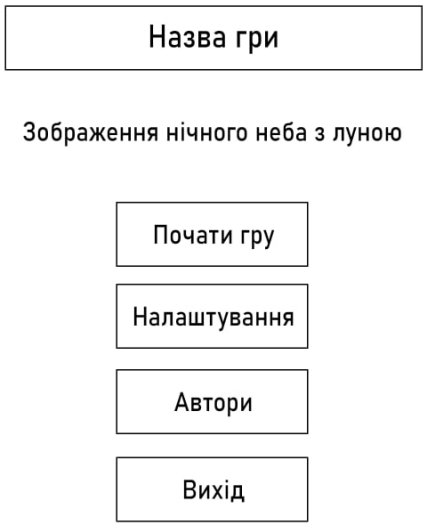


Рисунок 2.20 – Зображення зовнішнього вигляду головного меню гри

Кнопка «Почати гру» дозволяє гравцеві розпочати нову гру або продовжити існуючу, якщо є файл збереження. У розділі «Налаштування» можна встановити параметри, такі як зміна розширення екрана, керування гучністю звуків та музики. Розділ «Автори» містить інформацію про розробників та використані засоби розробки гри. Кнопка «Вийти» дозволяє вийти з гри. Схема налаштувань зображена на рис. 2.21.

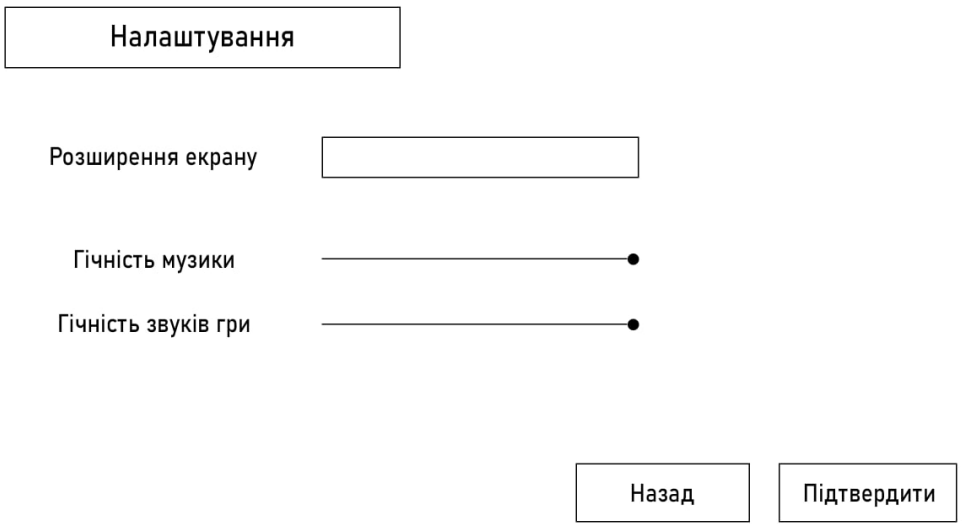


Рисунок 2.21 – Зображення зовнішнього вигляду меню налаштування

При першому запуску гри після натискання кнопки «Почати гру» гравцеві показується кат-сцена, яка розповідає передісторію головного героя. Потім гравець потрапляє в ігровий світ, де в нижньому лівому куті екрана відображається інтерфейс гравця. Схема інтерфейсу гравця зображена на рис. 2.22.

Рисунок 2.22 – Зображення зовнішнього вигляду інтерфейсу гравця

У цьому інтерфейсі представлені шкали здоров'я та зарядів крові. Також присутні три квадрати, що відображають кількість пляшок крові та перезарядку обраних гравцем вампірських здібностей з дерева вмінь. В демонстраційній версії гри відображаються перезарядження здібностей «Голодний укус» та «Тіньовий крок», які доступні гравцеві з самого початку

гри. Додатково відображаються кількість очок досвіду та поточний рівень головного героя.

У грі представлено 13 локацій, кожна з яких є кімнатою в замку лорда. Список усіх ігрових локацій наведено у табл. 2.6.

Таблиця 2.6 – Список ігрових локацій

Назва локації	Опис
Тюремні камери	Місце, в якому замикають жертв лорда, щоб зламати їхній дух і змусити забути про своє людське життя.
Прогулянковий хол	Місце призначене для роздумів лорда в дощову погоду або інші негоди, коли не можна вийти до саду.
Кухня	Місце, в якому лорд іноді тішить себе вишуканими людськими стравами.
Бібліотека	Місце розвитку розуму, заповнене тисячами книжок різного жанру.
Галерея мистецтв	Місце, де представлені мармурові статуї та статуї лицарів, гобелени та живопис відомих художників. Сюди лорд приводить своїх гостей.
Ритуальна кімната	Місце для проведення давніх ритуалів, які збільшують силу лорда-вампіра. Стіни вкриті гравюрами, що зображують дивні символи та моторошні істоти.
Кімната м'ясника	Похмурий простір, де висять гаки та інструменти, що використовуються для оброблення тіл та приготування їжі. Стіни вкриті кров'ю, і стоять столи, за якими слуга лорда проводить свої експерименти над бранцями.

Продовження таблиці 2.6

Назва локації	Опис
Читальний зал	Місце для вивчення давніх текстів та таємних знань, які зчерпає лорд.
Бальний зал	Порожня кімната, де колись проходили танці і лунала музика, але нинішній лорд більше не цікавиться цим.
Алхімічна лабораторія	У цьому місці лорд-вампір проводить експерименти, щоб відтворити унікальне зілля заворожування, яке допоможе йому не використовувати свої вампірські чари на полюванні людей.
Дахи замку	Високий та старий простір, розташований над головним корпусом замку. У цьому місті лорд може розмірковувати над своїми планами, спостерігаючи своє володіння з висоти пташиного польоту.
Оранжерея	У цій світлій кімнаті вирощуються різноманітні екзотичні рослини, які майстерно доглядають садівники лорда.
Покої Лорда	Просторі та розкішні покої, де лорд відпочиває та планує свої подальші дії, іноді граючи на фортепіано. Стіни прикрашені портретами предків та трофеями з полювання, а з вікна відкривається вид на темні ліси та далекі гірські вершини.

У демонстраційній версії гри доступні перші три локації: «Тюремні камери», «Прогулянковий хол» та «Бібліотека».

Крім описаних локацій, у грі передбачені різні шляхи, якими може рухатися гравець, досліджуючи замок. Схема навігаційної карти гри зображена на рис. 2.23.

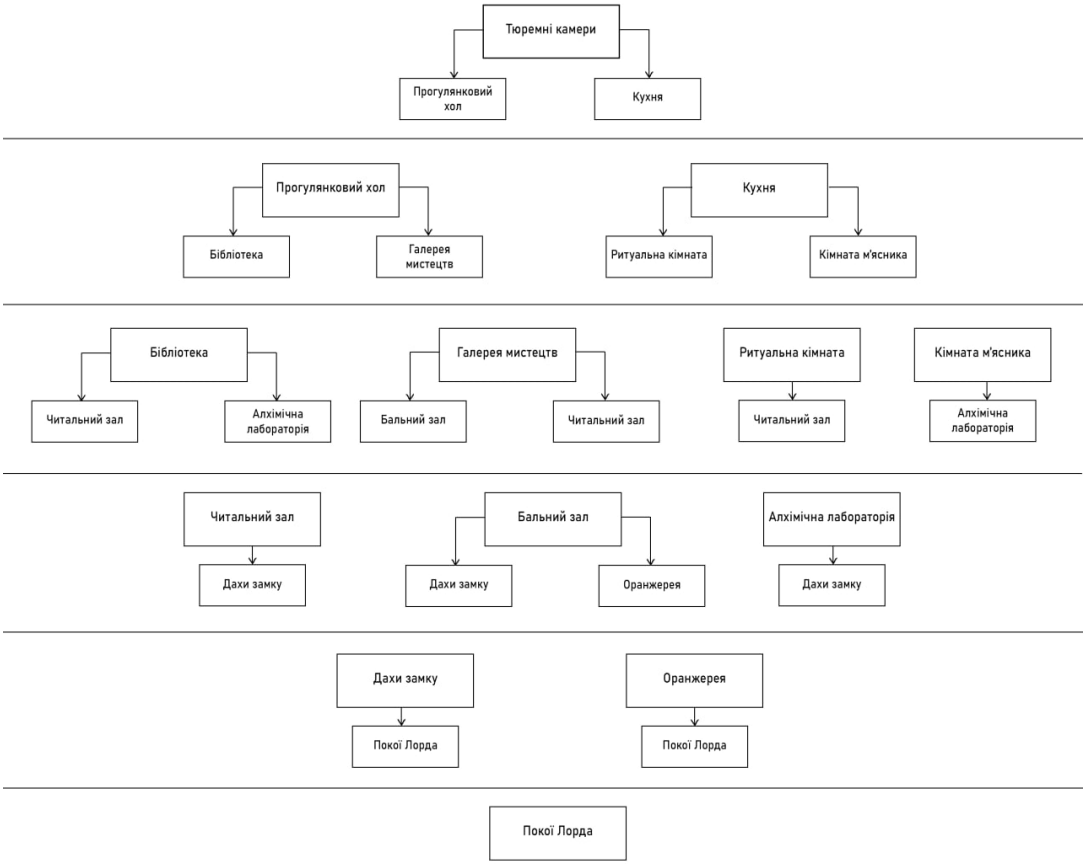


Рисунок 2.23 – Схема навігаційної карти гри

Усі локації поділені на рівні складності. Усього налічується шість рівнів, які показані за допомогою ліній на схемі. Наприклад, початкова локація «Тюремні камери» знаходиться на першому рівні, а «Прогулянковий хол» і «Кухня» – на другому рівні. У міру просування вглиб замку рівень складності поступово зростає, надаючи гравцеві більш небезпечних ворогів. Така структура забезпечує постійний інтерес для гравця протягом усієї гри.

Остання локація, «Покої Лорда», є фінальною точкою, де гравець очікує сутичку з головним босом гри. Водночас, у кількох локаціях, таких як «Читальний зал», «Бальний зал» та «Кімната м'ясника», також очікується бій з босами. Гравець має можливість вибрати свій шлях і обійти одного з цих босів, слідуючи іншому маршруту. Наприклад, гравець може пропустити боса з «Читального залу» або з «Бального залу».

2.12 Формули

У грі використовується певна кількість важливих формул, розрахунок яких визначають різні аспекти ігрового процесу.

Формула розрахунку максимального здоров'я головного героя та ворога після підняття рівня H_{max}:

$$H_{max} = H_{base} \cdot (1 + H_{inc} \cdot (L - 1)), \quad (2.1)$$

де H_{base} – базове значення максимального здоров'я персонажа;

H_{inc} – процент збільшення здоров'я за кожен рівень (наприклад, 0,6 або 60%);

L – поточний рівень персонажа.

Формула розрахунку шкоди атакуючого вміння головного героя та ворога після підняття рівня D:

$$D = D_{base} \cdot (1 + D_{inc} \cdot (L - 1)), \quad (2.2)$$

де D_{base} – базове значення шкоди атакуючого вміння персонажа;

D_{inc} – процент збільшення шкоди атакуючого вміння за кожен рівень (наприклад, для головного героя – 0,06 або 6%, для ворога – 0,05 або 5%);

L – поточний рівень персонажа.

Формула розрахунку шансу випадіння пляшки з кров'ю зі знищеного ворога P:

$$P = P_{base} + k \cdot (L_e - L_p), \quad (2.3)$$

де P_{base} – базовий шанс випадіння пляшки з кров'ю (наприклад, 0,2 або 20%);

k – коефіцієнт змінення шансу на кожен рівень різниці головного героя

та ворога (наприклад, 0,05 або 5%);

L_e – поточний рівень ворога;

L_p – поточний рівень головного героя.

Формула розрахунку максимальної кількості пляшок з кров'ю після підняття рівня головного героя F_{max}:

$$F_{max} = F_{base} + (\frac{L-1}{k}), \tag{2.4}$$

де F_{base} – базова максимальна кількість пляшок з кров'ю;

L – поточний рівень головного героя;

k – коефіцієнт збільшення кількості пляшок з кров'ю за кожні певні рівні головного героя (наприклад, 5).

Формула розрахунку проценту відновлення здоров'я головного героя H_{rest}:

$$H_{rest} = max(H_{max} \cdot \big(1 - (L_{curr} \cdot P)\big), 1), \tag{2.5}$$

де H_{max} – максимальна кількість здоров'я головного героя;

L_{curr} – поточний рівень головного героя;

P – коефіцієнт зниження ефективності відновлення здоров'я за рівень головного героя (наприклад, 0,05 або 5%).

Формула розрахунку кількості отриманих зарядів крові після використання здібності головного героя «Голодний укус» G:

$$G = 1 + \left(\frac{L_{curr}}{k}\right), \tag{2.6}$$

де L_{curr} – поточний рівень головного героя;

k – коефіцієнт збільшення кількості зарядів крові за кожні певні рівні головного героя (наприклад, 3).

Формула розрахунку максимальної кількості зарядів крові після підняття рівня головного героя B_{char}:

$$B_{char} = B_{base} + 1 \cdot (\frac{L_{curr}}{k}), \quad (2.7)$$

де B_{base} – базова максимальна кількість зарядів крові;

L_{curr} – поточний рівень головного героя;

k – коефіцієнт збільшення максимальної кількості зарядів крові за кожні певні рівні головного героя (наприклад, 3).

Формула розрахунку базової кількості очок досвіду головного героя E_{base}:

$$E_{base} = \frac{E_{next} + L_{mult}}{N}, \quad (2.8)$$

де E_{next} – кількість очок досвіду, яка необхідна для підняття рівня головного героя;

L_{mult} – коефіцієнт для збільшення рівня при знищенні ворогів (наприклад, 1,5);

N – кількість ворогів на рівні (наприклад, 15).

Формула розрахунку кількості отриманих очок досвіду зі знищеного ворога E:

$$E = E_{base} + D_{mult} \cdot (L_e - L_p), \quad (2.9)$$

де E_{base} – базова кількість очок досвіду;

D_{mult} – коефіцієнт для різниці рівнів головного героя та ворога (наприклад, 0,05 або 5%);

L_e – поточний рівень ворога;

L_p – поточний рівень головного героя.

Формула розрахунку кількості очок досвіду для підняття рівня головного героя L_{next}:

$$L_{next} = L \cdot k, \tag{2.10}$$

де L – поточний рівень головного героя;

k – коефіцієнт збільшення очок досвіду для кожного рівня головного героя (наприклад, 150).

2.13 Графіки та анімація

Всі присутні графічні елементи в грі наведені в табл. 2.7.

Таблиця 2.7 – Графічні елементи гри

Елемент	Опис
Інтерфейс	Графічні зображення, логотип гри, кнопки, іконки вмінь та бонусів, шкали гравця
Набір снарядів	Стріли, магічні кулі, криваві голки
Джерела світла	Підлогові та настінні смолоскипи
Бонус елементи	3 вида префабів
Елементи інтер’єру замку	Вази, портрети, книжні полки та шафи, величезні вікна, клітки

В розробленій грі головний герой повинен мати наступний перелік анімацій, який можна розглянути в табл. 2.8.

Таблиця 2.8 – Анімації головного героя

Тип анімації	Опис
Анімація простою	Програється під час бездіяльності персонажа
Анімація бігу	Звичайний біг персонажа

Продовження таблиці 2.8

Тип анімації	Опис
Анімація стрибку	Стрибок вгору та спуск вниз
Анімація отримання шкоди	Підстрибування з місця, піднімаючи трохи руки вгору
Анімація гибелі	Персонаж падає та розчиняється в пилюку. Програється під час втрати усієї кількості здоров’я
Анімація базової атаки	Мах кігтями в сторону цілі
Анімація тіньового кроку	Зникнення в тінь і поява з неї за мить
Анімація гострих гілок	Кидок тоненькою голкою з крові в ціль
Анімація голодного укусу	Ривок вперед, чіпляючись іклами
Анімація лазіння по драбині	Програється під час лазіння по драбині, змінюючи положення рук та ніг

2.14 Звуки та музика

Звуковий супровід у грі дуже збагачує ігровий досвід гравця. Музичні ефекти гри представлені у табл. 2.9.

Таблиця 2.9 – Музикальні ефекти гри

Елемент	Опис
Фонова музика головного меню та локацій	-
Звук бігу	-
Звук отримання шкоди	-
Звук приземлення	-
Звуки використання магії крові	-
Звук ривку	-
Звук гибелі	

2.15 Висновки до другого розділу

а) Розроблено концептуальний та дизайнерський документи, які описують основні компоненти та особливості ігрового проекту.

б) Розроблено матрицю взаємодії базових елементів гри (токенів) та кінцеві автомати головного героя та ворогів, за допомогою яких промодельовано поведінку персонажів.

3 РОЗРОБКА ДЕМОНСТРАЦІЙНОЇ ВЕРСІЇ ГРИ

3.1 Засоби розробки

Для створення демонстраційної версії гри використано три засоби розробки: Unity версії 2022.3.9f1, Microsoft Visual Studio 2022 та Adobe Photoshop 2020.

Unity – ігровий рушій для розробки двовимірних та тривимірних відеоігор та інтерактивних додатків. Основні особливості Unity включають кроссплатформенну підтримку, що дозволяє розробляти ігри для більш ніж 25 різних платформ, включаючи настільні комп'ютери, мобільні пристрої, ігрові консолі та пристрої віртуальної реальності, що дозволяє іграм запускатися на різних пристроях без змін коду.

Перевагами даного рушія є наявність потужного візуального середовища розробки, підтримка безлічі плагінів та бібліотек, а також активна спільнота розробників. Unity підтримує кілька мов програмування для написання ігрової логіки, включаючи C# та Boo (подібний до Python). Це дозволяє розробникам вибирати найбільш зручну для них мову програмування або навіть комбінувати їх для оптимального результату.

Інтерфейс редактора Unity складається з набору вікон, які можна організувати на свій розсуд. Він включає інспектор об'єктів, оглядач ресурсів, сцени, ієрархії та ігрове вікно. Завдяки гнучкому середовищу розробники можуть зручно керувати проектом, змінювати параметри об'єктів, тестувати та переглядати зміни у реальному часі. Потужна система управління ресурсами полегшує організацію та використання ассетів, таких як моделі, текстури та звуки.

Як основний інструмент для написання коду обрано Microsoft Visual Studio 2022 завдяки його інтеграції з Unity. Він надає розробникам зручний інтерфейс для написання, налагодження та тестування коду мовою програмування C#, який використовується для створення ігрових скриптів. C# – об'єктно-орієнтована мова програмування, яка розроблена компанією

Microsoft.

Для створення графічного вмісту гри використано Adobe Photoshop, який є інструментом для роботи з растровою графікою. Він був вперше випущений у 1988 році і став стандартом в індустрії. Одним із ключових переваг Photoshop є його широкий набір інструментів для редагування та створення зображень, що робить його ідеальним для розробки ігрового контенту. Photoshop дозволяє малювати піксельну графіку з високою точністю, використовуючи такі інструменти, як пензель, заливка та шари. Це робить його зручним для створення графіки у стилі піксель-арт, дозволяючи художникам легко редагувати та анімувати зображення.

3.2 Розробка графіки гри

Візуальна складова є однією із ключових частин будь-якої комп'ютерної гри. Перш ніж перейти до розгляду розробки графіки гри, важливо зрозуміти основні терміни, які широко використовуються у геймдизайні.

Спрайт (від англ. Sprite) – «окреме двовимірне зображення, що застосовується в комп’ютерній графіці як частина більшого зображення. Найчастіше – растрове зображення, що переміщується по екрану, незалежно від фону. Спрайт може бути статичним зображенням або анімованим» [26].

Спрайт-лист (з англ. Sprite Sheet) – графічний файл, який містить кілька зображень (спрайтів) в одному, упорядкованих у вигляді таблиці або сітки. Основне призначення спрайт-листа – оптимізація та спрощення роботи з графікою гри. Використання одного файлу для зберігання безлічі спрайтів допомагає знизити кількість запитів до графічної пам'яті та зменшує розмір даних, що завантажуються.

Тайл або тайлова графіка (від англ. Tile) – у комп’ютерній графіці «метод створення великих зображень, що складаються з невеликих фрагментів однакових габаритів, розташованих у сітці» [27]. «Тайлова

графіка складається з набору тайлів (тайлсету) та матриці клітинок, що визначає який тайл яку клітину займатиме» [27].

Набір тайлів або тайлсет (від англ. Tileset) – «це набір, що включає усі окремі фрагменти, використовувані в грі. Кожен тайл в наборі представляє певний графічний елемент, наприклад, ділянку місцевості, об’єкт або персонажа» [27].

Для реалізації графічної складової гри розробляється набір спрайтів, частина з яких відповідає анімаціям всіх персонажів з урахуванням їх кінцевих автоматів. Для кожної анімації створюється відповідний документ в Adobe Photoshop 2020, де у вікні створення документа вказується розмір зображення. Вікно створення документа зображено на рис. 3.1.

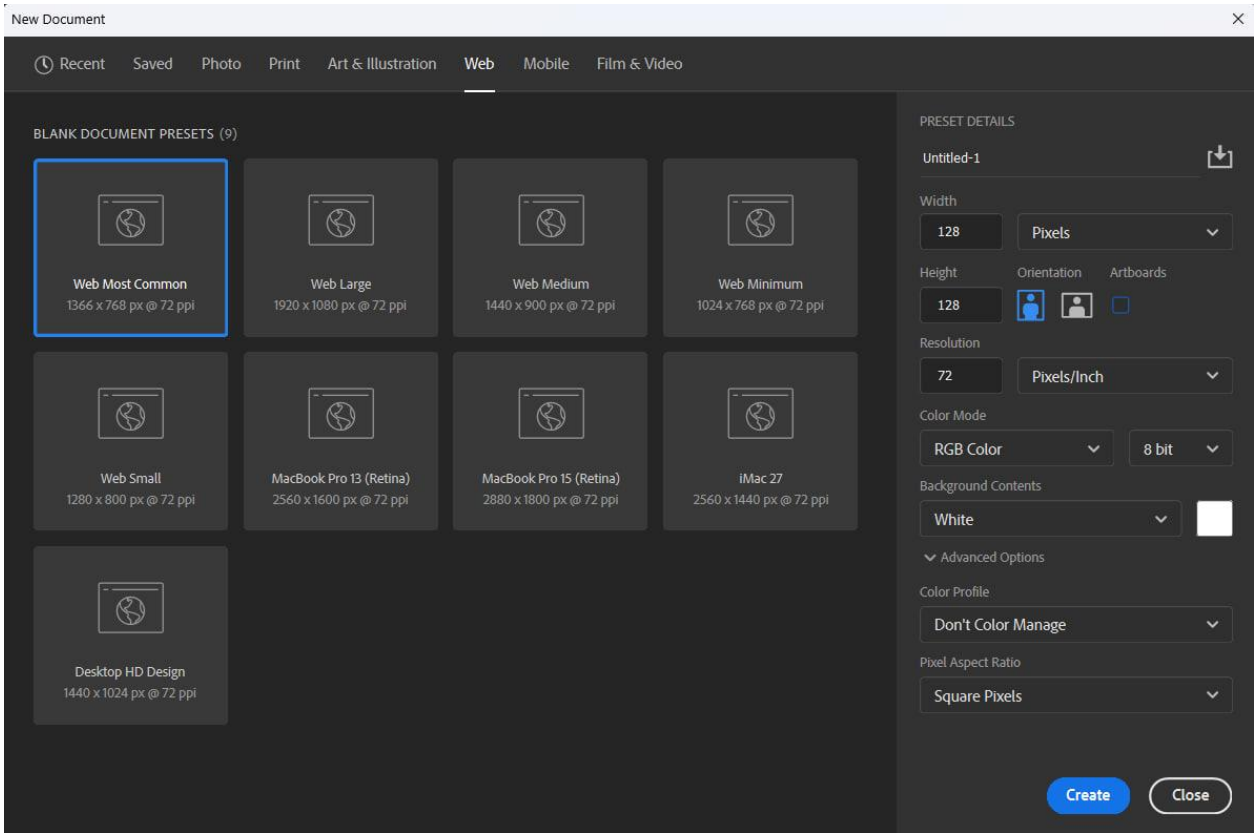


Рисунок 3.1 – Вікно створення документу

Наприклад, для анімації дальньої атаки ворога Вампір-лучник, як і наступних анімацій цього ворога, створюється документ розміру 128 пікселів завширшки та заввишки. Слід зазначити, що розміри спрайтів різних

персонажів можуть відрізнятися між собою. Наприклад, спрайти головного героя мають розмір 96 пікселів завширшки і заввишки, що менше розміру спрайтів Вампіра-лучника.

Після створення документа з'являється робоче місце, що складається з полотна, вікна шарів та інших інструментів. Вікно робочого простору зображено на рис. 3.2.

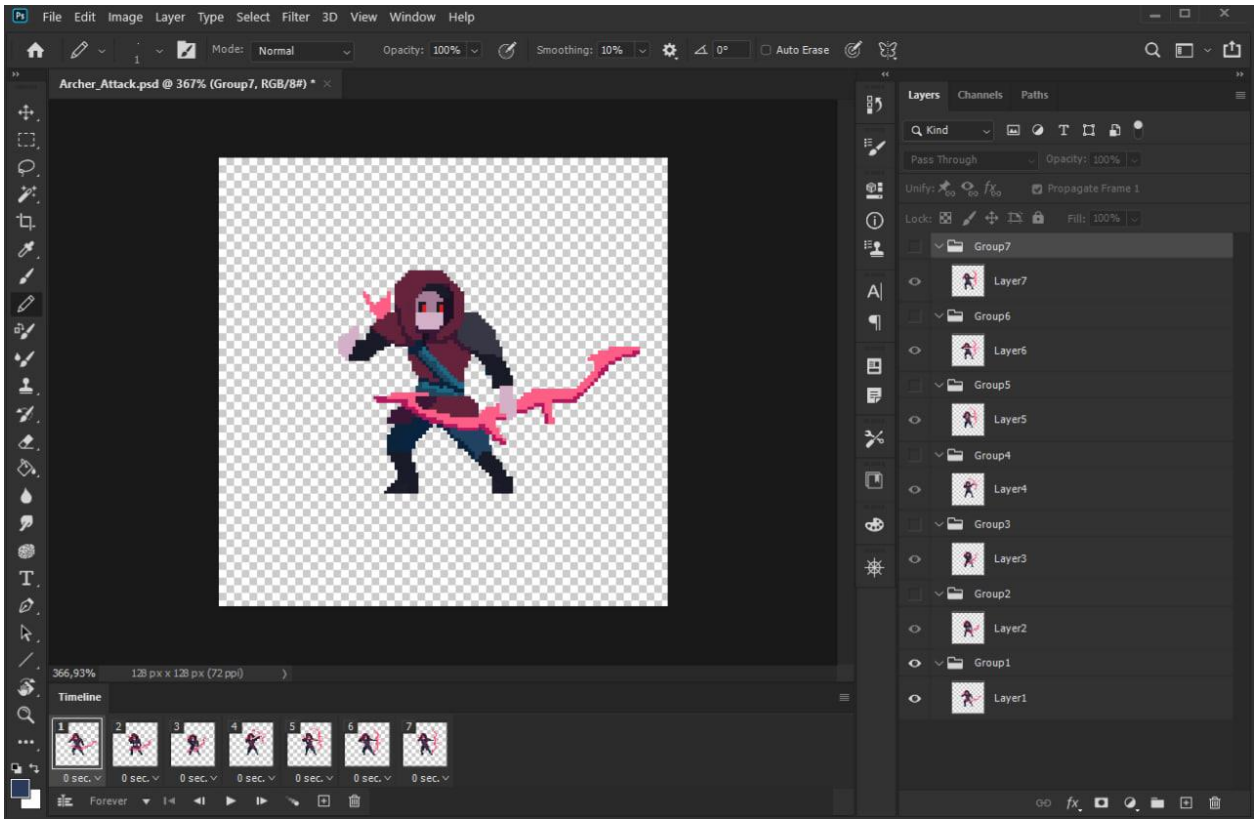


Рисунок 3.2 – Вікно робочого простору

Перед початком роботи слід налаштувати середовище. Для цього необхідно вибрати «Редагування» у верхній панелі, потім натиснути «Налаштування» і далі «Основні». У відкритому вікні вибирається «Інтерполяція зображення» як «По сусідніх пікселів (зберігає чіткі краї)». Вікно основних налаштувань зображено на рис. 3.3.

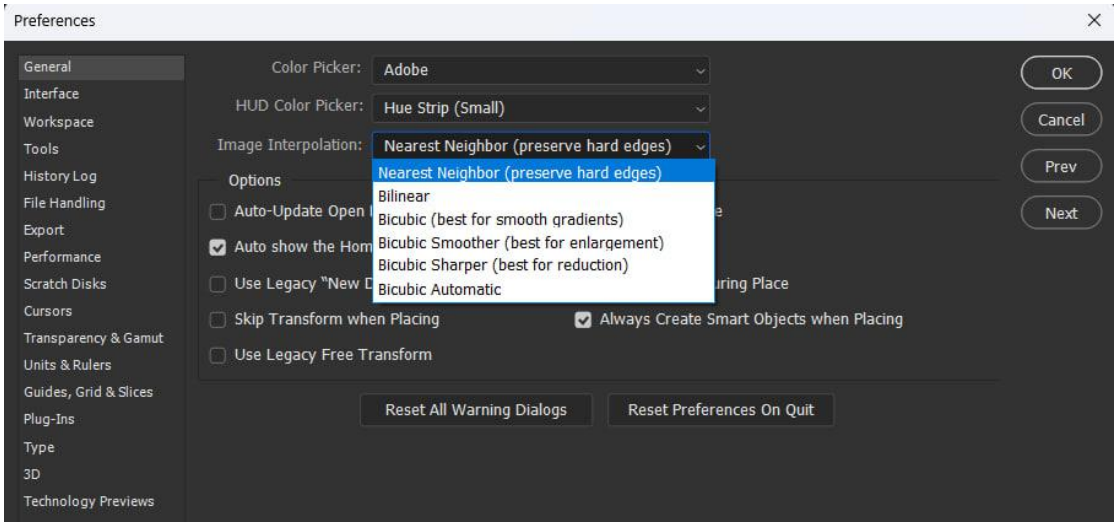


Рисунок 3.3 – Вікно основних налаштувань

У процесі роботи над спрайтами використовується вікно «Шкала часу», де створюється покадрова анімація. Вікно «Шкала часу» зображено на рис. 3.4.

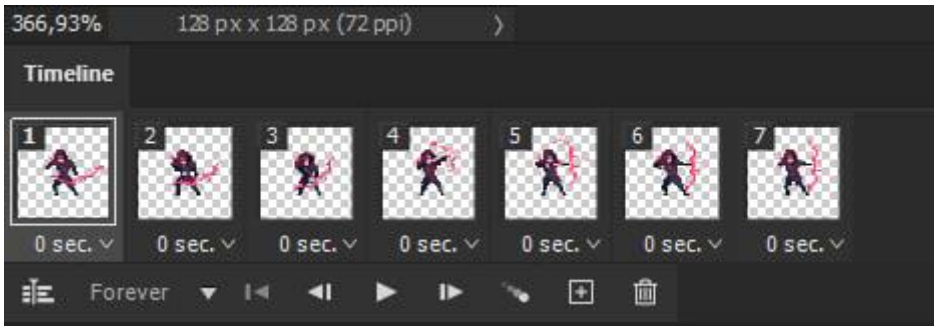


Рисунок 3.4 – Вікно «Шкала часу»

Для кожного кадру створюється окрема папка у вікні шарів, що дозволяє легко вносити необхідні зміни до спрайтів. Наприклад, для анімації дальньої атаки Вампіра-лучника створено 7 папок, кожна з яких відповідає окремому кадру анімації. Для налаштування анімації потрібно обрати один відповідний кадр у вікні «Шкала часу» та натиснути на іконку ока поруч тієї папки, вміст якої має відображатися на кадрі. Усередині кожної папки створюються шари, які відповідають за спрайт анімації персонажа. Вікно «Шари» зображено на рис. 3.5.

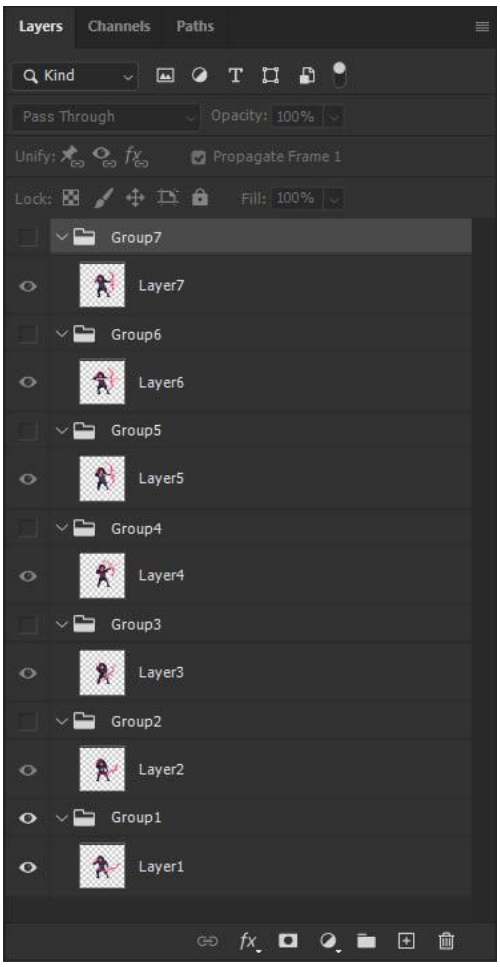


Рисунок 3.5 – Вікно «Шари»

У процесі створення спрайтів у стилі піксель-арту важливо використовувати інструмент «Олівець», оскільки інструмент «Пензель» залишає за собою на полотні розмитий слід. Розмір інструменту треба налаштувати в один піксель. Також потрібно налаштувати інструмент «Гумка», вибравши режим «Олівець» у верхній панелі інструментів. Налаштування інструмента «Гумка» зображено на рис. 3.6.

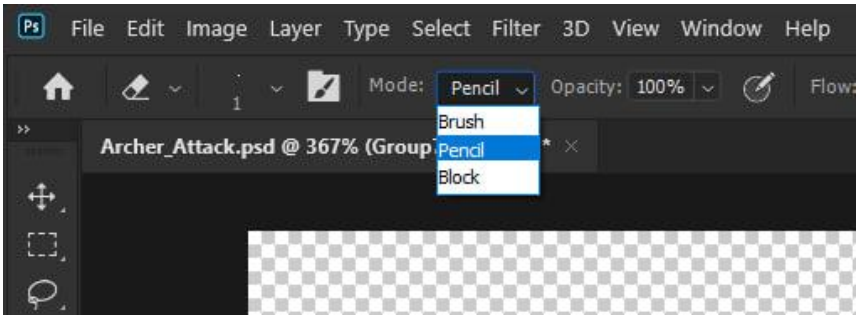


Рисунок 3.6 – Налаштування інструмента «Гумка»

Головний герой має анімації покою, бігу, стрибка, використання драбини, базової атаки, отримання шкоди та смерті, спрайт-листи яких зображено на рис. Б.1, Б.2, Б.3, Б.4, Б.5, Б.6 та Б.7 в додатку Б відповідно. Додатково розроблено анімації використання вампірських здібностей, таких як «Голодний укус» (додаток Б, рис. Б.8), «Гострі голки» (додаток Б, рис. Б.9) та «Тіньовий крок» (додаток Б, рис. Б.10). Оскільки здібність «Гострі голки» є далекою атакою, то для неї необхідно розробити спрайт снаряда – кровавої голки (додаток В, рис. В.11). Також розробляється анімація відновлення здоров'я, де головний герой п'є пляшку крові (додаток В, рис. В.12).

Створення спрайтів для ворогів також включає розробку різноманітних анімацій. Наприклад, Кістлявий вовк має анімації спокою, бігу, отримання шкоди, смерті та ближньої атаки (додаток В, рис. В.1). Вампір-лучник має анімації спокою, бігу, стрибка, отримання шкоди, смерті та стрілянини із зачарованого рожевого лука (додаток В, рис. В.2). Спрайтом снаряда дальньої атаки ворога є стріла таке самого кольору, як зброя (додаток В, рис. В.3). Вампір-стражник має анімації спокою, бігу, отримання шкоди, смерті, блокування щитом та ближньої атаки одноручним мечем (додаток В, рис. В.4). Відмінною рисою Вампіра-стражника є розмір спрайтів, що становить 160 пікселів завширшки та 120 пікселів заввишки для анімації розмаху мечем та зображення герба вампірського клану на щиті. Палаючий дух має анімації польоту, спокою, вибуху та смерті (додаток В, рис. В.5). Кажан володіє анімаціями польоту, очікування, отримання шкоди, смерті та ближньої атаки (додаток В, рис. В.6).

Інтерфейс гравця (UI) оформлений у середньовічному вампірському стилі, всі елементи зображені як старі пожовклі аркуші паперу. Спрайти умінь головного героя, таких як відновлення здоров'я, «Голодний укус» та «Тіньовий крок», мають унікальні іконки: пляшка крові, посмішка з іклами та рухома тінь. Розроблено спрайт квадратного аркуша паперу меншого розміру, який призначений для надписів, таких як поточний рівень головного героя та кількість пляшок крові. На спрайті шкали здоров’я відображаються

лише межі паперу, а на спрайті шкали зарядів крові відображаються спрайти краплі крові, кожен з яких символізує один заряд. Спрайти основних елементів UI зображено на рис. Г.1 в додатку Г. Також використовується спрайт кнопки синього кольору з червоним обведенням (додаток Г, рис. Г.2). Головне меню містить зображення нічного неба з місяцем (додаток Г, рис. Г.3) та логотипом гри (додаток Г, рис. Г.4).

Для створення оточення використовуються тайли розміром 32 пікселі. Створюються тайлсети локацій «Тюремні камери» (додаток Д, рис. Д.1), «Прогулянковий хол» (додаток Д, рис. Д.2) та «Бібліотека» (додаток Д, рис. Д.3).

3.3 Діаграма класів

Важливим елементом при проектуванні системи, заснованої на об'єктно-орієнтованому підході, є діаграма класів. Діаграма класів (англ. class diagram) – структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів (полів), методів, інтерфейсів та взаємозв'язків між ними [28]. У додатку Е наведено розширену діаграму класів, пов'язану з класами кінцевих автоматів.

Основні класи, задіяні в системі, включають State, StateMachine і StateMachineCore. Клас State є абстрактним класом, що представляє стан у кінцевому автоматі. Усі конкретні стани успадковуються від нього і реалізують специфічну для кожного стану логіку. Клас StateMachine управляє перемиканням станів з урахуванням певної логіки. Клас StateMachineCore є абстрактним базовим класом, від якого успадковуються всі конкретні кінцеві автомати, такі як кінцевий автомат головного героя (PlayerStateMachine), і кінцеві автомати ворогів (DogNPCStateMachine, ArcherNPCStateMachine, DefenderNPCStateMachine, GhostNPCStateMachine. Спрощену діаграму класів зображено на рис. 3.7.

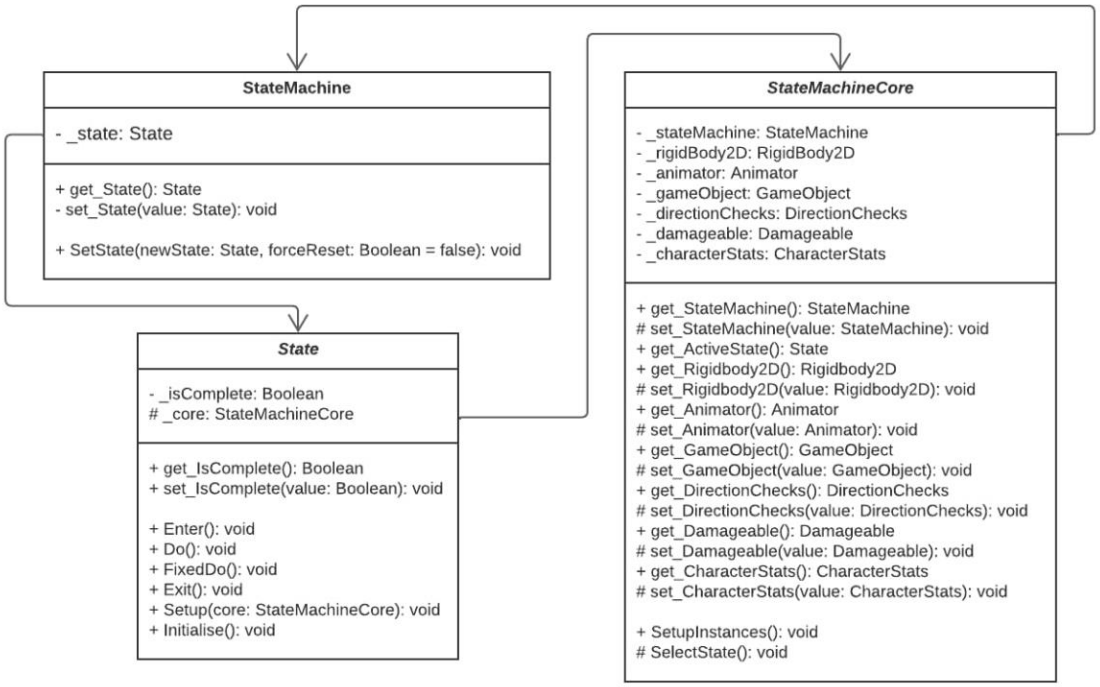


Рисунок 3.7 – Спрощена діаграма класів

У межах даної архітектури додатково присутні класи, які реалізують різні вхідні події, необхідні для функціонування кінцевих автоматів. Наприклад, клас StateMachineCore має властивість класу DirectionChecks, який відповідає за визначення напрямку руху та орієнтації об'єкта у грі. За допомогою цього класу можна перевірити, чи є перешкоди чи межі платформи з різних боків об'єкта. Також можна побачити, що кінцеві автомати персонажів мають поля класу CooldownChecks, що дозволяє керувати перезарядженням умінь персонажів за допомогою відліку часу. Найчастіше в кінцевих автоматах персонажів зустрічаються поля класу PlayerDetectionIncludingGroundChecks, який відповідає за виявлення головного героя у вибраній області ворога з урахуванням перешкод, таких як платформи. Цей клас дозволяє ворогам бачити головного героя лише тоді, коли між ними немає перешкод, що робить гру більш реалістичною. Клас PlayerInputChecks відповідає за обробку введення від гравця.

Дана архітектура дозволяє розширювати функціональність кінцевих автоматів, додаючи нові стани та вхідні події, а також легко керувати поведінкою персонажів у грі.

3.4 Програмна реалізація станів та кінцевих автоматів

Програмна реалізація станів та кінцевих автоматів у системі заснована на трьох ключових класах: State, StateMachine та StateMachineCore.

Клас State є абстрактним класом, який моделює стан в кінцевому автоматі. Код класу State наведено у лістингу 3.1.

```
using UnityEngine;

public abstract class State : MonoBehaviour
{
    public bool IsComplete {  get; set; }

    protected StateMachineCore _core;

    public virtual void Enter() { }
    public virtual void Do() { }
    public virtual void FixedDo() { }
    public virtual void Exit() { }

    public void Setup(StateMachineCore core)
    {
        _core = core;
    }

    public void Initialise()
    {
        IsComplete = false;
    }
}
```

Лістинг 3.1 – Код класу State

Цей клас містить методи життєвого циклу стану, такі як Enter, Do, FixedDo та Exit. Ці методи дозволяють визначити дії, що виконуються при вході в стан, під час його активності та при виході з нього.

Відмінність між методами Do і FixedDo полягає в тому, що перший викликається кожен кадр і використовується для виконання операцій, що залежать від часу, таких як перевірка введення команд із клавіатури гравця або оновлення анімацій. Метод FixedDo викликається кожен фізичний кадр, що робить його придатним для виконання операцій, пов'язаних із фізикою,

таких як оновлення положення об'єкта або обробка зіткнень. Поділ цих методів дозволяє більш точно контролювати логіку гри та забезпечує стабільне виконання як візуальних, так і фізичних аспектів стану.

Метод Setup пов'язує стан з основним компонентом StateMachineCore для забезпечення доступу до загальних параметрів та компонентів кінцевого автомата із усіх його станів. Оскільки в методі Exit стан позначається як завершений, існує метод Initialise, який скидає стан, встановлюючи властивість IsComplete false.

Клас StateMachine управляє поточним станом об'єкта та перемиканням між станами. Код класу StateMachine наведено у лістингу 3.2.

```
using UnityEngine;

public class StateMachine
{
    public State State { get; private set; }

    public void SetState(State newState, bool forceReset =
false)
    {
        if (State != newState || forceReset)
        {
            State?.Exit();
            State = newState;
            State.Initialise();
            State.Enter();
        }
    }
}
```

Лістинг 3.2 – Код класу StateMachine

Цей клас містить метод SetState, який відповідає за зміну стану. Якщо поточний стан відрізняється від нового або встановлений прапор forceReset, поточний стан завершується методом Exit, новий стан ініціалізується методом Initialize та активується методом Enter. Це забезпечує плавне та коректне перемикання між різними станами об'єкта.

Властивість State в класі StateMachine є поточним станом, яким управляє кінцевий автомат. Ця властивість оголошена з модифікатором

доступу public, тому її можна отримати ззовні класу, але встановлювати її можна лише всередині класу завдяки приватному сеттеру (private set). Це дозволяє контролювати процес зміни станів через метод SetState, що гарантує правильне виконання методів Exit, Initialize та Enter під час перемикання між станами.

Клас StateMachineCore служить абстрактним базовим класом для всіх конкретних кінцевих автоматів у системі. Код класу StateMachineCore наведено у лістингу 3.3.

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D), typeof(Animator))]
[RequireComponent (typeof(DirectionChecks), typeof(Damageable))]
[RequireComponent(typeof(CharacterStats))]
public abstract class StateMachineCore : MonoBehaviour
{
    public StateMachine StateMachine { get; protected set; }
    public State ActiveState => StateMachine.State;

    public Rigidbody2D Rigidbody2D {  get; protected set; }
    public Animator Animator { get; protected set; }
    public GameObject GameObject { get; protected set; }
    public DirectionChecks DirectionChecks { get; protected set;
    public Damageable Damageable { get; protected set; }
    public CharacterStats CharacterStats { get; protected set; }

    public void SetupInstances()
    {
        Rigidbody2D = GetComponent<Rigidbody2D>();
        DirectionChecks = GetComponent<DirectionChecks>();
        Animator = GetComponent<Animator>();
        CharacterStats = GetComponent<CharacterStats>();
        Damageable = GetComponent<Damageable>();
        Damageable.Core = this;
        GameObject = gameObject;

        StateMachine = new StateMachine();
        State[] allStates = GetComponentsInChildren<State>();
        foreach (var state in allStates)
        {
            state.Setup(this);
        }
    }
    protected abstract void SelectState();
}
```

Лістинг 3.3 – Код класу StateMachineCore

Цей клас включає різні компоненти, такі як Rigidbody2D, Animator, DirectionChecks, Damageable, CharacterStats і GameObject. Компонент Rigidbody2D відповідає за фізичну поведінку об'єкта, включаючи рух та зіткнення. Animator управляє анімаціями об'єкта, забезпечуючи плавні переходи між різними анімаційними станами. DirectionChecks використовується для визначення напрямку руху та перевірки перешкод навколо об'єкта. Damageable відповідає за отримання та обробку шкоди об'єктом, включаючи зменшення здоров'я. CharacterStats містить загальні характеристики персонажа, такі як поточна та максимальна кількість здоров'я та рівень. GameObject представляє сам об'єкт в ігровому світі, забезпечуючи доступ до всіх його компонентів та властивостей.

Ці властивості визначаються в класі StateMachineCore, щоб забезпечити доступ до всіх необхідних компонентів, які можуть знадобитися різним станам для виконання своєї логіки. Це сприяє централізованому управлінню та покращує модульність системи.

Метод SetupInstances ініціалізує ці компоненти та налаштовує стани, знаходячи всі екземпляри класу State в дочірніх об'єктах і пов'язуючи їх з даними StateMachineCore. Абстрактний метод SelectState використовується для опису механізму роботи кінцевого автомата, визначаючи переходи між його станами.

Робота будь-якого кінцевого автомата, що успадковується від класу StateMachineCore, починається з ініціалізації, при якій викликається метод SetupInstances, який пов'язує всі необхідні компоненти та стани з об'єктом. Далі визначається початковий стан, який стає активним за допомогою методу SetState властивості StateMachine. Коли необхідно змінити стан у методі SelectState, викликається метод SetState, який завершує поточний стан та активує новий. У ході виконання програми методи Do та FixedDo поточного стану викликаються у методах Update та FixedUpdate відповідно, забезпечуючи виконання відповідної логіки стану.

3.5 Реалізація віртуальних персонажів

В основі роботи всіх кінцевих автоматів персонажів лежить структура, що забезпечує перемикання між різними станами в залежності від певних умов. Оголошені стани представлені у вигляді приватних полів класу State.

Для керування переходами між станами використовуються різні перевірки та умови, такі як виявлення головного героя, доступність атаки, стан здоров'я та інші фактори, характерні для кожного персонажа. Для обробки вхідних подій у кожному кінцевому автоматі також оголошуються приватні поля, які відповідають за ці події.

При використанні атрибуту [SerializeField] приватні поля можуть бути призначені в інспекторі Unity, що дозволяє покращити читання коду та робить його більш модульним, оскільки значення цих полів можна легко змінити та перепризначити без зміни інших частин коду.

У методі Awake кожного кінцевого автомата викликається метод SetupInstances, який встановлює необхідні екземпляри та початкові параметри. Потім вибирається початковий стан кінцевого автомата за допомогою методу SetState властивості StateMachine.

Методи Update та FixedUpdate служать для перевірки поточного стану персонажа та виконання відповідної логіки роботи стану.

Для коректного перемикання станів після отримання шкоди персонажем, використовується метод StopHurt, який викликається для припинення отримання шкоди на останньому кадрі анімації та перемикання на необхідний стан.

Головний герой управляється кінцевим автоматом PlayerStateMachine (додаток Ж, лістинг Ж.1), який складається з наступних станів: спокою _idleState, бігу _runState, стрибка _jumpState, базової атаки _basicAttackState, смерті _dieState, використання драбини _ladderClimbState, відновлення здоров'я _healState, вампірських здібностей «Голодний укус» _hungryBiteState, «Гострі голки» _sharpNeedlesState і «Тіньовий крок»

shadowStepState.

Вхідні події у цьому класі контролюють різні аспекти поведінки персонажа. Наприклад, для здібностей «Голодний укус» та «Тіньовий крок» перевіряються перезарядки за допомогою полів _hungryBiteCooldownChecks та _shadowStepCooldownChecks відповідно. Перевірка доступності зарядів крові для здатності «Гострі голки» здійснюється через поле _sharpNeedlesBloodMagicAbility класу BloodMagicAbility. Контроль введення від гравця здійснюється за допомогою об'єкта _playerInputChecks, виявлення знаходження персонажа біля драбини через _playerDetectionInLadderChecks, а отримання характеристик головного героя через об'єкт _playerStats.

Далі розглядаються кінцеві автомати інших персонажів – ворогів. Кінцевий автомат DogNPCStateMachine (додаток Ж, лістинг Ж.2) описує поведінку ворога Кістлявий вовк. Цей автомат складається з наступних станів: патрулювання _patrollingState, переслідування _chasingState, спокою _idleState, атаки в ближньому бою _meleeAttackState і смерті _dieState. Вхідні події _playerInSightChecks та _playerInAttackActivationRangeChecks контролюють виявлення головного героя в області видимості та атаки, і _attackCooldownChecks перевіряє перезарядку атаки. У методі SelectState перевіряються умови, за яких ворог повинен переключитися між станами, наприклад, коли головного героя виявлено в одній із областей або коли ворог гине.

Другий ворожий персонаж – ArcherNPCStateMachine (додаток Ж, лістинг Ж.3), моделює поведінку Вампір-лучника. Він має стани патрулювання _patrollingState, спокою _idleState, атаки в дальньому бою _rangedAttackState, відскоку _bounceState та смерті _dieState. За допомогою об'єктів _playerInSightChecks та _playerInBounceChecks перевіряється виявлення головного героя в областях видимості та відскоку відповідно. Об'єкти _attackCooldownChecks та _bounceCooldownChecks дозволяють перевіряти перезарядки атаки та відскоку ворога.

Клас DefenderNPCStateMachine (додаток Ж, лістинг Ж.4) описує

поведінку Вампір-стражника. Цей автомат містить стани: патрулювання _patrollingState, спокою _idleState, переслідування _chasingState, атаки в ближньому бою _meleeAttackState, блокування щитом _shieldBlockState і смерті _dieState. Об'єкти _playerInSightChecks та _playerInAttackActivationRangeChecks дозволяють виявити головного героя в областях видимості та атаки відповідно. Також контролюються перезарядки атаки та блокування щитом за допомогою об'єктів _attackCooldownChecks та _shieldBlockCooldownChecks.

Палаючий дух управляється кінцевим автоматом GhostNPCStateMachine (додаток Ж, лістинг Ж.5). Цей автомат має стан: спокій _idleState, переслідування _chasingState, вибух _explosionState і смерть _dieState. Аналогічно ворог реагує на виявлення головного героя в областях видимості та атаки за допомогою об'єктів _playerInSightChecks та _playerInAttackActivationRangeChecks і вирішує, коли використовувати свою атакуючу здібність.

Кінцевий автомат BatNPCStateMachine (додаток Ж, лістинг Ж.6) моделює поведінку Кажана. Він складається зі станів сну _sleepingState, спокою _idleState, переслідування _chasingState, атаки в ближньому бою _meleeAttackState і смерті _dieState. Цей автомат має аналогічні вхідні події, як і в кінцевого автомата Кістлявого вовка. Кажан вирішує, коли прокидатися і атакувати залежно від виявлення головного героя і перезарядки атакуючого вміння.

Для кінцевих автоматів персонажів реалізовано велика кількість різних станів. Тому для прикладу розглядається найпоширеніший стан у ворогів – стан атаки ближнього бою MeleeAttackState (додаток К, лістинг К.1). Для порівняння взято стан базової атаки головного героя PlayerBasicAttackState (додаток К, лістинг К.2). Також розглядається простий за структурою клас IdleState (додаток К, лістинг К.3), який описує стан спокою у всіх персонажів.

Обидва стани MeleeAttackState і PlayerBasicAttackState контролюють анімації атаки, перевірку влучення по цілі та застосування шкоди. Однак у

класі PlayerBasicAttackState здійснюється ще переміщення головного героя у напрямку атаки. На відміну від ворогів, головний герой має ще можливість завдавати шкоди декілька персонажів через цикл foreach у методі FixedDo.

Стан IdleState контролює, щоб рухів об'єкта не виникало, і змінює параметри, такі як гравітація, якщо персонаж літає.

3.6 Налаштування демонстраційної версії гри

Розробка та налаштування демонстраційної версії гри виконується в середовищі Unity, яке складається з кількох основних вікон, що забезпечують зручний інтерфейс для розміщення та налаштування ігрових об'єктів. Основні вікна Unity включають Scene, Hierarchy, Inspector та Game. Вікно Scene дозволяє бачити та редагувати розташування об'єктів на рівні. Вікно Hierarchy показує всі об'єкти, що знаходяться в поточній сцені. Вікно Inspector дозволяє змінювати властивості вибраного об'єкта, а вікно Game показує, як виглядає гра.

У вікні створення сцени кожного рівня серед ігрових об'єктів у вкладці Hierarchy можна побачити кілька ключових елементів: Grid, PlayerCharacter, Enemies, PlayerUI та NextLevelPoint.

Об'єкт Grid містить різні об'єкти з компонентом Tilemap, які є основою ігрового світу. Об'єкт PlayerCharacter містить префаб головного героя. Об'єкт Enemies містить префаби ворогів, такі як DogNPC, ArcherNPC, DefenderNPC, GhostNPC та BatNPC. Об'єкт PlayerUI відповідає за відображення елементів інтерфейсу гравця. Об'єкт NextLevelPoint при зіткненні з головним героєм дозволяє гравцеві перейти на наступний рівень. Вікно створення сцени зображено на рис. 3.8.

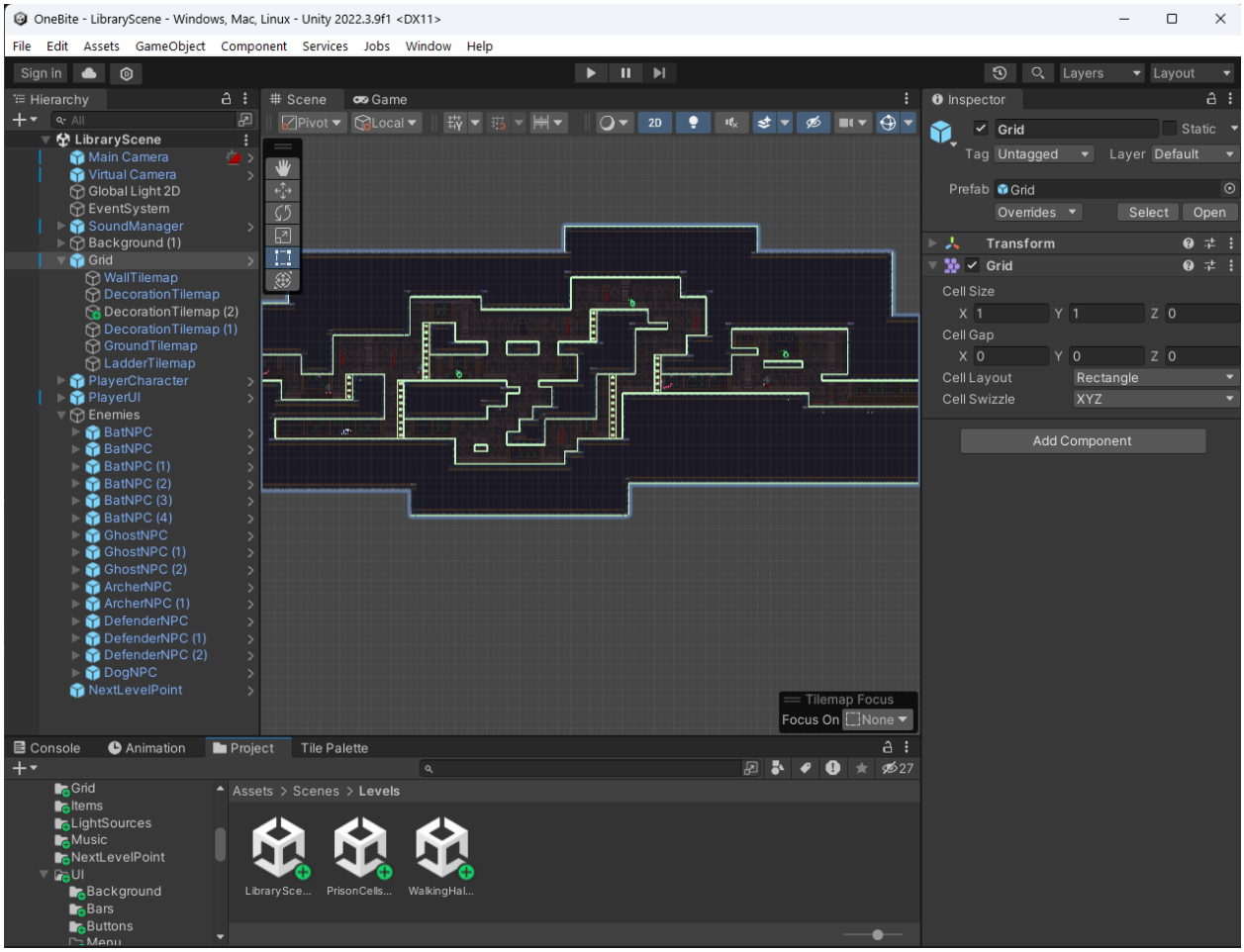


Рисунок 3.8 – Вікно створення сцени

Для формування префабу персонажа до об'єкта додаються відповідний скрипт кінцевого автомата та необхідні для його роботи компоненти, зокрема: Rigidbody2D, Collider2D, Animator, AnimationEvent, DirectionChecks та Damageable. Додатково додається компонент із характеристиками персонажа: для головного героя – PlayerStats та для ворога – EnemyStats. Також для головного героя додається компонент PlayerInputChecks, який дозволяє налаштувати керування гравця. Наприклад, кнопкою відновлення здоров'я головного героя встановлюється Q, кнопкою використання здібності «Голодний укус» – ПКМ, «Гострі голки» – F, «Тіньовий крок» – Shift. Налаштування префабу головного героя зображено на рис. 3.9.

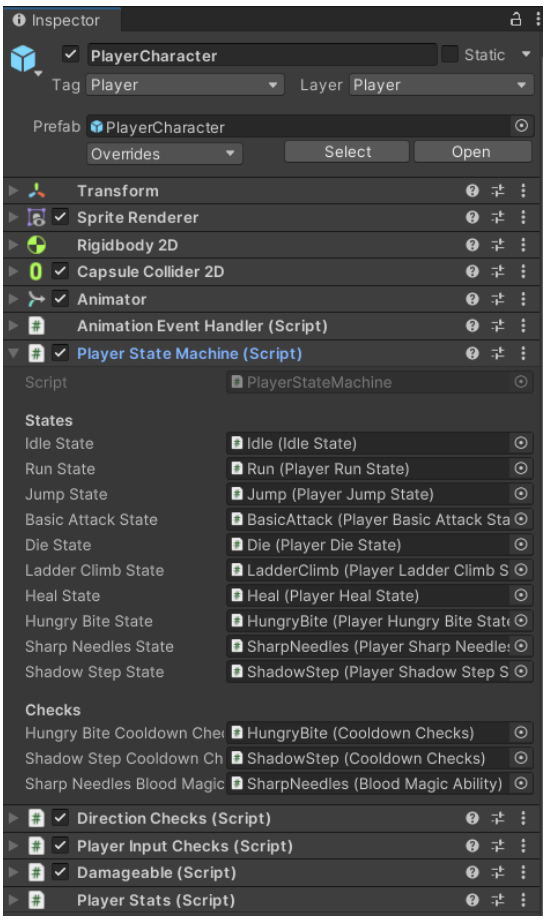


Рисунок 3.9 – Налаштування префабу головного героя

В об'єкті персонажа створюється дочірній об'єкт Behaviours, де розташовуються об'єкти, які відповідають певним станам персонажа. Наприклад, для головного героя PlayerCharacter дочірніми об'єктами Behaviours є Idle, Run, Jump, BasicAttack, Die, LadderClimb, Heal, HungryBite, SharpNeedles і ShadowStep. Кожен із цих об'єктів має прикріплений скрипт стану та необхідні йому компоненти. Наприклад, до об'єкта BasicAttack прикріплюється скрипт PlayerBasicAttackState і AttackAbility, який дозволяє вказати кількість шкоди, що наноситься. Потім налаштовуються всі стани персонажа, клацаючи по дочірнім об'єктам об'єкта Behaviours і заповнюючи значення поля. Налаштування стану базової атаки головного героя зображено на рис. 3.10.

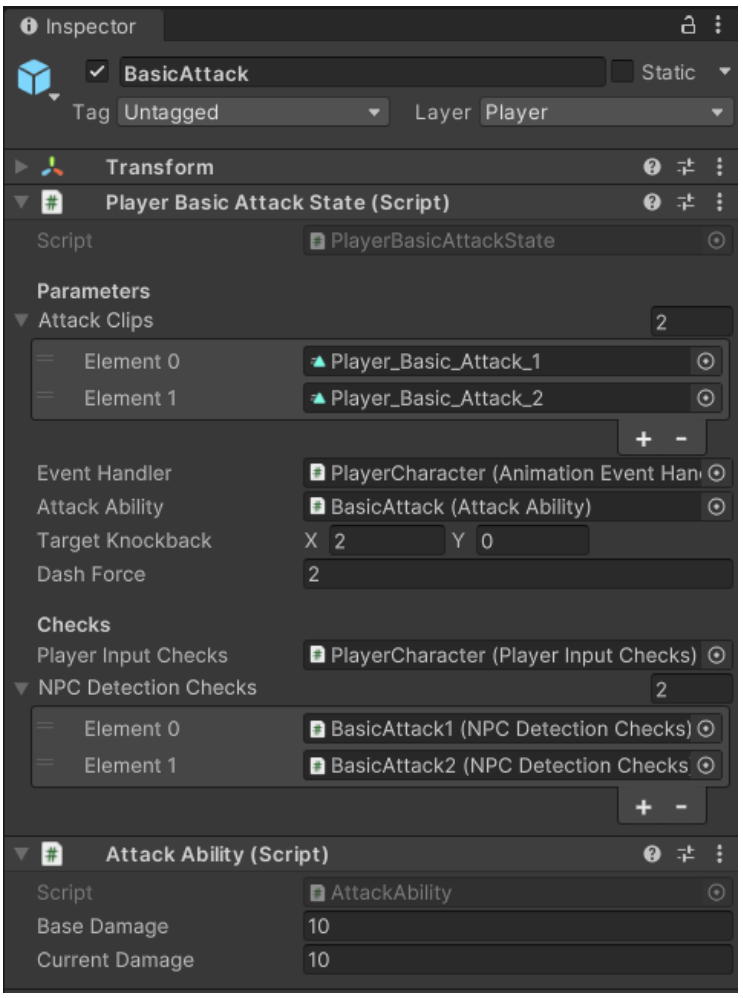


Рисунок 3.10 – Налаштування стану базової атаки головного героя

Для станів та кінцевого автомата персонажа також потрібні області виявлення. У всіх ворогів є область видимості, і у деяких ще є область радіусу атаки ближнього бою. Наприклад, у префабі DogNPC ворога Кістлявий вовк створюються дочірні об'єкти DetectionZone, MeleeAttackZone та MeleeAttackActivationZone, які містять колайдер з позначкою IsTrigger та прикріпленим скриптом для виявлення головного героя в області PlayerDetectionIncludingGroundChecks. Об'єкт DetectionZone відповідає за область видимості, MeleeAttackZone – область атаки, в якій завдається шкода головному герою, та MeleeAttackActivationZone – область активації атаки. Налаштування області виявлення персонажа зображено на рис. 3.11.

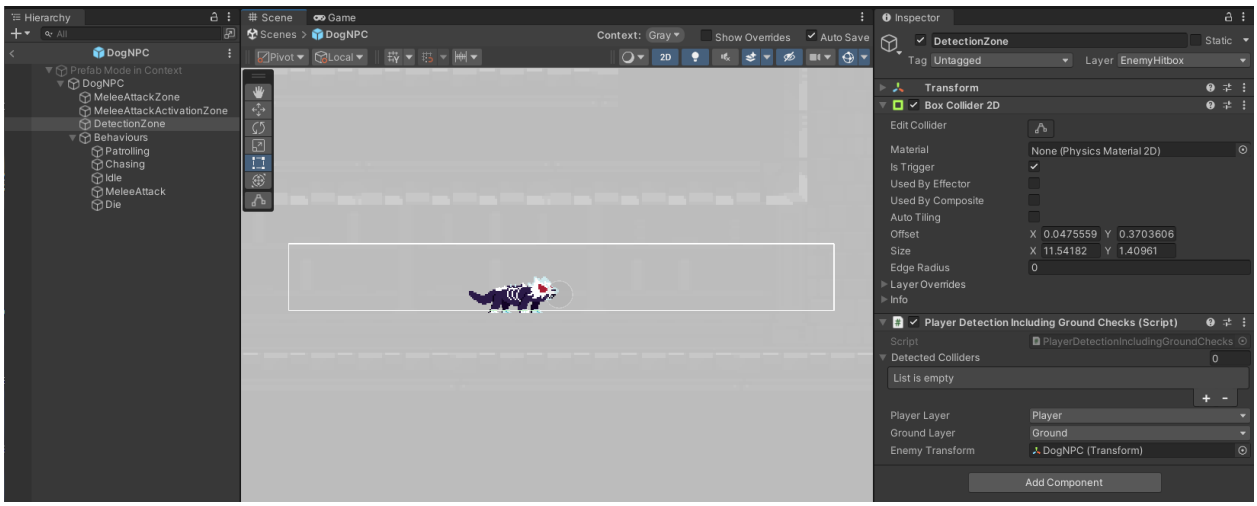


Рисунок 3.11 – Налаштування області виявлення персонажа

У вікні Tile Palette для кожної ігрової локації створюється палітра тайлів, за допомогою якої створюється ігровий світ кожного рівня. У цьому вікні вибирається інструмент Пензель, тайл та об'єкт із компонентом Tilemap, на якому необхідно розмістити тайли. Наприклад, палітра тайлів з назвою «LibraryPalette» для локації «Бібліотека» зображено на рис. 3.12.

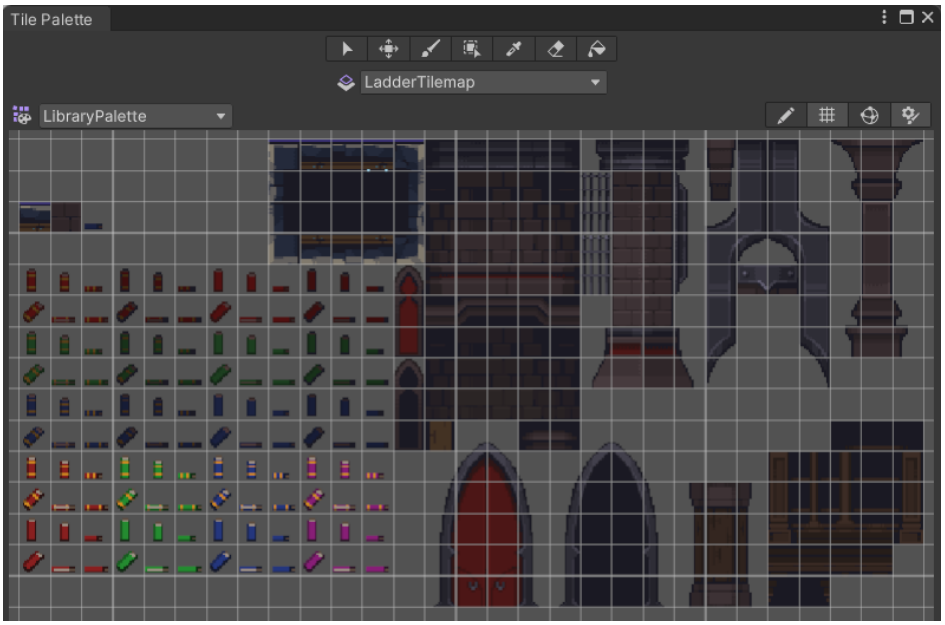


Рисунок 3.12 – Палітра тайлів для локації «Бібліотека»

Для налаштування префабу PlayerUI додаються до нього елементи інтерфейсу гравця, такі як іконки вмінь головного героя в об'єкті Abilities,

його шкали здоров'я та зарядів крові в об'єкті Bars, кількість очок досвіду та рівень персонажа в об'єкті TextWithIcons. Також у цьому префабі створюються меню паузи та меню після смерті головного героя. У компоненті Rect Transform дочірнього об'єкта префабу Group вибирається розміщення по нижньому лівому краю екрану. Налаштування префабу PlayerUI зображено на рис. 3.13.

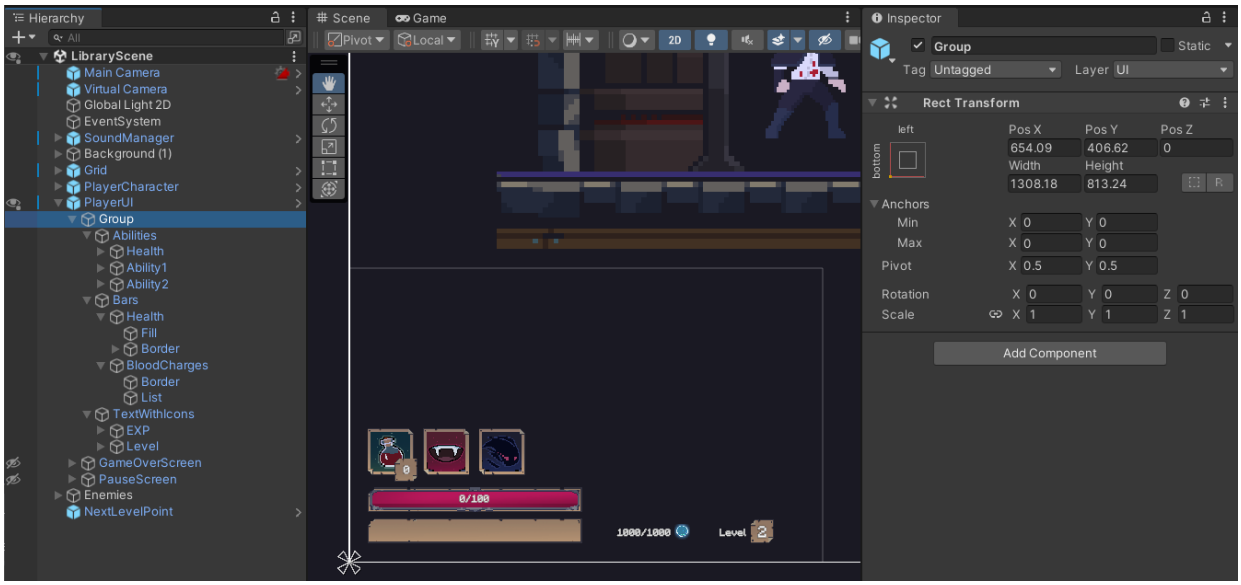


Рисунок 3.13 – Налаштування префабу PlayerUI

3.7 Перевірка працездатності системи

У процесі перевірки працездатності демонстраційної версії гри проведено серію тестів на машинах з різними конфігураціями для переконання у стабільності та продуктивності системи. Тестування проведено на двох типах машин: високопродуктивному ігровому комп'ютері та бюджетному ноутбуці. Характеристики машин наведено у табл. 3.1.

Таблиця 3.1 – Характеристики машин

Параметр	Високопродуктивний ігровий комп’ютер	Бюджетний ноутбук
Процесор	AMD Ryzen 5 7500F	Intel Core i3-7020U

Продовження таблиці 3.1

Параметр	Високопродуктивний ігровий комп’ютер	Бюджетний ноутбук
Оперативна пам’ять	32 ГБ	8 ГБ
Відеокарта	AMD Radeon RX 6800	Intel HD Graphics 620
Накопичувач	SSD 1 ТБ	SSD 512 ГБ
Операційна система	Windows 11 Pro	Windows 10 Pro

На кожній з цих машин протестована гра протягом не менше півгодини. Перевірка FPS проведено за допомогою програми Fraps. «Fraps – це програмне забезпечення, яке дозволяє робити скріншоти та відео екрану, а також підраховувати кількість FPS під час тестування ігор» [29]. Для початку проведення тестів необхідно запустити дану програму. Налаштування програми Fraps відображено на рис. 3.14.

Рисунок 3.14 – Налаштування програми Fraps

Далі запускається гра та у верхньому лівому куті екрану відображається показник FPS в реальному часі. Перевірка показника FPS відображена на рис. 3.15.

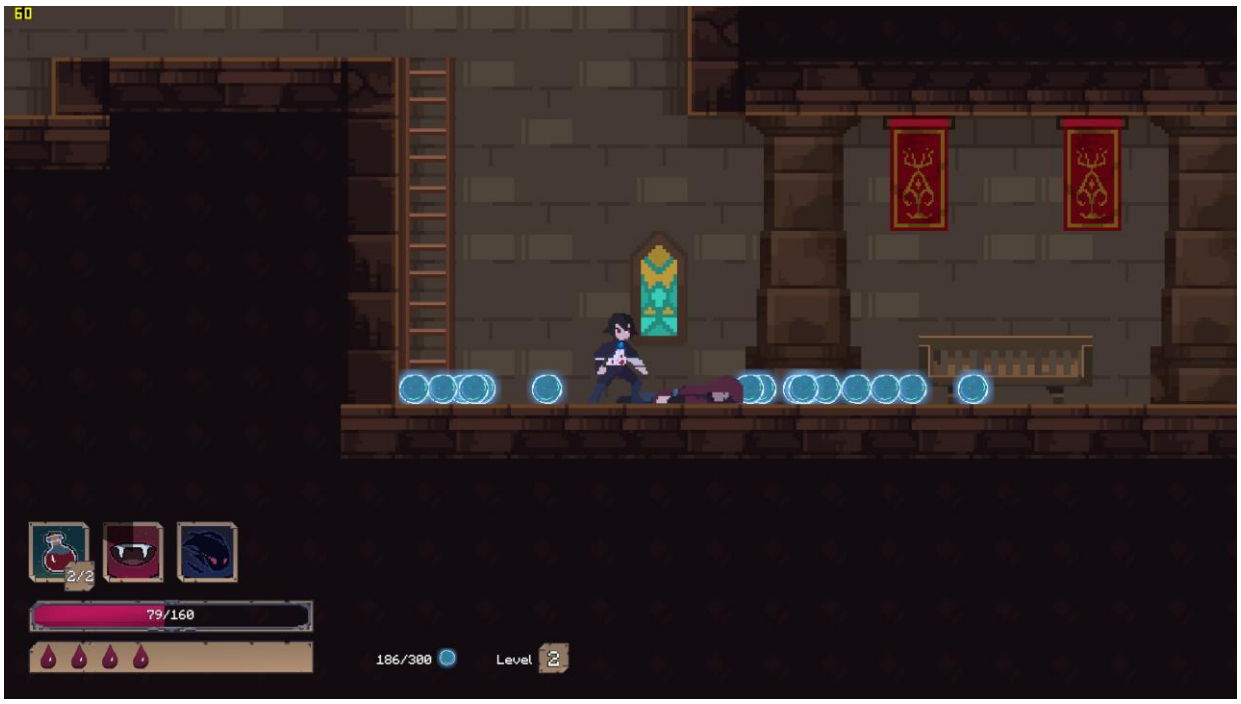


Рисунок 3.15 – Перевірка показника FPS

Під час тестів на всіх типах машин зафіксовано, що частота кадрів не падає нижче 60 кадрів за секунду. Такі показники підтверджують, що гра може комфортно працювати на широкому спектрі обладнання.

Крім того, в ході тестування особливу увагу приділено виявленню можливих багів. Баг – «помилка, вада або дефект в комп'ютерній програмі або системі, що викликає в ній неправильний або неочікуваний результат чи неочікувану поведінку» [30]. Проведено ретельне дослідження всіх ігрових механік, інтерфейсів та взаємодій для упевненості у коректній роботі всіх елементів гри. За час тестування не виявлено серйозних багів, що свідчить про високий рівень завершеності поточної версії продукту.

Таким чином, результати тестування демонструють, що демонстраційна версія гри є стабільною та продуктивною на різних апаратних конфігураціях.

3.8 Висновки до третього розділу

а) Вибрано засоби розробки демонстраційної версії гри, такі як Unity

версії 2022.3.9f1, Microsoft Visual Studio 2022 та Adobe Photoshop 2020

б) Розроблено графічну складову для реалізації демонстраційної версії гри, а саме спрайт-листи анімацій персонажів, спрайти елементів UI та тайлсети ігрових локацій.

в) Реалізована система кінцевих автоматів для управління поведінкою віртуальних персонажів. Ця система включає створення базових класів State, StateMachine та StateMachineCore, класів кінцевих автоматів PlayerStateMachine, DogNPCStateMachine, ArcherNPCStateMachine, DefenderNPCStateMachine, GhostNPCStateMachine та BatNPCStateMachine, класів станів та класів вхідних подій.

г) Виконано компілювання проекту та проведено тестування роботи демонстраційної версії гри.

ВИСНОВКИ

Представлену кваліфікаційну роботу спрямовано на розробку кінцевих автоматів для моделювання поведінки віртуальних персонажів.

В результаті аналізу предметної області розглянуто походження жанру «2D-Platformer» та визначено його основні характеристики та різновиди. Відзначено важливість застосування кінцевого автомата для регулювання поведінки персонажів в іграх жанру «2D-Platformer». Розглянуто такі інструменти моделювання поведінки віртуальних персонажів, як матриця взаємодій та кінцевий автомат. Розглянуто існуючі аналоги ігор жанру «2D-Platformer». Проаналізовано поведінку ворогів у зазначених проектах та виявлено переваги та недоліки реалізації їхньої поведінки. Проведено постановку завдання з урахуванням бажаного результату, необхідного для розробки проектної документації та демонстраційної версії гри.

Розроблено концептуальний та дизайнерський документи, які описують основні компоненти та особливості ігрового проекту. Розроблено матрицю взаємодії базових елементів гри (токенів) та кінцеві автомати головного героя та ворогів, за допомогою яких промодельовано поведінку персонажів.

Вибрано засоби розробки демонстраційної версії гри. Розроблено графічну складову для реалізації демонстраційної версії гри. Реалізована система кінцевих автоматів для управління поведінкою віртуальних персонажів. Виконано компілювання проекту та проведено тестування роботи демонстраційної версії гри.

Результатом даної роботи є проектна документація та демонстраційна версія гри в жанрі «2D-Platformer» з елементами Metroidvania, яка не має високих технічних вимог, має просте меню, управління та інтерфейс. Даний прототип гри може бути доповнений новими ворожими персонажами та вампірськими здібностями головного героя та офіційно випущений як гра на базі інтернет-ресурсу.

Система кінцевих автоматів, яка включає в себе три базові класи State, StateMachine та StateMachineCore, має потенціал для подальшого використання не лише в ігровій сфері, але й при проектуванні апаратних засобів комп'ютерних систем та мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ken Horowitz, Beyond Donkey Kong: A History of Nintendo Arcade Games // McFarland & Company, Inc.: North Carolina, 2020. – 264 p.
2. Super Meat Boy`s Hidden Depth [Електронний ресурс] – Режим доступу: https://www.ign.com/articles/2009/10/14/super-meat-boys-hidden-depth
3. Katana Zero review: The cutting edge [Електронний ресурс] – Режим доступу: https://www.shacknews.com/article/111305/katana-zero-review-the-cutting-edge
4. Dead Cells is a perfect example of Early Access done right [Електронний ресурс] – Режим доступу: https://www.pcgamer.com/dead-cells-is-a-perfect-example-of-early-access-done-right/
5. 10 Types of Platforms in Platform Video Games [Електронний ресурс] – Режим доступу: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games
6. Space Panic [Електронний ресурс] – Режим доступу: https://archive.org/details/arcade_panic
7. Josiah Lebowitz, Interactive Storytelling for Video Games: A Player-centered Approach to Creating Memorable Characters and Stories / Josiah Lebowitz, Chris Klug // Elsevier Inc.: Kidlington, 2011. – 318 p.
8. Castlevania – Developer Commentary [Електронний ресурс] – Режим доступу: https://shmuplations.com/castlevania/
9. Jump Bug [Електронний ресурс] – Режим доступу: https://www.arcade-museum.com/Videogame/jump-bug
10. The Video Game Canon: Contra [Електронний ресурс] – Режим доступу: https://www.warpzoned.com/2017/05/the-scientifically-proven-best-video-games-of-all-time-82-contra/
11. The Lost Vikings [Електронний ресурс] – Режим доступу: https://playclassic.games/games/puzzle-solving-dos-games-online/play-the-lost-vikings-online/

12. 1930s cartoon-inspired Cuphead targeting late 2014 on PC [Електронний ресурс] – Режим доступу: https://www.engadget.com/2014-01-04-1930s-cartoon-inspired-cuphead-targeting-late-2014-launch-on-pc.html
13. Gris from Nomada Studio and Devolver Digital Has Just Launched on the App Store [Електронний ресурс] – Режим доступу: https://toucharcade.com/2019/08/21/gris-available-now-price-app-store-nomada-studio-devolver/
14. Celeste is exactly how the Nintendo Switch continues to win [Електронний ресурс] – Режим доступу: https://web.archive.org/web/20180126151957/https://www.wired.com/story/celeste-nintendo-switch-review/
15. Spelunky 2 Review [Електронний ресурс] – Режим доступу: https://www.ign.com/articles/spelunky-2-review
16. Hollow Knight feels too familiar, despite being a solid metroidvania [Електронний ресурс] – Режим доступу: https://www.rockpapershotgun.com/hollow-knight-review
17. Crash Bandicoot 4: It`s About Time Review | An absolutely N. Credible sequel [Електронний ресурс] – Режим доступу: https://www.gamerevolution.com/review/661746-crash-4-review-ps4-xbox-one-ps5-xbox-series-x
18. Stylish 3D Platformer Demon Turf is Now Being Published by Playtonic Games [Електронний ресурс] – Режим доступу: https://gamerant.com/demon-turf-playtonic-games/
19. It Takes Two lovingly marries story and gameplay together [Електронний ресурс] – Режим доступу: https://www.unrealengine.com/en-US/developer-interviews/it-takes-two-lovingly-marries-story-and-gameplay-together
20. Project Feline [Електронний ресурс] – Режим доступу: https://www.projectfeline.com

21. A Hat in Time dev talks five-year development, Switch port, and what`s next [Електронний ресурс] – Режим доступу: https://nintendoeverything.com/interview-a-hat-in-time-dev-talks-five-year-development-switch-port-and-whats-next/
22. Unit-1 Game Core Design & Initial Design [Електронний ресурс] – Режим доступу: https://www.kdkce.edu.in/pdf/Unit%201-NEW%20-%20Game%20Core%20Design.pdf
23. Програмне забезпечення ігрових систем: конспект лекцій / С.В. Шестопалов, І.В. Колумба // О.: ОНТУ, 2024. – 121 с.
24. Кінцевий автомат (FSM) [Електронний ресурс] – Режим доступу: https://docs.aiogram.dev/uk-ua/latest/dispatcher/finite_state_machine/index.html
25. Д.В. Щербин, Дослідження підходів реалізації штучного інтелекту ворогів в іграх жанру «Shooter»: пояснювальна записка // О.: ОНТУ, 2023. – 144 с.
26. Sprite [Електронний ресурс] – Режим доступу: https://web.archive.org/web/20090525155624/http://encarta.msn.com/dictionary_/sprite.html
27. Tile-Based Texture Mapping on Graphics Hardware [Електронний ресурс] – Режим доступу: https://graphics.stanford.edu/papers/tile_mapping_gh2004/
28. Використання уніфікованої мови моделювання (UML) [Електронний ресурс] – Режим доступу: http://iwanoff.inf.ua/oop_kn/LabTraining05.html
29. Інструмент Fraps та його використання в тестуванні [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/technical-articles/fraps-tool/
30. Що таке Bug і Bug Report в тестуванні? [Електронний ресурс] – Режим доступу: https://training.epam.ua/ua/blog/460

ДОДАТОК А

Кінцевий автомат головного героя

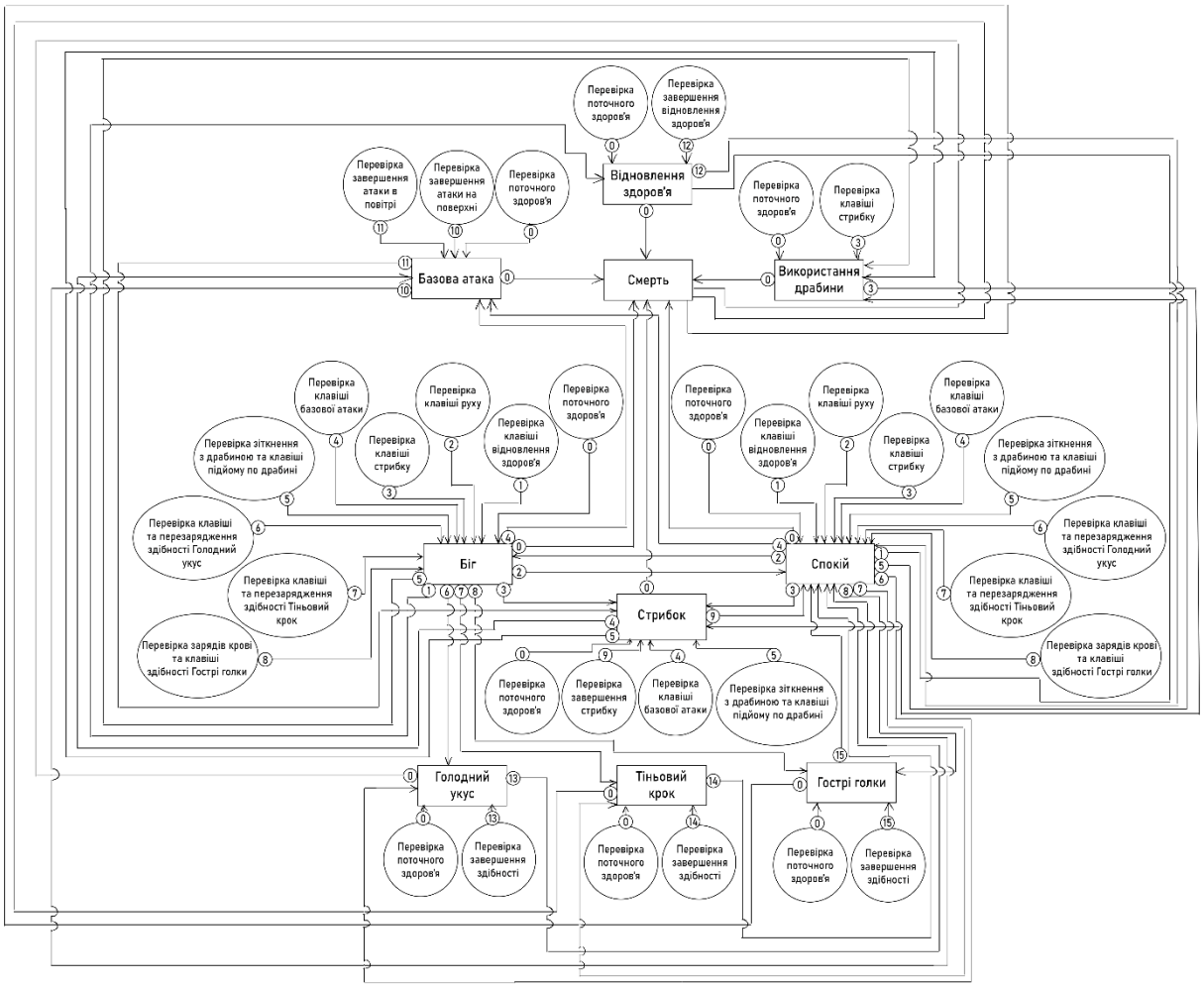


Рисунок А.1 – Спрощена схема кінцевого автомата головного героя

ДОДАТОК Б

Спрайти головного героя

Рисунок Б.1 – Спрайт-лист анімації спокою головного героя

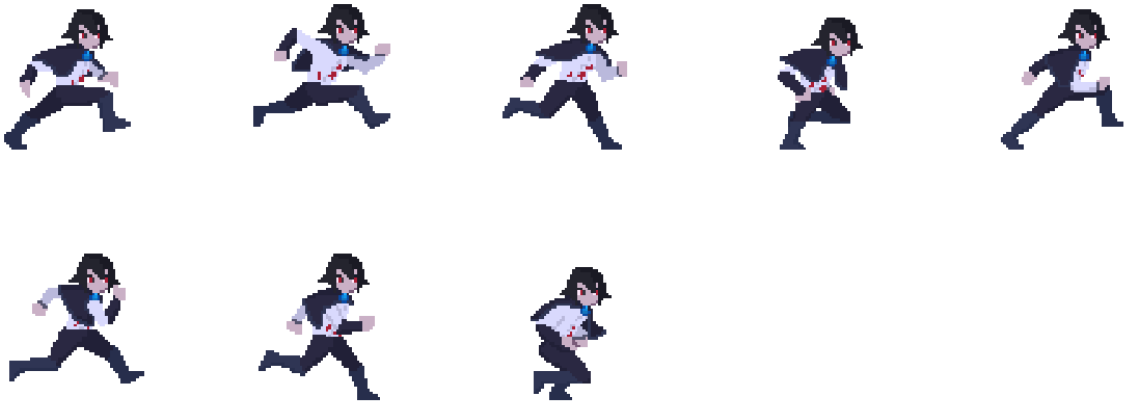


Рисунок Б.2 – Спрайт-лист анімації бігу головного героя

Рисунок Б.3 – Спрайт-лист анімації стрибка головного героя

Рисунок Б.4 – Спрайт-лист анімації використання драбини головного героя

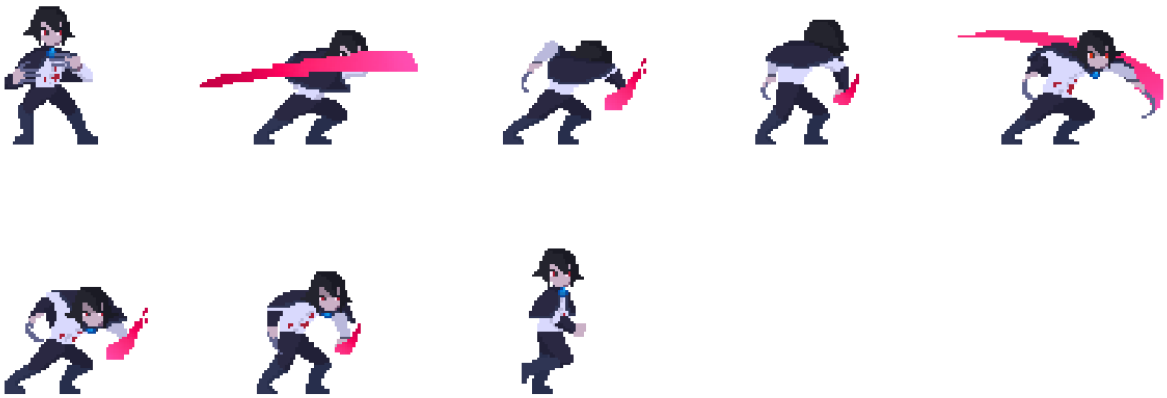


Рисунок Б.5 – Спрайт-лист анімації базової атаки головного героя

Рисунок Б.6 – Спрайт-лист анімації отримання шкоди головного героя

Рисунок Б.7 – Спрайт-лист анімації смерті головного героя

Рисунок Б.8 – Спрайт-лист анімації здібності «Голодний укус»

Рисунок Б.9 – Спрайт-лист анімації здібності «Гострі голки»

Рисунок Б.10 – Спрайт-лист анімації здібності «Тіньовий крок»

Рисунок Б.11 – Спрайт снаряда здібності «Гострі голки»

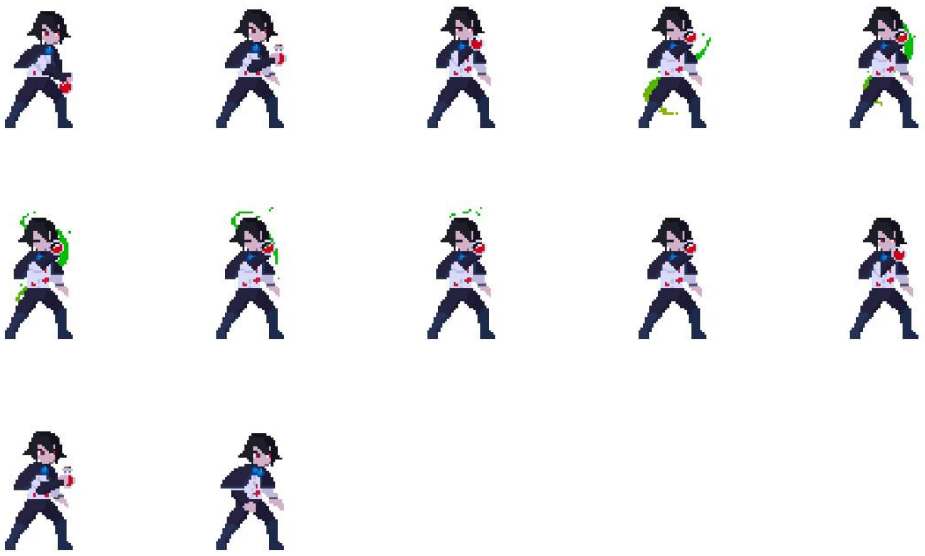


Рисунок Б.12. – Спрайт-лист анімації відновлення здоров'я головного героя

ДОДАТОК В

Спрайти ворогів

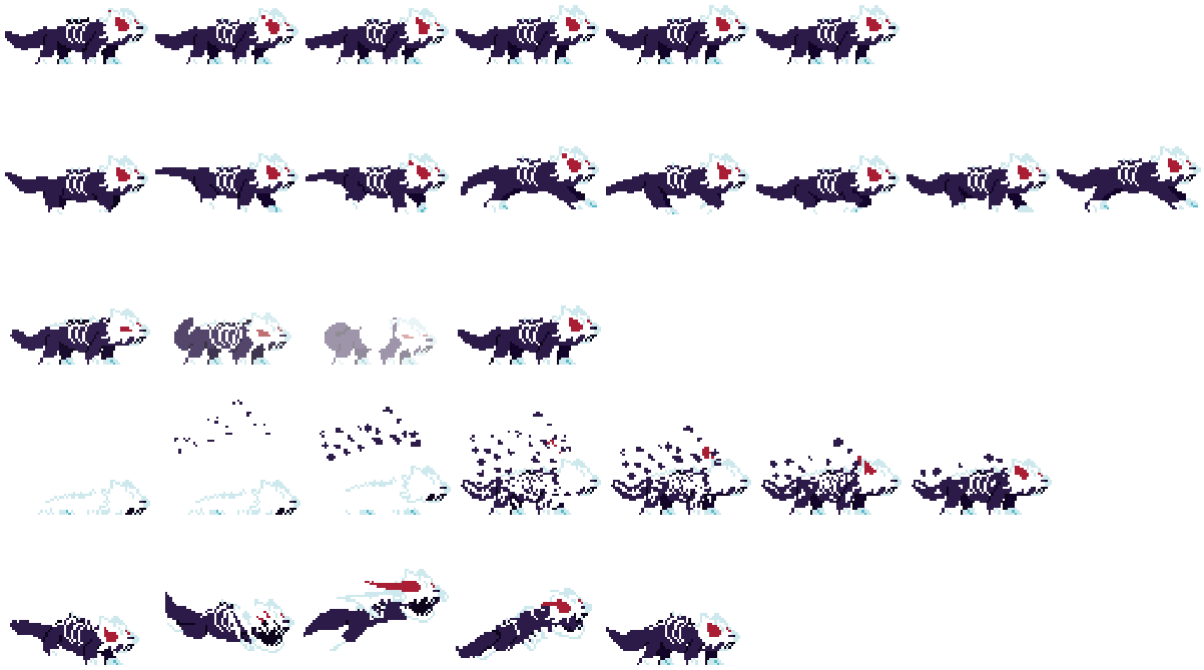


Рисунок В.1 – Спрайт-лист ворога Кістлявий вовк

Рисунок В.2 – Спрайт-лист ворога Вампір-лучник

Рисунок В.3 – Спрайт снаряда дальньої атаки ворога Вампір-лучник

Рисунок В.4 – Спрайт-лист ворога Вампір-стражник

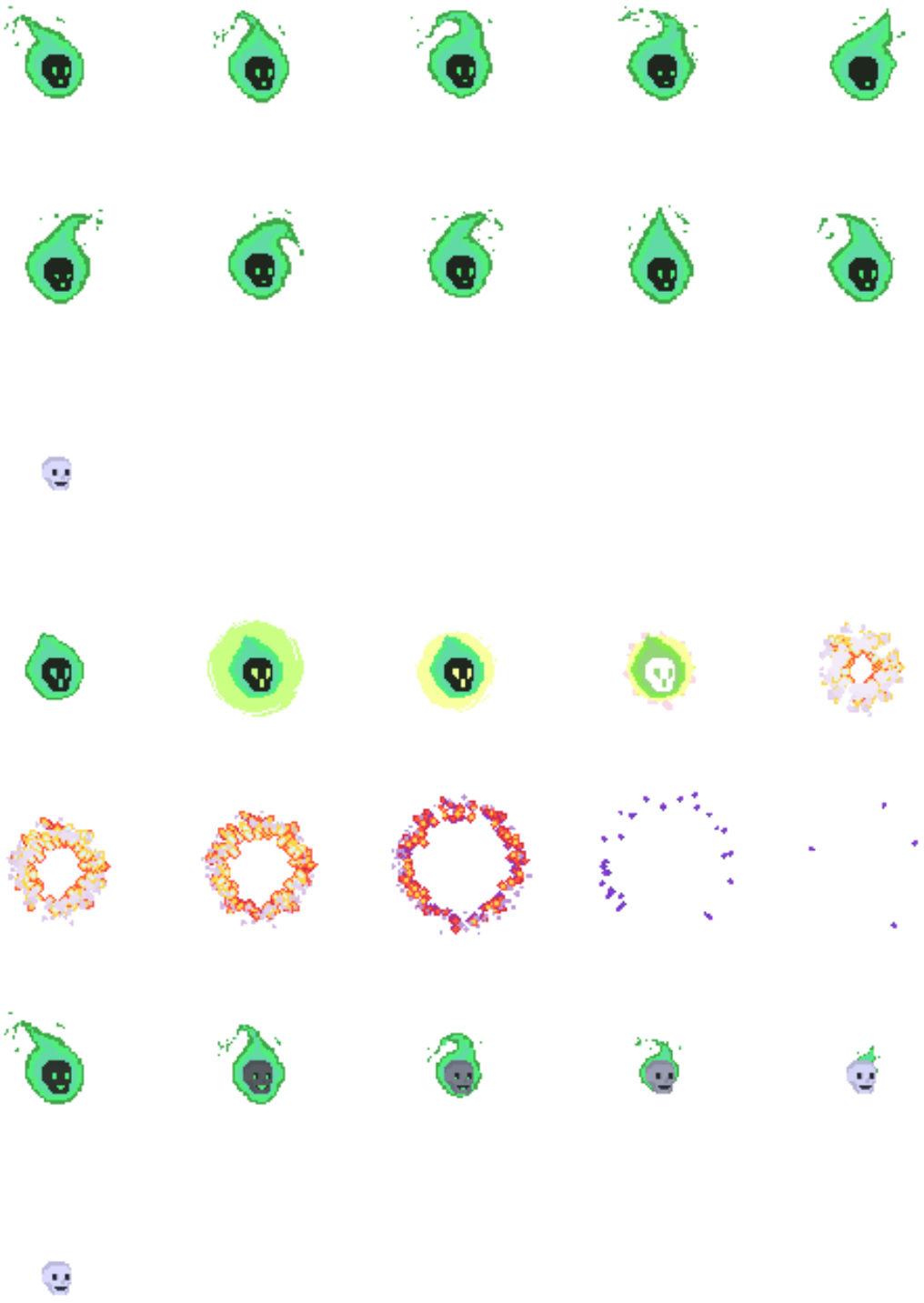


Рисунок В.5 – Спрайт-лист ворога Палаючий дух

Рисунок В.6 – Спрайт-лист ворога Кажан

ДОДАТОК Г

Спрайти елементів UI

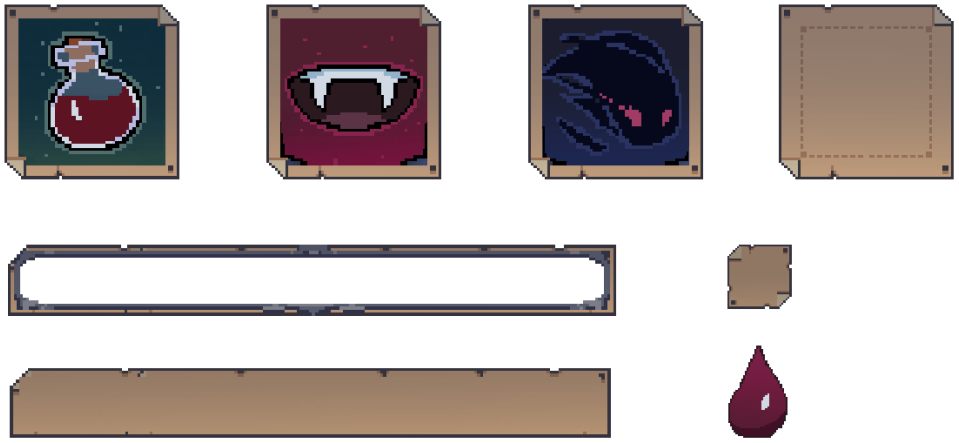


Рисунок Г.1 – Спрайти основних елементів UI

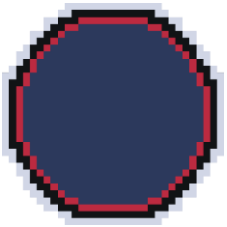


Рисунок Г.2 – Спрайт кнопки

Рисунок Г.3 – Фон головного меню гри

Рисунок Г.4 – Логотип гри

ДОДАТОК Д

Тайлсети локацій гри

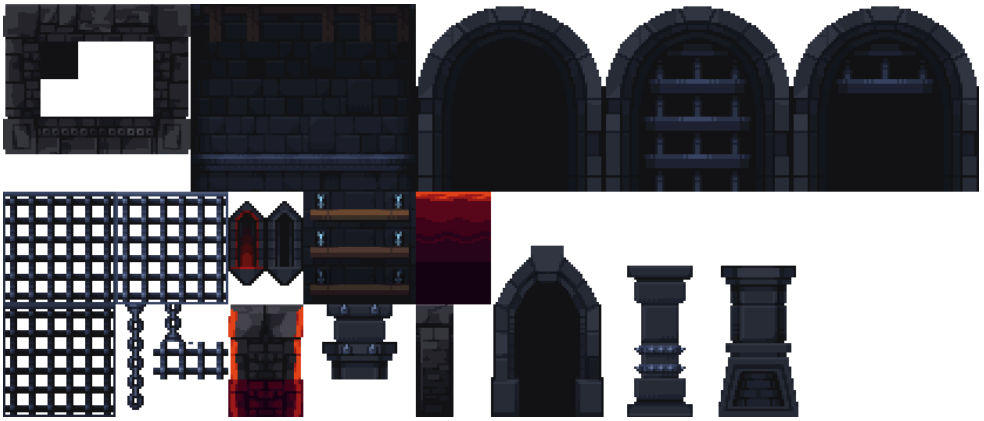


Рисунок Д.1 – Тайлсет локації «Тюремні камери»

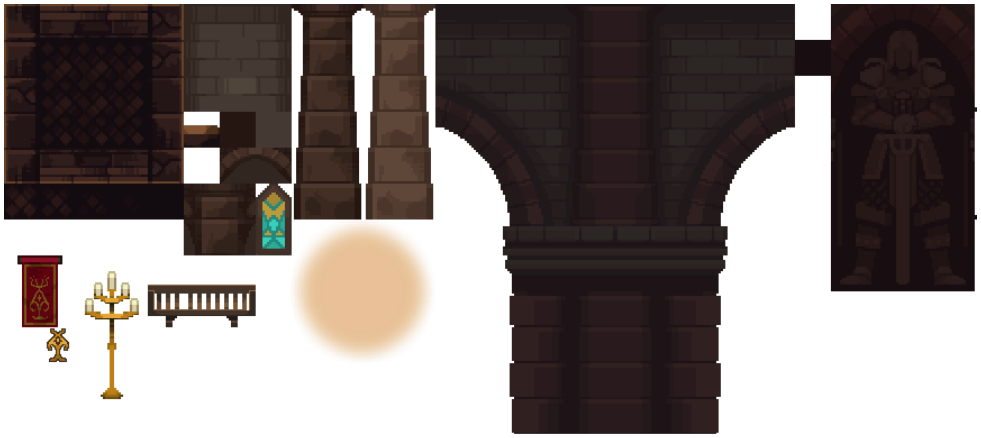


Рисунок Д.2 – Тайлсет локації «Прогулянковий хол»

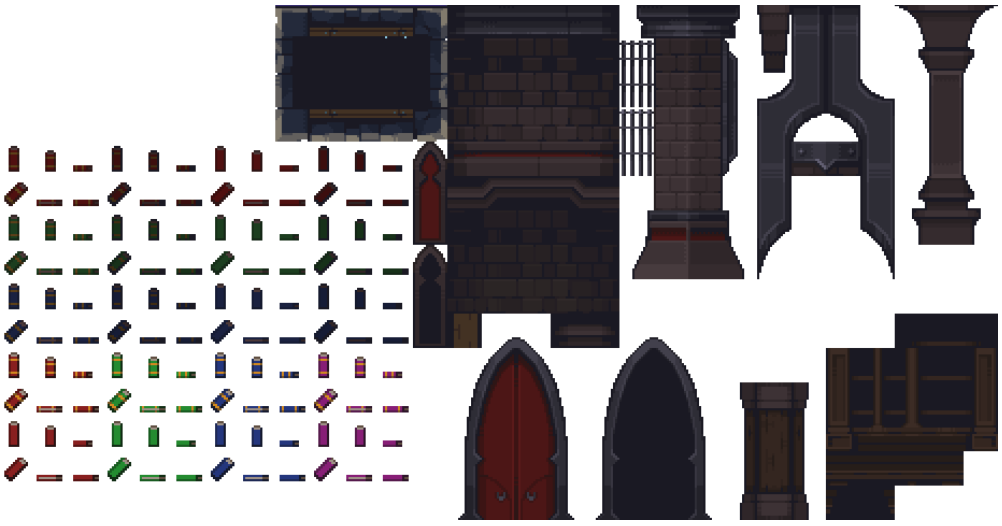


Рисунок Д.3 – Тайлсет локації «Бібліотека»

ДОДАТОК Е

Діаграма класів спроектованої системи

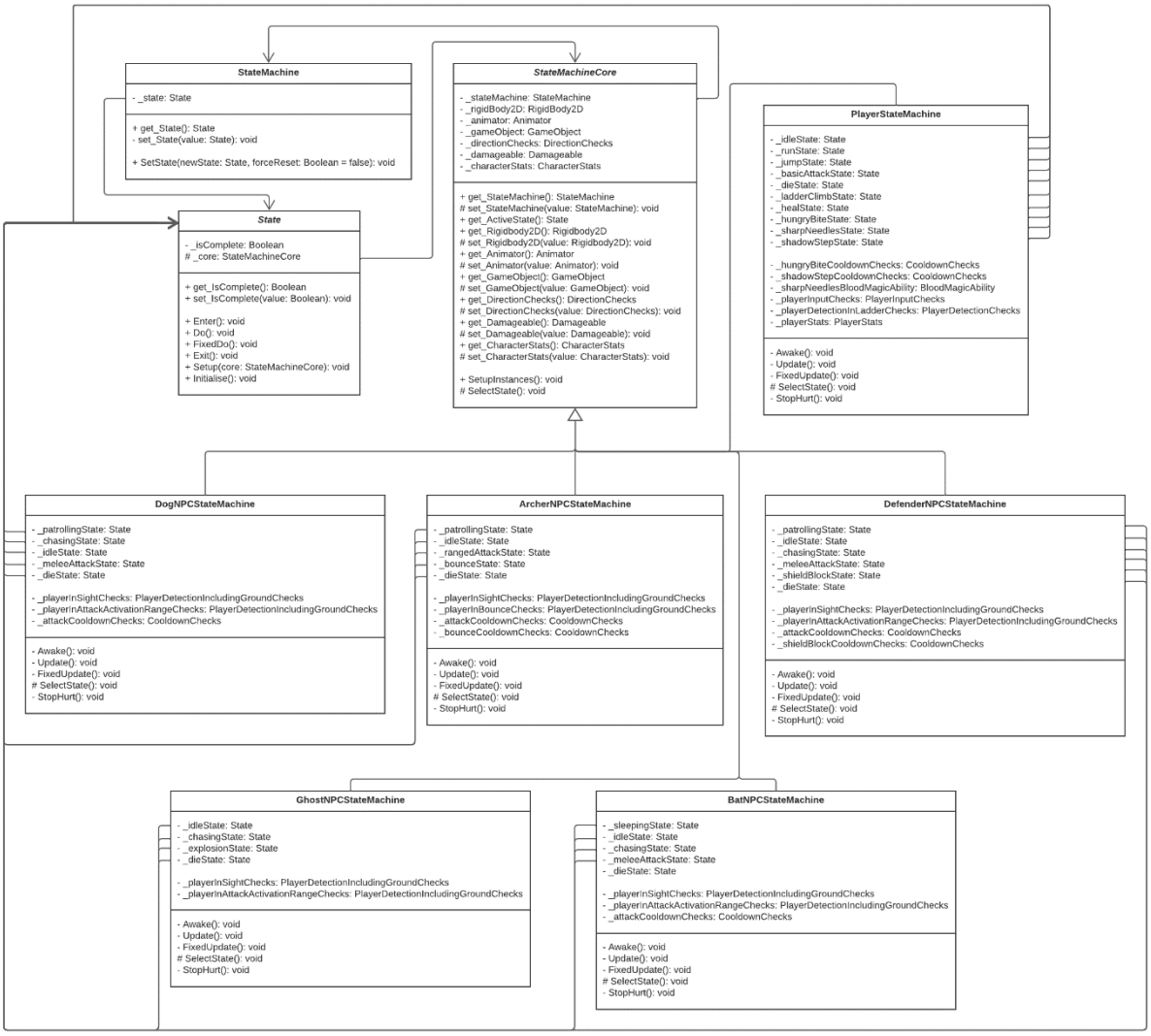


Рисунок Е.1 – Діаграма класів, пов’язана з кінцевими автоматами

ДОДАТОК Ж

Реалізація кінцевих автоматів віртуальних персонажів

```
using UnityEngine;

[RequireComponent (typeof(PlayerInputChecks),
typeof(AnimationEventHandler))]
public class PlayerStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _runState;

    [SerializeField]
    private State _jumpState;

    [SerializeField]
    private State _basicAttackState;

    [SerializeField]
    private State _dieState;

    [SerializeField]
    private State _ladderClimbState;

    [SerializeField]
    private State _healState;

    [SerializeField]
    private State _hungryBiteState;

    [SerializeField]
    private State _sharpNeedlesState;

    [SerializeField]
    private State _shadowStepState;

    [Header("Checks")]
    [SerializeField]
    private CooldownChecks _hungryBiteCooldownChecks;
    [SerializeField]
    private CooldownChecks _shadowStepCooldownChecks;
    [SerializeField]
    private BloodMagicAbility _sharpNeedlesBloodMagicAbility;
```

Лістинг Ж.1 – Програмна реалізація кінцевого автомату головного героя

```
    private PlayerInputChecks _playerInputChecks;
    private PlayerDetectionChecks
_playerDetectionInLadderChecks;
    private PlayerStats _playerStats;

    private void Awake()
    {
        SetupInstances();

        _playerInputChecks = GetComponent<PlayerInputChecks>();
        _playerStats = GetComponent<PlayerStats>();

        GameObject ladder =
GameObject.FindGameObjectWithTag("Ladder");
        _playerDetectionInLadderChecks =
ladder.GetComponent<PlayerDetectionChecks>();

        StateMachine.SetState(_idleState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt && !UIManager.Instance.IsPaused)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt && !UIManager.Instance.IsPaused)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
        if (StateMachine.State == _idleState)
        {
            if (_playerInputChecks.IsMovingHorizontally &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_runState);
            }
```

Лістинг Ж.1, аркуш 2

```
            else if (_playerInputChecks.IsJumpPressed &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_jumpState);
            }
            else if (_playerInputChecks.IsBasicAttackPressed &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_basicAttackState);
            }
            else if (_playerDetectionInLadderChecks.HasTarget &&
_playerInputChecks.IsLadderClimbPressed)
            {
                StateMachine.SetState(_ladderClimbState);
            }
            else if (_playerInputChecks.IsHealPressed &&
_playerStats.CurrentBloodFlasks > 0 &&
_playerStats.CurrentHealth < _playerStats.MaxHealth)
            {
                StateMachine.SetState(_healState);
            }
            else if (_playerInputChecks.IsHungryBitePressed &&
_hungryBiteCooldownChecks.HasEnded &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_hungryBiteState);
            }
            else if (_playerInputChecks.IsSharpNeedlesPressed &&
_playerStats.CurrentBloodCharges -
_sharpNeedlesBloodMagicAbility.BloodChargeNumber >= 0 &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_sharpNeedlesState);
            }
            else if (_playerInputChecks.IsShadowStepPressed &&
_shadowStepCooldownChecks.HasEnded &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_shadowStepState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _runState)
        {
            if (!_playerInputChecks.IsMovingHorizontally &&
DirectionChecks.IsGrounded)
```

Лістинг Ж.1, аркуш 3

```
            {
                StateMachine.SetState(_idleState);
            }
            else if (_playerInputChecks.IsJumpPressed &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_jumpState);
            }
            else if (_playerInputChecks.IsBasicAttackPressed &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_basicAttackState);
            }
            else if (_playerDetectionInLadderChecks.HasTarget &&
_playerInputChecks.IsLadderClimbPressed)
            {
                StateMachine.SetState(_ladderClimbState);
            }
            else if (_playerInputChecks.IsHealPressed &&
_playerStats.CurrentBloodFlasks > 0 &&
_playerStats.CurrentHealth < _playerStats.MaxHealth)
            {
                StateMachine.SetState(_healState);
            }
            else if (_playerInputChecks.IsHungryBitePressed &&
_hungryBiteCooldownChecks.HasEnded &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_hungryBiteState);
            }
            else if (_playerInputChecks.IsSharpNeedlesPressed &&
_playerStats.CurrentBloodCharges -
_sharpNeedlesBloodMagicAbility.BloodChargeNumber >= 0 &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_sharpNeedlesState);
            }
            else if (_playerInputChecks.IsShadowStepPressed &&
_shadowStepCooldownChecks.HasEnded &&
DirectionChecks.IsGrounded)
            {
                StateMachine.SetState(_shadowStepState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _jumpState)
        {
```

Лістинг Ж.1, аркуш 4

```
            if (_jumpState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (_playerDetectionInLadderChecks.HasTarget &&
_playerInputChecks.IsLadderClimbPressed)
            {
                StateMachine.SetState(_ladderClimbState);
            }
            else if (_playerInputChecks.IsBasicAttackPressed)
            {
                StateMachine.SetState(_basicAttackState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _basicAttackState)
        {
            if (_basicAttackState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _ladderClimbState)
        {
            if (!_playerDetectionInLadderChecks.HasTarget ||
_playerInputChecks.IsJumpPressed)
            {
                StateMachine.SetState(_jumpState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _healState)
        {
            if (_healState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
```

Лістинг Ж.1, аркуш 5

```
        }
        else if (StateMachine.State == _hungryBiteState)
        {
            if (_hungryBiteState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _sharpNeedlesState)
        {
            if (_sharpNeedlesState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _shadowStepState)
        {
            if (_shadowStepState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
        Damageable.IsHurt = false;
        Damageable.IsDamageIgnored = false;
        if (StateMachine.State == _ladderClimbState)
        {
            StateMachine.SetState(_ladderClimbState, true);
        }
        else
        {
            StateMachine.SetState(_idleState, true);
        }
    }
}
```

Лістинг Ж.1, аркуш 6

```
using UnityEngine;

public class DogNPCStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _patrollingState;

    [SerializeField]
    private State _chasingState;

    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _meleeAttackState;

    [SerializeField]
    private State _dieState;

    [Header("Checks")]
    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInSightChecks;

    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInAttackActivationRangeChecks;

    [SerializeField]
    private CooldownChecks _attackCooldownChecks;

    private void Awake()
    {
        SetupInstances();
        StateMachine.SetState(_patrollingState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
```

Лістинг Ж.2 – Програмна реалізація кінцевого автомату ворога Кістлявий вовк

```
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
        if (StateMachine.State == _patrollingState)
        {
            if (_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _chasingState)
        {
            if (!_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_patrollingState);
            }
            else if
(_playerInAttackActivationRangeChecks.HasTarget)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _idleState)
        {
            if (!_playerInAttackActivationRangeChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if (!_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_patrollingState);
            }
```

Лістинг Ж.2, аркуш 2

```
            else if (_attackCooldownChecks.HasEnded &&
_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_meleeAttackState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _meleeAttackState)
        {
            if (_meleeAttackState.IsComplete ||
!_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
        Damageable.IsHurt = false;
        Damageable.IsDamageIgnored = false;
        StateMachine.SetState(_idleState, true);
    }
}
```

Лістинг Ж.2, аркуш 3

```
using UnityEngine;

public class ArcherNPCStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _patrollingState;

    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _rangedAttackState;
```

Лістинг Ж.3 – Програмна реалізація кінцевого автомату ворога Вампір-лучник

```
    [SerializeField]
    private State _bounceState;

    [SerializeField]
    private State _dieState;

    [Header("Checks")]
    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInSightChecks;

    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInBounceChecks;

    [SerializeField]
    private CooldownChecks _attackCooldownChecks;

    [SerializeField]
    private CooldownChecks _bounceCooldownChecks;

    private void Awake()
    {
        SetupInstances();
        StateMachine.SetState(_patrollingState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
```

Лістинг Ж.3, аркуш 2

```
        if (StateMachine.State == _patrollingState)
        {
            if (_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _idleState)
        {
            if (!_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_patrollingState);
            }
            else if (_attackCooldownChecks.HasEnded &&
_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_rangedAttackState);
            }
            else if (_bounceCooldownChecks.HasEnded &&
_playerInBounceChecks.HasTarget &&
!_attackCooldownChecks.HasEnded)
            {
                StateMachine.SetState(_bounceState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _rangedAttackState)
        {
            if (_rangedAttackState.IsComplete ||
!_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _bounceState)
        {
            if (_bounceState.IsComplete ||
!_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_idleState);
```

Лістинг Ж.3, аркуш 3

```
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
        Damageable.IsHurt = false;
        Damageable.IsDamageIgnored = false;
        StateMachine.SetState(_idleState, true);
    }
}
```

Лістинг Ж.3, аркуш 4

```
using UnityEngine;

public class DefenderNPCStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _patrollingState;

    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _chasingState;

    [SerializeField]
    private State _meleeAttackState;

    [SerializeField]
    private State _shieldBlockState;

    [SerializeField]
    private State _dieState;

    [Header("Checks")]
    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInSightChecks;

    [SerializeField]
```

Лістинг Ж.4 – Програмна реалізація кінцевого автомату ворога Вампір-стражник

```
    private PlayerDetectionIncludingGroundChecks
_playerInAttackActivationRangeChecks;

    [SerializeField]
    private CooldownChecks _attackCooldownChecks;

    [SerializeField]
    private CooldownChecks _shieldBlockCooldownChecks;

    private void Awake()
    {
        SetupInstances();
        StateMachine.SetState(_patrollingState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
        if (StateMachine.State == _patrollingState)
        {
            if (_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
```

Лістинг Ж.4, аркуш 2

```
        else if (StateMachine.State == _chasingState)
        {
            if (!_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_patrollingState);
            }
            else if
(_playerInAttackActivationRangeChecks.HasTarget)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _idleState)
        {
            if (!_playerInAttackActivationRangeChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if (!_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_patrollingState);
            }
            else if (_attackCooldownChecks.HasEnded &&
!_shieldBlockCooldownChecks.HasEnded &&
_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_meleeAttackState);
            }
            else if (_shieldBlockCooldownChecks.HasEnded &&
_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_shieldBlockState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _meleeAttackState)
        {
            if (_meleeAttackState.IsComplete ||
!_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
```

Лістинг Ж.4, аркуш 3

```
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _shieldBlockState)
        {
            if (_shieldBlockState.IsComplete ||
!_playerInSightChecks.IsTargetAlive)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
        Damageable.IsHurt = false;
        Damageable.IsDamageIgnored = false;
        StateMachine.SetState(_idleState, true);
    }
}
```

Лістинг Ж.4, аркуш 4

```
using UnityEngine;

public class GhostNPCStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _chasingState;

    [SerializeField]
    private State _explosionState;

    [SerializeField]
    private State _dieState;

    [Header("Checks")]
    [SerializeField]
```

Лістинг Ж.5 – Програмна реалізація кінцевого автомату ворога Палаючий дух

```
    private PlayerDetectionIncludingGroundChecks
_playerInSightChecks;

    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInAttackActivationRangeChecks;

    private void Awake()
    {
        SetupInstances();
        StateMachine.SetState(_idleState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
        if (StateMachine.State == _idleState)
        {
            if (_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _chasingState)
        {
            if (_playerInAttackActivationRangeChecks.HasTarget)
```

```
            {
                StateMachine.SetState(_explosionState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _explosionState)
        {
            if (_explosionState.IsComplete)
            {
                Destroy(GameObject);
            }
            if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
        Damageable.IsHurt = false;
        Damageable.IsDamageIgnored = false;
        StateMachine.SetState(_idleState, true);
    }
}
```

Лістинг Ж.5, аркуш 3

```
using UnityEngine;

public class BatNPCStateMachine : StateMachineCore
{
    [Header("States")]
    [SerializeField]
    private State _sleepingState;

    [SerializeField]
    private State _idleState;

    [SerializeField]
    private State _chasingState;

    [SerializeField]
    private State _meleeAttackState;

    [SerializeField]
```

Лістинг Ж.6 – Програмна реалізація кінцевого автомату ворога Кажан

```
    private State _dieState;

    [Header("Checks")]
    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInSightChecks;

    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInAttackActivationRangeChecks;

    [SerializeField]
    private CooldownChecks _attackCooldownChecks;

    private void Awake()
    {
        SetupInstances();
        StateMachine.SetState(_sleepingState);
    }

    private void Update()
    {
        if (!Damageable.IsHurt)
        {
            SelectState();
            StateMachine.State.Do();
        }
        else
        {
            Rigidbody2D.velocity = new Vector2(0,
Rigidbody2D.velocity.y);
        }
    }

    private void FixedUpdate()
    {
        if (!Damageable.IsHurt)
        {
            StateMachine.State.FixedDo();
        }
    }

    protected override void SelectState()
    {
        if (StateMachine.State == _sleepingState)
        {
            if (_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
```

Лістинг Ж.6, аркуш 2

```
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _idleState)
        {
            if (_playerInSightChecks.HasTarget &&
!_playerInAttackActivationRangeChecks.HasTarget)
            {
                StateMachine.SetState(_chasingState);
            }
            else if
(_playerInAttackActivationRangeChecks.HasTarget &&
_attackCooldownChecks.HasEnded &&
_playerInAttackActivationRangeChecks.IsTargetAlive)
            {
                StateMachine.SetState(_meleeAttackState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _chasingState)
        {
            if (_playerInAttackActivationRangeChecks.HasTarget
|| !_playerInSightChecks.HasTarget)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
        else if (StateMachine.State == _meleeAttackState)
        {
            if (_meleeAttackState.IsComplete)
            {
                StateMachine.SetState(_idleState);
            }
            else if (!Damageable.IsAlive)
            {
                StateMachine.SetState(_dieState);
            }
        }
    }

    private void StopHurt()
    {
        Animator.StopPlayback();
```

Лістинг Ж.6, аркуш 3

```
            Damageable.IsHurt = false;
            Damageable.IsDamageIgnored = false;
            StateMachine.SetState(_idleState, true);
        }
    }
```

Лістинг Ж.6, аркуш 4

ДОДАТОК К

Реалізація станів віртуальних персонажів

```
using UnityEngine;

public class MeleeAttackState : State
{
    [Header("Parameters")]
    [SerializeField]
    private AnimationClip _attackClip;
    [SerializeField]
    private AnimationEventHandler _eventHandler;
    [SerializeField]
    private AttackAbility _attackAbility;
    [SerializeField]
    private Vector2 _knockback = Vector2.zero;
    [SerializeField]
    private bool _isFlying = false;

    [Header("Checks")]
    [SerializeField]
    private CooldownChecks? _cooldownChecks;
    [SerializeField]
    private PlayerDetectionIncludingGroundChecks
_playerInAttackRangeChecks;

    private bool _hasAttacked;
    private Vector2 _deliveredKnockback = Vector2.zero;
    private float _defaultGravityScale;
    private bool _hasAttackCooldown = false;

    public override void Enter()
    {
        if (_isFlying)
        {
            _defaultGravityScale =
_core.Rigidbody2D.gravityScale;
            _core.Rigidbody2D.gravityScale = 0f;
        }

        if (_cooldownChecks != null)
        {
            _hasAttackCooldown = true;
        }

        _eventHandler.OnFinished += StopAttack;
        _core.Animator.Play(_attackClip.name);
        _hasAttacked = false;
```

Лістинг К.1 – Програмна реалізація стану атаки ближнього бою ворога

```
    }

    public override void FixedDo()
    {
        if (_isFlying)
        {
            _core.Rigidbody2D.velocity = Vector2.zero;
        }
        else
        {
            _core.Rigidbody2D.velocity = new Vector2(0,
_core.Rigidbody2D.velocity.y);
        }

        if (_playerInAttackRangeChecks.HasTarget && _hasAttacked
== false)
        {
            Damageable damageable =
_playerInAttackRangeChecks.Target.GetComponent<Damageable>();

            if (damageable != null)
            {
                _hasAttacked = true;
                _deliveredKnockback = new Vector2(_knockback.x *
_core.GameObject.transform.localScale.x, _knockback.y);
                bool gotHit =
damageable.Hit(_attackAbility.CurrentDamage,
_deliveredKnockback);
            }
        }
    }

    private void StopAttack()
    {
        _core.Animator.StopPlayback();
        if (_cooldownChecks != null && _hasAttackCooldown)
        {
            _cooldownChecks.ResetCooldown();
        }
        IsComplete = true;
    }

    public override void Exit()
    {
        _eventHandler.OnFinished -= StopAttack;
        if (_isFlying)
        {
            _core.Rigidbody2D.gravityScale =
_defaultGravityScale;
            _core.Rigidbody2D.velocity = Vector2.zero;
        }
```

Лістинг К.1, аркуш 2

```
        IsComplete = true;
    }
}
```

Лістинг К.1, аркуш 3

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

public class PlayerBasicAttackState : State
{
    [Header("Parameters")]
    [SerializeField]
    private List<AnimationClip> _attackClips = new();
    [SerializeField]
    private AnimationEventHandler _eventHandler;
    [SerializeField]
    private AttackAbility _attackAbility;
    [SerializeField]
    private Vector2 _targetKnockback = Vector2.zero;
    [SerializeField]
    private float _dashForce = 5f;

    [Header("Checks")]
    [SerializeField]
    private PlayerInputChecks _playerInputChecks;
    [SerializeField]
    private List<NPCDetectionChecks> _NPCDetectionChecks =
new();

    private bool _hasAttacked;
    private int _currentAttackIndex = 0;
    private Vector2 _deliveredKnockback = Vector2.zero;

    public override void Enter()
    {
        _eventHandler.OnFinished += StopAttack;

_core.Animator.Play(_attackClips[_currentAttackIndex].name);
        _hasAttacked = false;
        _core.Rigidbody2D.velocity = new Vector2(_dashForce *
Mathf.Sign(_core.GameObject.transform.localScale.x),
_core.Rigidbody2D.velocity.y);
    }

    public override void FixedDo()
    {
        if (_core.Damageable.IsHurt)
        {
            IsComplete = true;
```

Лістинг К.2 – Програмна реалізація стану базової атаки головного героя

```
        }

        if (_hasAttacked == false)
        {
            foreach (var collider in
_NPCDetectionChecks[_currentAttackIndex].DetectedColliders)
            {
                Damageable damageable =
collider.GetComponent<Damageable>();
                if (damageable != null)
                {
                    _deliveredKnockback = new
Vector2(_targetKnockback.x *
_core.DirectionChecks.RunDirectionVector.x, _targetKnockback.y);
                    _hasAttacked = true;
                    bool gotHit =
damageable.Hit(_attackAbility.CurrentDamage,
_deliveredKnockback);
                }
            }
        }
    }

    private void StopAttack()
    {
        _core.Animator.StopPlayback();
        IsComplete = true;
    }

    public override void Exit()
    {
        _eventHandler.OnFinished -= StopAttack;

_NPCDetectionChecks[_currentAttackIndex].ResetDetectedColliders(
);
        _currentAttackIndex++;
        if (_currentAttackIndex >= _attackClips.Count)
        {
            _currentAttackIndex = 0;
        }
        IsComplete = true;
    }
}
```

Лістинг К.2, аркуш 2

```
using UnityEngine;

public class IdleState : State
{
```

Лістинг К.3 – Програмна реалізація стану спокою

```
    [SerializeField]
    private AnimationClip _idleClip;
    [SerializeField]
    private bool _isFlying = false;

    private float _defaultGravityScale;

    public override void Enter()
    {
        if (_isFlying)
        {
            _defaultGravityScale =
_core.Rigidbody2D.gravityScale;
            _core.Rigidbody2D.gravityScale = 0f;
        }

        _core.Animator.StopPlayback();
        _core.Animator.Play(_idleClip.name);
    }

    public override void FixedDo()
    {
        if (_isFlying)
        {
            _core.Rigidbody2D.velocity = Vector2.zero;
        }
        else
        {
            _core.Rigidbody2D.velocity = new Vector2(0,
_core.Rigidbody2D.velocity.y);
        }
    }

    public override void Exit()
    {
        if (_isFlying)
        {
            _core.Rigidbody2D.gravityScale =
_defaultGravityScale;
        }
    }
}
```

Лістинг К.3, аркуш 2