Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
"Програмне забезпечення для навчання основам інформатики школярів молодших класів"
08-54.БКР.004.00.000 ПЗ

Виконала: студентка 4 курсу, групи 1КІ-23мс
спеціальності 123 — "Комп'ютерна інженерія"
освітня програма — "Комп'ютерна інженерія"
Вареник А. О.
(прізвище та ініціали)

Керівник к.т.н., доц. каф. ОТ
Черняк О. І.
(прізвище та ініціали)

Рецензент к.т.н., доц. каф ПЗ
Хошаба О.М.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«11» 06 2025 р.

Вінниця ВНТУ — 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалавр
Галузь знань — 12 "Інформаційні технології"
Спеціальність — 123 "Комп'ютерна інженерія"
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф._____Азаров О. Д.
"21" березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Вареник Анні Олегівні

1 Тема роботи: "Програмне забезпечення для навчання основам інформатики школярів молодших класів", керівник роботи Черняк О. І. к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від "20" березня 2025 року № 97.

2 Строк подання студентом роботи 13.05.2025

3 Вихідні дані до роботи — навчальна програма з інформатики для молодших класів, принципи створення навчальних ігор, базові вимоги до інтерактивного контенту та піксельної графіки, вікові особливості цільової аудиторії.

4 Зміст пояснювальної записки: вступ, аналіз предметної області, проектування та реалізація програмного забезпечення, тестування функціоналу, підсумки реалізації, висновки.

5 Перелік графічного матеріалу: головне меню гри, схема навігації, ігрові сцени, приклади інтерфейсу користувача, зразки спрайтів у піксельному стилі.

6 Консультанти розділів роботи представлені у Таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Черняк О. І.	21.03.2025	13.05.2025
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.2025	30.05.2025

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план виконання БКР наведений у таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання	Примітка
1.	Постановка задачі роботи	21.03.25	Вик.
2.	Аналіз предметної області	22.03.25-27.03.25	Вик.
3.	Аналіз потреб школярів	28.03.25-06.04.25	Вик.
4.	Вибір технологій та інструментів для розробки	07.04.25-09.04.25	Вик.
5.	Розробка структурної схеми	10.04.25-13.04.25	Вик.
6.	Програмна реалізація ПЗ	14.04.25-04.05.25	Вик.
7.	Тестування ПЗ	05.05.25-07.05.25	Вик.
8.	Оформлення пояснювальної записки	08.05.25-12.05.25	Вик.
9.	Перевірка якості виконання БКР	13.05.25	Вик.

Студентка Анна ВАРЕНИК

Керівник роботи Олександр ЧЕРНЯК

АНОТАЦІЯ

УДК 004.42

Вареник А. О. Програмне забезпечення для навчання основам інформатики школярів молодших класів. Бакалаврська кваліфікаційна робота зі спеціальності 123 – Комп'ютерна інженерія; освітня програма – Комп’ютерна інженерія. Вінниця: ВНТУ, 2025. 54 с.

На укр. мові. Бібліогр.: 20 назв; рис.: 28.

У бакалаврській кваліфікаційній роботі розглянуто процес створення програмного забезпечення у вигляді навчальної гри, що має на меті ознайомити школярів молодших класів з основами інформатики. Проведено аналіз педагогічних та технічних вимог до навчального програмного забезпечення, розглянуто сучасні підходи до використання ігрових механік у сфері освіти.

У роботі описано архітектуру програмного забезпечення, логіку руху та анімацій головного героя, реалізацію бою з акторами, сценарії отримання шкоди та завершення гри. Створено інтерфейс користувача з головним меню, системою вибору розділів за темами та екраном завершення проходження рівнів. Для зручності масштабування параметри акторів реалізовано за допомогою ScriptableObject.

Ключові слова: інформатика, навчальна гра, Unity, школярі, гейміфікація, головний герой, актор, поведінка, атака.

ABSTRACT

Varenyk A. O. Educational Software for Teaching the Basics of Informatics to Primary School Students. Bachelor's thesis in specialty F7 – Computer Engineering; educational program – Computer Engineering. Vinnytsia: VNTU, 2025. 54 p.

In Ukrainian language. Bibliographer: 20 titles; Fig.: 28.

This bachelor’s thesis examines the process of developing educational software in the form of a game aimed at introducing primary school students to the fundamentals of informatics. The work includes an analysis of pedagogical and technical requirements for educational software and explores modern approaches to using gamification in education.

The paper describes the game architecture, the logic of character movement and animations, the implementation of combat with enemies, scenarios of taking damage, and game over conditions. A user interface has been developed with a main menu, thematic section selection, and a game-over screen. For easier scalability, enemy parameters are implemented using ScriptableObject.

Keywords: informatics, educational game, Unity, primary school, gamification, main character, enemy, behavior, attack.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології відіграють важливу роль у житті кожної людини, і особливо це стосується освіти. Впровадження цифрових технологій у навчальний процес сприяє підвищенню ефективності засвоєння знань, розвитку логічного мислення та формуванню навичок роботи з комп'ютером. Одним із перспективних напрямків є використання ігрових методик у навчанні, що особливо актуально для школярів молодших класів. Саме тому тема даного дипломного проекту присвячена розробці програмного забезпечення для навчання основ інформатики молодших школярів у процесі гри.

Актуальність полягає у тому, що навчання через гру є одним із найефективніших методів для дітей молодшого шкільного віку, оскільки поєднує в собі пізнавальний та розважальний аспекти. Враховуючи постійний розвиток інформаційних технологій, важливо створювати інтерактивні засоби навчання, які допомагають дитині не лише засвоїти базові знання з інформатики, а й розвинути критичне мислення та творчі здібності.

Метою даного дипломного проекту є розробка програмного забезпечення для навчання основ інформатики учнів молодших класів на основі ігрових методик, що сприятиме підвищенню їхньої мотивації школярів до правильного користування комп’ютерною технікою.

Для досягнення мети, потрібно виконати такі завдання:

— аналіз теоретичних основ та існуючих рішень;

— визначення основних складових програмного забезпечення для навчання основ інформатики;

— реалізація програмного забезпечення для навчання основ інформатики.

Об'єкт дослідження — процес розробки програмного забезпечення для навчання основ інформатики школярів молодших класів із використанням цифрових технологій.

Предмет дослідження — програмне забезпечення, яке реалізує ігрові методики для вивчення основ інформатики на платформі Unity із використанням мови програмування C#.

Інноваційність полягає у тому, що на відміну від існуючих програм у даній розробці запропоновано нові аспекти взаємодії з комп`ютерною технікою, що допомагає користувачам отримати досвід обслуговування технічних засобів.

Практичне застосування розробленого програмного забезпечення може бути використане в освітніх закладах для викладання основ інформатики молодшим школярам. Воно допоможе зробити навчальний процес більш цікавим, інтерактивним і ефективним, що сприятиме розвитку сучасних навичок роботи з інформаційними технологіями у дітей.

Апробація. Матеріали дослідження за темою бакалаврської роботи "Програмне забезпечення для навчання основам інформатики школярів молодших класів" доповідались на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2025). Опубліковано тези доповіді.

У ході виконання дипломного проекту застосовано методи аналізу та узагальнення наукової літератури, методи програмної розробки та тестування програмного забезпечення.

1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Сучасний стан використання програмного забезпечення у навчанні

Інформатизація суспільства кардинально змінила вимоги до освітніх програм, особливо в початковій школі. Уже з перших класів діти мають опановувати базові навички роботи з комп’ютером, розуміти поняття інформації, алгоритму, безпечної роботи в інтернеті. Відповідно до положень реформи Нової української школи (НУШ) [1], інформатика є складовою інтегрованого курсу "Я досліджую світ" або викладається окремим предметом, починаючи з другого класу.

Одним із важливих напрямків навчання інформатики в початковій школі є розвиток цифрової грамотності: вміння ефективно користуватися пристроями та розуміння принципів безпечної поведінки у цифровому середовищі. Крім цього, сучасні навчальні програми акцентують увагу на формуванні алгоритмічного мислення, навичок вирішення задач, критичного аналізу інформації.

Окрему увагу в навчальних планах не дуже приділяють темі технічного обслуговування пристроїв. Молодші школярі повинні розуміти базові правила догляду за комп’ютерами, планшетами та іншими гаджетами: дотримання чистоти, уникнення потрапляння рідин, правильне транспортування пристроїв, регулярне оновлення програмного забезпечення [2]. Роз'яснення таких основних правил є надзвичайно важливим, оскільки сприяє формуванню в дітей відповідального ставлення до техніки та запобігає її поломкам через необережність.

Сучасні методи викладання інформатики активно використовують ігрові підходи. Відповідно до вікових особливостей молодших школярів, навчання через гру сприяє кращому засвоєнню матеріалу, підвищенню інтересу до предмета та розвитку пізнавальної активності. Інтерактивні завдання, навчальні ігри, симуляції — усе це дозволяє легко і невимушено пояснювати навіть складні для сприйняття поняття.

Однією з перспективних форм подання інформації є ігрове моделювання ситуацій догляду за пристроями. У такому підході діти можуть навчатися

правильним діям через взаємодію з віртуальним світом — наприклад, боротися з "пилом", що шкодить комп’ютеру, або уникати "крапель води", що символізують загрозу для електроніки. Такий метод дозволяє закріпити знання через емоційно-позитивний досвід та сприяє кращому засвоєнню інформації.

Разом із тим, існують і певні виклики. Не всі школи мають достатню кількість сучасної техніки або доступ до якісних навчальних програм. Також залишається питання підготовки педагогів до впровадження інноваційних методів навчання у сфері інформатики.

Отже, сучасний стан викладання інформатики в початковій школі характеризується інтеграцією цифрової грамотності, основ алгоритмізації, правил технічного обслуговування пристроїв і активним застосуванням ігрових технологій у навчальному процесі. Формування відповідального ставлення до використання комп’ютерної техніки є важливим елементом загальної підготовки молодших школярів до життя в сучасному цифровому середовищі.

1.2 Ігровий підхід до навчання дітей у молодших класах

Комп’ютерні ігри є потужним інструментом розвитку дитини, який при правильному використанні може значно підвищити ефективність навчання. Особливо актуальним їх застосування є у початковій школі, коли учні перебувають на етапі формування базових навичок мислення, сприйняття, уваги та пам’яті.

Гра — діяльність людини з моделювання іншого виду діяльності з розважальною чи навчальною метою [3]. І це природний вид діяльності для дітей молодшого віку. Саме через гру вони краще пізнають навколишній світ, набувають соціального досвіду та навичок співпраці. Відповідно, використання комп’ютерних ігор у навчальному процесі дозволяє забезпечити високу мотивацію до вивчення нового матеріалу, активізувати пізнавальну діяльність та зробити навчання більш емоційно забарвленим і захоплюючим.

Ігрові навчальні програми дають змогу наочно демонструвати поняття, які складно пояснити лише теоретично. Завдяки візуалізації, інтерактивності та

динамічності учні легше засвоюють інформацію, розвивають логічне мислення та навички вирішення проблемних ситуацій. Комп’ютерні ігри сприяють також розвитку дрібної моторики, координації рухів та уважності.

Однією з важливих особливостей навчальних ігор є можливість створення умов для самостійного відкриття знань. У процесі гри дитина не просто отримує готову інформацію, а навчається через спроби і помилки, аналізує наслідки своїх рішень, що стимулює розвиток аналітичного мислення. Більш того, гейміфікація навчання дозволяє індивідуалізувати процес: кожен учень може рухатися у власному темпі, отримуючи миттєвий зворотний зв'язок про свої досягнення.

Ігрові засоби навчання особливо ефективні у викладанні інформатики, де важливо сформувати не тільки знання, але й практичні вміння користування пристроями. Через сюжетні комп’ютерні ігри учні можуть моделювати різні ситуації використання техніки, тренувати навички правильного догляду за комп’ютерами, планшетами та іншими гаджетами.

Навчальне програмне забезпечення у вигляді гри можуть також підвищити рівень цифрової безпеки дітей, оскільки через ігрові механіки легше пояснити поняття про небезпеки у мережі, захист персональних даних, правила поводження з електронною поштою та іншими сервісами.

Переваги ігрових технології у тому, що їх використання допомагає зняти психологічні бар’єри у взаємодії учнів з комп’ютером, розвивати потребу в творчій діяльності, створювати умови для самовираження дитини, підсилювати мотивацію до навчання, а вчитель займає активну позицію і за необхідності стає проміжною ланкою між комп’ютером і учнем під час занять [4].

Водночас застосування комп’ютерних ігор у навчанні потребує правильного педагогічного супроводу. Важливо ретельно добирати зміст програмного забезпечення, її складність і тривалість, враховуючи вікові особливості учнів. Програмне забезпечення з ігровими методиками повинні не просто розважати, а відповідати навчальним цілям, сприяти розвитку певних умінь та навичок.

Отже, використання комп’ютерного програмного забезпечення в ігровій формі у навчанні молодших школярів є ефективним засобом підвищення мотивації, розвитку критичного мислення та формування практичних навичок. В умовах правильної організації гейміфіковане навчання дозволяє гармонійно поєднати розвагу та освітній процес, забезпечуючи всебічний розвиток дитини в інформаційному суспільстві.

1.3 Платформи для розробки навчальних продуктів

Сучасна індустрія комп’ютерних ігор базується на використанні спеціалізованих ігрових рушіїв (game engines), які значно спрощують створення графіки, анімації, фізичних взаємодій та логіки ігрового процесу. Завдяки готовим наборам інструментів, рушії дозволяють зосередитися на геймдизайні та педагогічних цілях, не витрачаючи час на розробку "з нуля" низькорівневих систем. Найпоширенішими платформами для розробки ігор сьогодні є Unreal Engine, Godot та Unity, кожен із яких має свої особливості та сфери найкращого застосування.

Unreal Engine, створений компанією Epic Games, заслужено вважається одним із найпотужніших рушіїв для реалізації фотореалістичної графіки та складних 3D-сцен. Він використовує мову C++ та систему візуального скриптування Blueprints, що дає змогу поєднувати високу продуктивність із відносною простотою прототипування.

Завдяки інтегрованим засобам для роботи з матеріалами, освітленням і частинками, Unreal Engine широко застосовується в розробці, програмного забезпечення, масштабних проектів AAA-класу, у візуалізації архітектурних об’єктів, а також у сфері VR/AR. Проте через високу вимогливість до обладнання та довшу криву навчання даний рушій рідше використовується в освітніх проектах для молодшої вікової групи. Інтерфейс Unreal Engine показано на Рисунку 1.1 [5].

Рисунок 1.1 — Інтерфейс Unreal Engine

Godot Engine — відкрита платформа з повністю безкоштовним ліцензуванням, яка швидко завоювала популярність серед інді-розробників. Використовуючи власну скриптову мову GDScript (схожу на Python), а також підтримуючи C# і VisualScript, Godot спрощує створення як 2D-, так і 3D-ігор.

Основні переваги цього рушія — невеликий розмір середовища розробки, зрозумілий інтерфейс та можливість швидко опанувати базові інструменти. Однак через обмежену екосистему плагінів і меншій кількості готових активів Godot зазвичай вибирають для невеликих або доволі специфічних проектів, де контроль над кожним аспектом кодування є важливішим за комплексний набір інструментарію. Інтерфейс Godot Engine можна побачити на Рисунку 1.2 [6].

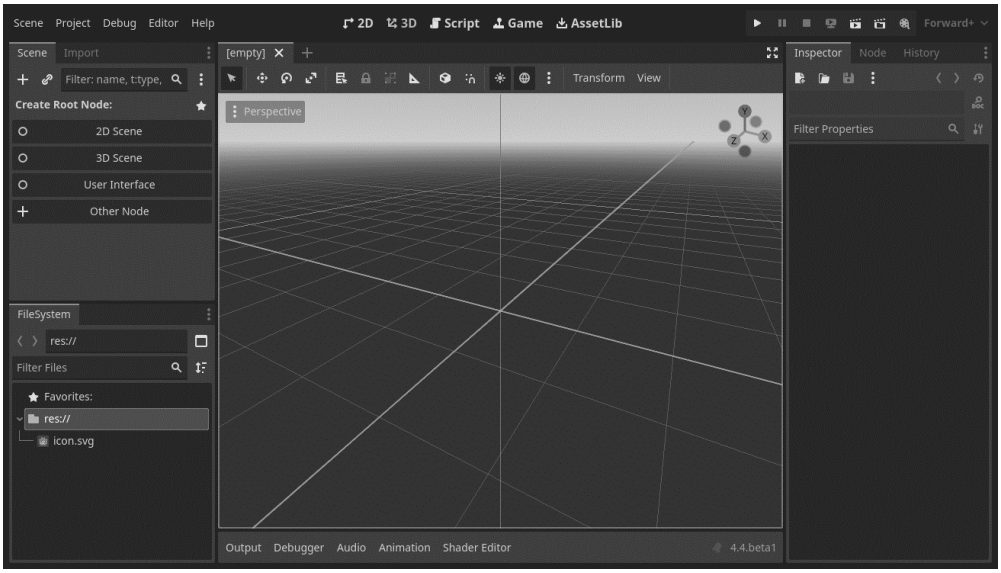

Рисунок 1.2 — Інтерфейс Godot Engine

Unity стоїть на перехресті простоти освоєння та широких можливостей. Від свого дебюту в 2005 році цей рушій зосередився на підтримці великої кількості платформ (Windows, macOS, Android, iOS, WebGL, консолі та VR/AR), що робить його ідеальним вибором для створення мультиплатформених програм.

Розробка в Unity ведеться на мові C#, яка поєднує в собі зрозумілість синтаксису та потужність сучасного об’єктно-орієнтованого підходу. Інтерфейс редактора побудований таким чином, щоб навіть користувачі з невеликим або мінімальним досвідом могли швидко зорієнтуватися у створенні сцен, налаштуванні анімацій і фізики, а також у редагуванні матеріалів і освітлення. Інтерфейс Unity зображено на Рисунку 1.3.

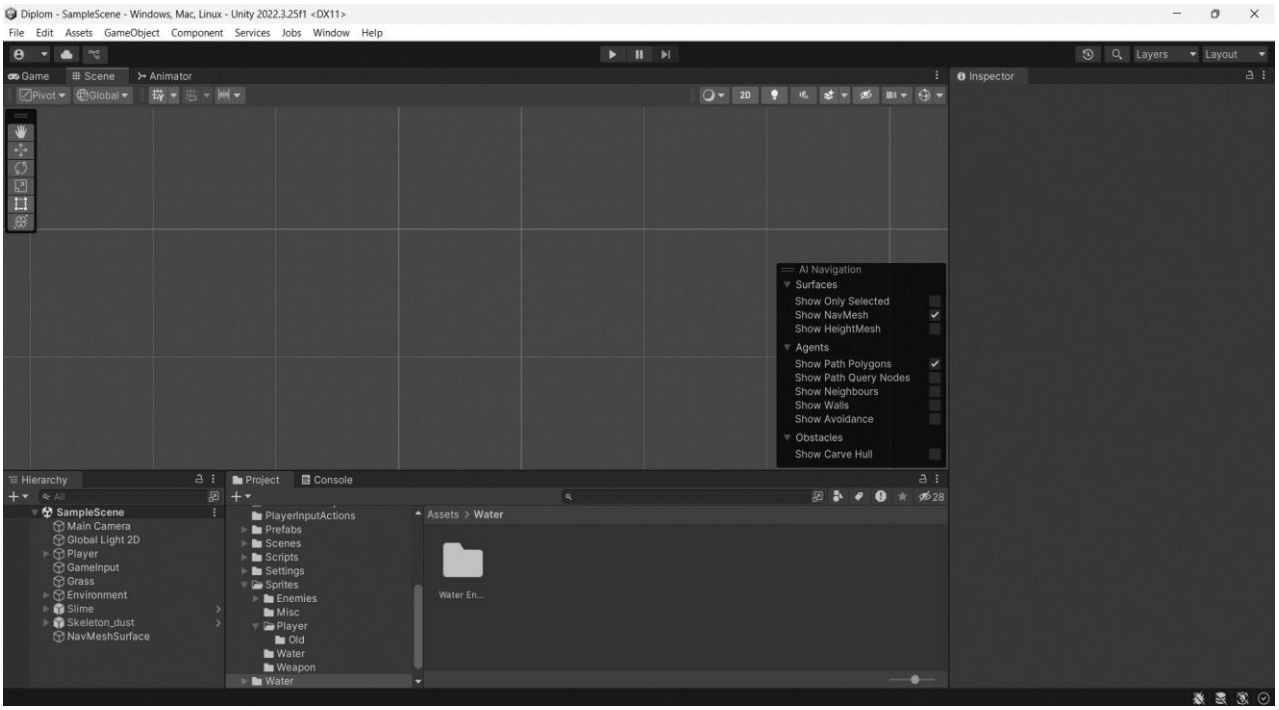

Рисунок 1.3 — Інтерфейс Unity

Крім того, в Unity Asset Store містяться тисячі готових моделей, текстур, скриптів і шаблонів, що суттєво скорочує час та полегшує прототипування, що дозволяє освітнім розробникам більше зосередитися на детальнішій та глибшій розробці програмного забезпечення, не відволікаючись на створення власних спрайтів [7]. Ресурс Unity Asset Store показано на Рисунку 1.4.

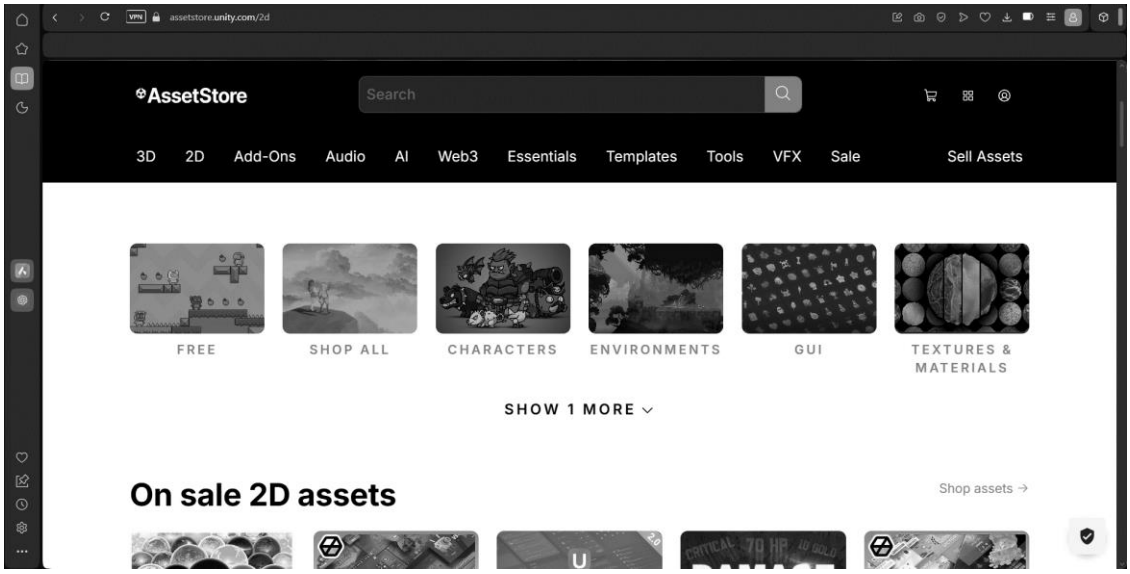

Рисунок 1.4 — Ресурс Unity Asset Store

Особливо важливою при розробці освітніх ігор є можливість з легкістю створювати 2D-інтерфейси й контролювати порядок відображення графічних елементів. Unity надає вбудовані компоненти UI, які дозволяють реалізувати меню, підказки, лічильники очок та інші важливі елементи навчального інтерфейсу без необхідності писати власні системи розмітки. А вбудований фізичний двигун і редактор анімацій допомагають робити взаємодію користувача з об’єктами максимально природною та наочною.

Для цього дипломного проекту, спрямованого на навчання молодших школярів основам інформатики та догляду за комп’ютерною технікою, Unity було обрано з кількох причин.

По-перше, рушій дозволяє швидко створювати адаптивні навчальні рівні й ігрові механіки з використанням простих C#-скриптів. По-друге, завдяки кросплатформеності готовий додаток можна запускати як у шкільних комп’ютерних класах, так і на планшетах чи в браузері, що підвищує доступність гри для учнів.

По-третє, наявність великої кількості безкоштовних ресурсів у Asset Store суттєво спрощує інтеграцію ілюстрацій, анімацій та звукових ефектів, необхідних для створення цікавого й мотивуючого ігрового середовища. Саме такі можливості

роблять Unity найприйнятнішим інструментом для реалізації освітнього проекту із гейміфікацією процесу вивчення інформатики в початковій школі.

1.4 Огляд існуючих програмних продуктів для навчання інформатики

Серед найвідоміших та найпоширеніших програмних засобів для введення дітей у світ інформатики — онлайн-платформи Code.org і Scratch.

Code.org [8] пропонує структуровані "уроки" з блокового програмування, де учні складають алгоритми з кольорових блоків, а кожне завдання відразу показує результат у вигляді анімацій чи простих ігор. Це дозволяє дітям молодшого шкільного віку засвоювати базові поняття алгоритмізації та логіки без початкових знань із текстового коду. На Рисунку 1.5 показано, наявні курси на Code.org.

Рисунок 1.5 — Курси Code.org

Також даний ресурс дозволяє не лише навчатися школярам. Є можливість зареєструватися як викладач, що надає можливості створення власних "уроків", відстеження прогресу учнів тощо, як показано на Рисунку 1.6. Такий функціонал в свою чергу показує гнучкість, легку інтегрованість з шкільною програмою та націленість на результат з Code.org.

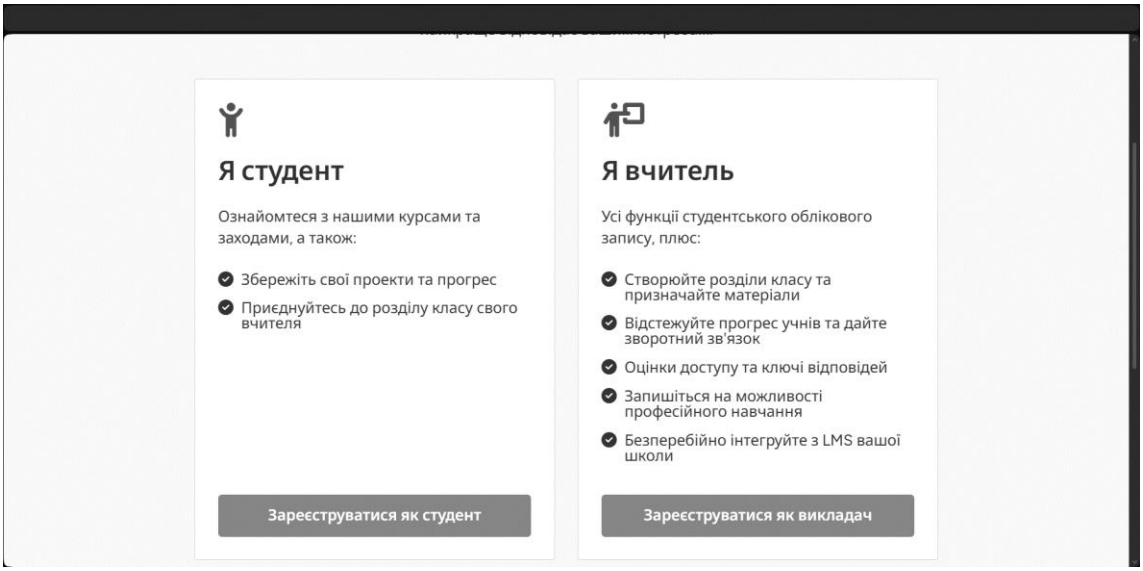

Рисунок 1.6 — Вибір реєстрації на Code.org

Scratch, розроблений MIT Media Lab, поширюється як десктопний додаток та веб-інструмент і дозволяє створювати інтерактивні історії, ігри й мультимедіа-проекти, що розвиває креативне мислення й дає перший досвід роботи з подіями та змінними [9]. Ресурс можна побачити на Рисунку 1.7.

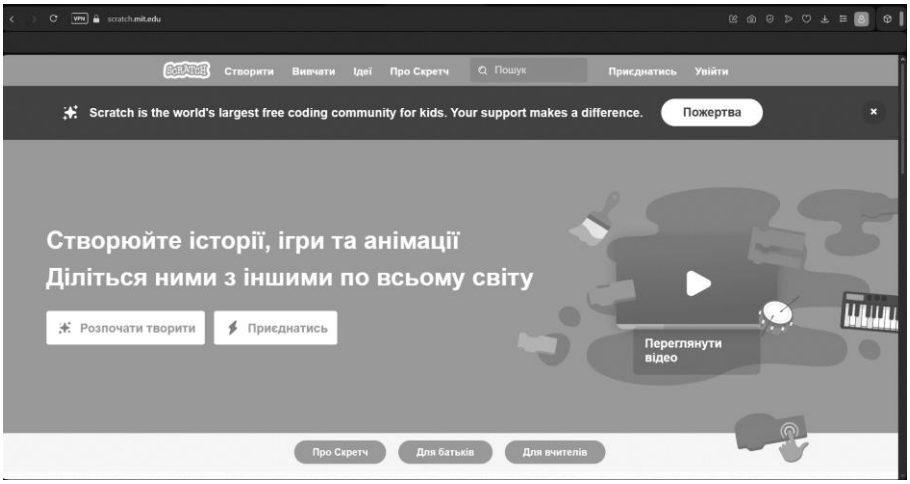

Рисунок 1.7 — Вибір реєстрації на Scratch

Інший популярний ресурс — Tynker[10], який містить набір уроків ігор із різним рівнем складності, а також готові шаблони проектів. Tynker інтегрує не лише блокове програмування, а й елементи текстових мов JavaScript та Python на просунутому рівні, що підходить для дітей від 8 років.

Він має зручний інтерфейс і підтримує мобільні додатки, але більшість контенту доступно за платною підпискою, що може бути обмеженням для шкільних кабінетів із вузьким бюджетом. Показаний даний ресурс на Рисунку 1.8.

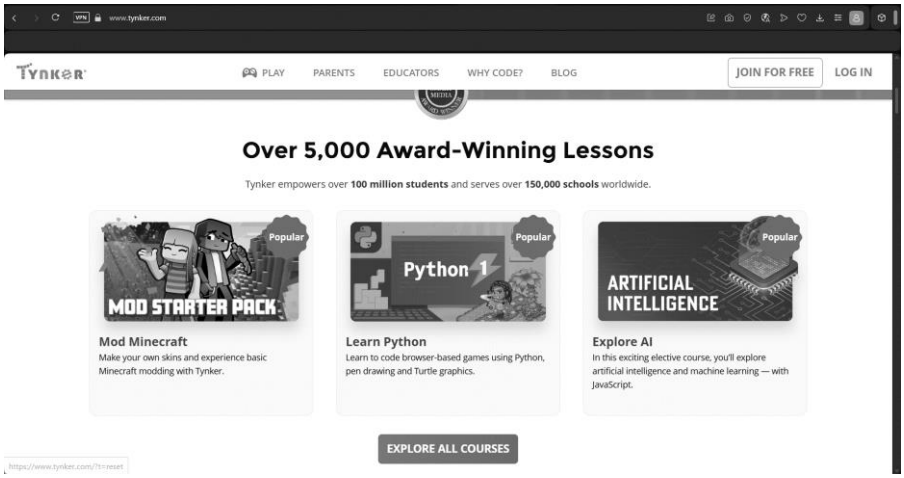

Рисунок 1.8 — Ресурс Tynker

Усі згадані рішення добре ілюструють базові підходи до навчання та розробки програмного забезпечення: від блокового кодування до симуляцій в ігровій формі. Проте більшість із них або не дозволяють вбудовувати власні сценарії догляду за пристроями, або вимагають платної підписки за повний функціонал.

На відміну від них, проект на Unity забезпечить повний контроль над механікою, включаючи моделювання догляду за технікою в інтерактивній формі, а також доступний без додаткових ліцензійних витрат.

2 ВИЗНАЧЕННЯ ОСНОВНИХ СКЛАДОВИХ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ НАВЧАННЯ ОСНОВ ІНФОРМАТИКИ

В основу даної роботи покладена концепція розробки програмного забезпечення, що привчає в ігровій формі дітей молодшого шкільного віку до правильного користування комп'ютерною технікою. У зв'язку з цим з'явилася ідея створити гру, в якій користувач бореться із двома ворожими акторами: Пилом та Водою. Даний підхід направлений на те, щоб привчити дітей постійно звертати увагу на ці два негативних чинника з метою їх нейтралізації. Ця ідея реалізована в ігровій формі може не нав'язливо і зацікавлено вволікати молодших школярів до уважного і бережного відношення до засобів комп'ютерної техніки, спонукаючи їх до дотримання правил користування будь-якими електроприладами.

Особливістю даного підходу є те, що навчальні цілі досягаються без всякого напруження і примусу у ненав'язливій формі. У процесі такого навчання школярі комфортно себе почиватимуть та будуть заохочувати своїх друзів і знайомих. На основі даного підходу було розроблено ігрове програмне забезпечення.

Для реалізації даної ідеї, визначимо такі основні складові проекту:

— вибір ігрового рушія та інструментальних засобів розробки;
— визначення структури проекту;
— визначення принципів управління персонажем;
— проектування анімаційного супроводу станів головного героя;
— проектування поведінки акторів та системи бою;
— моделювання станів та анімаційної поведінки акторів;
— проектування зброї головного героя;
— проектування взаємодії атаки героя та порогів.

2.1 Вибір рушія та інструментальних засобів розробки

Для створення навчальної гри було обрано ігровий рушій Unity версії 2022.3.25f1 [11], що є одним із найбільш популярних середовищ для розробки інтерактивних додатків різної складності. Рішення використовувати саме цю

платформу ґрунтується на її універсальності, широких можливостях для створення двовимірних і тривимірних проектів, а також доступності великої кількості навчальних матеріалів та готових ресурсів.

Unity має зручний візуальний редактор, що дозволяє легко розміщувати об'єкти на сцені, налаштовувати їх властивості, а також створювати ієрархії об'єктів без потреби у складних налаштуваннях. Однією з вагомих переваг рушія є підтримка роботи з великою кількістю різноманітних форматів графічних і анімаційних ресурсів, що значно полегшує імпорт та налаштування об'єктів у грі. Окрім того, середовище Unity надає потужний набір інструментів для створення анімацій, управління фізикою об'єктів, обробки взаємодії між персонажами та об'єктами середовища.

Важливо зазначити, що Unity підтримує програмування на мові C#, яка є сучасною об'єктно-орієнтованою мовою із чітким і зрозумілим синтаксисом. Це дозволяє гнучко реалізовувати логіку програмного забезпечення, зручно працювати із подіями, обробляти введенні дані користувачем та налаштовувати взаємодію між об'єктами сцени.

Крім того, обрана версія Unity надає доступ до нової системи обробки вводу Input System (Рисунок 2.1), що забезпечує більш гнучке та масштабоване управління діями гравця, що є важливим для реалізації сучасних вимог до користувацького досвіду.

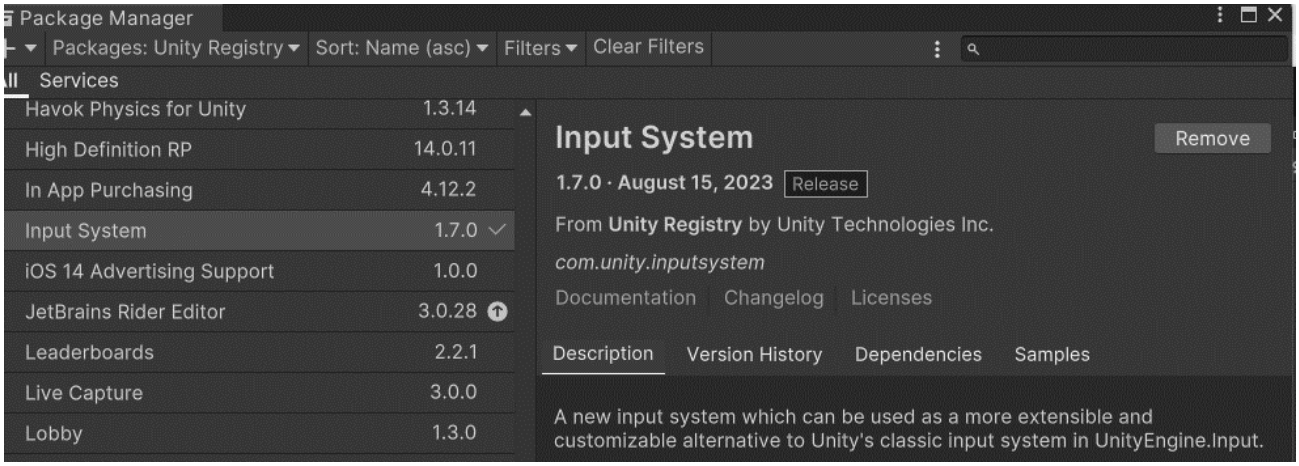

Рисунок 2.1 — Система обробки Input System

Інструменти розробки доповнюються вбудованим редактором анімацій Animator та можливістю створення навігаційної сітки NavMesh [12] для реалізації штучного інтелекту супротивників. Завдяки цьому стає можливим створення складнішої поведінки акторів, що переслідують персонажа і взаємодіють із навколишнім середовищем.

Таким чином, вибір Unity як основного інструменту розробки дозволяє не лише скоротити час створення гри, але й забезпечити високу якість візуальної та інтерактивної складової проекту, що є критично важливим для досягнення освітніх цілей гри.

2.2 Визначення структури проекту

На етапі проектування було особливу увагу приділено правильній організації структури файлів та ресурсів у середовищі розробки Unity. Грамотно побудована структура дозволяє підтримувати порядок у проекті, полегшує навігацію між об'єктами та значно спрощує подальшу розробку і тестування нових функціональних можливостей програмного забезпечення.

Проект поділений на кілька основних каталогів відповідно до логічного призначення їхнього вмісту. Такий підхід є загальноприйнятим у професійній розробці і значно підвищує зручність роботи з великими обсягами даних.

Усі сцени проекту, на яких розміщуються ігрові об’єкти, середовище та персонажі, зберігаються в окремій папці Scenes. На даному етапі проекту створено базову сцену під назвою SampleScene, яка служить тестовим майданчиком для розробки основної логіки програмного забезпечення та перевірки взаємодії між об'єктами.

Графічні ресурси, такі як моделі персонажа, ворожих акторів і об'єктів середовища, об’єднані в папку Prefabs. Кожен префаб це заздалегідь налаштований шаблон об’єкта, який можна багаторазово використовувати при розробці програмного забезпечення. Це забезпечує не лише зручність розміщення об'єктів

на сцені, а й можливість централізованого оновлення властивостей усіх копій префаба при зміні оригінального об'єкта.

Анімаційні дані винесено до окремого каталогу Animations. У цій папці зберігаються анімаційні кліпи для головного героя та ворожих акторів, а також не мало важливий компонент — аніматори. Вони використовуються для налаштування переходів між різними станами анімацій. Використання централізованого сховища для анімацій дозволяє легко оновлювати чи модифікувати рухи персонажів без необхідності змінювати численні об'єкти вручну. Такий підхід сприяє підтриманню чистоти структури проєкту та зменшує кількість помилок, що можуть виникати при дублюванні анімацій або неправильній синхронізації станів.

Скрипти, які реалізують логіку поведінки персонажа, ворожих акторів та обробку користувацького вводу інформації, розташовані у папці Scripts. Кожен скрипт має чітко визначене призначення, що сприяє модульності коду, його кращому тестуванню та легшому внесенню змін у майбутньому.

Така структуризація дозволяє розробнику зосередитися на окремих аспектах програмного забезпечення — наприклад, бойовій механіці, керуванні, штучному інтелекті ворожих акторів або взаємодії з середовищем — не порушуючи при цьому загальну архітектуру проекту. Крім того, це значно спрощує командну роботу та повторне використання компонентів у разі розширення гри або створення нових рівнів.

Окремо зберігаються налаштування введення користувача у папці Settings. Це дозволяє централізовано управляти способами взаємодії гравця із програмним забезпеченням, що особливо важливо для підтримки адаптивності управління в різних режимах (наприклад, для комп'ютерів або мобільних пристроїв).

Організація структури проекту, яку можна побачити на Рисунку 2.2, у такий спосіб забезпечує не лише зручність поточної роботи над програмним забезпеченням, але й створює міцний фундамент для подальшого розширення

програмного забезпечення. При потребі додавання нових механік, персонажів або елементів середовища, така структура дасть змогу швидко інтегрувати зміни без ризику порушити вже реалізовану логіку. У довгостроковій перспективі це також полегшує обслуговування коду, пошук та виправлення помилок, що особливо важливо в контексті програмного забезпечення, який може доповнюватися новими темами або рівнями відповідно до необхідних тем.

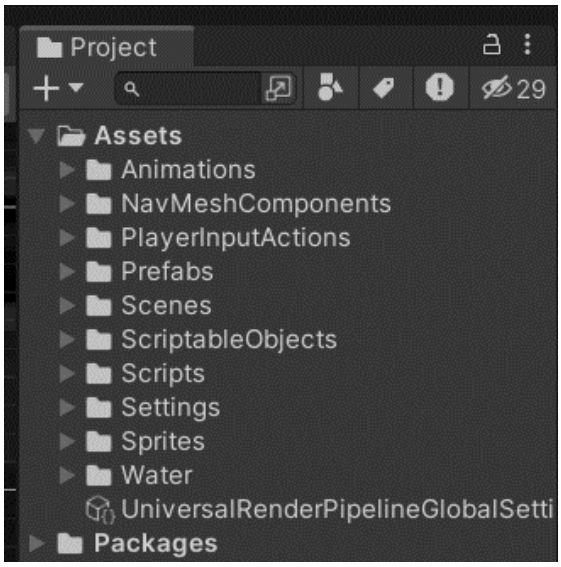

Рисунок 2.2 — Структура проекту

Таким чином, правильна організація структури проекту є важливим етапом розробки і дозволяє забезпечити високу якість ігрового продукту навіть на етапі його активного вдосконалення. Вона дозволяє забезпечити високу якість програмного продукту навіть на етапі його активного вдосконалення та тестування.

Завдяки чітко структурованим каталогам і логічному розподілу відповідальностей між компонентами, розробник отримує можливість швидко орієнтуватися в коді, ефективно відслідковувати взаємозв’язки між об’єктами та з легкістю вносити необхідні зміни без ризику дестабілізації проекту.

Отже, дотримання принципів структурованості на рівні файлової системи проекту є запорукою стабільної роботи програмного забезпечення, його зручної підтримки та подальшого розвитку в майбутньому.

2.3 Принципи управління персонажем

У комп’ютерних навчальних іграх важливо забезпечити інтуїтивно зрозуміле та зручне управління головним персонажем. Це дозволяє користувачеві швидко адаптуватися до інтерфейсу, зосередитись на виконанні завдань і сприяє формуванню позитивного досвіду взаємодії з програмним продуктом. У межах розроблюваного програмного забезпечення управління реалізовано з урахуванням вікових особливостей молодших школярів — воно мінімалістичне, базується на клавішах руху з клавіатури (W, A, S, D або стрілками) та натисканням лівої кнопки миші для атаки.

Основним функціональним елементом системи управління виступає клас Player (Додаток Г), який відповідає за переміщення, обробку отриманого ушкодження, стан "живий/мертвий", а також взаємодію з іншими об’єктами. Клас Player реалізований через патерн Singleton, що гарантує наявність лише єдиного екземпляра гравця в межах сцени.

Переміщення персонажа реалізовано за допомогою компонента Rigidbody2D, через метод MovePosition, що забезпечує плавний та фізично коректний рух. Напрямок руху зчитується із клавіатури за допомогою InputAction, налаштованих через систему InputSystem. Крім того, реалізовано перевірку мінімальної швидкості, яка дозволяє визначити, чи персонаж перебуває у стані бігу (IsRunning). Це значення необхідне для синхронізації з візуальним відображенням анімації бігу.

Додатково в класі Player передбачено обробку ушкоджень. При влучанні актором по герою відбувається зниження значення рівня здоров’я. Якщо значення дорівнює нулю, викликається метод смерті, який фіксує цей стан, зупиняє рух персонажа і надсилає сигнал іншим об’єктам за допомогою події OnPlayerDeath.

Щоб зрозуміти, чи дійсно герой отримав пошкодження, впроваджено механізм "мерехтіння" персонажа. Це реалізується в класі PlayerVisual (див.

Додаток Д) через короткочасне вимикання та ввімкнення компонента SpriteRenderer, що створює ефект блиму.

Для запобігання надмірному зниженню рівня здоров’я протягом короткого часу, також внесено змінну _canTakeDamage. Вона дозволяє впровадити проміжок відновлення між отриманням ушкоджень, який реалізується за допомогою корутини DamageRecoveryRoutine.

Обертання персонажа в сторону миші дозволяє клас PlayerVisual, який аналізує координати миші на екрані та змінює орієнтацію саме спрайту відповідно до положення користувацького курсора. Завдяки цьому забезпечується логічна та наочна поведінка героя, яка відповідає очікуванням користувача.

У підсумку, система управління в межах розроблюваного ПЗ є одночасно простою та функціонально повною. Вона враховує вікові особливості цільової аудиторії, забезпечує чітке зворотне повідомлення, а також дозволяє легко масштабувати логіку з урахуванням майбутніх ускладнень або додаткових механік.

2.4 Визначення способів анімаційного супроводу станів

Анімації це ключовий елемент візуального сприйняття персонажа в інтерактивному середовищі програмного забезпечення. Його основне призначення полягає у наданні зворотного зв’язку користувачу, що дозволяє легко розпізнавати дії, емоційний стан та ситуацію, у якій перебуває головний герой. У межах цього проекту передбачено реалізацію системи анімацій, яка реагує на зміну станів персонажа, а саме: переміщення, переслідування, атака, отримання ушкодження та смерть.

Анімації реалізуються з використанням системи аніматора Unity, що базується на створенні станів та переходів між ними у вигляді графу. Для головного персонажа створено анімаційний контролер, що містить кілька основних станів: "Idle" (очікування/спокій), "Run" (біг), "Attack" (атака), "TakeHit" (отримання

ушкодження) та "Die" (смерть). Кожен з цих станів супроводжується відповідною по кадровою 2D анімацією, створеної на основі спрайтів.

Перехід між анімаціями керується параметрами, що задаються у відповідному скрипті впродовж виконання гри. Наприклад, логічний параметр IsRunning активується в разі виявлення переміщення героя та ініціює анімацію бігу. У разі відсутності руху відбувається повертається стану анімації до спокою. Аналогічно, при натисканні на клавішу лівою кнопки миші активується тригер Attack, який викликає відповідну анімацію атаки.

Особливої уваги заслуговує реалізація анімації смерті. Після зменшення рівня здоров’я героя до нуля запускається тригер IsDie, що блокує подальші дії та відтворює відповідну анімацію. Після завершення цієї анімації система блокує можливість керування персонажем.

Задля кращого візуального сприйняття та розуміння, що головний об'єкт дійсно отримав ушкодження від ворожого актора, реалізовано ефект мерехтіння спрайта головного героя у разі отримання урону. Цей ефект створено через тимчасове вимкнення та ввімкнення спрайта з використанням корутини, що дозволяє гравцю чітко усвідомити момент нанесення ушкодження.

Таким чином, анімаційний супровід у розробленому програмному продукті виконує не лише естетичну, а й функціональну роль, чим в свою чергу сприяє кращій взаємодії користувача з ігровим середовищем та забезпечує інтуїтивне розуміння дій головного героя.

2.5 Визначення елементів поведінки та системи бою

Динамічна та ефективна реалізація системи бою є ключовим елементом ігрового процесу, що дозволяє сформувати у школярів відчуття змагальності, виклику та залученості. У контексті навчального застосунку для молодших школярів, бойова система має залишатися інтуїтивно зрозумілою, візуально виразною, але при цьому достатньо динамічною, щоб стимулювати інтерес до подальшого проходження гри.

Одним із базових компонентів бойової взаємодії виступає проектування поведінки акторів. У розробленому проекті реалізовано актора типу "Пил", що втілює образ опору у вигляді умовного фізичного забрудника. Пил здійснює патрулювання по карті, реагує на наближення гравця, розпізнає його в межах заданої дистанції та розпочинає переслідування.

При достатньому наближенні Пил атакує гравця, що призводить до втрати частини очок здоров’я. В основі логіки його поведінки реалізовано машинний автомат станів, що передбачає перемикання між станами: спокій (Idle), бродіння (Roaming), переслідування (Chasing), атака (Attacking) та смерть (Death). Цей підхід дозволяє гнучко реагувати на дії гравця, забезпечуючи варіативність поведінки актора без необхідності складних алгоритмів штучного інтелекту.

У подальших етапах розробки планується реалізація додаткового типу актора — "Вода". Цей супротивник матиме відмінні поведінкові характеристики, зокрема, буде здійснювати дистанційні атаки у вигляді водяних кульок. Такий актор виконуватиме роль складнішого противника, що вимагатиме від гравця прийняття нових та швидких тактичних рішень. Поведінка Води також буде ґрунтуватися на автоматі станів, однак з розширеним набором дій: вибір напрямку стрільби, очікування між залпами, уникнення наближення до гравця тощо.

Взаємодія між гравцем і акторами реалізується за допомогою системи колайдерів та обробки подій зіткнення. У випадку атаки гравця актор отримує пошкодження, яке зменшує його поточний рівень здоров’я. При досягненні нульового значення HP актор змінює свій стан на смерть, відтворюється відповідна анімація, після чого об'єкт актора поступово зникає зі сцени, тому що пил в приладах ніхто не залишає.

Для розуміння, що актор дійсно отримав пошкодження, було реалізовано анімацію отримання удару, а також реакція на смерть шляхом відтворення спеціального ефекту зникнення. Такий підхід не тільки забезпечує ігрову логіку, але й сприяє формуванню у користувача уявлення про причинно-наслідкові зв’язки між діями та результатами і в свою чергу виробіток дофаміну,, від виконаного завдання.

2.6 Визначення основних етапів моделювання станів та анімаційної поведінки

У контексті створення інтерактивного навчального програмного забезпечення не менш важливою складовою є моделювання поведінки віртуальних супротивників. З цією метою у проекті передбачено проектування системи станів акторів на основі принципів скінченних автоматів, а також створення відповідних анімацій для кожного з можливих станів.

Система поведінки акторів базується на зміні таких основних станів: бродіння (Roaming), переслідування головного героя (Chasing), атака (Attacking) отримання урону (TakeHit) та смерть (Death). Логіка переходів між станами залежить від поточної ситуації в грі — зокрема, від положення головного персонажа, відстані до нього, наявності у гравця здоров’я тощо. Наприклад, актор, який не бачить гравця, знаходиться у стані бродіння. Щойно гравець з’являється в полі зору актора, активується стан переслідування, а при досягненні допустимої дистанції — атака.

У процесі проектування було визначено ключові параметри, що регулюють зміну станів, зокрема: радіус переслідування, дистанція атаки, швидкість руху актора у різних режимах тощо. Для збереження структурованості та керованості поведінки акторів усі ці параметри було інкапсульовано в окремому класі EnemyAI (див. Додаток К), який виконує функції контролера станів. Цей клас реалізує поведінкову логіку через автомат станів Finite State Machine, де кожен стан — бродіння, переслідування, атака, смерть — обробляється окремим блоком умов і дій. Такий підхід забезпечує прозорість внутрішньої логіки та полегшує налагодження на етапі розробки.

Застосування подібної архітектури також сприяє масштабованості системи, дозволяючи легко додавати нові типи акторів із власними сценаріями дій без необхідності повністю змінювати вже реалізовану логіку. Завдяки цьому у майбутньому можлива інтеграція додаткових негативних чинників зі специфічною

поведінкою, що значно підвищує потенціал розширення програмного забезпечення.

Особлива увага приділяється проектуванню анімаційного супроводу. Для кожного зі станів актора передбачено відповідну анімацію: анімація покою, швидкого переслідування, атаки, а також окремі анімації для отримання урону та смерті. Перемикання анімацій у середовищі Unity здійснюється за допомогою параметрів аніматора, таких як IsRunning, Attack, TakeHit, IsDie. Їх зміна ініціюється подіями, що виникають у відповідь на зміну стану або взаємодію з об’єктами оточення, що дозволяє синхронізувати логіку поведінки з візуальним відображенням процесів у грі.

На рівні концептуального проектування було передбачено наявність щонайменше двох типів ворожих акторів: Пил — супротивник ближнього бою, та Вода — ворожий актор дальнього бою, що здійснює атаку водяними снарядами. Для кожного з них створено окрему модель поведінки та індивідуальний набір анімацій, адаптованих до специфіки їх атакуючих дій.

Це дозволяє забезпечити гнучкість системи, підвищити різноманітність ігрових ситуацій, а також створити більш захопливу динаміку боротьби. Завдяки різниці в тактиці поведінки ворожих акторів, головний об'єкт має змогу адаптувати свою стратегію залежно від типу противника, що також сприяє розвитку логічного мислення у цільової аудиторії.

Таким чином, проектування системи переходів станів і супровідних анімацій акторів відіграє ключову роль у формуванні ігрової динаміки. Чітка структура автомату станів у поєднанні з активним візуальним супроводом сприяє створенню цілісного інтерактивного середовища, що не лише привертає увагу учнів молодшого шкільного віку, а й мотивує до подальшого засвоєння матеріалу.

2.7 Визначення засобів взаємодії

Важливим елементом бойової системи програмного забезпечення є зброя, за допомогою якої головний герой може протистояти акторам. У контексті

розроблюваного програмного забезпечення, що має на меті навчання основам інформатики для молодших школярів, особлива увага приділяється адаптації ігрових механік до відповідного віку цільової аудиторії. Саме тому було замінено традиційну зброю (наприклад, меч) на перо, яке виконує функцію ударної зброї.

Цей підхід має на меті сформувати у школяра асоціації, пов’язані з чистотою та порядком, оскільки перо слугує засобом для "прибирання" акторів, таких як Пил. Таким чином, з одного боку, уникається демонстрація жорстоких бойових дій, що є неприйнятним для дитячої аудиторії, а з іншого — створюється позитивне образне сприйняття дій головного об'єкта. Крім того, такий образ зброї добре вписується у загальну тематику програмного забезпечення, що також підтримує екологічну тематику.

З технічного боку, механіка роботи пера реалізована через систему тригерів і колайдерів. При натисканні гравцем лівої кнопки миші активується відповідна подія атаки, яка запускає метод Attack() поточного активного об’єкта зброї. У момент удару активується полігональний колайдер, який перевіряє наявність актора в зоні ураження. Якщо відбувається зіткнення з об’єктом, що позначений як актор, той отримує шкоду згідно з алгоритмом обробки урону.

Оскільки атака — це не миттєва дія, а частина анімації взмаху, перо вмикає колайдер лише у певний проміжок часу, що відповідає візуальному анімаційному руху зброї. Це досягається за допомогою анімаційних подій, які вказують точні моменти активації й деактивації колайдера пера. Такий підхід дозволяє досягти візуально та логічно узгодженої бойової механіки, а також запобігти випадковому нанесенню шкоди акторам поза межами активного бою.

Таким чином, у програмному забезпеченні реалізовано простий, але ефективний механізм бою, який дозволяє дитині отримати базове уявлення про логіку подій та взаємодію об’єктів, водночас підтримуючи легку й доброзичливу атмосферу програмного забезпечення. Такий підхід сприяє інтеграції освітніх елементів до геймплейного процесу.

2.8 Визначення способів всіх варіантів взаємодії

У структурі навчальної гри важливою складовою є механізм взаємодії між головним героєм та ворожими об’єктами. Цей елемент не лише додає динамічності геймплею, а й демонструє логіку та обробки зіткнень в ігровому середовищі.

Основна логіка взаємодії полягає у перевірці на зіткнення між колайдером атаки головного героя зброєю-пером та тілом актора. Кожен актор у грі має змінну, що зберігає його поточний рівень здоров’я. У разі успішного удару по актору відповідний метод зменшує кількість його здоров’я на визначену кількість одиниць.

Ворожі актори Пил та Вода мають різну модель взаємодії при отриманні урону. Так, ворожий актор Пил реагує на атаку головного героя отриманням шкоди, а при зменшенні здоров’я до нуля переходить у стан смерті, який супроводжується відповідною анімацією та подальшим зникненням об'єкта. Водночас ворожий актор Вода матиме ускладнену логіку поведінки, зокрема, контратаку у вигляді запуску водяних кульок, що вимагає додаткових перевірок на зіткнення з тілом героя.

Щоб унеможливити одночасне багаторазове нанесення шкоди при одному ударі, реалізовано механізм короткочасного включення колайдера лише на період активної фази анімації атаки. Це дозволяє зменшити кількість хибних зіткнень і підвищити точність відображення бойових подій.

У свою чергу, герой також може отримувати шкоду в наслідок атаки ворожих акторів. При зіткненні актора з тілом персонажа викликається метод TakeDamage(), що активує анімацію відкидання та короткочасного мерехтіння героя. Для уникнення повторного урону впродовж певного часу, реалізовано таймер, протягом якого герой не може повторно отримати шкоду.

Таким чином, система взаємодії між атакою гравця і ворожими акторами побудована з урахуванням базових принципів обробки подій, анімаційних станів і фізичних зіткнень у 2D-середовищі Unity. Такий підхід забезпечує інтерактивність, логічність бойових ситуацій.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ НАВЧАННЯ ОСНОВ ІНФОРМАТИКИ

3.1 Розробка головного меню

Головне меню є першим екраном, з яким взаємодіє користувач після запуску гри. Воно виконує роль навігаційного хабу, забезпечуючи доступ до ключових режимів і функцій. Оскільки програмне забезпечення орієнтоване на школярів молодшого віку, важливою вимогою до меню стала його простота, інтуїтивна зрозумілість та візуальна привабливість.

Зовнішній вигляд головного меню (Рисунок 3.1) витримано в піксельному стилі, відповідно до загальної концепції гри та стилізовано під тематику мікросхем та цифрового середовища. Такий підхід візуально поєднує елементи інтерфейсу з ігровим процесом та сприяє кращому зануренню школяра в навчальний контекст, який асоціюється з основами інформатики.

Рисунок 3.1 — Головне меню

Меню реалізовано у вигляді окремої сцени, де на тлі стилізованої текстури материнської плати розташовано дві основні інтерактивні кнопки:

— "Play" перенаправляє на наступну сторінку меню вибору тематики (Рисунок 3.2), де на вибір є поки 2 концепції "Enemy" та "Safety rules" де вже учень обирає тему, з якою бажає ознайомитися через механіку взаємодії з віртуальним середовищем;

— "Exit" завершує роботу програми.

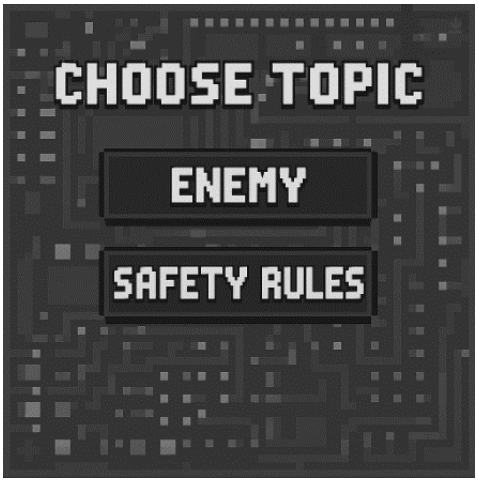

Рисунок 3.2 — Меню вибору тематики

При натисканні на інтерактивні кнопки, реалізовані в інтерфейсі на Рисунку 3.3, здійснюється навігація між ключовими ігровими сценами:

— при натисканні на кнопку "Enemy" здійснюється перехід до сцени вибору ворожого чинника, де учень має змогу обрати один із типів загроз: "Пил" або "Воду".

— при натисканні на "Safety Rules" відкривається сцена, присвячена ознайомленню із правилами техніки безпеки та поводження з технікою, що сприяє формуванню у молодших школярів відповідального ставлення до комп’ютерної безпеки. На даний момент дана сцена знаходиться на стадії розробки.

Рисунок 3.3 — Меню вибору рівня з негативним чинником

Розміщення елементів здійснювалося за допомогою системи Canvas у Unity. Кожна кнопка реалізована як об’єкт UI Button, до якого прикріплені події для обробки натискань. Стилізація кнопок — включно з фоном, шрифтами та межами — виконана з імітацією старих відеоігор. Це дозволяє не лише підтримувати загальний художній стиль, але й зберігати читабельність навіть для користувачів з низьким рівнем комп’ютерної підготовки.

Важливим елементом також є розміщення кнопок з достатнім інтервалом одна від одної для уникнення помилкових натискань. Розробка передбачає, що в майбутньому в меню можуть бути додані додаткові розділи (наприклад, інструкції, налаштування або інформація про розділи навчання), однак базова функціональність була реалізована першочергово.

3.2 Реалізація сцени рівнів та навігації

Одним із ключових етапів під час розробки ігрового середовища стало створення сцени, де відбувається безпосередня взаємодія користувача з навчальним контентом у формі гри. Для цього була розроблена ігрова карта та додано плату (Рисунок 3.4) [13], на якій персонаж може вільно переміщуватись, досліджуючи територію, взаємодіяти з об'єктами, уникаючи небезпек або вступаючи у взаємодію з акторами.

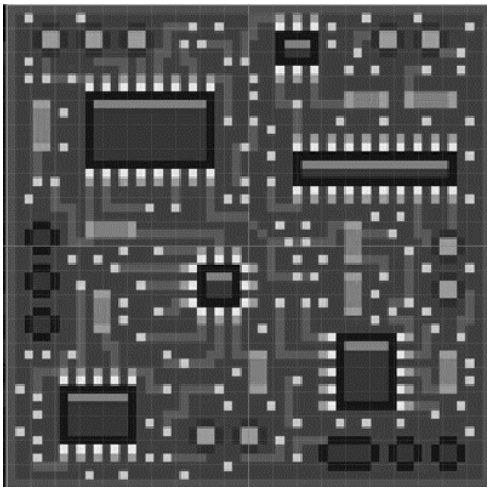

Рисунок 3.4 — Площина для руху

Щоб забезпечити коректний рух персонажів сценою, у проекті використано інструмент NavMesh [12] — навігаційну сітку, яка дозволяє визначити прохідні області рівня, як показано на Рисунку 3.5. Цей інструмент є стандартною частиною середовища розробки Unity і широко використовується для створення поведінки штучного інтелекту та переміщення об'єктів.

Поверхня мапи гри складається з піксельного тайлу, на який накладено NavMeshSurface — компонент, що дозволяє згенерувати сітку навігації. Поверхня, по якій може ходити гравець та актори, була окреслена вручну за допомогою Polygon Collider 2D, який відображає межі доступної території. Саме ці межі були використані як орієнтир при створенні навігаційної сітки.

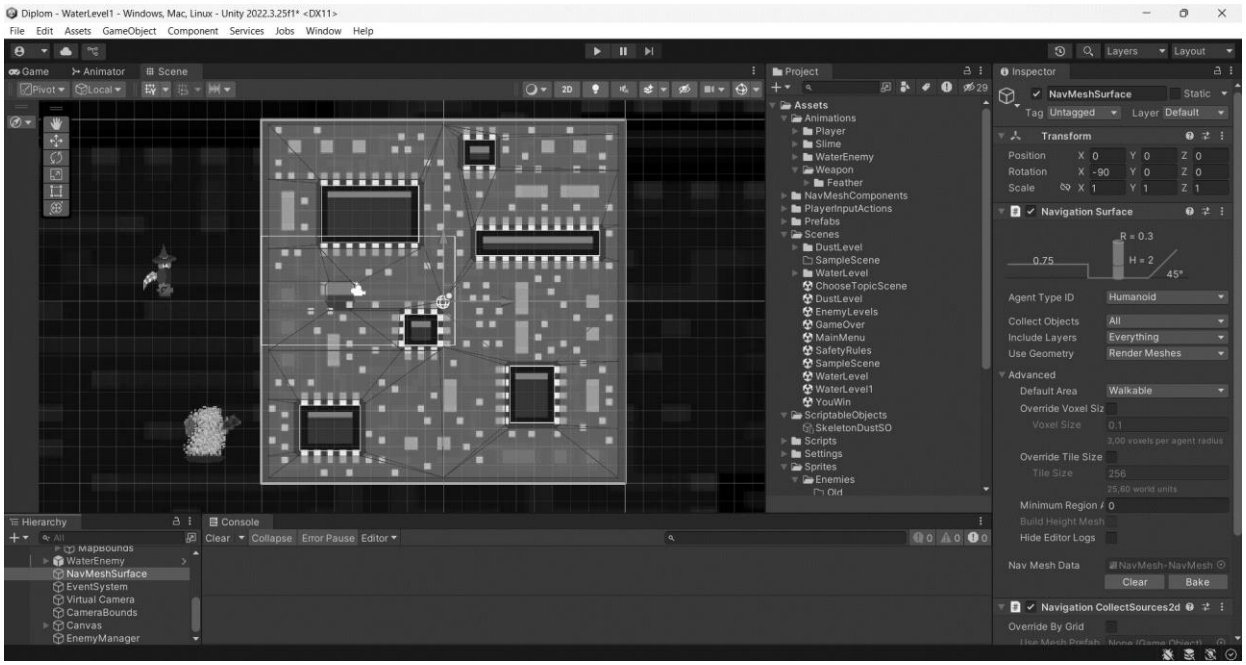

Рисунок 3.5 — NavMesh

Після конфігурації Polygon Collider 2D його контури були використані в якості обмежувача для камери Cinemachine Confiner 2D, завдяки чому камера слідує тільки за персонажем.

Для реалізації переміщення акторів по сітці використовується компонент NavMeshAgent, який автоматично обчислює шлях до цілі та враховує обмеження, задані NavMesh. Це дозволяє акторам переслідувати гравця або патрулювати територію без зіткнень із непрохідними об’єктами або ще шукати шляхи

досягнення до точки призначення та обходити фізичні об’єкти. Така реалізація пересування акторів по карті робить їх більш непередбачуваними та "живими", що в свою чергу надає динамічності та жвавості процесу дослідження.

Головний герой, навпаки, керується вручну за допомогою отримання сигналів з клавіатури, але логіка його руху враховує ті самі фізичні обмеження мапи, що й для акторів.

Такий підхід дозволяє забезпечити плавне та реалістичне пересування персонажів у межах ігрової сцени, одночасно обмеживши їхні дії лише передбаченою та виділеною територією. Використання NavMesh у 2D-проекті є доволі нетиповим, проте в даному випадку це дозволило ефективно керувати логікою пересування акторів та забезпечити контроль за межами карти.

3.3 Візуалізація та керування головним об'єктом

Одним з ключових компонентів програмного забезпечення є головний ігровий персонаж, із яким взаємодіє гравець протягом усього ігрового процесу. Реалізація цього об’єкта потребувала створення зручної системи управління, візуалізації та логіки взаємодії з середовищем, включаючи анімацію, пошкодження, атаку та реакцію на зовнішні події.

У Unity головний персонаж представлений об’єктом Player, який зображений на Рисунку 3.6 [14]. Герой розміщується у сцені та містить ряд компонентів: Rigidbody2D, Animator, Collider2D, KnockBack, а також скрипти Player.cs і PlayerVisual.cs. (див. Додатки Г та Д).

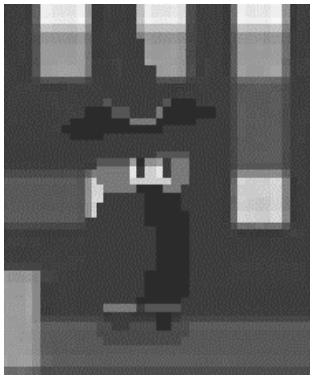

Рисунок 3.6 — Головний герой

3.3.1 Реалізація керування рухом

Рух головного героя є ключовим компонентом інтерактивної частини гри, адже забезпечує гравцеві можливість дослідження віртуального середовища та взаємодії з його об'єктами. Управління персонажем реалізоване з використанням вхідної системи Unity Input System (Рисунок 3.7), що дозволяє зчитувати натискання клавіш та координати миші.

Основна логіка переміщення сконцентрована у скрипті Player.cs, де реалізовано обробку руху, орієнтацію персонажа у напрямку вказівника миші та обмеження щодо переміщення під час станів отримання шкоди чи смерті.

Завдяки такій реалізації досягається плавність управління та висока відповідність дій гравця очікуваному результату, що особливо важливо для забезпечення комфортного використання програмного забезпечення учнями молодшого шкільного віку.

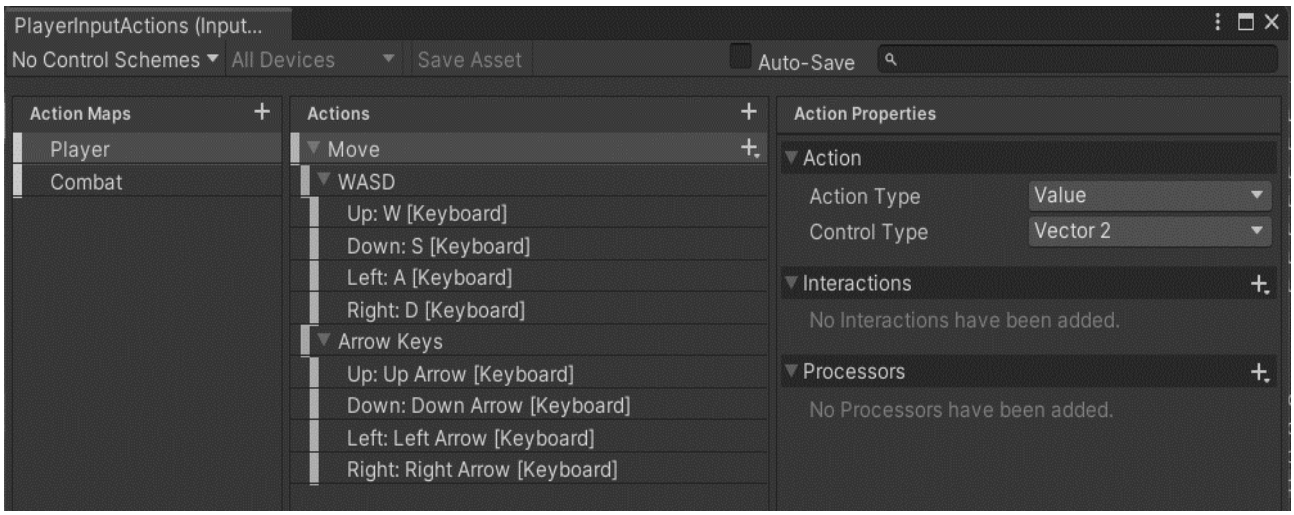

Рисунок 3.7 — Система InputSystem зчитування рухів героя

У цьому скрипті за допомогою компоненту Rigidbody2D задається фізичне переміщення головного об'єкту. Зчитування сигналів з клавіатури відбувається через систему введення, після чого формується вектор напрямку руху, який передається до компонента фізики.

Рух здійснюється відповідно до вхідного вектора, що задається гравцем за допомогою клавіш управління. Для досягнення однакової швидкості незалежно від напрямку, в тому числі при діагональному русі, вектор нормалізується, тобто його довжина приводиться до одиниці. Це дозволяє забезпечити плавне, природнє пересування персонажа у двох-вимірному середовищі програмного забезпечення, зберігаючи контрольованість рухів у будь-якому напрямку.

Додатково реалізовано логіку перевірки стану руху головного об'єкта, що дозволяє визначати його поточний стан. Якщо інтенсивність руху перевищує встановлений поріг, персонаж вважається таким, що перебуває у стані бігу і автоматично переходить у цей стан. У подальшому така логіка впливає на візуалізацію анімацій. На Рисунку 3.8 зображена анімація головного об'єкта у стані бігу.

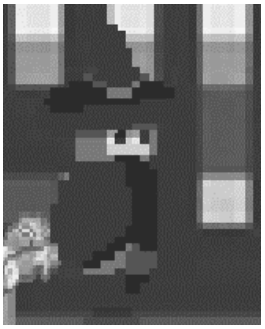

Рисунок 3.8 — Анімація бігу головного об'єкта

Контроль переміщення вмикається автоматично на старті запуску рівня. У випадку смерті персонажа (тобто коли рівень здоров’я дорівнює нулю), система автоматично блокує подальше зчитування рухів. Це реалізовано шляхом виклику методу DisableMovement() зі скрипту GameInput.cs (див. Додаток Е), який повністю деактивує контроль введення.

Такий підхід гарантує логічну завершеність ігрової сесії, не допускаючи взаємодії з оточенням після настання умови поразки, а також покращує загальну цілісність сценарію.

3.3.2 Система анімацій

У проекті була реалізована система анімацій, яка дозволяє наочно відображати стани героя — такі як спокій, біг, атака, отримання шкоди та смерть (Рисунок 3.9). Завдяки цьому гравець краще орієнтується в ситуації та може інтуїтивно розуміти, коли персонаж виконує певну дію або зазнає впливу з боку оточення.

Кожна анімація пов’язана з відповідним станом у логіці головного об'єкта та автоматично активується через параметри Animator. Такий підхід не лише покращує візуальну привабливість гри, а й сприяє кращому сприйняттю динаміки подій, що особливо важливо для молодшої аудиторії.

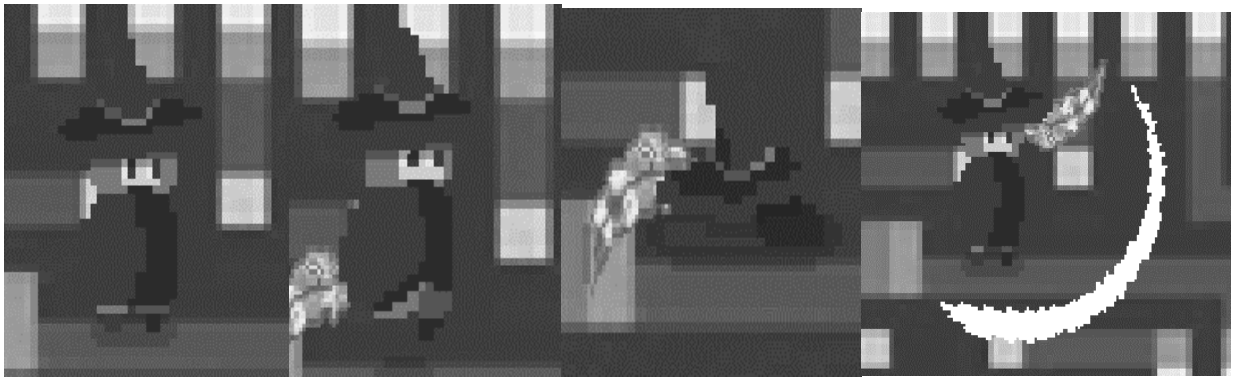

Рисунок 3.9 — Стани анімацій героя (Спокій, Біг, Смерть, Атака)

Основна логіка керування анімаціями персонажа реалізована в окремому скрипті PlayerVisual.cs (див. Додаток Д). У цьому скрипті налаштовано взаємодію з параметрами компонентів аніматора головного об'єкта, які були попередньо створені у вікні Animator. Вказані параметри дозволяють систематизовано, наочно та доступно керувати візуальними відображеннями змін станів персонажа в реальному часі.

До таких параметрів належать логічні змінні IsRunning та IsDie, які активують відповідні анімації бігу та знищення головного об’єкта. Візуальне представлення цих параметрів у середовищі Unity наведено на Рисунку 3.10 в меню Animator.

Завдяки цьому підходу досягається узгодженість між ігровою логікою та візуальною складовою, що дозволяє інтуїтивно та без ускладнень розуміти, у якому стані перебуває головний об'єкт у кожен момент проходження рівня.

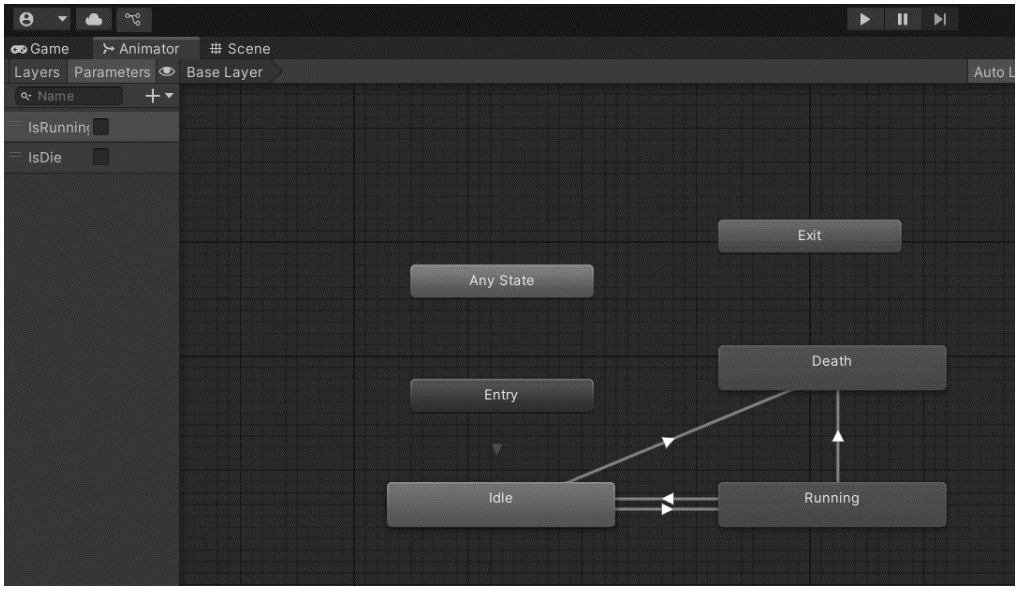

Рисунок 3.10 – Animator Player

Реакція на отримання шкоди супроводжується анімацією мерехтіння, яка реалізована за допомогою спеціальної корутини. Візуальний ефект досягається шляхом тимчасового вимикання й повторного вмикання компонента SpriteRenderer. Завдяки цьому персонаж ніби "блимає", що слугує чітким візуальним сигналом для гравця про те, що герой зазнав удару.

Упродовж цього періоду головний герой не може отримувати додаткову шкоду, що запобігає випадковому переудару від одного й того ж ворожого актора. Це дозволяє створити більш передбачувану і керовану ігрову взаємодію, підвищуючи якість геймплею. Вся логіка мерехтіння після шкоди реалізована в скрипті PlayerVisual.cs.

Анімація смерті активується у відповідь на подію, яка генерується з основного скрипту героя — Player.cs — після того, як лічильник здоров’я досягає нуля. Внаслідок цього візуальна складова героя переходить у відповідний стан, і у вікні Animator програється анімація загибелі героя.

Для візуалізації руху персонажа було використано параметр IsRunning, значення якого змінюється залежно від того, чи перебуває герой в стані руху. Його значення визначається на основі аналізу вхідного вектору руху, отриманого через систему обробки введення сигналів з пристроїв вводу інформації, реалізовану у скрипті GameInput.cs.

Також для повороту героя у бік курсора миші реалізовано логіку, щоб забезпечити відповідність між напрямком атаки та орієнтацією персонажа у просторі на карті. Цей поворот не впливає на фізику руху, а здійснюється лише за допомогою дзеркального відображення спрайта (flipX) відносно вертикальної осі. Таким чином, система анімацій забезпечує плавний та інтуїтивно зрозумілий зворотний зв’язок гравця із головним героєм.

3.3.3 Тестування компоненту "Profile"

Однією з важливих складових ігрового процесу є реалізація механізму атаки головного героя. У поточній версії програмного забезпечення головний об'єкт має змогу атакувати ворожих акторів за допомогою зброї — пера [15], яке виконує функцію інструмента взаємодії та завдання шкоди.

Атака активується натисканням лівої кнопки миші, що обробляється через систему вводу у скрипті GameInput.cs. Після зчитування дії користувача, викликається подія атаки, яка передається в систему активної зброї, реалізовану у ActiveWeapon.cs (див. Додаток Н). Саме через цей скрипт відбувається звернення до конкретного об’єкта зброї, що має метод атаки.

Перо як зброя має власний колайдер (Рисунок 3.11), який під час анімації атаки активується на обмежений проміжок часу. За активацію колайдера відповідає тригер атаки, який спрацьовує при натисканні лівої кнопки миші. Це зроблено для того, щоб колайдер не був активний на постійній основі.

Якщо в цей момент відбувається зіткнення з об’єктом актора, зокрема з його колайдером, то ворожий актор отримує шкоду. Відповідна логіка обробляється в скрипті EnemyEntity.cs (див. Додаток К), де визначено метод TakeDamage(). У

ньому здоров’я ворожого актора зменшується, а також генерується подія взаємодії об'єктів отримання шкоди.

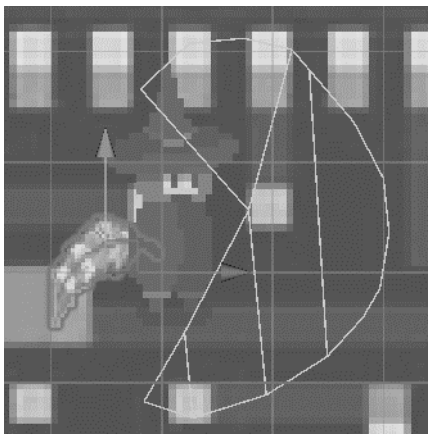

Рисунок 3.11 — Колайдер для атаки пера

Для гнучкості та зручності подальшого розширення бойової системи реалізовано окремий контролер ActiveWeapon, який зберігає поточну активну зброю й дозволяє в майбутньому змінювати типи зброї без потреби модифікації базової логіки атаки. Це відкриває можливості для розвитку та розширення програмного забезпечення.

Таким чином, у програмному забезпеченні реалізовано простий, але ефективний механізм бою, який дозволяє дитині отримати базове уявлення про логіку подій, взаємодію об’єктів, синхронізацію дій з анімацією, а також про структуру бойової механіки.

Водночас реалізація засобів бою у формі доброзичливого і метафоричного інструмента (пера) підтримує легку та позитивну атмосферу програмного забезпечення. Такий підхід не лише підвищує інтерес до процесу, а й сприяє формуванню у дитини важливих ціннісних орієнтирів — наприклад, що «чистота — це зброя», а турбота про оточення є важливою частиною поведінки.

3.4 Реалізація негативних чинників

У програмному забезпеченні реалізовано двох основних типів ворожих акторів — Пил та Воду. Кожен з них має власні особливості поведінки, атаки та

візуального супроводу, що створює різноманітність у геймплеї та сприяє кращому засвоєнню навчального матеріалу в інтерактивній формі.

Ворожого актора Пил реалізовано як ближнього противника, який переміщується по ігровому середовищу, виявляє гравця на певній відстані та починає його переслідувати. За умови досягнення критичної дистанції, ворожий актор переходить у стан атаки. Основна атака Пилу виконується за допомогою удару мечем. Логіка його поведінки реалізована у скрипті EnemyAI.cs (див. Додаток И), де визначено машинну модель станів: очікування, блукання, переслідування, атака та смерть. Перемикання між станами залежить від дистанції до гравця, яка визначається через методи Vector3.Distance().

Анімації руху та атаки ворожого актора Пил [16] [17] супроводжуються активацією колайдера меча, реалізованого через методи PolygonColliderTurnOn() та PolygonColliderTurnOff() у скрипті EnemyEntity.cs. Після отримання достатньої кількості шкоди ворожий актор переходить у стан смерті — його колайдери деактивуються, відтворюється відповідна анімація та зрештою об'єкт зникає зі сцени. Зображення ворожого актора "Пил" представлено на Рисунку 3.12.

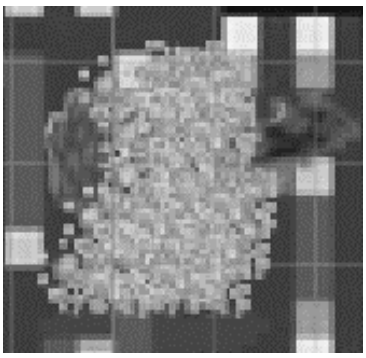

Рисунок 3.12 — Ворожий актор "Пил"

Ворожий актор Вода [18] має іншу механіку атаки: він атакує дистанційно, запускаючи у гравця водяні кульки. Після виявлення героя на відстані, ворожий актор зупиняється та починає стріляти снарядами з інтервалом у часі. Візуальна складова забезпечується відповідними анімаціями атаки та запуску снарядів. Самі

водяні кульки є окремими об’єктами з колайдерами, що перевіряють зіткнення з гравцем. Зображення ворожого актора "Вода" показано на Рисунку 3.13.

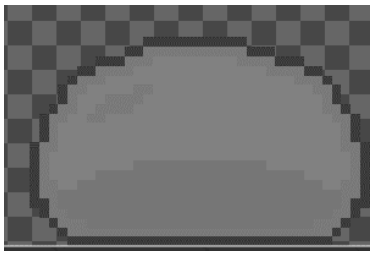

Рисунок 3.13 — Ворожий актор "Вода"

Механізм запуску снарядів реалізовано у скрипті EnemyWaterAttack.cs. Ворожі актори типу Вода мають власний набір станів, що частково перетинаються з Пилом, але доповнені логікою відстрілу та візуалізацією снарядів.

Усі ворожі актори мають змінну кількість здоров’я, що задається у відповідному параметрі. Шкода від гравця наноситься через метод TakeDamage(), реалізований у EnemyEntity.cs, що реагує на зіткнення зброї з тілом ворожого актора.

Таким чином, реалізація обох типів ворожих акторів забезпечує різні сценарії взаємодії, дозволяє урізноманітнити рівні гри та створити повноцінну навчальну атмосферу з гейміфікованими елементами.

3.5 Реалізація взаємодії та отримання шкоди

Система бою в грі реалізована як сукупність взаємодій між гравцем та ворожими акторами: гравець може завдавати шкоди ворожим акторам, а ворожі актори — відповідати атакою, у разі якщо перебувають на відповідній відстані. Таким чином, бойова система є двосторонньою й динамічно змінюється відповідно до ситуації на полі бою.

Головний герой використовує зброю — перо, яким завдає шкоди ворожим акторам. При натисканні гравцем на ліву кнопку миші викликається подія атаки [19]. Ця подія передається через об’єкт ActiveWeapon, який зберігає поточну на даний момент зброю гравця. У скрипті зброї виконується запуск

анімації удару, а також активується відповідний колайдер, який відповідає за фізичне зіткнення з ворожим актором. На Рисунку 3.14 показані стани анімацій ворожого актора "Пил".

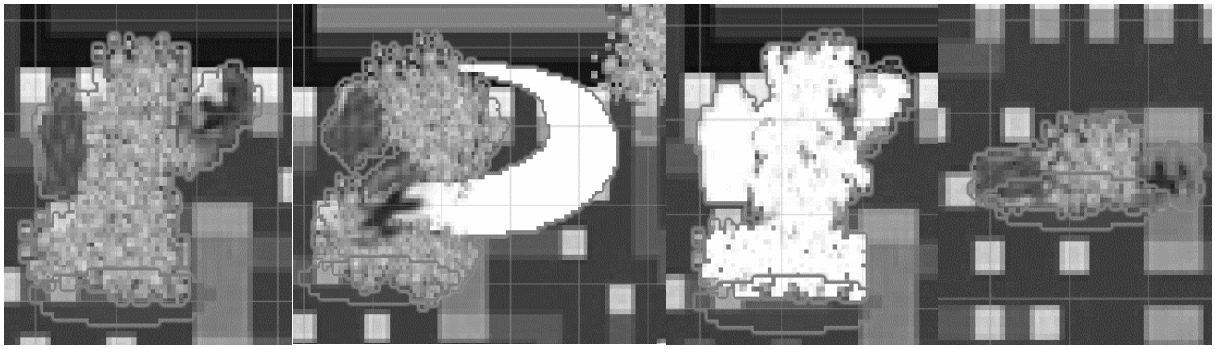

Рисунок 3.14 — Стани анімацій ворожого актора "Пил"

У разі контакту з ворожим актором, у скрипті ворожого актора EnemyEntity.cs спрацьовує метод отримання шкоди TakeDamage(). У ньому зменшується кількість здоров’я, відтворюється анімація отримання шкоди, а за умови досягнення нульового значення здоров’я — відбувається перехід у стан смерті та знищення об’єкта ворожого актора. На Рисунку 3.15 показані стани переходів анімацій ворожого актора.

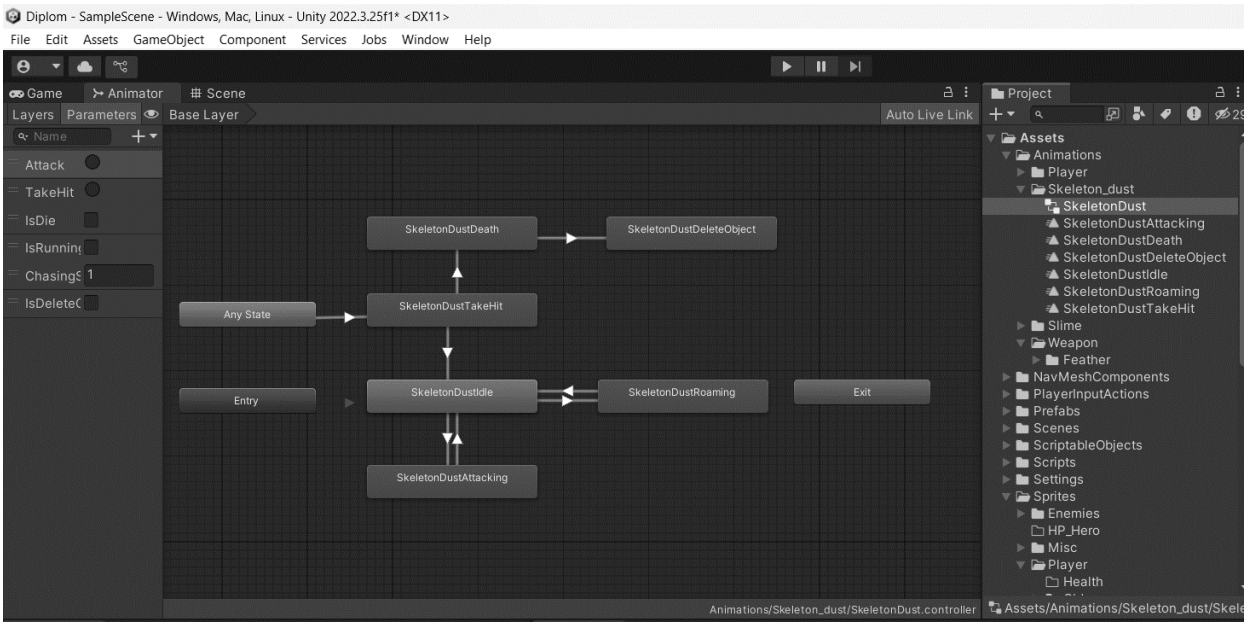

Рисунок 3.15 — Стани переходів анімацій ворожого актора

Ворожі актори також можуть завдати шкоди гравцеві. У актора "Пил" це реалізується через Polygon Collider 2D меча, який активується під час анімації атаки. У скрипті героя Player.cs реалізовано метод TakeDamage(), який приймає джерело атаки та величину шкоди. Після завдання шкоди гравець отримує ефект відкидання назад (knockback), який реалізовано у скрипті KnockBack.cs (див. Додаток Д). Це додає динаміки та наочності бою.

Щоб уникнути багатократного зчитування шкоди за один удар, реалізовано період невразливості героя. Протягом короткого проміжку часу після отримання шкоди герой не може отримати нову. Цей механізм реалізовано за допомогою корутини, яка встановлює затримку через WaitForSeconds() перед тим, як дозволити повторне нанесення шкоди.

Також у класі PlayerVisual.cs реалізовано візуальний ефект мерехтіння головного об'єкта під час отримання шкоди та зменшення рівня шкали здоров'я [20]. Це досягається завдяки циклічному вимиканню та вмиканню видимості спрайта протягом певного проміжку часу, що підвищує інформативність бою для гравця.

Обидві сторони бойового процесу — гравець і ворожі актори — мають змінну кількість здоров’я та анімації смерті. При досягненні нульового значення здоров’я, відтворюється відповідна анімація, після чого об'єкт або видаляється зі сцени, або блокується його взаємодія.

Таким чином, система бою реалізована як через візуальні ефекти, так і через фізичні компоненти як колайдери, дозволяючи створити цілісний ігровий досвід із чітким зворотним зв'язком на дії гравця.

У результаті реалізації вказаних етапів було створено повноцінний рівень, у межах якого головний об'єкт взаємодіє з ворогом типу "Пил". На цьому рівні відтворюється базова розроблена механіка: пересування, отримання шкоди та застосування атаки. Візуалізацію сцени рівня наведено на Рисунку 3.16.

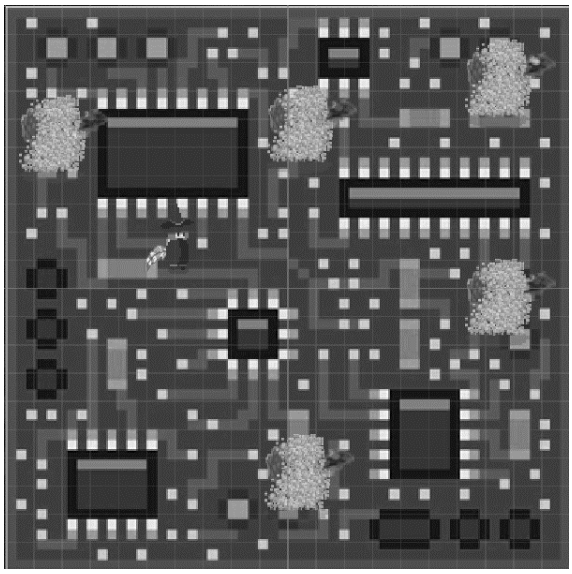

Рисунок 3.16 — Реалізований рівень "Пил"

3.6 Реалізація закінчення ресурсів та завершення програми

Смерть головного героя є важливою частиною ігрової логіки програмного забезпечення, оскільки вона сигналізує про завершення поточної ігрової сесії та формує відчуття відповідальності за дії навіть у віртуальному середовищі. Реалізація смерті гравця включає кілька взаємопов’язаних елементів: припинення руху, візуальне відображення смерті, блокування подальшої взаємодії та подальший перехід у фінальний стан.

Для виявлення смерті героя у скрипті Player.cs стан здоров’я контролюється змінною _currentHealth, що зменшується кожного разу, коли герой отримує шкоду. Після зменшення здоров’я до нуля викликається метод DetectDeath(), який перевіряє, чи герой ще живий. Якщо ні — встановлюється прапорець _isAlive = false, що означає припинення будь-якої активності персонажа.

Також блокується можливість повторного отримання шкоди або руху. У цьому ж методі викликається подія OnPlayerDeath, яка передається іншим компонентам, відповідальним за візуалізацію смерті.

У скрипті PlayerVisual.cs реалізована підписка на подію OnPlayerDeath, тобто смерті героя. Коли вона спрацьовує, в аніматорі змінюється булевий параметр IsDie, що активує відповідну анімацію загибелі героя. Завдяки цьому гравець

бачить, як персонаж падає або припиняє діяти, що підсилює ефект кінця гри (Рисунок 3.17).

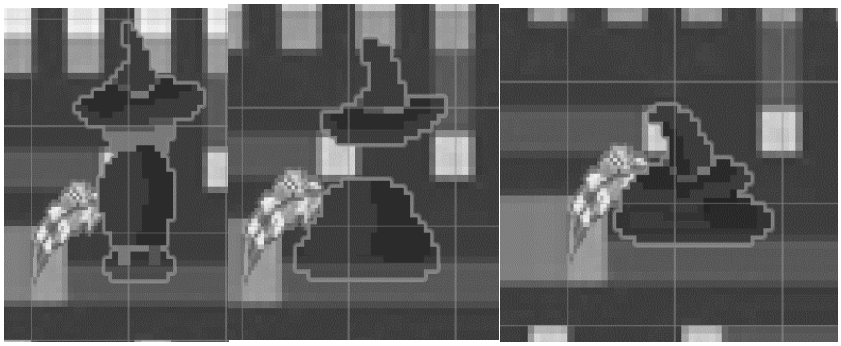

Рисунок 3.17 — Анімація закінчення життя героя

Після спрацювання механізму смерті викликається метод DisableMovement() зі скрипта GameInput.cs, що повністю блокує можливість управління героєм. Таким чином, користувач більше не має контролю над персонажем і не може взаємодіяти із середовищем. Це дозволяє уникнути непередбачуваної поведінки гри після завершення життєвого циклу персонажа та підкреслює фінальність ситуації в рамках поточної сесії.

Після загибелі головного героя внаслідок отримання критичної шкоди від ворожих акторів у грі відбувається автоматичний перехід до спеціального екрану завершення. Цей екран та сигналізує про завершення ігрової сесії, інформуючи гравця про поразку або перемогу (Рисунок 3.18).

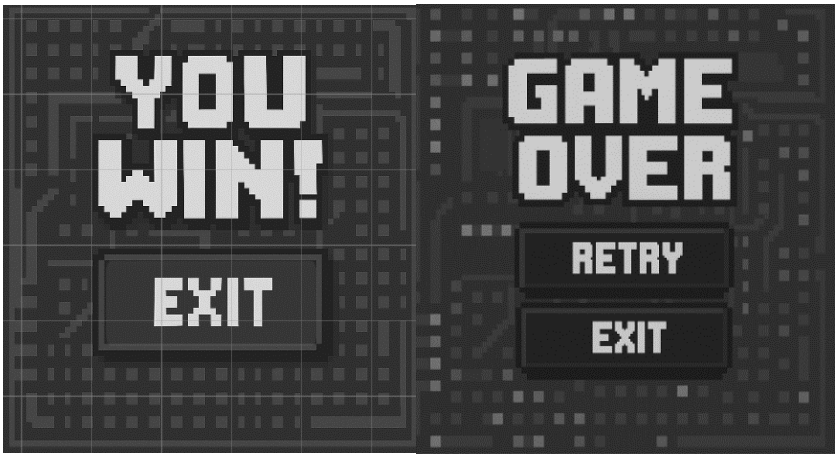

Рисунок 3.18 — Екран після завершення проходження рівня

3.7 Використання ScriptableObject для параметризації негативних чинників

У проекті, присвяченому розробці програмного забезпечення для навчання основам інформатики молодших школярів, важливу роль відіграє ефективне управління даними, пов’язаними з ігровими об’єктами. Одним із найзручніших і рекомендованих підходів у середовищі Unity є застосування архітектурного шаблону ScriptableObject, який дозволяє зберігати конфігураційні параметри окремо від логіки поведінки об'єкта.

У даному проекті ScriptableObject використовується для збереження та організації параметрів ворожих акторів. Такий підхід забезпечує централізоване редагування характеристик одночасно всіх ворожих акторів без потреби змінювати логіку у відповідних скриптах.

Для кожного типу ворожого актора створюються окремі екземпляри об’єкта на основі спільного шаблону, де задаються такі властивості, як:

— кількість очок здоров’я;

— швидкість пересування;

— дистанція виявлення гравця;

— параметри шкоди та атаки;

— час зникнення об'єкта після знищення;

— час відновлення між атаками тощо.

Реалізація побудована на базі скрипту EnemySO.cs (див. Додаток М), який описує набір змінних, що відповідають за поведінкові параметри ворожого актора. Ці об’єкти зберігаються у вигляді окремих файлів у папці ScriptableObjects і можуть легко бути повторно використані або змінені у редакторі Unity без перекомпіляції коду. Такий підхід знижує ймовірність помилок при оновленнях та дозволяє оперативно адаптувати баланс гри.

При ініціалізації ворожих акторів у грі, відповідні параметри підвантажуються безпосередньо з пов’язаного ScriptableObject, що робить систему модульною та легко розширюваною. Наприклад, додавання нового ворожого

актора не потребує створення нового скрипту — достатньо створити новий ScriptableObject на основі наявного шаблону та прикріпити його до префабу ворожого актора.

Подібна архітектура є ефективною не лише з точки зору повторного використання коду, а й з огляду на масштабованість гри у майбутньому, зокрема при плануванні додаткових типів ворожих акторів або складніших механік.

3.8 Тестування

Після реалізації основних функціональних компонентів гри було проведено тестування з метою перевірки працездатності програмного забезпечення та відповідності поставленим навчальним цілям. Тестування здійснювалося у формі спостереження за реальними представниками цільової аудиторії — учнями молодших класів, яким було запропоновано ознайомитись з ігровим процесом без попередніх інструкцій або навчання. Такий підхід дозволив об'єктивно оцінити інтуїтивність керування, доступність ігрової логіки та загальний емоційний відгук на ігровий контент.

У результаті взаємодії з ігровим середовищем було виявлено, що користувачі швидко адаптуються до запропонованої механіки управління. Рух головного персонажа, атака ворожих акторів, уникнення небезпеки та переміщення по локації були інтуїтивно зрозумілими навіть для дітей, які раніше не мали досвіду користування подібними цифровими продуктами. Такий рівень інтуїтивності інтерфейсу свідчить про високу юзабіліті розробленого програмного продукту, що особливо важливо при роботі з молодшою віковою категорією.

Окрему увагу під час тестування було приділено сприйняттю контенту та рівню засвоєння основних меседжів, закладених у структуру програмного забезпечення. Після проходження рівнів з ворожими акторами, які уособлюють

певні загрози для комп’ютерного обладнання, учні продемонстрували розуміння відповідних правил безпечної поведінки.

Зокрема, після взаємодії з актором Пилом діти правильно сформулювали думку про необхідність підтримання чистоти у комп’ютерному класі, своєчасного прибирання пилу та обережного поводження з обладнанням. Аналогічно, після рівня з актором Водою більшість учнів зазначили, що відкриті ємності з водою не можна розміщувати поряд із комп’ютерною технікою, і що вживання напоїв за комп’ютером є небезпечним.

Крім того, під час проходження гри було зафіксовано позитивну динаміку зацікавлення дітей як самим процесом, так і тематикою інформаційної безпеки. Учні охоче обговорювали побачене, ставили запитання та демонстрували прагнення до правильного поводження з технікою.

Таким чином, можна стверджувати, що програмний продукт виконує не лише розважальну, але й навчально-просвітницьку функцію. Після проходження рівнів дитина засвоює базові правила безпеки при роботі з комп’ютером, що повністю відповідає меті створення даного програмного забезпечення.

3.9 Підсумки реалізації

Основною ідеєю розробки стала необхідність створення програмного забезпечення у формі, яка була б привабливою, доступною та зрозумілою для цільової аудиторії. Ігровий підхід дозволяє ефективніше формувати базові знання, адже взаємодія в процесі ігрового підходу активізує когнітивні процеси дитини, мотивує до навчання та підвищує емоційну залученість.

На основі обраної платформи Unity було реалізовано всі ключові компоненти програмного забезпечення. Створено головне меню з інтуїтивно зрозумілим інтерфейсом у піксельному стилі. Такий підхід дозволяє зробити інтерфейс візуально привабливим для дітей, а також забезпечує легке сприйняття навігації.

Меню дає можливість користувачу обрати одну з доступних тем або перейти до обраного рівня.

які включають фонове середовище та логіку переміщення ворогів за допомогою технології NavMesh, що дозволяє створити реалістичну навігацію у двовимірному середовищі. Просторове розташування об'єктів, а також можливість реагувати на дії гравця, додає динаміки та інтерактивності.

Головний об'єкт, озброєний пером як засобом впливу на навколишнє середовище, має розширену логіку керування, анімаційний супровід та систему бою. Реалізовано систему отримання шкоди, яка супроводжується ефектом мерехтіння та відкидання, що посилює візуальний зворотній зв’язок.

Крім того, введено два типи ворожих акторів — "Пил" і "Вода", які уособлюють відповідні загрози для комп’ютерного обладнання. Кожен з них має поведінкову модель, засновану на автоматі станів: бродіння, виявлення героя, переслідування, атака, отримання шкоди та знищення.

Механіка бою передбачає взаємодію через колайдери, що активуються на момент атаки. Це дозволяє моделювати зіткнення головного об'єкта з негативним чинником і здійснювати обчислення шкоди згідно з заданими параметрами. При загибелі ворожого актора активується відповідна анімація, а після знищення усіх ворожих акторів через певний проміжок часу, реалізований за допомогою корутини LoadVictorySceneAfterDelay(), що прописана в скрипті EnemyManager.cs, відображається екран перемоги "You Win", що забезпечує логічне завершення рівня та досягнення цілі.

Особливу увагу було приділено архітектурі проекту. Для оптимального керування параметрами ворожих акторів впроваджено ScriptableObject — спеціальні об’єкти Unity, які дозволяють зберігати параметри поза межами сцен і компонентів. Це забезпечує гнучкість налаштування, дозволяє повторно використовувати налаштування для різних ворогів і значно спрощує масштабування проекту.

Таким чином, реалізована версія програмного забезпечення не лише відповідає технічним вимогам, але й закладає міцну основу для подальшого розвитку. У майбутньому можливо легко додавати нових ворожих акторів, нові сцени та інтерактивні завдання без потреби змінювати існуючу логіку гри. Проект повністю відповідає поставленій меті — залучити школярів молодших класів до пізнання основ інформатики у доступній та захоплюючій формі.

ВИСНОВКИ

У рамках дипломної роботи було реалізовано програмне забезпечення навчального призначення для учнів молодших класів, метою якого є ознайомлення дітей з основами інформатики та формування у них елементарних навичок безпечного користування комп’ютерною технікою. Розроблена гра має форму інтерактивної пригоди, у якій учень керує персонажем, взаємодіє з віртуальним середовищем, долає ворожих акторів та отримує візуальне підтвердження правильних і неправильних дій. Такий підхід забезпечує високий рівень залученості та емоційної взаємодії з навчальним контентом.

У ході реалізації проекту було вирішено такі задачі.

— проаналізовано теоретичні основи та існуючі рішення;

Розглянуто сучасний стан викладання інформатики в початковій школі, визначено особливості використання комп’ютерних ігор у навчанні дітей, досліджено платформи для розробки ігрових продуктів та проведено огляд існуючих програмних продуктів для навчання інформатики.

— визначено основні складові програмного забезпечення для навчання основ інформатики;

Обрано ігровий рушій та інструментальні засоби розробки, визначено структуру проекту, визначено принципи управління персонажем, визначено способи анімаційного супроводу станів головного героя, визначено елементи поведінки ворожих акторів та системи бою, визначено основні етапи моделювання станів та анімаційної поведінки ворожих акторів, визначено види зброї головного героя, визначено способи взаємодії атаки героя та ворожих акторів.

Структуроване проектування логіки персонажа, механіки руху, бойової системи, анімацій станів, відтворення звукових та візуальних ефектів. Всі ці компоненти дозволяють створити реалістичний та послідовний віртуальний світ, який не лише розважає, а й навчає. У грі реалізовані два ворожих актори — "Пил" та "Вода", кожен з яких є уособленням певної загрози для комп’ютерної техніки. Для боротьби з ними головний герой використовує перо — символічний

інструмент очищення. Такий наочний і образний підхід сприяє кращому засвоєнню інформації учнями молодших класів.

— реалізовано програмне забезпечення для навчання основ інформатики.

Розроблено головне меню гри, реалізовано сцени гри та навігації, розроблено візуалізацію та керування головним героєм, реалізовано ворожих акторів "Пил" та "Вода", реалізовано систему бою та отримання шкоди, реалізовано смерть героя та завершення гри, реалізовано закінчення життя героя та завершення гри, використано ScriptableObject для параметризації ворожих акторів, проведено тестування та підсумки реалізації.

Особливу увагу в роботі приділено візуальній складовій — усі сцени реалізовані у стилістиці піксельної графіки, що робить гру зрозумілою та привабливою для дітей. Функціональність реалізовано на платформі Unity, що забезпечує високу продуктивність, кросплатформеність і можливість масштабування проекту. В роботі також використано інструменти Cinemachine для динамічного супроводу героя камерою, ScriptableObject — для зручного параметрування поведінки ворожих акторів, та системи подій, що забезпечують реактивність і модульність коду.

Тестування показало, що реалізоване програмне забезпечення є ефективним засобом навчання. Учні після проходження гри демонструють кращі знання щодо того, як правильно поводитися з комп’ютером, наприклад: не допускати потрапляння води, не залишати техніку в пилу, слідкувати за чистотою, вимикати пристрої після роботи тощо. Ігрова форма подачі інформації сприяє глибшому засвоєнню матеріалу в порівнянні з традиційними методами.

Архітектура проекту дозволяє легко додавати нові рівні, типи ворожих акторів, освітні сценарії, що робить програму перспективною для подальшого розвитку та адаптації до різних тем інформатики. Таким чином, всі поставлені завдання були виконані і мета дослідження була досягнута. Слід відзначити, що дипломна робота також заклала основу для створення повноцінного навчального продукту, який може бути використаний у школах як один з інтерактивних інструментів викладання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мій клас [Електронний ресурс]. — Режим доступу: https://www.miyklas.com.ua/p/informatika-nush

2. Мій клас "Інформатика" [Електронний ресурс]. — Режим доступу: https://www.miyklas.com.ua/p/informatika-nush/2-klas

3. Вікіпедія [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Гра

4. На Урок [Електронний ресурс]. — Режим доступу: https://naurok.com.ua/vikoristannya-igrovih-tehnologiy-na-urokah-informatiki-340312.html

5. Epicgames [Електронний ресурс]. — Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-editor-interface

6. Godot docs [Електронний ресурс]. — Режим доступу: https://docs.godotengine.org/uk/4.x/getting_started/introduction/first_look_at_the_editor.html

7. AssetStore [Електронний ресурс]. — Режим доступу: https://assetstore.unity.com/2d

8. Code [Електронний ресурс]. — Режим доступу: https://code.org/student/elementary

9. Scratch [Електронний ресурс]. — Режим доступу: https://scratch.mit.edu

10. Tynker [Електронний ресурс]. — Режим доступу: https://www.tynker.com

11. Unity 2022.3.25 [Електронний ресурс]. — Режим доступу: https://www.npmjs.com/package/yup

12. Github [Електронний ресурс]. — Режим доступу: https://github.com/h8man/NavMeshPlus

13. Pinterest [Електронний ресурс]. — Режим доступу: https://www.pinterest.com/pin/292874782019719645/

14. Itch.io [Електронний ресурс]. — Режим доступу: https://9e0.itch.io/witches-pack

15. Pinterest [Електронний ресурс]. — Режим доступу: https://www.pinterest.com/pin.it/1wg3cMums

16. Itch.io [Електронний ресурс]. — Режим доступу: https://luizmelo.itch.io/monsters-creatures-fantasy

17. Pinterest [Електронний ресурс]. — Режим доступу: https://www.pinterest.com/pin/35254809571234426/

18. Asset Store [Електронний ресурс]. — Режим доступу: https://assetstore.unity.com/packages/2d/characters/slime-enemy-pixel-art-228568#content

19. Google Drive [Електронний ресурс]. — Режим доступу: https://drive.google.com/drive/folders/1GsFqatZQQoaqs_irMupAAgxQrWFn4cNZ

20. Itch.io [Електронний ресурс]. — Режим доступу: https://pumpkyn-i.itch.io/heartshplife-pixel-art-16x16-assets

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
"21"березня 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
"Програмне забезпечення для навчання основам інформатики школярів молодших класів"
08-54.БКР.004.00.000.ТЗ

Науковий керівник: к.т.н., доцент
_____ Черняк О. І.

Студентка групи 1КІ-23мс
_____ Вареник А. О.

ВНТУ 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність БКР полягає у тому, що навчання через гру є одним із найефективніших методів для дітей молодшого шкільного віку, оскільки поєднує в собі пізнавальний та розважальний аспекти. Враховуючи постійний розвиток інформаційних технологій, важливо створювати інтерактивні засоби навчання, які допомагають дитині не лише засвоїти базові знання з інформатики, а й розвинути критичне мислення та творчі здібності.

1.2 Наказ про затвердження теми БКР

2 Мета БКР і призначення розробки

2.1 Метою даного дипломного проекту є розробка програмного забезпечення для навчання основ інформатики учнів молодших класів на основі ігрових методик, що сприятиме підвищенню їхньої мотивації до навчання та кращому засвоєнню матеріалу.

2.2 Практичним застосуванням розроблене програмне забезпечення може бути використане в освітніх закладах для викладання основ інформатики молодшим школярам. Воно допоможе зробити навчальний процес більш цікавим, інтерактивним і ефективним, що сприятиме розвитку сучасних навичок роботи з інформаційними технологіями у дітей.

3 Вихідні дані для виконання БКР

3.1 Аналіз теоретичних основ та існуючих рішень.

3.2 Визначення основних складових програмного забезпечення для навчання основ інформатики.

3.3 Реалізація програмного забезпечення для навчання основ інформатики.

4 Вимоги до виконання БКР

Головна вимога — розроблене програмне забезпечення повинно сприяти мотивації школярів до правильного користування комп’ютерною технікою.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 – Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Теоретичні основи та аналіз існуючих рішень	21.03.25	06.04.25	Аналітичний огляд літературних джерел, задачі досліджень, розділ 1 ПЗ
2	Визначення основних складових програмного забезпечення для навчання основ інформатики	07.04.25	13.04.25	Розділ 2
3	Розробка програмного забезпечення для навчання основ інформатики	14.04.25	04.05.25	Розділ 3
4	Апробація та впровадження результатів дослідження	05.05.25	07.05.25	Тези доповідей
5	Оформлення пояснювальної записки, графічного матеріалу і презентації	08.05.25	12.05.25	ПЗ, графіч. матеріал і презентація
6	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	13.05.25	13.05.25	Оформлені документи

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист

БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються

— ДСТУ 3008: 2015 "Звіти в сфері науки і техніки. Структура та правила оформлювання";

— ДСТУ 8302: 2015 "Бібліографічні посилання. Загальні положення та правила складання";

— міждержавний ГОСТ 2.104-2006 "Єдина система конструкторської документації. Основні написи";

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – "Комп’ютерна інженерія" (освітня програма

"Комп’ютерна інженерія"). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в "Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21".

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмне забезпечення для навчання основам інформатики школярів молодших класів.

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ: кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98,5 Схожість 1,5%

Аналіз звіту подібності (відмітити потрібне):

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Вареник А.О
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О.І.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАВЧАННЯ ОСНОВАМ ІНФОРМАТИКИ ШКОЛЯРІВ МОЛОДШИХ КЛАСІВ

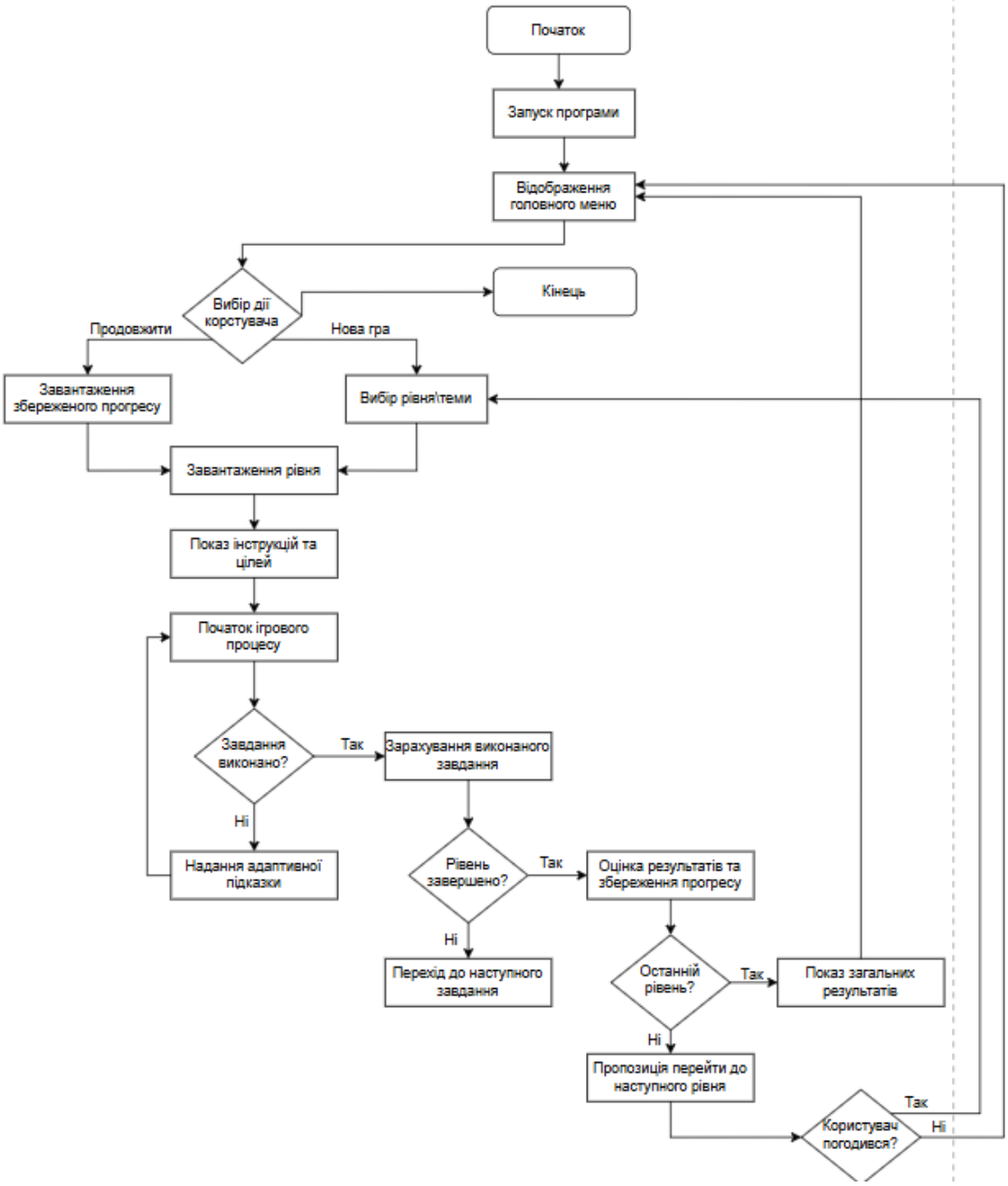

Рисунок В.1 — Структурна схема програмного забезпечення для навчання основам інформатики школярів молодших класів

ДОДАТОК Г

Лістинг файлу Player.cs

Лістинг програмного забезпечення

```
using System;
using System.Collections;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.InputSystem.LowLevel;
using UnityEngine.SceneManagement;

[SelectionBase]
public class Player : MonoBehaviour
{
   public static Player Instance { get; private set; } //Singleton, тому що перс один, і клас його один
   public event EventHandler OnPlayerDeath;//подія смерті Героя
   [SerializeField] private float _movingSpeed = 5f; //швидкість персонажу
   [SerializeField] private int _maxHealth = 10; //Max знач HP героя
   [SerializeField] private float _damageRecoveryTime = 0.5f; //t відновлення героя (при отриманні урону 0.5c)
   [SerializeField] private float _deathRecoveryTime = 1.5f;//t перемикання сцени Гру закінчено після смерті героя ( 0.5c)
   [SerializeField] private PlayerVisual _playerVisual; //посилання на PlayerVisual
   Vector2 inputVector;
   private Rigidbody2D _rb;
   private KnockBack _knockBack;
   private float _minMovingSpeed = 0.1f; //Перевірка мінімальної швидкості Player щоб зрозуміти, чи він біжить чи стоїть
   private bool _isRunning = false; //Перевірка, чи біжить Player (ні)
   private int _currentHealth;
   private bool _canTakeDamage;//чи може зараз герой отримати урон
   private bool _isAlive;
   private void Awake()//Awake спрацьовує до старту
   {
      Instance = this;//Singleton
      _rb = GetComponent<Rigidbody2D>();
      _knockBack = GetComponent<KnockBack>();
   }
   private void Start()
   {
      _currentHealth = _maxHealth;
      _canTakeDamage = true;
```

```
        _isAlive = true;
        GameInput.Instance.OnPlayerAttack += GameInput_OnPlayerAttack;//функція, яка буде спрацьовувати при натисканні лівої кнопки миші
    }
    private void Update()
    {
        inputVector = GameInput.Instance.GetMovementVector();
        //inputVector = inputVector.normalized;//діагональні рухи такі самі по швидкості як інші (нормалізація верктора =1)
    }
    private void FixedUpdate() //Рух персонажа, зчитує напрвлення руху
    {
        if (_knockBack.IsGettingKnockedBack)//якщо герой в стані відльоту, то покидаємо метод і не виконуємо рух персонажем
            return;
        HandleMovement();
    }
    public bool IsAlive() => _isAlive;//метод щоб коли тільки живий, то відбувається поворот в стотону миші
    public bool IsDead => _currentHealth <= 0; //повертає true, якщо HP героя = 0 або менше
    public void TakeDamage(Transform damageSourse, int damage) // метод отримання урону Героєм
    {
        if (_canTakeDamage && _isAlive)//якщо можемо зараз отримати урон
        {
            _canTakeDamage = false;
            _currentHealth = Mathf.Max(0, _currentHealth -= damage); // (вичислення зарашнього HP) і вибивається максимальне на даний момент
            Debug.Log(_currentHealth);
            _knockBack.GetKnockedBack(damageSourse);
            _playerVisual.StartBlinking(); //викликаємо метод мерехтіння
            StartCoroutine(DamageRecoveryRoutine());//запуск КАРУТИНИ отримання пошкодження
        }
        DetectDeath();
    }
    public int GetCurrentHealth()
    {
        return _currentHealth;
    }
    public int GetMaxHealth()
    {
        return _maxHealth;
    }
```

```
    private void DetectDeath()//якщо HP героя = 0 - його буде знищено - МЕТОД СМЕРТІ ГЕРОЯ
    {
        if (_currentHealth == 0 && _isAlive)
        {
            _isAlive = false;
            _knockBack.StopKnockBackMovement();//зупитання відлітання при смерті
            GameInput.Instance.DisableMovement();//виклик методу блокування рухів після смерті
            OnPlayerDeath?.Invoke(this, EventArgs.Empty);//перевірка події, чи помер герой
            StartCoroutine(DeathRecoveryRoutine());
        }
    }
    private IEnumerator DeathRecoveryRoutine()
    {
        yield return new WaitForSeconds(_deathRecoveryTime);
        SceneManager.LoadScene("GameOver");
    }
    //КОРУТИНА - тут очікує час (в цьому випадку йде очкування 0,5 секунди, перед отримуванням героєм урону)
    private IEnumerator DamageRecoveryRoutine()
    {
        yield return new WaitForSeconds(_damageRecoveryTime);
        _canTakeDamage = true;
    }
    //КІНЕЦЬ КОРУТИНИ
    public bool IsRunning() //повернення значення змінної чи біжить персонаж, ?щоб мати змогу викликати її в PlayerVisual?
    {
        return _isRunning;
    }
    private void GameInput_OnPlayerAttack(object sender, System.EventArgs e)//коли в ігрока виконається функція, він запитає у ActiveWeapon, яка на даний момент зброя в перса,
    {
        ActiveWeapon.Instance.GetActiveWeapon().Attack();//і в зарашньої зброї повернеться перо і викличеться функція атаки (звертаємося до ActiveWeapon)
    }
    private void HandleMovement()//Рух персонажу! Як має рухатися (в залежності від того які кнопки натиснуті)
    {
        _rb.MovePosition(_rb.position + inputVector * (_movingSpeed * Time.fixedDeltaTime)); //Друга половина впливає на швидкість руху (0.02 с і вкиликаємо 50 разів за 1с)
```

```
        if (Mathf.Abs(inputVector.x) > _minMovingSpeed || Mathf.Abs(inputVector.y) > _minMovingSpeed) //В загалі перевірка, чи біжить герой. Якщо рух по осі Х більше мін швидкості або швидкість руху по осі У швидше в будь якому напрямку за мін швидкість, то наж персонаж БІЖИТЬ
        {
            _isRunning = true;
        }
        else
        {
            _isRunning = false;
        }
    }
    public Vector3 GetPlayerScreenPosition()//зчитування положення ігрока на екрані
    {
        Vector3 playerScreenPosition = Camera.main.WorldToScreenPoint(transform.position);
        return playerScreenPosition;
    }

}
```

ДОДАТОК Д

Лістинг файлу PlayerVisual.cs

```
using System.Collections;
using UnityEngine;

public class PlayerVisual : MonoBehaviour
{
   [SerializeField] private SpriteRenderer _spriteRenderer; //посилання на
SpriteRenderer гравця
   private Animator animator;
   private const string IS_RUNNING = "IsRunning";
   private const string IS_DIE = "IsDie";
   private void Awake()
   {
      animator = GetComponent<Animator>(); //отримуємо компонент Animator
      // НЕ потрібно GetComponent для _spriteRenderer, бо ми його вже прив'язали
вручну через Inspector
   }
   private void Start()
   {
      Player.Instance.OnPlayerDeath += Player_OnPlayerDeath; //підписка на подію
СМЕРТІ героя
   }
   private void Player_OnPlayerDeath(object sender, System.EventArgs e)
   {
      animator.SetBool(IS_DIE, true); //вмикаємо анімацію смерті
   }
   //Звертаємося до класу Player (методу public bool IsRunning() ), щоб знати чи
біжить песонаж
   private void Update()
   {
      animator.SetBool(IS_RUNNING, Player.Instance.IsRunning());
      if (Player.Instance.IsAlive()) //умова, що тільки живий — повертається до
мишки
         AdjustPlayerFacingDirection();
   }
   private void AdjustPlayerFacingDirection() //Поворот спрайту героя в сторону
мишки
   {
      Vector3 mousePos = GameInput.Instance.GetMousePosition();
      Vector3 playerPosition = Player.Instance.GetPlayerScreenPosition();
      if (mousePos.x < playerPosition.x)
      {
```

```
            _spriteRenderer.flipX = true;
        }
        else
        {
            _spriteRenderer.flipX = false;
        }
    }
    public void StartBlinking() //Метод запуску мерехтіння
    {
        StartCoroutine(BlinkCoroutine());
    }
    private IEnumerator BlinkCoroutine() //Корутина мерехтіння спрайта
    {
        float blinkDuration = 0.5f; //тривалість мерехтіння
        float blinkInterval = 0.1f; //інтервал між вкл/викл
        float elapsed = 0f;
        while (elapsed < blinkDuration)
        {
            _spriteRenderer.enabled = !_spriteRenderer.enabled; //перемикаємо видимість
            yield return new WaitForSeconds(blinkInterval);
            elapsed += blinkInterval;
        }
        _spriteRenderer.enabled = true; //вмикаємо спрайт після завершення
    }
}
```

ДОДАТОК Е

Лістинг файлу GameInput.cs

```
using System;
using UnityEngine;
using UnityEngine.InputSystem;

public class GameInput : MonoBehaviour //клас для зчитування вводу з клавіатури
{
   //Singleton клас, тому що клас вводу з клавіатури у нас один
   public static GameInput Instance { get; private set; }
   private PlayerInputActions _playerInputActions;
   public event EventHandler OnPlayerAttack; //ПОДІЯ АТАКИ ГЕРОЯ
   private void Awake()
   {
      Instance = this;
      _playerInputActions = new PlayerInputActions();//ініціалізація
PlayerInputActions
      _playerInputActions.Enable();//виклик Enable, щоб запрацювала
      _playerInputActions.Combat.Attack.started += PlayerAttack_started;//зчитування
клацання лівої кнопки миші (натискання вниз)
   }
   public Vector2 GetMovementVector()
   {
      Vector2 inputVector = _playerInputActions.Player.Move.ReadValue<Vector2>();
      return inputVector;
   }
   public Vector3 GetMousePosition()//Зчитування положення миші (x знач + чи -)
   {
      Vector3 mousePos = Mouse.current.position.ReadValue();
      return mousePos;
   }
   public void DisableMovement()//блокування рухів після смерті героя
   {
      _playerInputActions.Disable();
   }
   public void UnsubscribeAllEvents()
   {
      OnPlayerAttack = null;
   }
   private void PlayerAttack_started(InputAction.CallbackContext obj)
   {
```

```
        OnPlayerAttack?.Invoke(this, EventArgs.Empty);//Перевірки, чи виконується подія (не=0)Виклик самої події удару при натиску лівої кнопки миші. Тепер треба передати цю подію герою
    }
}
```

ДОДАТОК Ж

Лістинг файлу Utils.cs

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

namespace Diplom.Utils
{
   public static class Utils
   {
      public static Vector3 GetRandomDir() //функція для задання рандомного знаяення x та y
      {
         return new Vector3(Random.Range(-1f, 1f), Random.Range(-1f, 1f)).normalized;//створюємо новий Вектор3, передаємо туди випадкове направлення по осі x та y і нормалізуємо
      }
   }
}
```

ДОДАТОК И

Лістинг файлу EnemyAI.cs

```
using UnityEngine;
using UnityEngine.AI;
using Diplom.Utils;
using System;

public class EnemyAI : MonoBehaviour
{
    [SerializeField] private State _startingState;
    [SerializeField] private float _roamingDistanceMax = 7f; //Максимальна відстань бродіння ворога
    [SerializeField] private float _roamingDistanceMin = 3f;//Мінімальна відстань бродіння ворога
    [SerializeField] private float _roamingTimerMax = 2f;//час бродіння ворога до точки (якщо за цей час не досяг потрібної точки, шукає іншу і йде до неї)
    [SerializeField] private bool _isChasingEnemy = false;//змінна чи переслідує ворог??
    [SerializeField] private float _chasingDistance = 4f; //відстань з якої ворог починає переслідувати героя
    [SerializeField] private float _chasingSpeedMultiplier = 2f; //змінна, яка відповідає за прискорення ворога при переслідуванні
    [SerializeField] private bool _isAttackingEnemy = false;//змінна, чи ворог атакуючий
    [SerializeField] private float _attackingDistance = 2f;  //змінна відстані, з якої буде атакувати ворог
    [SerializeField] private float _attackRate = 2f;//швидкість атаки ворога
    private float _nextAttackTime = 0f;//останнє положення Пилу
    private UnityEngine.AI.NavMeshAgent _navMeshAgent;
    private State _currentState;
    private float _roamingTimer;//змінна, яка зберігає скільки часу вже об`єкт пробродив
    private Vector3 _roamPosistion;//вибір нової точки, до якої рухатиметься Агент (змінна для збереження цієї інфи)
    private Vector3 _startingPosition;//початкова точка руху Агента, від якої і опиратиметься подальша побудова руху (змінна для збереження цієї інфи)
    private float _roamingSpeed;//змінна швидкості ходьби простої
    private float _chasingSpeed;//змінна швидкості ходьби переслідування
    private float _nextCheckDirectionTime = 0f;//чи в правильному напрямку повернуті
    private float _checkDirectionDuration = 0.1f;//як часто перевіряти, чи правильно повернуті
    private Vector3 _lastPosition;
```

```
    public event EventHandler OnEnemyAttack;//оголошення події атаки ворога
    public bool IsRunning //Відслідкування, чи біжить Пил-Скелет
    {
        get
        {
            if (_navMeshAgent.velocity == Vector3.zero)//якщо швидкість ворога(агента)
= 0
            {
                return false; //то він не біжить
            }
            else
            {
                return true;
            }
        }
    }
    public enum State//створено машину станів спокою та руху
    {
        Idle,
        Roaming,
        Chasing,//переслідування
        Attacking,
        Death
    }
    private void Awake()
    {
        _navMeshAgent = GetComponent<UnityEngine.AI.NavMeshAgent>();
        _navMeshAgent.updateRotation = false; //щоб слайм не обертався при русі
        _navMeshAgent.updateUpAxis = false;//щоб орієнтація МавМеша не впливала
на орієнтацію об`єкту
        _currentState = _startingState;//коеф прискорення =1
        _roamingSpeed = _navMeshAgent.speed;
        _chasingSpeed = _navMeshAgent.speed * _chasingSpeedMultiplier;//швидкість
переслідування = шв ворога*
    }
    private void Update()//постійна перевірка
    {
        StateHandler();
        MovementDirectionHandler();
    }
    public void SetDeathState()//стан смерті Ворога
    {
        _navMeshAgent.ResetPath();//скидання шляху агента (Ворога)
        _currentState = State.Death;//новий стан це СМЕРТЬ
    }
```

```
   private void StateHandler()//(метод перевірки, коли герой далеко втіче від ворога, то вород закінчить переслідування) або перевірка в якому стані знаходиться Агент(ворог)
   {
      switch (_currentState) // по замовчення наш агент в стані бродіння, якщо нічого не відбуавється
      {
         case State.Roaming://Якщо об`єкт в стані бродіння, то виконуємо логіку бродіння (коли починаємо бродити):
            _roamingTimer -= Time.deltaTime;//зменшуємо час бродіння (має бути не більше 2 с)
            if (_roamingTimer < 0)//якщо час бродіння менше 0, тоді
            {
               Roaming();//виконуємо пошук нової точки, до якої хочемо пройти
               _roamingTimer = _roamingTimerMax;//назначаємо новий час бродіння (максимальний, у нас це 2с)
            }
            CheckCurrentState();
            break;
         case State.Chasing:
            ChasingTarget();//щось має робитися при переслідуванні
            CheckCurrentState();//повернення значення зарашнього стану ворога
            break;
         case State.Attacking:
            AttackingTarget();
            CheckCurrentState();
            break;
         case State.Death:
            break;
         default:
         case State.Idle:
            break;
      }
   }
   private void ChasingTarget()//метод переслідування цілі
   {
      _navMeshAgent.SetDestination(Player.Instance.transform.position);//цілю переслідування є герой
   }
   public float GetRoamingAnimationSpeed()//метод отримання V значення прискорення ворога
   {
      return _navMeshAgent.speed / _roamingSpeed;//якщо агент(ворог), просто гуляє, то його V=_roamingSpeed(бродання) = _navMeshAgent.speed;(швидкості агента).
```

```
    }
    private void CheckCurrentState()//отримання зарашнього статну ворога
    {
        float distanceToPlayer = Vector3.Distance(transform.position,
Player.Instance.transform.position);//визначення відстані між ворогом та героєм
        State newState = State.Roaming;//бродитиме
        if (_isChasingEnemy) //якщо ворог атакує
        {
            if (distanceToPlayer <= _chasingDistance) //якщо відстань до героя відстані
переслідування, то
            {
                newState = State.Chasing;//стан переслідування
            }
        }
        if (_isAttackingEnemy) //якщо ворог атакуючий
        {
            if (distanceToPlayer <= _attackingDistance) //якщо відстань не героя менша
ніж дистанція для атаки, то
            {
                newState = State.Attacking;//новий стан = атака
            }
        }
        if (newState != _currentState)//якщо новий стан не = стану, що був раніше
        {
            if (newState == State.Chasing)//і якщо зараз в стані переслідування
            {
                _navMeshAgent.ResetPath();//то ворог забуває про точку, до якої рухався
в стані бродіння Roaming
                _navMeshAgent.speed = _chasingSpeed;//зміна швидкості на швидкість
переслідування
            }
            else if (newState == State.Roaming)//якщо ми перейшли в новий стан
бродіння (не переслідуємо героя)
            {
                _roamingTimer = 0f;//миттєвий перехід стану
                _navMeshAgent.speed = _roamingSpeed; //тоді зменшуємо швидкість до
шв бродіння
            }
            else if (newState == State.Attacking)//якщо новий стан, це стан атаки
            {
                _navMeshAgent.ResetPath();//то не перслідуємо героя (скидаємо шлях)
            }
            _currentState = newState;//повернення зарашнього нового стану
        }
```

```
        if (Player.Instance == null || Player.Instance.IsDead) //припинення бити героя після його смерті
        {
            _navMeshAgent.ResetPath(); //скидаємо шлях, якщо гравець мертвий
            _currentState = State.Roaming; //ворог нічого не робить
            return;
        }
    }
    private void AttackingTarget()//логіка атаки ворога
    {
        if (Time.time > _nextAttackTime)//якщо зарашній час більше часу наступної атаки (а воно в нас =0), тобто одразу
        {
            OnEnemyAttack?.Invoke(this, EventArgs.Empty);//виклик події
            _nextAttackTime = Time.time + _attackRate;//і новий час атаки, це зарашній час + 2с (бо зм =2с) (кажемо коли буде наступна атака)
        }
        if (Player.Instance == null || Player.Instance.IsDead) return; //не атакуємо, якщо герой мертвий
    }
    private void MovementDirectionHandler()//метод повороту ворога в сторону руху
    {
        if (Time.time > _nextCheckDirectionTime)//якщо час перевірки наступив, йдемо далі
        {
            if (IsRunning)//якщо ворог біжить, то
            {
                ChangeFacingDirection(_lastPosition, transform.position);//дивимося де була остання позиція ворога і яке зарашнє положення і повертаємося в ту ж сторону
            }
            else if (_currentState == State.Attacking)//якщо ворог в стані атаки
            {
                ChangeFacingDirection(transform.position, Player.Instance.transform.position);//то повертаємося до героя
            }
            _lastPosition = transform.position;//оновлюємо останнє положення (робимо його тепершнім)
            _nextCheckDirectionTime = Time.time + _checkDirectionDuration;
        }
    }
    private void Roaming()
    {
        _startingPosition = transform.position;//онолвення початкової точки, від якої потім будемо рухатися
```

```
        _roamPosistion = GetRoamingPosition();//Якщо значення часу бродіння менше 0, тоді виконуємо пошук нової рандомної точки
        _navMeshAgent.SetDestination(_roamPosistion);//рух до нової рандомної точки нашим Агентом
    }
    private Vector3 GetRoamingPosition()//отримання Агентом випадкової нової точки руху
    {
        return _startingPosition + Utils.GetRandomDir() * UnityEngine.Random.Range(_roamingDistanceMin, _roamingDistanceMax);//пошук нової точки: беремо стару точку + направлення руху, що * на довжину руху  (тут типу рухається біля своєї точки спавна)
    }
    private void ChangeFacingDirection(Vector3 sourcePosition, Vector3 targetPosition) //sourcePosition(місце знаходження ворога зараз) targetPosition(точка куди потрібно рухатися ворогу)
    {
        if (sourcePosition.x > targetPosition.x) //якщо зарашня позиція по х правіще від точки, куди має йти ворог, то
        {
            transform.rotation = Quaternion.Euler(0, -180, 0);//то розвернути на 180 градусів
        }
        else
        {
            transform.rotation = Quaternion.Euler(0, 0, 0);//або залишаємо як є
        }
    }
}
```

ДОДАТОК К

Лістинг файлу EnemyEntity.cs

```
using System;
using UnityEngine;

[RequireComponent(typeof(PolygonCollider2D))]
[RequireComponent(typeof(BoxCollider2D))]
[RequireComponent(typeof(EnemyAI))]
public class EnemyEntity : MonoBehaviour
{
   [SerializeField] private EnemySO _enemySO;
   public event EventHandler OnTakeHit;//подія Отримання урону
   public event EventHandler OnDeath;//подія СМЕРТІ ворога
   private Animator _animator;
   private const string IS_DIE = "IsDie";
   public event EventHandler OnDeleteDustObjectAfterDeath;//подія видалення тіла
ворога після знищення
   //[SerializeField] private int _maxHealth;
   private int _currentHealth;
   private PolygonCollider2D _polygonCollider2D;
   private BoxCollider2D _boxCollider2D;
   private EnemyAI _enemyAI;
   private void Awake()
   {
      _polygonCollider2D = GetComponent<PolygonCollider2D>();
      _boxCollider2D = GetComponent<BoxCollider2D>();
      _enemyAI = GetComponent<EnemyAI>();
   }
   private void Start()//при запуску гри зарашній стан HP - максимальний
   {
      _currentHealth = _enemySO.enemyHealth;
   }
   private void OnTriggerStay2D(Collider2D collision) //приймає колейдер, який його
пересік
   {
      if (collision.transform.TryGetComponent(out Player player)) //якщо перетений
об'єкт це герой
      {
         player.TakeDamage(transform, _enemySO.enemyDamageAmount); //герой
отримує урон від Пилу
      }
   }
   public void TakeDamage(int damage)//метод отримання урону
```

```
    {
        _currentHealth -= damage;
        OnTakeHit?.Invoke(this, EventArgs.Empty);//виклаємо подію АТАКА
(ОТРИМАННЯ УРОНУ)
        DetectDeath();
    }
    public void PolygonColliderTurnOff() //метод для виключення колайдеру "меча"
ворога Пилу
    {
        _polygonCollider2D.enabled = false;
    }
    public void PolygonColliderTurnOn() //метод для включення колайдеру "меча"
ворога Пилу
    {
        _polygonCollider2D.enabled = true;
    }
    private void DetectDeath()//якщо значення HP ворога <=0, то ворог помирає і
видаляється його об'єкт зі сцени
    {
        if (_currentHealth <= 0)
        {
            _boxCollider2D.enabled = false;//"вимкнення" колайдеру Пилу, при смерті
            _polygonCollider2D.enabled = false;//"вимкнення" колайдеру взмаху меча
Пилу, при смерті
            _enemyAI.SetDeathState();//метод
            OnDeath?.Invoke(this, EventArgs.Empty);//якщо значення HP<0,
викликається подія смерті
            DeleteEnemyObject();
        }
    }
    private void DeleteEnemyObject()//видалення об'єкта ворога, після його смерті
    {
            OnDeleteDustObjectAfterDeath?.Invoke(this, EventArgs.Empty);//перевірка,
чи відбувся тригер смерті. Якщо так тоді викликається подія зникнення
знищеного тіла ворога
    }
}
```

ДОДАТОК Л

Лістинг файлу SkeletonDustVisual.cs

```
using UnityEngine;
using System;

[RequireComponent(typeof(Animator))]
public class SkeletonDustVisual : MonoBehaviour
{
    [SerializeField] private EnemyAI _enemyAI;
    [SerializeField] private EnemyEntity _enemyEntity;
    [SerializeField] private GameObject _enemyShadow;
    private Animator _animator;//доступ до аніматора
    //константи тригерів, щоб звертатися до них в коді (має бути ОБОВ'ЯЗКОВО
без помилок написано)
    private const string IS_RUNNING = "IsRunning";
    private const string CHASING_SPEED_MULTIPLIER = "ChasingSpeedMultiplier";
    private const string ATTACK = "Attack";
    private const string TAKEHIT = "TakeHit";
    private const string IS_DIE = "IsDie";
    private const string IS_DELETE_OBJECT = "IsDeleteObject";
    private void Awake()
    {
        _animator = GetComponent<Animator>();//ініціалізація аніматора
    }
    private void Start()
    {
        _enemyAI.OnEnemyAttack += _enemyAI_OnEnemyAttack;//підписуємося на
подію атаки
        _enemyEntity.OnTakeHit += _enemyEntity_OnTakeHit; //підписуємося на
подію отримання урону(ворог)
        _enemyEntity.OnDeath += _enemyEntity_OnDeath;//підписуємося на подію
смерті(ворог)
        _enemyEntity.OnDeleteDustObjectAfterDeath +=
_enemyEntity_OnDeleteDustObjectAfterDeath;//підписуємося на подію видалення
тіла ворога після знищення
    }
    private void _enemyEntity_OnDeleteDustObjectAfterDeath(object sender,
EventArgs e)
    {
        _animator.SetBool(IS_DELETE_OBJECT, true);
    }
    private void _enemyEntity_OnTakeHit(object sender, System.EventArgs e)//логіка
отримання урону ворогом
```

```
    {
        _animator.SetTrigger(TAKEHIT);
    }
    private void _enemyEntity_OnDeath(object sender, System.EventArgs e)//логіка
смерті ворога
    {
        _animator.SetBool(IS_DIE, true);
        _enemyShadow.SetActive(false);//значення тіні при смерті = фолс
    }
    private void OnDestroy()
    {
        if (_enemyAI != null)
            _enemyAI.OnEnemyAttack -= _enemyAI_OnEnemyAttack;//відписуємося від
події атаки (наприклад коли ворог помирає)
        if (_enemyEntity != null)
        {
            _enemyEntity.OnTakeHit -= _enemyEntity_OnTakeHit;
            _enemyEntity.OnDeath -= _enemyEntity_OnDeath;
            _enemyEntity.OnDeleteDustObjectAfterDeath -=
_enemyEntity_OnDeleteDustObjectAfterDeath;
        }
    }
    private void Update()
    {
        _animator.SetBool(IS_RUNNING, _enemyAI.IsRunning);//Звертаємося до
компоненту БІГУ ворога (біжить чи ні)
        _animator.SetFloat(CHASING_SPEED_MULTIPLIER,
_enemyAI.GetRoamingAnimationSpeed()); //а тут чому = параметр прискорення НА
ДАНИЙ МОМЕНТ
    }
    public void TrigerAttackAnimationTurnOff()
    {
        if (_enemyEntity != null)
            _enemyEntity.PolygonColliderTurnOff();
    }
    public void TrigerAttackAnimationTurnOn()
    {
        if (_enemyEntity != null)
            _enemyEntity.PolygonColliderTurnOn();
    }
    private void _enemyAI_OnEnemyAttack(object sender, System.EventArgs e)
    {
        _animator.SetTrigger(ATTACK);//запуск тригера АТАКИ
    }
    public void DeleteSelf()
```

```
    {
        Destroy(gameObject); // саме цей об'єкт зникне (візуальна частина ворога)
    }
}
```

ДОДАТОК М

Лістинг файлу EnemySO.cs

```
using UnityEngine;

[CreateAssetMenu()]
public class EnemySO : ScriptableObject
{
    public string enemyName;
    public int enemyHealth;
    public int enemyDamageAmount; //уваишщуи0по
}
```

ДОДАТОК Н

Лістинг файлу ActiveWeapon.cs

```
using UnityEngine;
public class ActiveWeapon : MonoBehaviour
{
   public static ActiveWeapon Instance { get; private set; } //Singleton, тому що зброя Пера в героя одна
   [SerializeField] private Feather feather;

   private void Awake()
   {
      Instance = this;
   }
   private void Update()
   {
      if (Player.Instance.IsAlive())//умтова, що коли тільки живий, то відбувається поворот в стотону миші
         FollowMousePosition();
   }
   public Feather GetActiveWeapon()  //повертає значення пера
   {
      return feather;
   }
   private void FollowMousePosition()//Повертання пера(зброї) за мишкою. (у цьому випадку повертається весь префаб пера)
   {
      Vector3 mousePos = GameInput.Instance.GetMousePosition();//Отримання значення миші на екрані
      Vector3 playerPosition = Player.Instance.GetPlayerScreenPosition();//Зчитування позиції героя на екрані
      if (mousePos.x < playerPosition.x) //якщо позиція миші лівіше позиції героя, то і повертаємося в ліво
      {
         transform.rotation = Quaternion.Euler(0, 180, 0);// Якщо позиція миші лівіше від позиції пера, то ми розвертаємо перо навколо вісі У. Quaternion для обертання об'єктів
      }
      else
      {
         transform.rotation = Quaternion.Euler(0, 0, 0); ;
      }
   }
}
```

ДОДАТОК П

Лістинг файлу Feather.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Rendering;
public class Feather : MonoBehaviour
{
   [SerializeField] private int _damageAmount = 2;
   public event EventHandler OnFeatherSwing;//ПОДІЯ ВЗМАХУ ПЕРА (і анцімація зробить анімацію взмаху)
   private PolygonCollider2D _polygonCollider2D;
   private void Awake()
   {
      _polygonCollider2D = GetComponent<PolygonCollider2D>();
   }
   private void Start()
   {
      AttackColliderTurnOff();//із самого початку коладер Пера буде вимкненим
   }
   public void Attack()
   {
      AttackColliderTurnOffOn();//при атаці колайдер Пера вмикається, без атаки вимикається (і при дуже швидкій атаці HP у ворога також буде змінатися)
      OnFeatherSwing?.Invoke(this, EventArgs.Empty);//перевірка, чи меч в стані взамаху??? (сталася подія взмаху)
   }
   private void OnTriggerEnter2D(Collider2D collision)
   {
      // Якщо колайдер або об'єкт уже знищений — просто вийти
      if (collision == null || collision.gameObject == null) return;
      // Якщо це не EnemyEntity — ігноруємо
      if (!collision.TryGetComponent(out EnemyEntity enemyEntity)) return;
      // Перевіряємо, чи об'єкт все ще активний (не знищений)
      if (!enemyEntity.gameObject.activeInHierarchy) return;
      // І лише тепер наносимо урон
      enemyEntity.TakeDamage(_damageAmount);
   }
   public void AttackColliderTurnOff()//вимкнення колайдера Пера, що не бігати із постійно ним включеним
   {
      //_polygonCollider2D.enabled = false;\
```

```
        if (UnityEngine.SceneManagement.SceneManager.GetActiveScene().name ==
"YouWin")
            return;
        if (_polygonCollider2D != null)
        {
            _polygonCollider2D.enabled = false;
        }
    }
    private void AttackColliderTurnOn()
    {
        if (UnityEngine.SceneManagement.SceneManager.GetActiveScene().name ==
"YouWin")
            return;
        if (_polygonCollider2D != null)
        {
            _polygonCollider2D.enabled = true;
        }
        //_polygonCollider2D.enabled = true;
    }
    private void AttackColliderTurnOffOn()//для того, щоб при дуже швидкій атаці
HP у ворога також буде змінатися, а не проскользувати)
    {
        AttackColliderTurnOff();
        AttackColliderTurnOn();
    }
}
```

ДОДАТОК Р

Лістинг файлу FeatherVisual.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Rendering;
public class FeatheVisual : MonoBehaviour
{
   [SerializeField] private Feather feather;//знає тепер, що існує клас Feather
   private Animator animator;
   private const string ATTACK = "Attack";//Константа, щоб 1 раз прописати в коді цей тригер, іпотім просто звертатися
   private void Awake()
   {
      animator = GetComponent<Animator>();
   }
   private void Start()
   {
      feather.OnFeatherSwing += Feather_OnFeatherSwing; //підписуємося на подію взмаху меча
   }
   private void Feather_OnFeatherSwing(object sender, System.EventArgs e)
   {
      if (animator != null)
      {
         animator.SetTrigger(ATTACK);
      }
   }
   public void TriggerEndAttackAnimation()//вимикає колайдер Пера, якщо закінчується анімація взмаху пера
   {
      feather.AttackColliderTurnOff();
   }
}
```

ДОДАТОК С

Лістинг файлу FeatherSlashVisual.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
    [SerializeField] private Feather feather; //звертаємося до об'єкту Пера, тому що буде прив'язка до його події (коли розмахує)
    private const string ATTACK = "Attack";//оголошення змінної АТАКА
    private Animator animator;//змінна Аніматора
    private void Awake()
    {
        animator = GetComponent<Animator>();//ініціалізація Аніматора
    }
    private void Start()
    {
        feather.OnFeatherSwing += Feather_OnFeatherSwing; //підписуємося на подію взмаху меча, коли він махає
    }
    private void Feather_OnFeatherSwing(object sender, System.EventArgs e)//спрацьоює тригер атаки
    {
        animator.SetTrigger(ATTACK);
    }
    private void OnDestroy()
    {
        if (feather != null)
        {
            feather.OnFeatherSwing -= Feather_OnFeatherSwing;
        }
    }
}
```