Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Проектування та розробка комп'ютерної гри із застосуванням нечіткої логіки*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Архипова О.М.*
(прізвище, ініціали)

4 курсу *КІ-542* групи

Керівники: *доцент Нєнов О.Л.*
(посада, прізвище та ініціали)
асистент Орел А.С.
(посада, прізвище та ініціали)

Консультанти: *проф. Басюркіна Н. Й.*
(посада, прізвище та ініціали)

(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *10.06* 2023 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса - 2023 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Розробка ігор та інтерактивних медіа у віртуальній реальності*

ЗАТВЕРДЖУЮ

Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *10* » *квітня* 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Архипова Олександра Миколайовича

1. Тема роботи *Проектування та розробка комп'ютерної гри із застосуванням нечіткої логіки*

Затверджена наказом університету від « *10* » *04* 2023 р., наказ № *147-03*

2 Термін здачі здобувачем закінченої роботи *5 червня 2023 р.*

3. Вихідні дані роботи

1. Технології для використання – нечітка логіка, кінцевий автомат.

2. Середовище розробки – Unity.

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Передпроектний аналіз. 3. Постановка завдання. Концепт-документ. 4. Проектування гри. Дизайн-документ. 5. Створення контенту і програмна реалізація. 6. Економічні розрахунки. 7. Охорона праці. 8. Загальні висновки.

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

Слайд 1. Характеристика кваліфікаційної роботи. Слайди 2-3. Технології в основі гри. Слайд 4. Концепт гри і вимоги до програмного застосунку. Слайд 5. Архітектура проекту. Слайд 7. Програмні алгоритми. Слайд 8. Елементи інтерфейсу користувача. Слайд 9. Техніко-економічні показники. Слайд 10. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Басюркіна Н.Й., д.е.н., професор*		
Охорона праці			
Нормоконтроль	*Нєнов О.Л., к.т.н., доцент*		

7. Дата видачі завдання *11.04.2023*

Керівники _____ *Олексій НЄНОВ*

_____ *Андрій ОРЕЛ*

Завдання прийняв до виконання _____ *Олександр АРХИПОВ*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Передпроектний аналіз предметної області.*	*24.04.2023*	
2.	*Постановка завдання на розробку.*	*25.04.2023*	
3.	*Проектування архітектури проекту гри.*	*01.05.2023*	
4.	*Проектування компонентів проекту гри.*	*07.05.2023*	
5.	*Проектування алгоритмів гри.*	*14.05.2023*	
6.	*Програмна реалізація гри.*	*21.05.2023*	
7.	*Тестування гри.*	*25.05.2023*	
8.	*Техніко-економічний аналіз проекту.*	*31.05.2023*	
9.	*Опрацювання питань охорони праці.*	*02.06.2023*	
10.	*Оформлення пояснювальної записки і презентації.*	*05.06.2023*	

Керівники роботи _____ *Олексій НЄНОВ*

_____ *Андрій ОРЕЛ*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач – дипломник _____ *Олександр АРХИПОВ*

АНОТАЦІЯ

Дана робота присвячена розробці комп’ютерної гри «Танковий лабіринт» змішаного жанру за допомогою *Unity* з використанням технологій нечіткої логіки, кінцевого автомату і штучного інтелекту. В роботі розглянуті основи цих технологій, представлений концепт і проект комп’ютерної гри, наведені фрагменти його реалізації. Також досліджено питання економічного обґрунтування проекту і охорони праці.

Обсяг текстової частини пояснювальної записки – 83 аркуші, додатків – 9 аркушів.

Ключові слова: *комп’ютерна гра, лабіринт, нечітка логіка, кінцевий автомат, штучний інтелект, Unity, гейм-дизайн.*

ABSTRACT

This work is devoted to the development of the computer game "Tank Labyrinth" of a mixed genre using Unity using the technologies of fuzzy logic, finite state machine and artificial intelligence. The paper examines the basics of these technologies, presents the concept and project of a computer game, and provides fragments of its implementation. The issue of economic justification of the project and labor protection was also investigated.

The volume of the textual part of the explanatory note is 83 sheets, appendices are 9 sheets.

Keywords: *computer game, maze, fuzzy logic, finite state machine, artificial intelligence, Unity, game design.*

ЗМІСТ

стор.

ВСТУП

Індустрія комп'ютерних ігор сьогодні є дуже актуальною галуззю, яка продовжує швидко зростати рік у рік. Згідно зі звітом *Global Games Market Report*, опублікованому в 2021 році, індустрія комп'ютерних ігор загалом заробила 175,8 мільярда доларів у 2020 році, що на 20 % більше, ніж у попередньому.

Індустрія геймдеву постійно розвивається, в ній з'являються нові технології та тренди, що створює можливості для розробників ігор та інших професіоналів у цій галузі. Більш того, з розвитком різних платформ, таких як мобільні пристрої, планшети та віртуальна реальність, індустрія ігор стає ще більш доступною та різноманітною. Таким чином, індустрія комп'ютерних ігор представляє собою дуже актуальний та перспективний *IT*-сектор на сьогоднішній день.

Велике значення у розробці сучасних комп'ютерних ігор має нечітка логіка (*fuzzy logic*). Особливо це стосується ігор, де відбувається активна взаємодія зі штучним інтелектом (*AI*). Нечітка логіка дозволяє розробникам створювати штучний інтелект, який може адаптуватися до середовища, що змінюється, і приймати рішення на основі нечітких вхідних даних. Таким чином, використання нечіткої логіки дозволяє створювати розумніші та адаптивніші *AI*, що може значно покращити ігровий процес та підвищити рівень задоволення гравців. Це свідчить про те, що дана дипломна робота є актуальною і важливою.

Об’єктом дослідження в роботі є узагальнений процес розробки комп’ютерних ігор з використанням нечіткої логіки.

Предметом дослідження є процес проектування і реалізації компонентів проекту комп’ютерної гри з використанням нечіткої логіки.

Основною прикладною метою даної роботи є розробка проекту і реалізація початкової версії комп’ютерної гри обраного жанру і у відповідності до сформульованих вимог.

Для здійснення поставленої мети вирішується низка завдань, серед яких:

- узагальнена постановка завдання;
- передпроектний аналіз технологій нечіткої логіки, кінцевого автомату і штучного інтелекту у комп'ютерних іграх;
- розробка концепт-документу майбутньої гри;
- створення проекту комп'ютерної гри у відповідності до розробленого концепту;
- реалізація проекту у тестовий варіант застосунку;
- техніко-економічне обґрунтування проекту;
- дослідження питань охорони праці;
- оформлення пояснювальної записки та підготовка презентації для захисту проекту.

РОЗДІЛ 1
АНАЛІЗ ТЕХНОЛОГІЙ НЕЧІТКОЇ ЛОГІКИ, КІНЦЕВОГО АВТОМАТУ І ШТУЧНОГО ІНТЕЛЕКТУ У КОМП'ЮТЕРНИХ ІГРАХ

1.1 Огляд поняття апарату нечіткої логіки

Нечітка логіка представляє собою розділ математики, який узагальнює класичну логіку та теорію множин. Вперше введена Лотфі Заде в 1965 році, нечітка логіка вивчає об'єкти, які мають функцію належності елементів до множини, що приймає значення у інтервалі від 0 до 1, а не тільки 0 та 1, якими оперує двійкова логіка. Відповідно, нечіткі значення, якими оперує нечітка логіка, зазвичай представлені в комп'ютерах як числа з плаваючою точкою, а не як цілі числа.

На базі цього поняття вводяться логічні операції над нечіткими множинами та формулюється поняття лінгвістичної змінної, яка відображає нечіткі множини. Предметом нечіткої логіки є вивчення суджень в умовах нечіткості, які подібні до звичайних суджень, та їх використання в обчислювальних системах.

Типовим прикладом, який використовується для опису нечіткої логіки, є температура. Сонячний літній день можна описати як «тепло», не знаючи при цьому точної температури. І навпаки, на Алясці взимку холодно, і точне значення температури тут не є необхідним. Ці поняття холодного, прохолодного, теплого та гарячого є нечіткими. Існує багато неясностей щодо того, в який момент відбувається перехід від теплого до гарячого. Нечітка логіка дозволяє моделювати ці поняття як набори та визначати їх достовірність або істинність за допомогою набору правил.

Прийняття рішення людьми або комп'ютерними програмами часто нагадує певні сірі зони (рис. 1.1). Припустимо, людина не їла декілька годин і починає відчувати легкий голод. Питання: в який момент вона настільки

зголодніла, що пішла перекусити? Можна розглядати час відразу після їжі як 0, а 1 буде точкою, коли людина наближається до голодування.

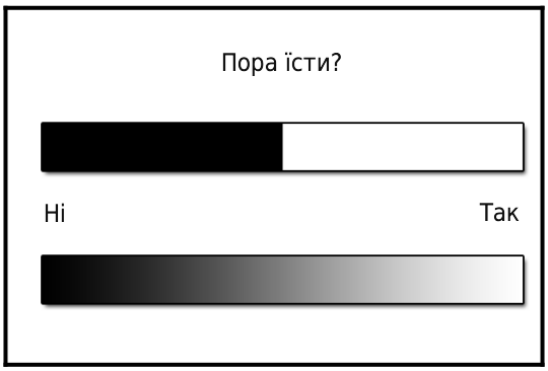


Рис. 1.1 – Нечітка шкала значень

Є багато факторів, які визначають остаточний вибір при прийнятті рішень. Це призводить до іншого аспекту контролерів нечіткої логіки – вони можуть враховувати стільки даних, скільки необхідно. Наприклад, вище було враховано лише одне значення для прийняття такого рішення, а саме час з моменту останнього прийому їжі. Однак існують інші фактори, які можуть вплинути на це рішення, наприклад, скільки енергії витрачає людина, наскільки калорійна їжа тощо. У більшості випадків саме множина вхідних значень впливає на результат, який можна інтерпретувати як «імовірність перекусити».

Системи нечіткої логіки можуть бути дуже гнучкими через їх загальний характер. На основі наданих вхідних даних нечітка логіка забезпечує вихідні дані. Що цей результат означає для, наприклад, гри, залежить виключно від розробника. Крім того, що вхідні дані впливають на рішення, яке полягає в отриманні результату та його використанні у спосіб, зрозумілий комп'ютеру, вихідні дані також можна використовувати для визначення того, скільки чогось потрібно зробити, як швидко або як довго щось відбувається тощо. Наприклад, якщо агент є автомобілем у фантастичній гоночній грі, який має здатність «нітрофорсування», нечітка логіка

може допомогти йому витрачати цей ресурс, щоб їхати швидше. Значення від 0 до 1 може представляти нормалізований проміжок часу, протягом якого він використовує це підвищення, або, можливо, нормалізовану кількість палива для використання.

Серед багатьох інших рішень, які приймаються розробниками при програмуванні ігор, при виборі найкращого способу вирішення того або іншого проектного питання треба оцінити вимоги нашої гри, а також технологічні та апаратні обмеження. Існує певна вартість продуктивності, пов'язана з переходом від простої системи «так/ні» до більш тонкої системи з нечіткою логікою. Це іноді стає однією з причин, які примушують відмовитися від використання нечіткої логіки. Звісно, не будь-яка складна система є кращою за більш просту. Бувають випадки, коли потрібні саме простота та передбачуваність двійкової системи, оскільки вона може краще підійти для гри. Але нерідко більш прийнятним є принцип «все треба робити якомога простіше, але не простіше за це». Стосовно штучного інтелекту, зокрема, його треба зробити настільки простим, наскільки це потрібно у грі, але не простішим, ніж це мінімально необхідно. Штучний інтелект Pac-Man, наприклад, ідеально підходить для цієї гри – він досить простий. Однак така простота, як правило, є недоречною в сучасному шутері або стратегії.

Існує багато способів, якими нечітка логіка може бути корисною при розробці відеоігор. Це досить ефективний інструмент, який бере деякі дані, оцінює їх подібно до того, як це зробила б людина (хоча й набагато простіше), а потім перетворює дані назад у інформацію, яку може використовувати система.

Контролери з нечіткою логікою мають багато варіантів використання в реальному світі. Серед них – системи опалення, вентиляції та кондиціонування повітря, автомобілі, оснащені складними комп'ютеризованими системами, смартфони, в яких, зокрема, операційні системи враховують різні параметри для оптимізації яскравості екрану, пральні машини, в яких

розмір завантаження, забрудненість води, температура та інші фактори враховуються від циклу до циклу для оптимізації використання води, споживання енергії та часу тощо.

1.2 Ігровий штучний інтелект

Штучний інтелект – досить потужний засіб, який широко застосовується в комп'ютерних іграх і не тільки. Взагалі, інтелект можна розуміти як здатність чомусь навчитися і потім застосувати ці знання. Штучний інтелект, що застосовується у відеоіграх, є ілюзією інтелекту. Розумні ігрові істоти не обов'язково повинні чомусь навчатися, але мають принаймні переконати гравця, що вони чомусь навчаються. Ігровий штучний інтелект повинен спостерігати за навколишнім середовищем і реагувати на нього, щоб виглядати розумним. У той час як природні істоти використовують очі, вуха та інші засоби для сприйняття, об'єкти ігрового штучного інтелекту мають у своєму розпорядженні інший набір датчиків. Так само, замість того, щоб використовувати великий складний мозок, програмний код обробляє ці дані і моделює логічну та правдоподібну реакцію на них.

Власне, «штучний інтелект» – це лише загальний термін; його різноманітні реалізації та застосування відрізняються для різних потреб і для вирішення різних груп проблем. Це і комп'ютерний зір з можливістю аналізувати дані для виконання певних операцій, таких як розпізнавання облич, символів та інших об'єктів, і обробка природної мови (*NLP, Natural language processing*) – здатність, яка дозволяє машині читати та розуміти людські мови, і машинне навчання – наука про створення алгоритмів і програм, які можуть вивчати програмно оброблені дані і застосовувати їх для майбутнього навчання. Підмножиною машинного навчання є нейронні мережі, які дозволяють комп'ютерам «навчатися» і завдяки повторному навчанню ставати все ефективнішими у вирішенні великої кількості проблем. Дуже популярна вправа для тестування машинного навчання

нейронної мережі полягає в тому, щоб навчити штучний інтелект розпізнавати значення набору рукописних символів.

Штучний інтелект в іграх бере свій початок аж від самих ранніх ігор, таких як аркадний хіт *Namco Pac-Man*. Штучний інтелект в цій грі був у кращому випадку елементарним, але при цьому кожен із ворогів мав унікальну поведінку. Вивчення такої поведінки та реагування на них додає цій грі величезної глибини та змушує гравців повертатися до неї навіть після 30 років після її випуску.

Робота ігрового дизайнера полягає в тому, щоб зробити гру достатньо складною, щоб вона була захоплюючою, але не настільки важкою, щоб гравець ніколи не міг перемогти. На шляху до цієї мети штучний інтелект є інструментом, що допомагає абстрагувати шаблони поведінки, яких дотримуються ігрові істоти (так звані неігрові персонажі, *NPC*), щоб зробити їх більш органічними та реалістичними.

Ігровий рушій Unity надає низку підходів до реалізації шаблонів штучного інтелекту. Деякі з них працюють «з коробки», а інші доводиться будувати з нуля. Багато концепцій, таких як пошук шляху та навігаційні сітки, є взаємопов’язаними та будуються одна на одній.

Одним з ключових термінів технології штучного інтелекту є так званий агент. Агент – це автономна штучно інтелектуальна сутність. Коли мова йде про штучний інтелект у грі, мається на увазі не конкретний персонаж, а сутність, що демонструє складні моделі поведінки, яку можна назвати інтелектуальною. Цією сутністю може бути персонаж, істота, транспортний засіб або щось інше.

1.3 Скінченні автомати

Скінченні автомати (*finite-state machine, FSM*) можна вважати однією з найпростіших моделей штучного інтелекту, які часто використовуються в іграх. Скінченний автомат в основному складається з заданої кількості

станів, які з'єднані в графі переходами між ними. Сутність гри починається з початкового стану, а потім відстежує події та правила, які ініціюють перехід до іншого стану. Ігровий об'єкт може бути лише в одному стані в будь-який момент часу.

В якості прикладу можна розглянути персонажа-охоронця зі штучним інтелектом у типовій грі-шутері. Його стани можуть бути такими: патрулювання, переслідування та стрілянина (рис. 1.4).

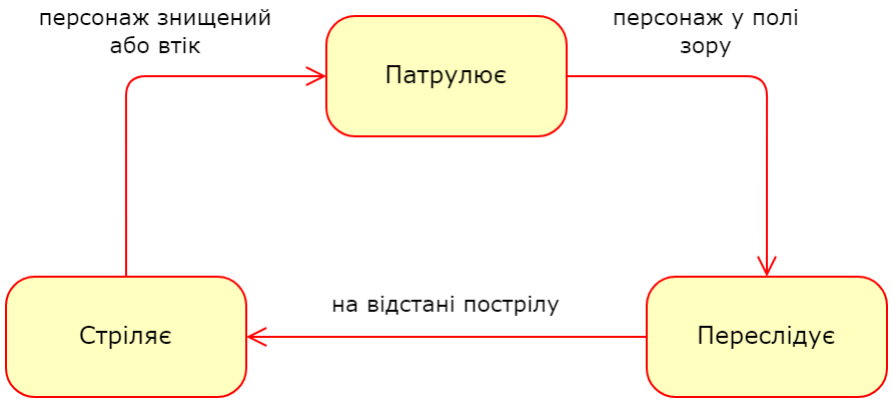


Рис. 1.4 – Стани персонажа-охоронця

В простому скінченному автоматі використовуються чотири типи компонентів:

- стани, в яких може опинятися ігрова сутність або *NPC* (патрулювання, погоня та стрілянина);
- переходи між різними станами (стрілки);
- правила, які використовується для ініціювання зміни стану («персонаж на місці», «персонаж на відстані пострілу» та «персонаж знищений або втік»)
- події, які запускатимуть перевірку правил (зона видимості охоронця, відстань до гравця тощо).

Скінченні автомати зазвичай використовуються в шаблонах штучного інтелекту при розробці ігор, оскільки їх відносно легко реалізувати,

візуалізувати та зрозуміти. Використовуючи прості оператори *if/else* або оператори *switch*, можна легко реалізувати скінченний автомат. Велика кількість станів і переходів потребує спеціальних прийомів управління скінченним автоматом.

1.4 Взаємодія агента штучного інтелекту з навколишнім середовищем

Ігровий штучний інтелект може поводитися достатньо переконливо, якщо агент має можливість реагувати на події навколо нього, оточення, гравця та інших агентів. Подібно до реальних живих організмів, агент може покладатися на зір, звук та інші «фізичні» стимули. Ми можемо отримати доступ до навіть більшої кількості даних у нашій грі, ніж реальний організм може отримати зі свого оточення. Наприклад, нам може бути відомо місцезнаходження неігрового персонажа, незалежно від того, чи він поблизу ігрового персонажа, склад його інвентаря, розташування предметів у всьому ігровому світі та, взагалі, поточне значення будь-якої змінної, яку вибрано для цього агента у програмному коді.

На рис. 1.5 поле зору агента може бути представлено конусом перед ним, а його діапазон слуху – сірим колом, який його оточує.

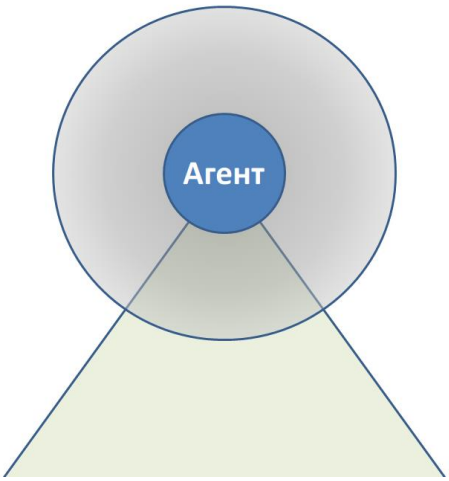


Рис. 1.5 – «Зір» та «слух» агента штучного інтелекту

Подібним способом ми можемо моделювати не лише ті сенсорні системи, що використовуються для біологічних істот (таких як зір, звук або нюх), але й цифрові та механічні системи, які можуть використовувати ворожі агенти (наприклад, радари і локатори роботів, веж, тощо).

У грі Metal Gear Solid, наприклад, ці концепції реалізовані таким чином. Поле зору ворога позначається на міні-карті гравця у формі конуса. Якщо увійти в конус, над головою ворога з'явиться знак оклику, а потім пролунає звуковий сигнал, повідомляючи гравцеві, що його помітили.

1.5 Відстеження шляху та керування

Іноді потрібно, щоб ігрові персонажі зі штучним інтелектом блукали в ігровому світі, дотримуючись приблизно керованого або чітко визначеного шляху. Наприклад, у гоночній грі супротивникам потрібно орієнтуватися на дорозі гоночного маршруту. У стратегіє реального часту наші підрозділи повинні мати можливість дістатися, де б вони не були, до вказаного місця, переміщаючись по місцевості та навколо один одного.

Щоб виглядати розумними, агенти повинні вміти визначати, куди вони прямують, і якщо вони можуть досягти цієї точки, мати здібність прокласти найефективніший маршрут. Також може знадобитися змінити цей шлях, якщо під час навігації з'явиться якась перешкода або виникнуть інші зміни вхідних умов. Слідування шляху та керування також можуть бути представлені через кінцевий автомат.

Припустімо, потрібно, щоб наш персонаж перемістився з точки A в точку B, але на шляху стоїть стіна, і він не може йти прямо до цілі (рис. 1.6). Отже, йому потрібно знайти спосіб дістатися від точки A до точки B, оминаючи стіну.

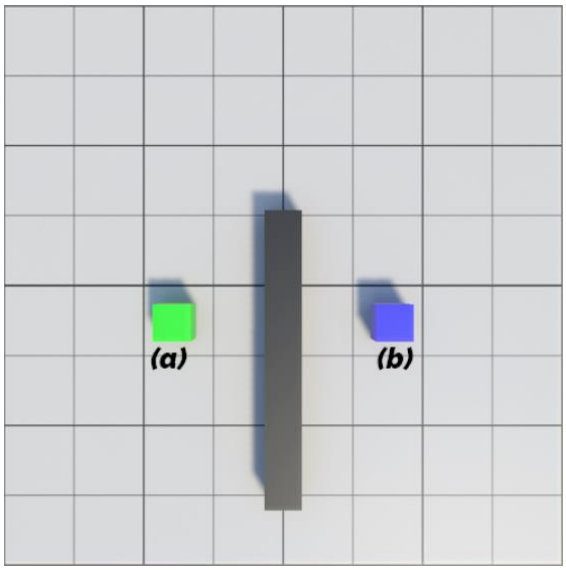


Рис. 1.6 – Проста задача з пошуку маршрута від пункту А до пункту В

Щоб знайти шлях із точки А в точку В, агентові потрібно знати про розташування перешкод. Для цього можна розділити карту на невеликі фрагменти, які представляють всю карту у форматі сітки квадратів (рис. 1.6) (плитки можуть бути й інших форм, наприклад шестикутні або трикутні). Після розбиття на квадрати можна посилатися на карту в 2D-масиві.

Одними з часто використовуваних способів організації пошуку шляху та навігації є реалізація власної системи пошуку шляху за алгоритмом A* або системи маршрутних точок. Проте, при зміні розташування перешкод в реальному часі використання цих способів ускладнюється.

Також для цього можна використовувати вбудовану функцію Unity «Навігаційна сітка» – *NavMesh* (*Navigation Mesh*). Ця функція використовує опуклі багатокутники для представлення областей на карті, до яких може пересуватися об’єкт штучного інтелекту.

Важливою перевагою використання навігаційної сітки є те, що вона дає агенту більше інформації про навколишнє середовище, ніж, наприклад, система маршрутних точок. При використанні NavMesh є можливість безпечно коригувати шлях, тому що є дані про безпечний регіон, у якому можуть пересуватися об’єкти штучного інтелекту. Ще одна перевага

використання навігаційної сітки полягає в тому, що можна використовувати одну й ту саму сітку для різних типів об’єктів штучного інтелекту. Різні об’єкти штучного інтелекту можуть мати різні властивості, такі як розмір, швидкість і можливості пересування як у площині, так і в тривимірному просторі.

Програмне створення навігаційної сітки на основі сцени може бути дещо складним процесом. Починаючи з персональної версії Unity 5, рушій пропонує вбудований генератор навігаційної сітки. Окрім генерації самої сітки NavMesh, Unity надає багато додаткових функцій «з коробки», такі як колізії агентів і пошук шляху на згенерованому графіку через алгоритм A*.

Висновки по першому розділу

Проведені дослідження, результати яких представлені в розділі, обґрунтовують актуальність і доцільність подальшої роботи. На базі розглянутих концепцій нечіткої логіки, штучного інтелекту і скінченного автомату здійснено загальну постановку завдання на дипломну роботу. Це дозволяє перейти до подальшого проектування і реалізації комп’ютерної гри з використанням розглянутих технології.

РОЗДІЛ 2
ПРОЕКТУВАННЯ КОМПОНЕНТІВ КОМП'ЮТЕРНОЇ ГРИ

2.1 Технічне завдання на розробку комп'ютерної гри

2.1.1 Загальна інформація про проект

Комп'ютерна гра «Танковий лабіринт» розрахована на дітей молодшого шкільного віку. Вона має бути нескладною в управлінні, з простою, але привабливою 3D-графікою.

Мета даного концепту – представити загальну ідею дитячої гри «Танковий лабіринт» з упором на простоту управління та цікавий ігровий процес. Гра дозволить розвинути логічне мислення та реакцію у дітей.

2.1.2 Концепція геймплею

Гра «Танковий лабіринт», яка розробляється в даній роботі, доволі проста. Зовнішньо вона дещо нагадує суміш традиційних жанрів симулятора танку, «Tower Defense», головоломки і стратегії реального часу. Гравцеві потрібно провести танк крізь горизонтальний лабіринт зі стін, на яких розташовані вежі. Гравець має клікати на певних точках лабіринту і формувати таким чином маршрут. Поки танк перетинає лабіринт, вежі на шляху намагатимуться знищити його, стріляючи в нього снарядами. Щоб допомогти танку перебратися до кінцевого пункту, гравець може використовувати по мірі проходження лабіринту такі здібності:

- *Boost* (поштовх) – подвоює швидкість руху танка;
- *Shield* (захист) – створює «щит» навколо танку, який блокує зустрічні снаряди;
- *Treatment* (лікування) – частково відновлює «здоров'я» (*HP*) танку.

2.1.3 Мета, правила гри та ігровий процес

Гравець має допомогти танку пройти крізь лабіринт, на стінах якого розташовані вежі, задаючи ключові точки маршруту і використовуючи спеціальні здібності. Вежі на шляху намагатимуться знищити танк, стріляючи в нього розривними снарядами. Мета гри (завдання гравця) – допомогти танку перебратися до виходу з лабіринту (цільової позиції) з мінімальними пошкодженнями.

Правила гри:

1. Ігровий світ представляє собою лабіринт, через який повинен пройти танк.
2. Танку початково дається 100 балів здоров’я.
3. На стінах лабіринту розташовані вежі. Коли танк до них наближається, вони атакують його снарядами. При цьому танк втрачає «здоров’я». Кожне влучення снаряду віднімає 10 балів здоров’я.
4. Гравець може керувати танком безпосередньо або застосувати автопілот для окремих його функцій. В режимі автопілоту танк буде автоматично вибирати маршрут і просуватися ним, а також задіювати інші свої спеціальні здібності для подолання перешкод та захисту від снарядів веж (описані далі).
5. Гра триває доти, доки танк не досягне кінцевої точки лабіринту або поки його не знищать вежі.

Здібності танка:

1. «Поштовх» – активує здатність, що подвоює швидкість руху танка на деякий час. Це дозволяє танку швидко втекти від снарядів у критичних ситуаціях чи випередити вежі.

2. «Захист» – активує здатність, що створює свого роду щит навколо танка на короткий період часу. У цей час танк стає невразливим для снарядів, що дозволяє безпечно проходити через ворожі атаки.
3. «Лікування» – активує здатність частково відновити своє «здоров'я», яке було знижено пошкодженнями від пострілів веж.

Ігровий процес:

1. Гравець управляє танком за допомогою миші, задаючи в лабіринті проміжні цільові точки маршруту. Після кліку миші танк самостійно прямує до вказаної точки, оминаючи стіни і шукаючи маршрут до неї.
2. У міру просування вежі починають атакувати танк, стріляючи в нього снарядами.
3. Гравець повинен в потрібні моменти активувати спеціальні здібності танка, щоб допомогти йому пережити атаку та просунутися далі по лабіринту.
4. Коли танк досягає кінцевої точки лабіринту, рівень вважається пройденим, і гравець переходить на наступний рівень з більш складним лабіринтом та більшою кількістю веж.

Гра закінчується, коли танк успішно досягає кінцевої точки на кожному рівні лабіринту, або гине під обстрілом веж. Після закінчення гри гравцеві пропонується можливість розпочати нову гру або повторно зіграти у вже пройдені рівні.

У грі передбачається використання:

- 3D-графіки з освітленням і тінями;
- фонової музики і звукових ефектів.

Конкретні деталі реалізації, графіка та звукове оформлення можуть бути розроблені додатково на етапі розробки гри.

2.1.4 Додаткові функції гри

1. Автоматична генерація рівнів (лабіринтів з вежами).
2. Різні види поверхонь, по яким може пересуватися танк – автоматично з'являються при генерації лабіринту. Відрізняються кольором та дією на танк: сірі – гальмують рух танка, червоні – роблять його менш керованим.
3. Предмети, які з'являються в деяких місцях лабіринту і за які гравець отримує додаткові можливості.
4. Додаткові здібності танка (окрім вищевказаних):
   - *Hulk mode* (режим Халка): танк збільшується в розмірах протягом певного часу;
   - *Shrink mode* (режим скорочення): танк зменшується протягом встановленого періоду часу (протилежність режиму Халка);
   - *Stealth* (скритність): у цьому режимі вежі не можуть виявити танк;
   - *DMV mode* (режим деформації часу): ця здатність сповільнює час.
5. Додаткові здібності веж: використання більш швидких і більш ушкоджуючих снарядів.
6. Контроль часу проходження ігрового рівня.
7. Реєстрація і облік гравців.
8. Ведення таблиці досягнень.
9. Можливість збереження незакінченої гри в файл і завантаження гри з нього.
10. Налаштування способів керування і параметрів графіки.
11. Керування за допомогою джойстика або шолома віртуальної реальності.

12. Підказки у грі, наприклад, про рух у невірному напрямі.
13. Озвучування підказок голосом.

2.1.5 Споживчі вимоги до ігрового застосунку

1. Використання 3D-графіки з освітленням і тінями.
2. Складність проходження ігрових рівнів – невелика (розрахована для дітей).
3. Керування за допомогою клавіатури і (або) миші.
4. Можливість налаштувань способів керування і параметрів графіки.
5. Фонова музика і звукові ефекти.

2.1.6 Системні вимоги

Мінімальні системні вимоги:

- процесор: Intel Core i3-4130 або AMD A8-5600K;
- відео-карта: вбудована;
- обсяг оперативної пам'яті: 2 ГБ;
- обсяг вільного місця на накопичувачі HDD/SSD: 200 МБ;
- наявність звуковідтворюючих пристроїв;
- операційна система: Windows 7 або вище;

Рекомендовані системні вимоги:

- процесор: Intel Core i5-4590 або AMD Ryzen 5 1500X;
- відео-карта: NVIDIA GTX 1060 або AMD Radeon RX 480;
- обсяг оперативної пам'яті: 4 ГБ;
- обсяг вільного місця на накопичувачі HDD/SSD: 400 МБ;
- вихід: HDMI 1.3, USB 2.0;
- операційна система: Windows 10 або вище.

2.1.7 Вимоги до транспортування і зберігання

Гра буде доступна на хмарному сервісі, а також може бути додана у спеціалізовані ігрові магазини.

Програмна документація поставляється в електронному та паперовому вигляді.

2.1.8 Спеціальні вимоги

Гра повинна повинно мати зрозумілий та лаконічний для користувача інтерфейс.

Документація, створена при реалізації та змінах гри, повинна містити всю інформацію, необхідну для роботи з ним розробників (програмістів та дизайнерів).

Мова програмування – C#, середовище розробки – Unity + Visual Studio.

Проект повинен забезпечувати можливість інтеграції програмного забезпечення з деякими видами периферійного обладнання для керування (миша, клавіатура, джойстик, шолом віртуальної реальності).

2.1.9 Спосіб ліцензування

Планується поширювати програму як freeware-продукт під GNU General Public License.

2.1.10 Порядок контролю та прийняття

Після передачі бета-версії програми замовнику, він має право тестувати гру протягом 7 днів. Після чого замовник повинен прийняти по даному етапі або в письмовій форми пояснити причини відмови від прийняття. У

випадку обґрунтованої відмови розробники повинні виправити помилки і допрацювати продукт.

2.2 Проектування рівнів

Кожен рівень гри представляє собою лабіринт зі стартовою та цільовою (кінцевою) точками. Танк гравця спочатку знаходиться у стартовій точці. Гравець має провести його крізь лабіринт до цільової точки. На стінах лабіринту у певних позиціях розташовані вежі, які атакуватимуть проїжджаючий танк.

Перший рівень гри представляє собою тестовий рівень, проходячи який, гравець знайомиться з гейплеєм і вчиться керувати танком. Схема першого рівня показана на рис. 2.1.

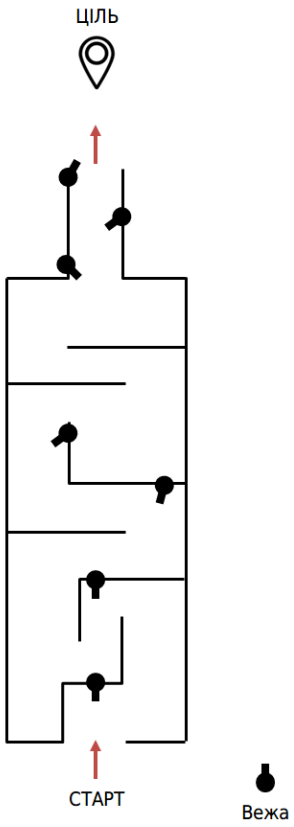


Рис. 2.1 – Схема першого рівня гри

Другий рівень містить вже більш складний лабіринт, для проходження якого гравцеві треба задіяти логіку, увагу і пам’ять. Схема другого рівня гри показана на рис. 2.2.

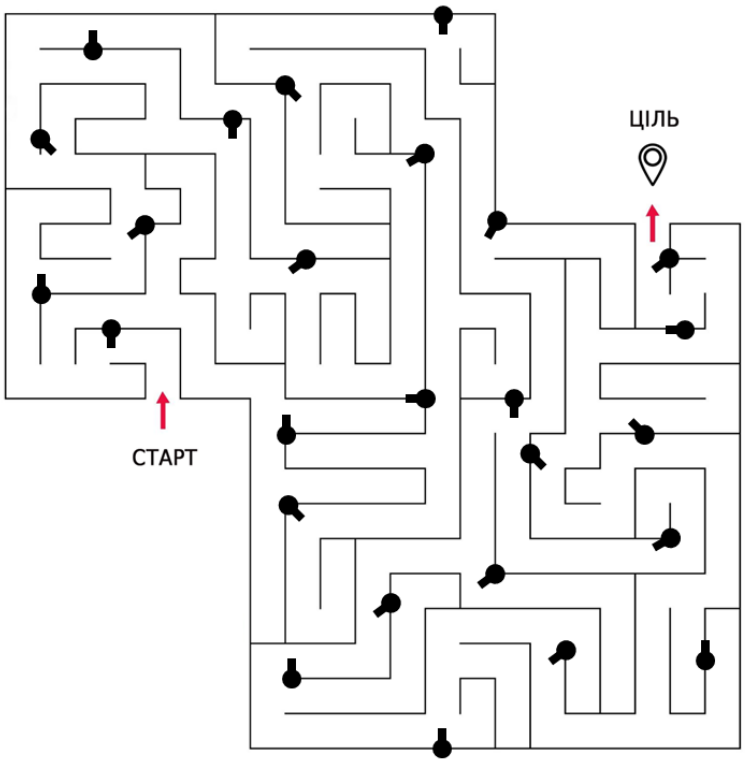


Рис. 2.2 – Схема другого рівня гри

Для проходження рівня гравець має одночасно думати, як пройти лабіринт оптимальним способом, відповідно, направляти танк до проміжних позицій маршруту, відслідковувати рівень здоров’я танку, по мірі проходження маршруту задіювати здібності «поштовх», «захист» і «лікування», щоб мінімізувати пошкодження і відновлювати «здоров’я» танку. Таким чином, гра містить елементи таких жанрів, як «головоломка», «action» і «стратегія реального часу».

2.3 Система «здоров’я» персонажу

Ігровий персонаж (танк) в нашій грі має «здоров’я» (*HP, Health Points*), від рівня якого залежить успішність проходження гри. Початково танк має

100 балів *HP*. Після кожного попадання снаряду з вежі *HP* знижується на 10 балів. Після застосування здібності «лікування» *HP* збільшується на 10 балів. Лікування може бути задіяно не більше деякої кількості разів, в залежності від рівня (2 рази на першому рівні, 5 разів на другому рівні), і не частіше ніж кожні *n* секунд (10 с для першого рівня, 15 с для другого).

В режимі автоматичного лікування танк має самостійно застосовувати цю здібність в необхідних ситуаціях. Таким чином, завданням контролера «здоров'я» є прийняття рішення щодо того, коли треба застосовувати для танка здібність «лікування» залежно від балів здоров'я, що в нього залишилися.

Взагалі, найпростішим способом реалізації цієї задачі є двійкова система процесу прийняття рішення, яка діє за типовим алгоритмом «якщо $HP < k$, застосувати лікування». В такій системі «здоров'я» нашого танку може бути в одному з двох станів – менше *k* або ні.

Нечітка версія цього сценарію починається з визначення стану «здоров'я» танка (рис. 2.3). Зіставимо ймовірність того, що танк використає здібність «лікування», з тим, скільки йому не вистачає здоров'я. Найпростіший варіант – використати лінійну функцію.

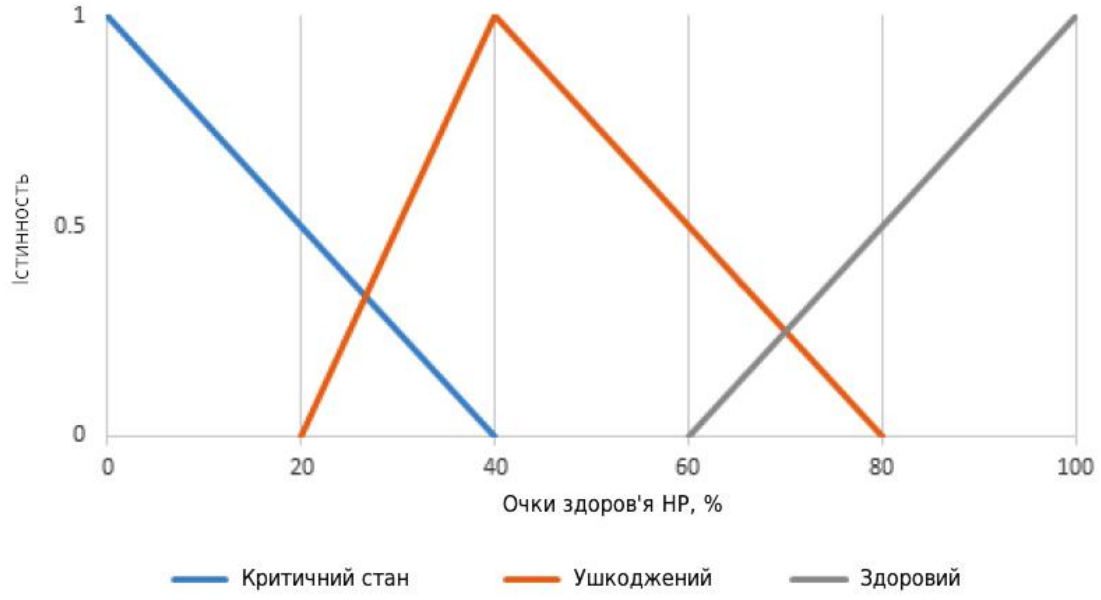


Рис. 2.3 – Функція, що представляє нечіткі значення «здоров'я» танка

Нечітка логіка оперує поняттям функції належності, яке дозволяє визначити, наскільки правдивим є деяке твердження. У цьому випадку нас цікавлять не просто необроблені значення, які дозволяють визначити, чи варто персонажу використовувати заклинання. Натомість ми розбиваємо його на логічні фрагменти інформації, які персонаж танку може використати, щоб визначити, яким має бути його напрямок дій.

Будемо порівнювати три твердження та оцінимо не лише те, наскільки кожне з них правдиве саме по собі, але й яке з них є найбільш правдивим:

- танк у критичному стані;
- танк ушкоджений;
- танк цілий (здоровий).

Це є визначенням ступеня приналежності до набору. Отримавши цю інформацію, контролер нечіткої логіки зможе визначити, що з нею робити далі.

Аналізуючи наведені вище твердження, можна помітити, що два з них можуть бути істинними одночасно. Танк може бути в критичному стані і при цьому ушкоджений, що цілком логічно. Він також може бути дещо ушкодженим і, відповідно, частково цілим. Для кожного твердження можна вибрати порогові значення, але в даному випадку ми оцінимо їх згідно з графіком на рис. 2.3. Вертикальне значення представляє ступінь істинності твердження як нормалізоване число з плаваючою точкою (від 0 до 1):

- при 0 відсотках здоров'я оцінка істинності твердження про критичний стан дорівнює 1 (тобто це твердження найбільш справедливо саме при цьому значенні здоров'я): стан танку є критичним, коли його «здоров'я» втрачено;
- при 40-відсотковому здоров'ї танк ушкоджений, і саме це є найправдивішим твердженням;
- при 100-відсотковому здоров'ї найправдивішим твердженням є те, що танк повністю цілий (тобто «здоровий»).

Усе, що виходить за межі цих абсолютно правдивих тверджень, характеризується певною нечіткістю. Розглянемо приклад, коли поточний рівень здоров'я танку становить 65 відсотків. На діаграмі це можна візуалізувати, як показано на рис. 2.4:

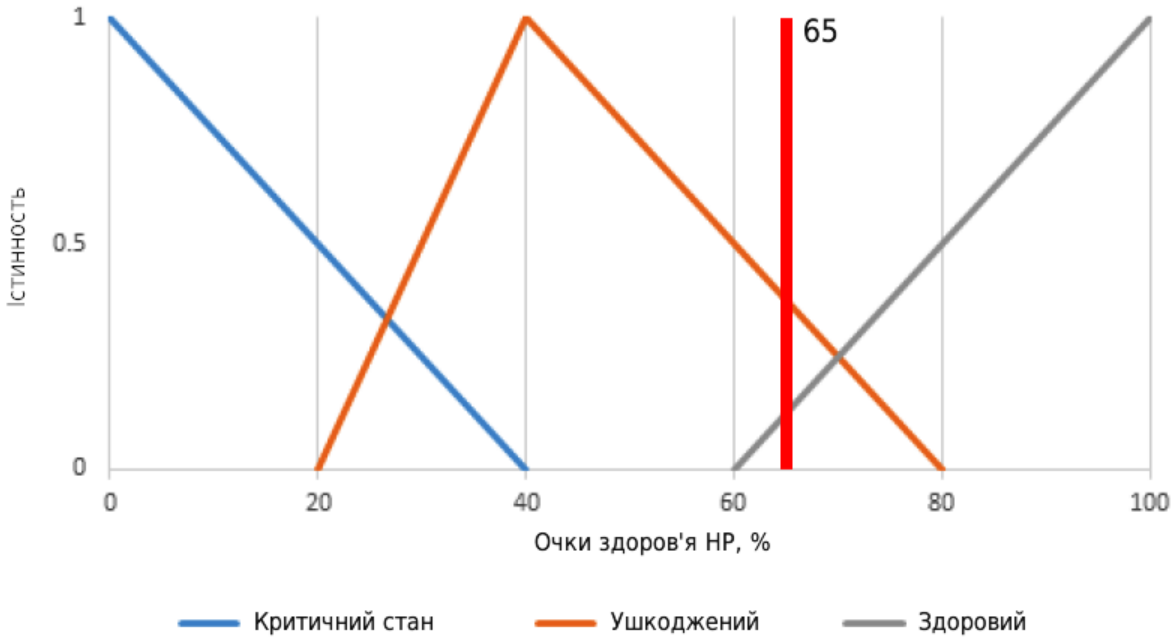


Рис. 2.4 – Рівень «здоров'я» танку 65 %

Вертикальна лінія, проведена через діаграму на 65 %, представляє рівень «здоров'я» танку. Як видно, ця лінія перетинає два набори – «ушкоджений» і «здоровий», що означає, що танк частково ушкоджений, але він також певною мірою «здоровий». Проте, вертикальна лінія перетинає набір «ушкоджений» у вищій точці на графіку. Це значить, що танк більше ушкоджений, ніж цілий; точніше, танк на 37,5 % ушкоджений, на 12,5 % цілий і на 0 % знаходиться у критичному стані.

Висновки по другому розділу

В результаті виконання даного етапу роботи були розроблені технічне завдання, вибрані засоби проектування і реалізації проекту гри «Танковий лабіринт», а також в представлені проектні рішення, які стосуються концепції, геймплею, рівнів та поведінкових аспектів гри. Вони дають змогу перейти до практичної реалізації гри з використанням відповідних інструментальних засобів в обраному середовищі розробки – Unity.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ І ІНТЕГРАЦІЯ КОМПОНЕНТІВ ГРИ

3.1 План реалізації комп'ютерної гри

В цьому розділі описано процес реалізації нашої комп'ютерної гри «Танковий лабіринт» на основі проектних рішень і за допомогою технологій та інструментів, які були розглянуті у попередніх розділах.

Процес розробки гри включає:

- створення моделей танка і вежі;
- створення агента штучного інтелекту для вежі;
- створення агента *NavMeshAgent* для танку;
- створення ігрового рівня і налаштування середовища;
- перевірку зразка сцени.

Результатом першого етапу проекту є реалізація тестового варіанту гри «Танковий лабіринт», який втілює головні ідеї проекту. Фактична логіка гри і поведінка агентів, конкретні показники здоров'я, шкоди та виграшу, можуть і мають бути конкретизовані або змінені на наступних ітераціях розробки.

Для реалізації веж підходить концепція скінченного автомату, оскільки вежі мають обмежену кількість станів і не вимагають додаткової складності дерева поведінки. Вежі також мають знати про своє оточення, а саме, чи є танк поблизу, щоб вони могли по ньому стріляти. Тому для моделювання поля зору та зондування веж доцільно використати сферичний тригер.

Танк повинен мати можливість самостійно орієнтуватися в середовищі, тому для нього використовуються компоненти *NavMesh* і *NavMeshAgent*.

3.2 Створення і налаштування танку

Проект гри містить префаб для танку, який називається *Tank* і розташований у папці *Prefabs*.

Геометрична основа танку представлена на рис. 3.1.

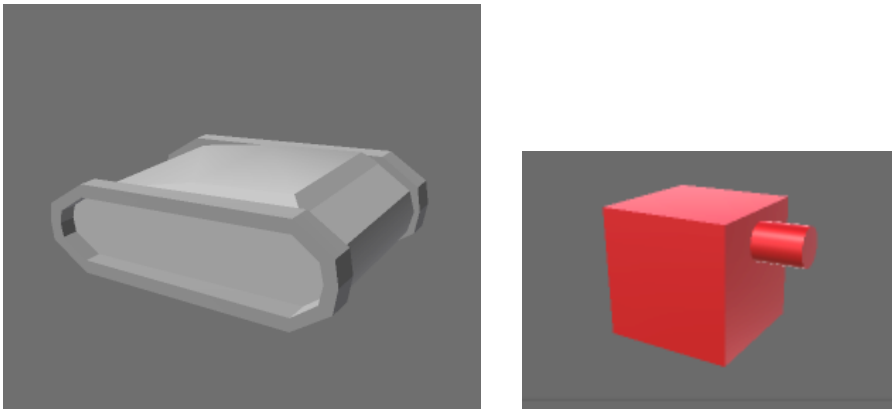


Рис. 3.1 — 3D-модель основи і верхньої частини танку

Параметри префабу танка інспекторі показані на рис. 3.2.

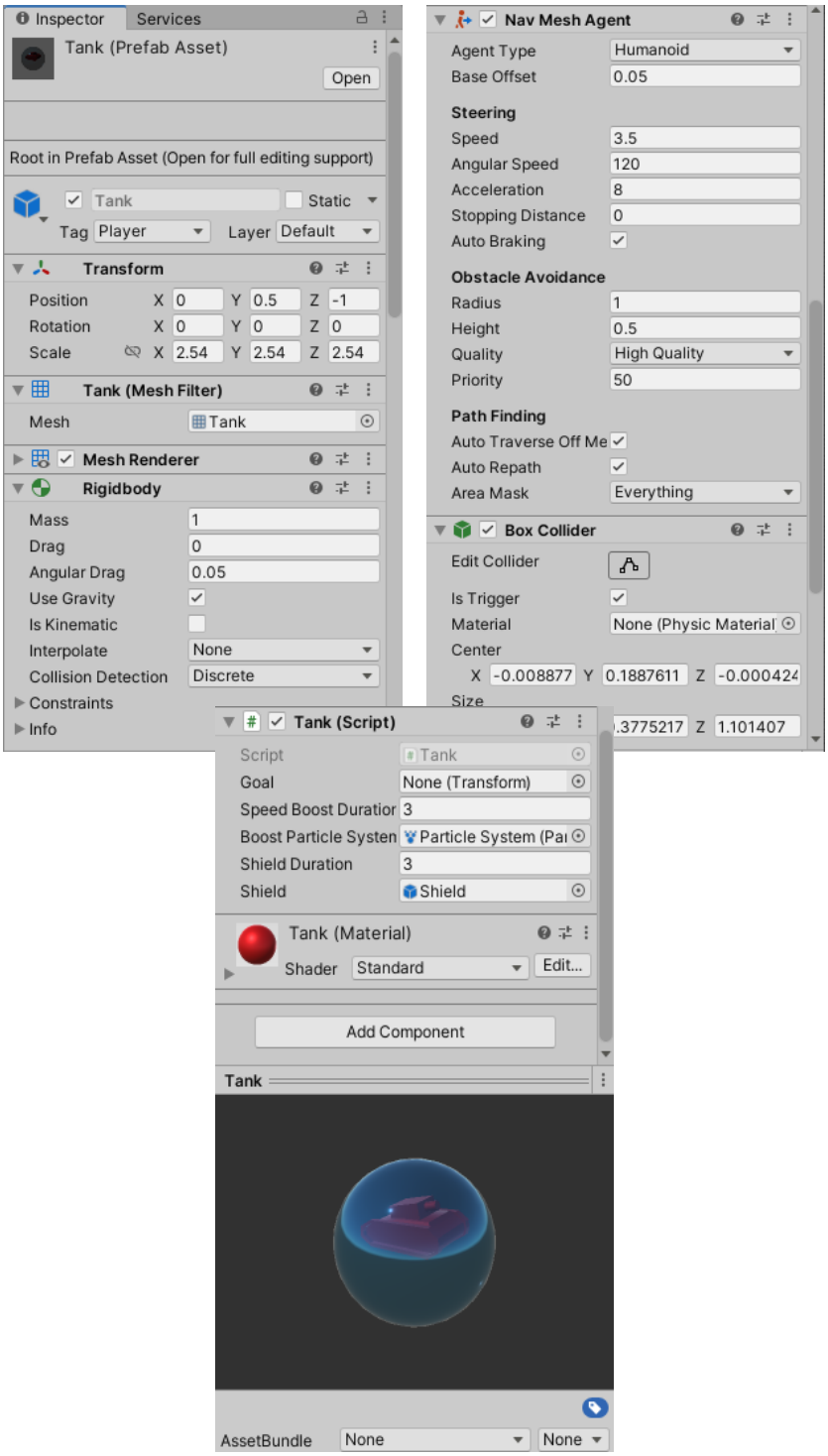


Рис. 3.2 — Налаштування префабу танка в інспекторі

Сам танк є простим агентом з однією метою: дістатися до кінця лабіринту. Як згадувалося раніше, гравець повинен допомогти танку вибратися по дорозі, активувавши його здібності, щоб захистити його від зустрічного вогню з веж.

До префабу танка приєднаний скрипт *Tank.cs*:

```
using UnityEngine;
using System.Collections;

public class Tank : MonoBehaviour {
    [SerializeField]
    private Transform goal;
    private UnityEngine.AI.NavMeshAgent agent;
    [SerializeField]
    private float speedBoostDuration = 3;
    [SerializeField]
    private ParticleSystem boostParticleSystem;
    [SerializeField]
    private float shieldDuration = 3f;
    [SerializeField]
    private GameObject shield;

    private float regularSpeed = 3.5f;
    private float boostedSpeed = 7.0f;
    private bool canBoost = true;
    private bool canShield = true;

    private bool hasShield = false;
```

Спочатку налаштовуються деякі параметри: тривалість здібностей, пов’язані з ними ефекти тощо.

```
    private void Start() {
        agent = GetComponent<UnityEngine.AI.NavMeshAgent>();
        agent.SetDestination(goal.position);
    }

    private void Update() {
        if (Input.GetKeyDown(KeyCode.B)) {
            if (canBoost) {
                StartCoroutine(Boost());
            }
        }
        if (Input.GetKeyDown(KeyCode.S)) {
            if (canShield) {
                StartCoroutine(Shield());
            }
        }
    }
```

Метод *Start* виконує деякі налаштування для танка; він захоплює компонент *NavMeshAgent* і встановлює його пункт призначення рівним цільовій змінній. Далі це описано більш детально.

Метод *Update* використовується, щоб отримати дані про здібності танка. Ми зв'язали клавішу *B* зі здібністю *boost* (посилення), а клавішу *S* – зі здібністю *shield* (щит). Оскільки це тимчасові здібності, як і здатність веж стріляти, їх реалізовано за допомогою співпрограм:

```
private IEnumerator Shield() {
    canShield = false;
    shield.SetActive(true);
    float shieldCounter = 0f;
    while (shieldCounter < shieldDuration) {
        shieldCounter += Time.deltaTime;
        yield return null;
    }
    canShield = true;
    shield.SetActive(false);
}

private IEnumerator Boost() {
    canBoost = false;
    agent.speed = boostedSpeed;
    boostParticleSystem.Play();
    float boostCounter = 0f;
    while (boostCounter < speedBoostDuration) {
        boostCounter += Time.deltaTime;
        yield return null;
    }
    canBoost = true;
    boostParticleSystem.Pause();
    agent.speed = regularSpeed;
}
```

Логіка двох цих здібностей схожа. Метод *Shield()* вмикає та вимикає ігровий об'єкт «shield», який відповідає за здібність «захист» і визначається у змінній в інспекторі. Після того, як мине час, рівний *shieldDuration*, він вимикається, після чого гравцеві дозволяється знову використовувати «захист».

Основна відмінність у методі *Boost()* полягає в тому, що замість того, щоб увімкнути та вимкнути ігровий об'єкт, *boost* викликає метод *Play* у системі частинок, яку було призначено через інспектор, а також встановлює швидкість агента *NavMeshAgent* так, щоб вона подвоювала вихідне значення, перш ніж скинути його в початкове значення в кінці тривалості здатності.

Цей простий шаблон можна використовувати для надання танку нових здібностей у наступних версіях проекту. Також тут можна буде запрограмувати додаткову логіку, щоб кастомізувати здібності *shield* та *boost*.

На сцені є екземпляр танка з налаштованими змінними. Змінній *Goal* буде призначено *transform*-об'єкт з такою ж назвою – він розташований на сцені в кінці створеного лабіринту. Тут також можна налаштувати тривалість здібностей, яка за замовчуванням встановлена на 3 секунди. Також у майбутньому можна поміняти зовнішній вигляд здібностей: задіяти систему частинок, яка використовуватиметься в посиленні *boost*, або деякий ігровий об'єкт, що використовуватиметься для щита *shield*.

Останній фрагмент коду, який потрібно зараз описати, – це код, який керує камерою. Ми хочемо, щоб камера слідувала за гравцем, але лише вздовж його значення *z*, горизонтально вниз по доріжці. Код для реалізації цього виглядає так:

```
using UnityEngine;
using System.Collections;

public class HorizontalCam : MonoBehaviour {
    [SerializeField]
    private Transform target;

    private Vector3 targetPositon;

    private void Update() {
        targetPositon = transform.position;
        targetPositon.z = target.transform.position.z;
        transform.position = Vector3.Lerp(transform.position, targetPositon,
Time.deltaTime);
    }
}
```

Спочатку тут просто встановлюється цільове положення камери рівним її поточному положенню на всіх осях. Далі координата *z* цільового положення прирівнюється до координати *z* цілі, яка налаштована на трансформацію танка. Потім використовується лінійна інтерполяція (*Vector3.Lerp*), щоб плавно переводити камеру з поточного положення в цільове положення кожного кадру.

3.3 Навігація танка при керуванні в ручному режимі

Коли ми вказуємо танку якусь точку на місцевості, до якої йому треба переміститися, він має це зробити певним чином інтелектуально, уникаючи перешкод. Його агент штучного інтелекту має знати, де знаходяться перешкоди, в першу чергу статичні. Уникнення зіткнень між об’єктами, що динамічно рухаються, є другою задачею, відомою як керування поведінкою. Для цього скористаємося інструментом *NavMesh*, який вбудований в Unity. Він представляє сцену у вигляді навігаційної сітки, що дозволяє агентам знайти оптимальний шлях до цілі.

На рис. 3.3 показаний тестовий рівень, який було створено для тестування функціональності системи *NavMesh*. Керування танком здійснюється клацанням по точці, куди танк повинен переміститися. Ця точка помічається жовтим індикатором.

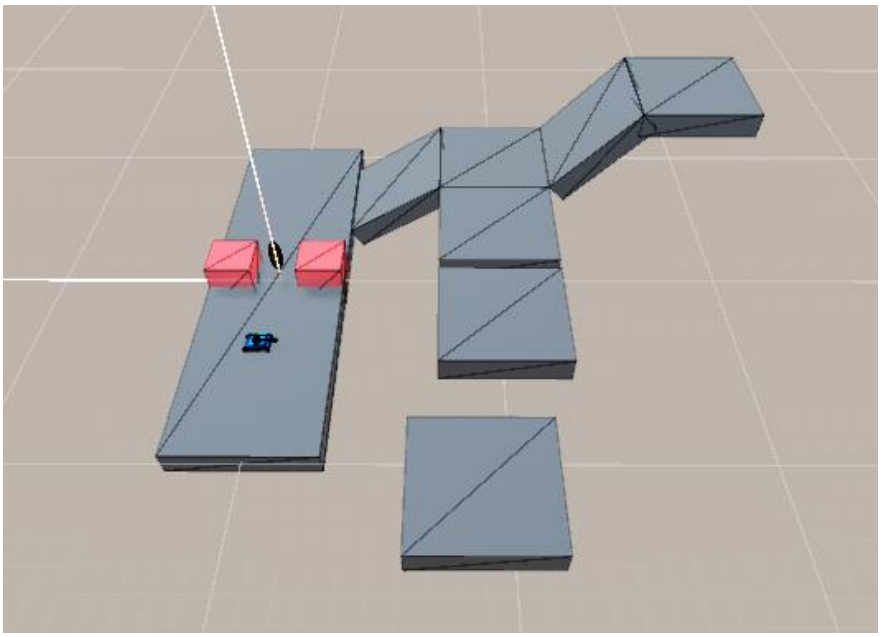


Рис. 3.3 – Сцена для тестування системи NavMesh

По-перше, позначаємо в Інспекторі об’єктів усі геометричні блоки в сцені, які будуть запікатися в NavMesh, як *Navigation Static* (навігаційна статика) (рис. 3.4).

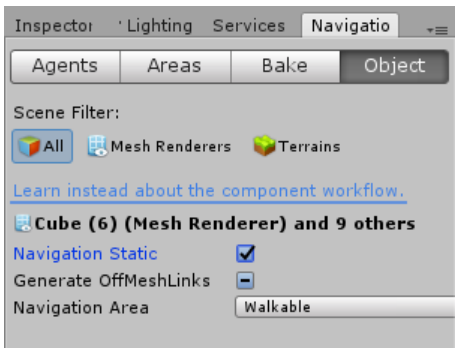


Рис. 3.4 – Налаштування статичної навігації для блоків NavMesh

Далі виконуємо формування навігаційної сітки. Спочатку визначаємо параметри агента для взаємодії з іншими агентами (рис. 3.5).

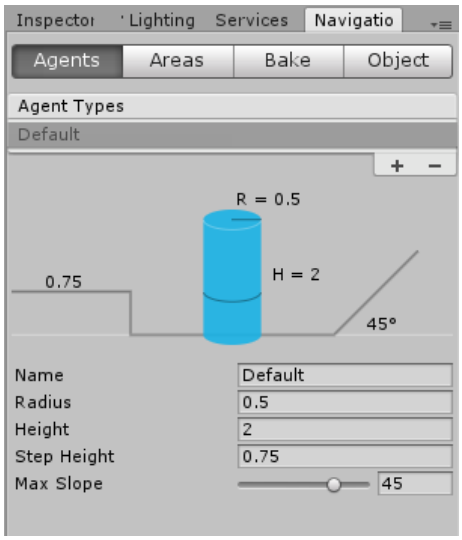


Рис. 3.5 – Налаштування параметрів агента при взаємодії з іншими агентами

Налаштування і призначення основних параметрів:

– *Radius* = 0,5: «особистий простір» агента, який використовується для уникнення взаємодії з іншими агентами;

- *Height* = 2: висота агента, яку він може використовувати для уникнення вертикального стикання (наприклад, при проходженні під якимось речами);
- *Step Height* = 0,75: висота перешкоди, через яку може перелізти агент;
- *Max Slope = 45*: максимальний нахил поверхні, на яку може піднятися агент; таким чином можна зробити деякі круті ділянки карти недоступними для агента.

На вкладці *Area* налаштовуємо стандартні типи областей: *Walkable*, *Not Walkable* і *Jump*, зокрема, призначаємо їм «вартість» (*Cost*) (рис. 3.6). За допомогою областей ми робимо певні зони доступними або недоступними для кожного агента, а також позначаємо їх як більш або менш бажані з точки зору вартості навігації.

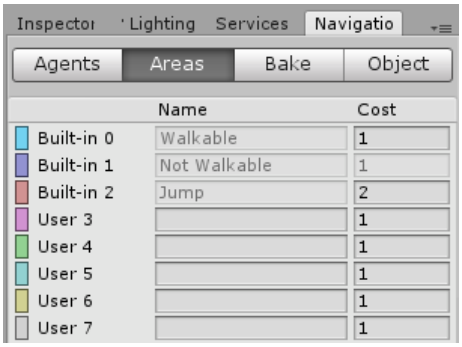


Рис. 3.6 – Налаштування типів областей навігації

На третій вкладці – *Bake* – налаштовуємо і створюємо («запікаємо») фактичну навігаційну сітку для нашої сцени (рис. 3.7). Параметри розміру агента на цій вкладці визначають, як агенти взаємодіють із середовищем, тоді як параметри на вкладці «Агенти» визначають, як вони взаємодіють з іншими агентами та рухомими об’єктами. Параметри *Drop Height* (висота падіння) та *Jump Distance* (відстань стрибка) контролюють, наскільки далеко агент може «стрибнути», щоб досягти якоїсь частини NavMesh, яка не пов’язана напряму з ділянкою, на якій зараз перебуває агент.

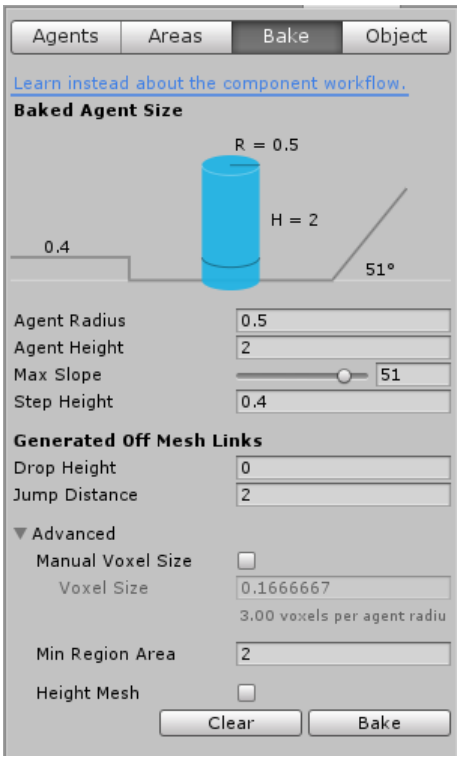


Рис. 3.7 – Налаштування навігаційної сітки для тестової сцени

Розширені налаштування дозволяють задати параметр *Manual Voxel Size* (*Розмір вокселя*) як параметр свого роду якості: чим менший розмір, тим більше деталей можна зафіксувати в сітці. Параметр *Min Region Area* (*Площа мінімальної області*) використовується для пропуску платформ для формування сітки, або для поверхонь нижче заданого порогу. Параметр *Height Mesh* надає більш детальні вертикальні дані для «запікання» навігаційної сітки. Наприклад, це може допомогти зберегти правильне розташування агента під час підйому по сходах.

За допомогою кнопки *Bake* («*Випікати*») створюємо сітку для нашої сцени. На тестовому рівні вона виглядає, як показано на рис. 3.8 (блакитні області).

Після налаштування сцени за допомогою NavMesh налаштовуємо агента на використання цієї інформації. Для цього прикріплюємо до нашого персонажа – ігрового об’єкту *Tank* – компонент *Nav Mesh Agent* (рис. 3.9).

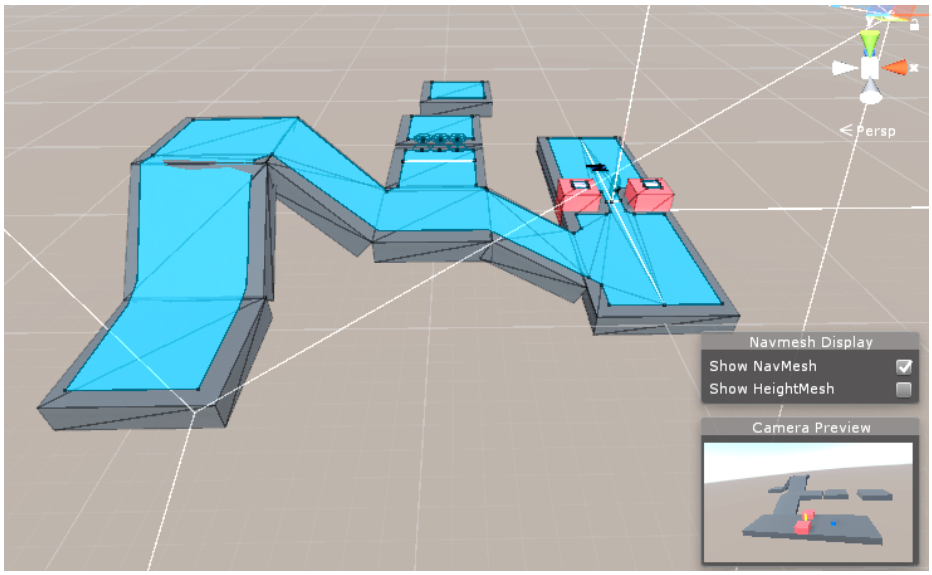


Рис. 3.8 – Сформована навігаційна сітки для тестової сцени

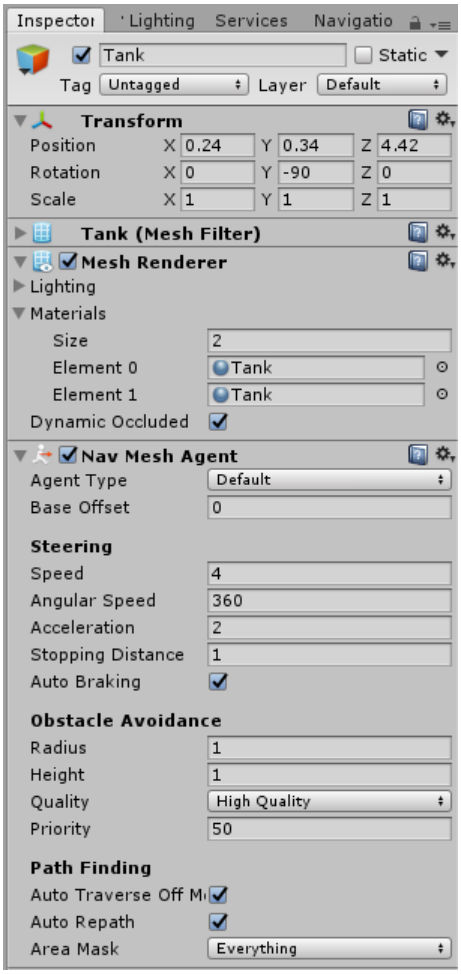


Рис. 3.9 – Налаштування навігаційної сітки для сцени

Налаштовуємо тут, зокрема, такі параметри:

- *Agent Type* (*Тип агента*): *Default* (за замовчуванням);
- *Auto Traverse Off Mesh Link*: увімкнуто (дозволяє агенту автоматично використовувати функцію автоматичного переходу без прив'язки до сітки;
- *Area Mask* (*Маска зони*): *Everything* (вибрати усі зони, які були встановлені на вкладці *Areas* вікна *Navigation*).

В результаті компонент NavMesh виконуватиме майже всю рутинну роботу: розміщення на доріжці, пошук шляху, уникнення перешкод тощо.

Залишається надати агенту цільовий пункт призначення. Для цього створюємо сценарій під назвою *Target.cs*. Цей клас виконуватиме три основні дії:

- вистрілює промінь від початку камери до позиції світу миші за допомогою променя;
- оновлює положення маркера;
- оновлює властивість *destination* (призначення) для всіх агентів NavMesh.

У методі *Start()* ми ініціалізуємо масив *navAgents* за допомогою методу *FindObjectsOfType()*:

```
using UnityEngine;
using UnityEngine.AI;

public class Target : MonoBehaviour
{
    private NavMeshAgent[] navAgents;
    public Transform targetMarker;

    private void Start ()
    {
          navAgents = FindObjectsOfType(typeof(NavMeshAgent)) as
NavMeshAgent[];
    }
```

Метод *UpdateTargets()* проходить через масив *navAgents* і встановлює для цільового призначення вхідні координати типу Vector3.

```
private void UpdateTargets ( Vector3 targetPosition )
{
      foreach(NavMeshAgent agent in navAgents)
    {
            agent.destination = targetPosition;
      }
}
```

У нашій грі використовується підхід *click-to-move* (клацнути для переміщення), тому щоразу, коли гравець клацає, ми випускаємо промінь від камери в світ до курсора миші. І якщо ми щось натискаємо, ми в такий спосіб призначаємо цю позицію попадання як нову цільову позицію (*targetPosition*) для агента. Ми також відповідно встановлюємо маркер цілі для легкої візуалізації цільового призначення агента в грі.

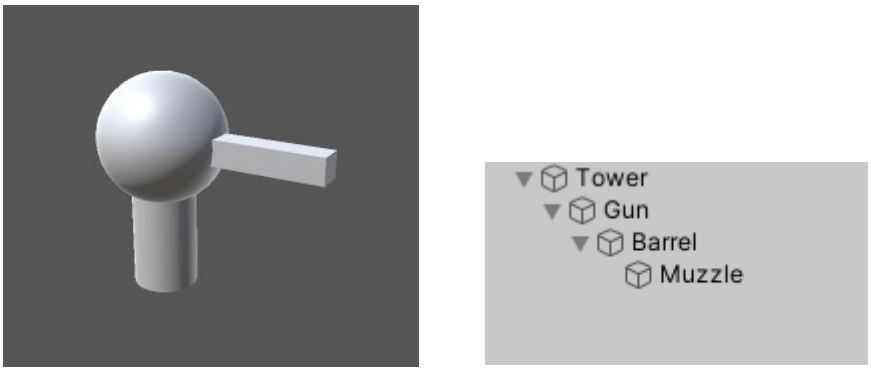
```
private void Update ()
{
    if(GetInput())
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hitInfo;

        if (Physics.Raycast(ray.origin, ray.direction, out hitInfo))
        {
            Vector3 targetPosition = hitInfo.point;
            UpdateTargets(targetPosition);
            targetMarker.position = targetPosition;
        }
    }
}
```

3.4 Створення вежі

Збірна вежа розташовується в папці *Prefabs*. У тестовій версії проекту вона досить проста: це група примітивів, об’єднаних у фігуру, схожу на гармату (рис. 3.10, а).

Ієрархія компонентів вежі показана на рис. 3.10, б. Її складові:

- *Tower* (вежа) – циліндр, який утримує інші частини; технічно це основа вежі;
- *Gun* (гармата) – сфера, встановлена на вежі зі стволом і дулом на ній; ця частина вежі рухається та стежить за гравцем;
- *Barrel* (ствол) і *Muzzle* (дуло) – дуло розташоване на кінчику ствола і використовується як точка появи снарядів, що вилітають з гармати.



а б

Рис. 3.10 – Зовнішній вигляд вежі (а) і ієрархія її компонентів (б)

Ствол гармати кріпиться до сферичної частини вежі. Гармата може вільно обертатися навколо своєї осі під час стеження за об’єктом, що дозволяє їй вести вогонь у напрямку своєї цілі; в усіх інших випадках вона нерухома. Як тільки танк відійде досить далеко, вежа перестане відслідковувати його і змінять положення. На сцені має бути декілька веж, розміщених по всьому рівню.

Гармата є головним діючим елементом вежі. Її властивості в інспекторі Unity Editor показані на рис. 3.11.

Додаємо компоненти, які впливатимуть на логіку гармати.

Sphere Collider (сферичний колайдер) – це, по суті, радіус дії вежі. Коли танк входить у цю сферу, вежа може виявити та зафіксувати його, щоб почати по ньому стріляти. Це реалізація зони сприйняття вежі. Радіус сфери встановлено на 7 одиниць (підібрано експериментально). Прапорець

Is Trigger встановлений у *true*, оскільки нам потрібно, щоб ця сфера не спричиняла зіткнення, а просто запускала тригерні події.

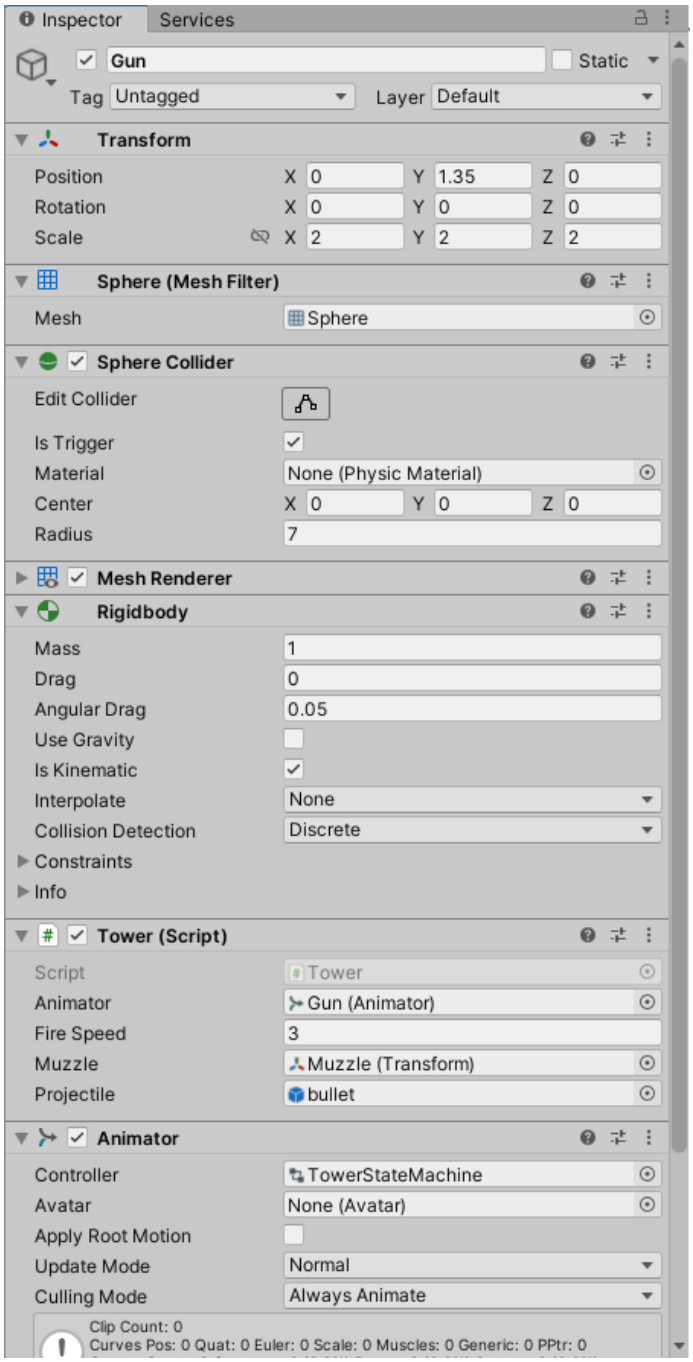


Рис. 3.11 — Гармата *Gun* в інспекторі об'єктів Unity Editor

Rigidbody – цей компонент необхідний для того, щоб колайдер працював правильно, незалежно від того, рухаються об'єкти чи ні. Unity не

надсилає події зіткнень і не запускає події ігровим об’єктам, які не рухаються, якщо вони не мають компонента *Rigidbody*.

Tower – це сценарій, що реалізує логіку для вежі. Він працює в тандемі з скінченним автоматом і поведінкою скінченного автомата, далі вони розглядаються більш детально.

Animator – це скінченний автомат вежі. Він фактично не опрацьовує анімацію.

Перш ніж розробляти код, який керуватиме вежею, необхідно описати відповідний скінченний автомат. Діаграма станів цього автомату, названого *LockedOnState*, представлена на рис. 3.12.

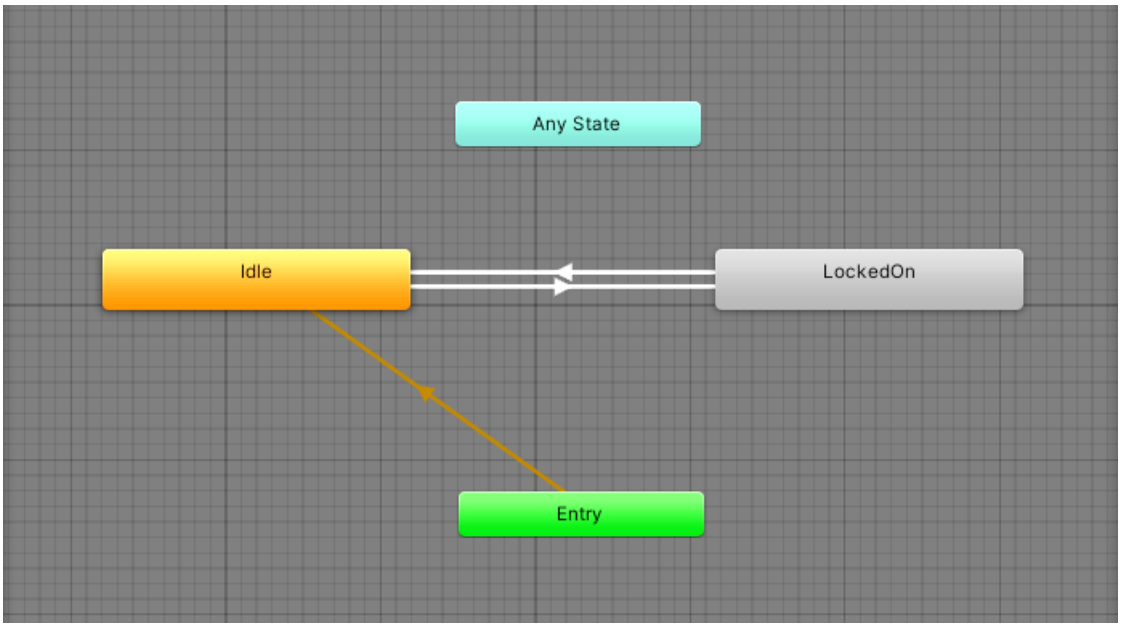


Рис. 3.12 — Діаграма станів скінченного автомату вежі

Перехід від стану за замовчуванням *Idle* (бездіяльність) до стану *LockedOn* (заблоковано, захоплено) відбувається, коли для логічної змінної *TankInRange* (танк в зоні дії) встановлено значення *true*. Коли для цієї змінної встановлено значення *false*, відбувається зворотний перехід.

До стану *LockedOn* приєднаний клас *StateMachineBehaviour*, який описано далі:

```
using UnityEngine;
using System.Collections;

public class LockedOnState : StateMachineBehaviour {

    GameObject player;
    Tower tower;

    // OnStateEnter викликається, коли починається перехід
    //  і скінченний автомат починає оцінювати цей стан
    override public void OnStateEnter(Animator animator, AnimatorStateInfo
stateInfo, int layerIndex) {
        player = GameObject.FindWithTag("Player");
        tower = animator.gameObject.GetComponent<Tower>();
        tower.LockedOn = true;
    }

    // OnStateUpdate викликається в кожному кадрі оновлення між зворотними
    //  викликами OnStateEnter і OnStateExit
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo
stateInfo, int layerIndex) {
        animator.gameObject.transform.LookAt(player.transform);
      }

    // OnStateExit викликається, коли перехід закінчується
    //  і скінченний автомат завершує оцінку цього стану
    override public void OnStateExit(Animator animator, AnimatorStateInfo
stateInfo, int layerIndex) {
        animator.gameObject.transform.rotation = Quaternion.identity;
        tower.LockedOn = false;
    }
}
```

Коли здійснюється вхід у даний стан і викликається метод *OnStateEnter*, відшукується посилання на гравця *player*. У наведеному вище коді гравця позначено тегом «*Player*», що дозволяє отримати посилання на нього за допомогою методу *GameObject.FindWithTag*. Далі отримується посилання на компонент *Tower*, приєднаний до збірної вежі, і встановлюється для його логічного поля *LockedOn* значення *true*.

Поки ми перебуваємо в цьому стані, в кожному кадрі викликається метод *OnStateUpdate*. Усередині цього методу отримується посилання на ігровий об’єкт (*GameObject*) *Gun* (до якого приєднано компонент *Tower*) через надане посилання на *Animator*. Посилання на гармату *Gun* використовується, щоб вона відстежувала танк за допомогою методу *Transform.LookAt*.

В якості альтернативи, оскільки поле *LockedOn* об'єкту *Tower* має значення *true*, цю логіку можна було б обробити в сценарії *Tower.cs*.

Нарешті, коли ми виходимо зі стану, викликається метод *OnStateExit*. В цьому методі виконується невелике очищення: скидається обертання гармати, щоб вказати гравцеві, що вона більше не стежить за танком, і повертається логічний параметр *LockedOn* вежі у значення *false*.

Клас *StateMachineBehaviour* взаємодіє зі сценарієм *Tower.cs*.

```
using UnityEngine;
using System.Collections;

public class Tower : MonoBehaviour {
    [SerializeField]
    private Animator animator;

    [SerializeField]
    private float fireSpeed = 3f;
    private float fireCounter = 0f;
    private bool canFire = true;

    [SerializeField]
    private Transform muzzle;
    [SerializeField]
    private GameObject projectile;

    private bool isLockedOn = false;

    public bool LockedOn {
        get { return isLockedOn; }
        set { isLockedOn = value; }
    }
```

Спочатку оголошуються необхідні змінні та властивості.

По-перше, потрібне посилання на наш скінченний автомат – для цього використовується змінна *Animator*.

Наступні три змінні, *fireSpeed*, *fireCounter* і *canFire*, пов'язані з логікою стрілянини вежі, яка розглянута далі.

Як вже згадувалось раніше, дульний зріз – це місце, звідки вилітають снаряди під час пострілу. Снаряд – це префаб, який необхідно створити.

Нарешті, змінна *isLockedOn* отримується та встановлюється через метод *LockedOn*. Принцип інкапсуляції передбачає дотримання конфіденцій-

ності значень, якщо немає явної вимоги для них бути загальнодоступними. Тому замість того, щоб робити змінну *isLockedOn* загальнодоступною, ми надаємо властивість для віддаленого доступу до неї (у нашому випадку – з поведінки *LockedOnState*):

```
    private void Update() {
        if (LockedOn && canFire) {
            StartCoroutine(Fire());
        }
    }

    private void OnTriggerEnter(Collider other) {
        if (other.tag == "Player") {
            animator.SetBool("TankInRange", true);
        }
    }

    private void OnTriggerExit(Collider other) {
        if (other.tag == "Player") {
            animator.SetBool("TankInRange", false);
        }
    }

    private void FireProjectile() {
        GameObject bullet = Instantiate(projectile, muzzle.position,
muzzle.rotation) as GameObject;
        bullet.GetComponent<Rigidbody>().AddForce(muzzle.forward * 300);
    }

    private IEnumerator Fire() {
        canFire = false;
        FireProjectile();
        while (fireCounter < fireSpeed) {
            fireCounter += Time.deltaTime;
            yield return null;
        }
        canFire = true;
        fireCounter = 0f;
    }
}
```

Далі реалізовані всі інші методи класу, в тому числі логіка вежі. У циклі *Update* перевіряються дві речі: чи захоплений танк зоною видимості вежі та чи можемо вежа стріляти. Якщо і те й інше вірно, запускається співпрограма *Fire()*.

Оскільки не потрібно, щоб вежа постійно стріляла в танк, використовуються змінні, визначені раніше, щоб створити амортизацію між кожним пострілом. Після того, як викликається метод *FireProjectile()* і встановлюється для змінної *canFire* значення *false*, запускається лічильник від 0 до *fireSpeed*, перш ніж знову встановити для *canFire* значення *true*. Метод *FireProjectile()* обробляє екземпляр снаряда та вистрілює його в напрямку, куди вказує гармата, за допомогою *Rigidbody.AddForce*. Справжня логіка снарядів обробляється в іншому місці і розглядається пізніше.

Нарешті, є дві події *OnTrigger* – одна, коли щось потрапляє в тригер, прикріплений до цього компонента, й інша, коли об'єкт залишає цей тригер. Логічний параметр *TankInRange*, який керує переходами для скінченного автомата, встановлюється тут у значення *true*, коли ми входимо у тригер, і повертається у значення *false*, коли ми виходимо. Коли танк потрапляє в сферу «бачення» гармати, остання миттєво замикається на танку (захоплює його), а коли танк залишає цю сферу, ця прив'язка знімається.

3.5 Реалізація стрілянини вежею

Як видно на рис. 3.11, при створенні в інспекторі компонента *Tower* змінній *projectile* (снаряд) призначено префаб під назвою *bullet* (куля). Цей префаб розташований в папці *Prefabs* проекту. Його вигляд і налаштування показані на рис. 3.13.

Ігровий об'єкт кулі досить простий – геометрично це просто яскраво-жовта куля. До неї під'єднано сферичний колайдер, для властивості *IsTrigger* встановлено значення *true*, а також приєднано компонент *Rigidbody* з вимкненою гравітацією. Компонент *Projectile* (Снаряд), приєднаний до префабу кулі, обробляє логіку колізій.

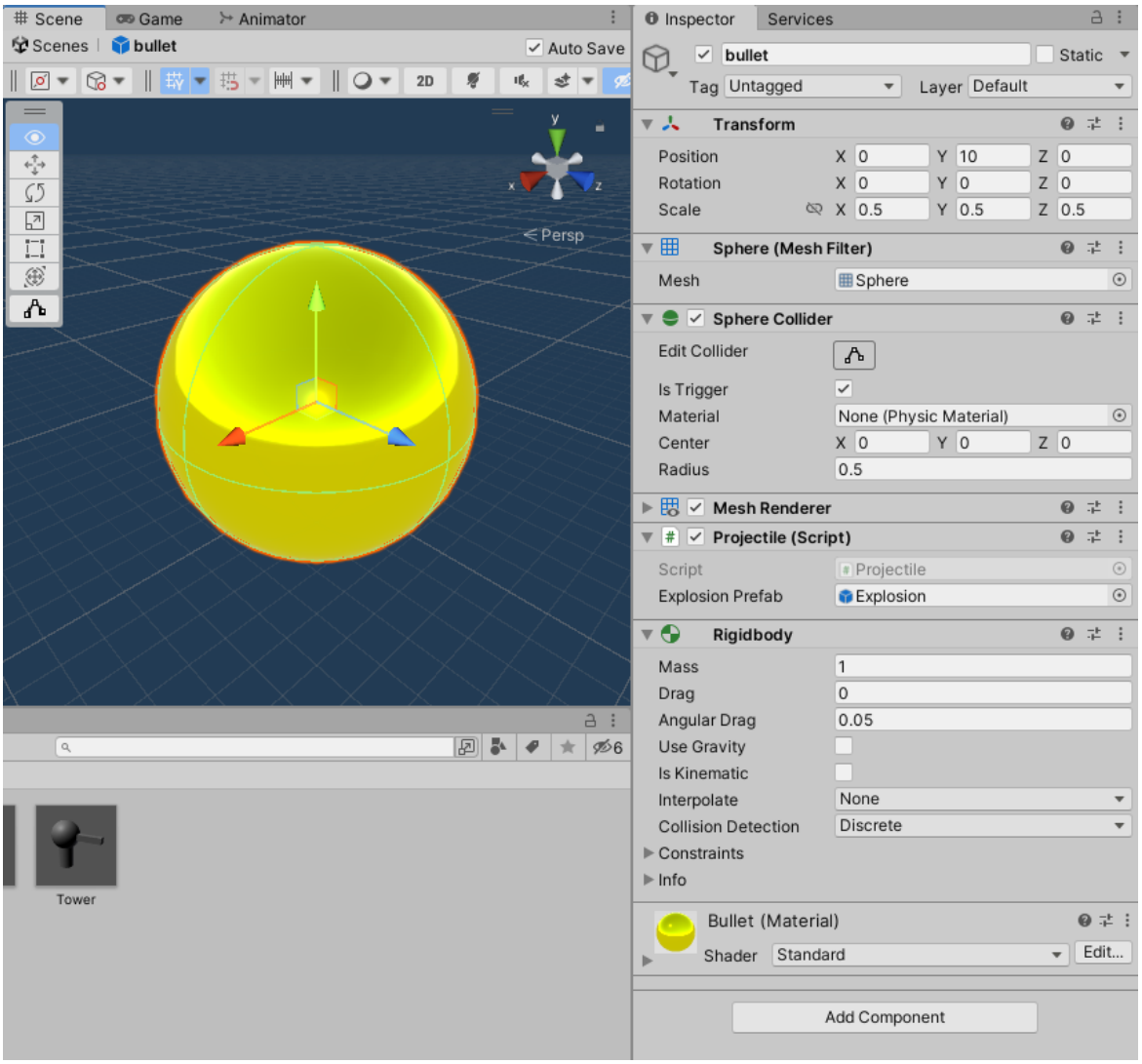


Рис. 3.13 — Префаб кулі *bullet*

Код класу кулі:

```
using UnityEngine;
using System.Collections;

public class Projectile : MonoBehaviour {

    [SerializeField]
    private GameObject explosionPrefab;

      void Start () {   }

    private void OnTriggerEnter(Collider other) {
        if (other.tag == "Player" || other.tag == "Environment") {
            if (explosionPrefab == null) {
                return;
            }
```

```
            GameObject explosion = Instantiate(explosionPrefab,
transform.position, Quaternion.identity) as GameObject;
            Destroy(this.gameObject);
        }
    }
}
```

На ігровому рівні всі підлоги та стіни позначено тегом *Environment* («Оточення»). В методі *OnTriggerEnter* здійснюється перевірка, що є тригером, з яким стикається цей снаряд: танк гравця або оточення. Після цього створюється об’єкт на основі префабу вибуху та знищується снаряд. Префаб вибуху виглядає, як показано на рис. 3.14.

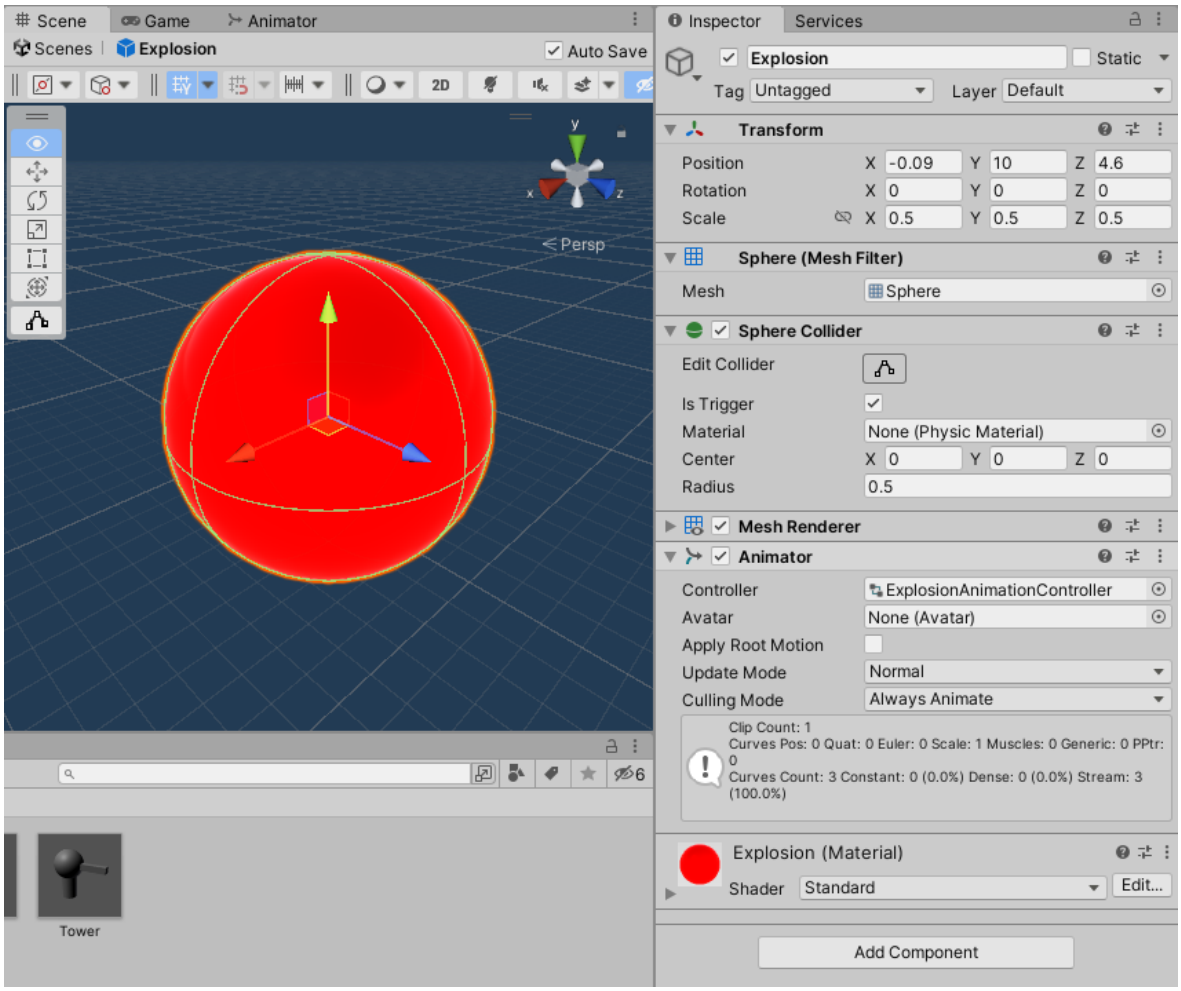


Рис. 3.14 — Префаб вибуху *Explosion*

Як видно, цей ігровий об'єкт дуже схожий на попередній. Тут також є сферичний колайдер із значенням *true* для поля *IsTrigger*. Головна відмінність – компонент аніматора. Коли вибух створюється, він розширюється так само, як і звичайний вибух, а потім використовується кінцевий автомат, щоб знищити екземпляр, коли він виходить зі свого стану вибуху. Контролер анімації виглядає в Unity Editor, як показано на рис. 3.15:

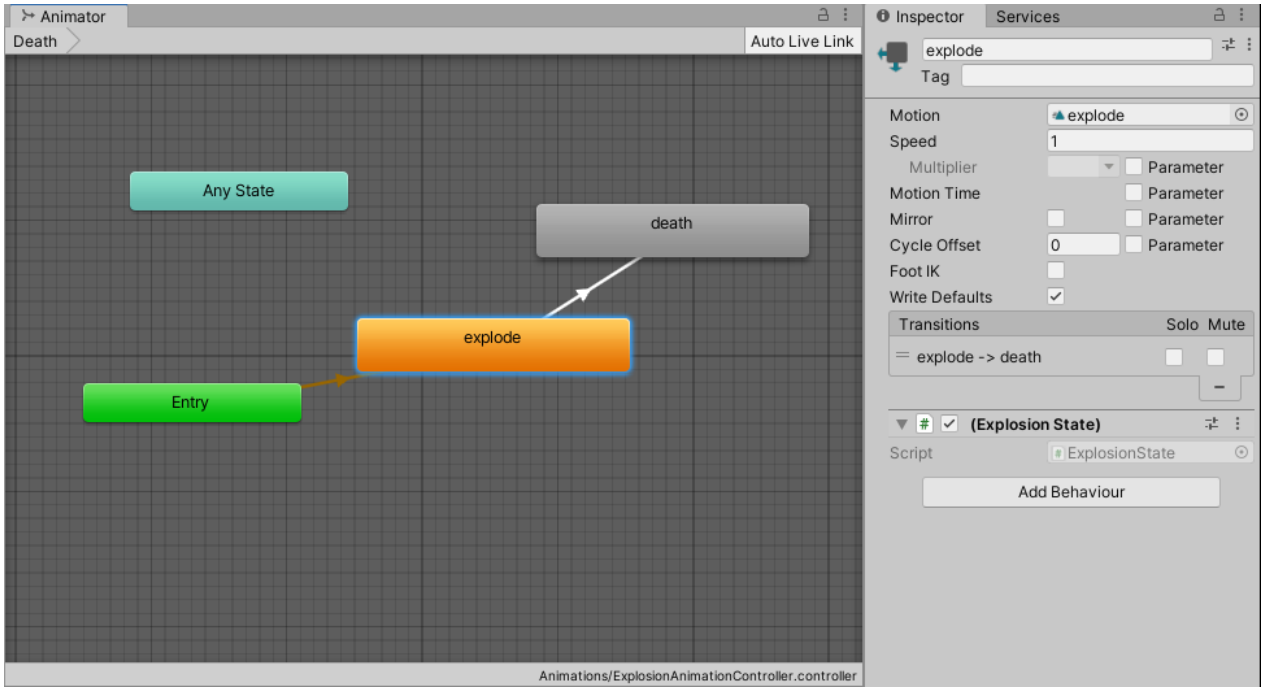


Рис. 3.15 — Контролер анімації, що управляє префабом вибуху *Explosion*

Стан вибуху *explode* має поведінку, яка описується таким програмним кодом:

```
using UnityEngine;
using System.Collections;

public class ExplosionState : StateMachineBehaviour {

// OnStateExit is called when a transition ends and the state machine
finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo
stateInfo, int layerIndex) {
        Destroy(animator.gameObject, 0.1f);
    }
}
```

Все, що тут робиться, це знищення екземпляра об’єкта, коли здійснюється вихід зі стану, що відбувається після завершення анімації. У подальшому гра може доповнюватися тут додатковою ігровою логікою, наприклад, для запуску будь-яких вторинних ефектів, таких як пошкодження, частинки середовища або щось подібне.

3.6 Створення першого рівня і налаштування його середовища

Оскільки танк використовує компонент *NavMeshAgent* для переміщення по навколишньому середовищу, потрібно, як і в тестовій сцені, налаштувати сцену першого рівня за допомогою статичних ігрових об’єктів, щоб процес запікання працював належним чином.

Лабіринт першого рівня влаштований таким чином, що танк має достатньо простору для легкого маневрування. На рис. 3.16 показано загальний макет лабіринту першого рівня:

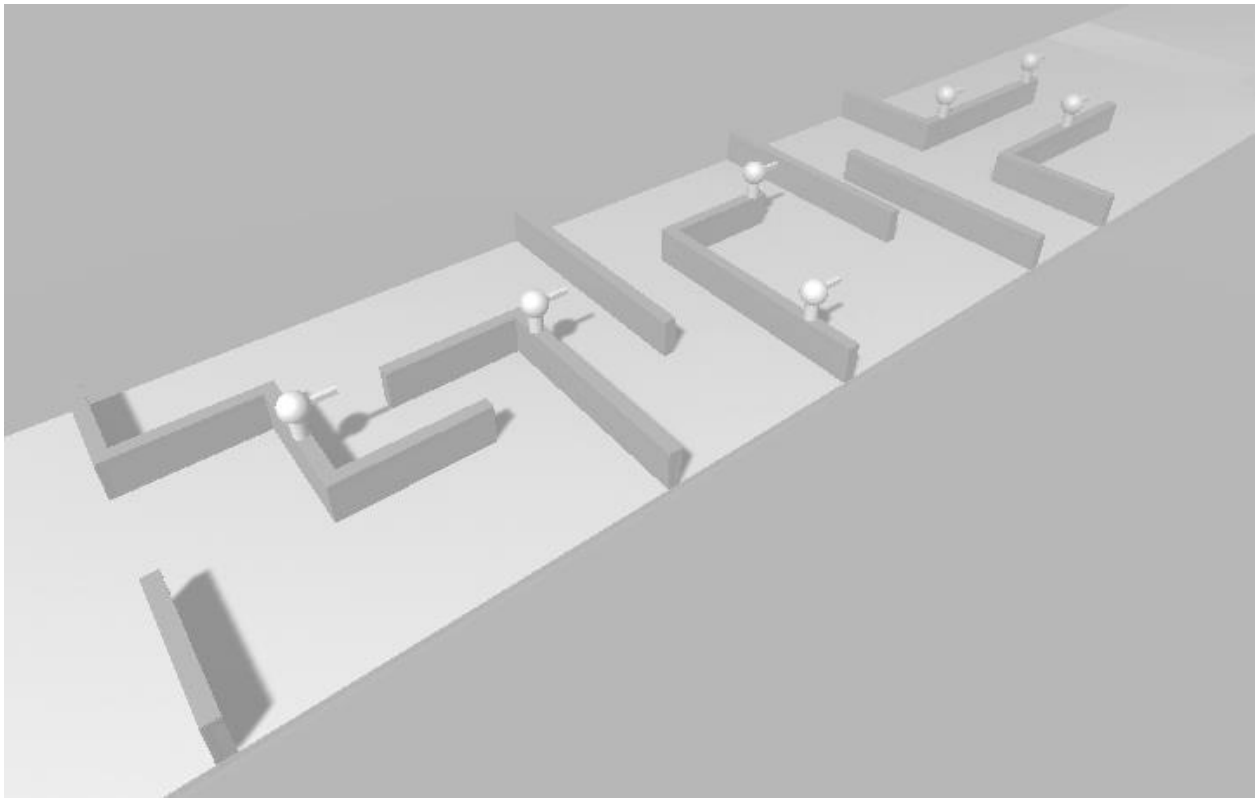


Рис. 3.16 — Лабіринт першого рівня гри

Як можна побачити, у лабіринті першого рівня розташовано сім веж і декілька поворотів. Щоб танк не зачепив стіни, були експериментально підібрані і налаштовані параметри у вікні навігації:

- *Radius* = 1,46 (мінімальна відстань між центром цього агента та будь-якими іншими агентами або перешкодами поблизу);
- *Height* = 1,6 (висота агента для проходження під перешкодами).

При формуванні конфігурації рівня блокуючі об’єкти, які було додано до сцени, були позначені як статичні. Після налаштування конфігурації лабіринту було сформовано навігаційну сітку для сцени. Після «запікання» *NavMesh* була отримана сцена першого рівня, яка показана на рис. 3.17:

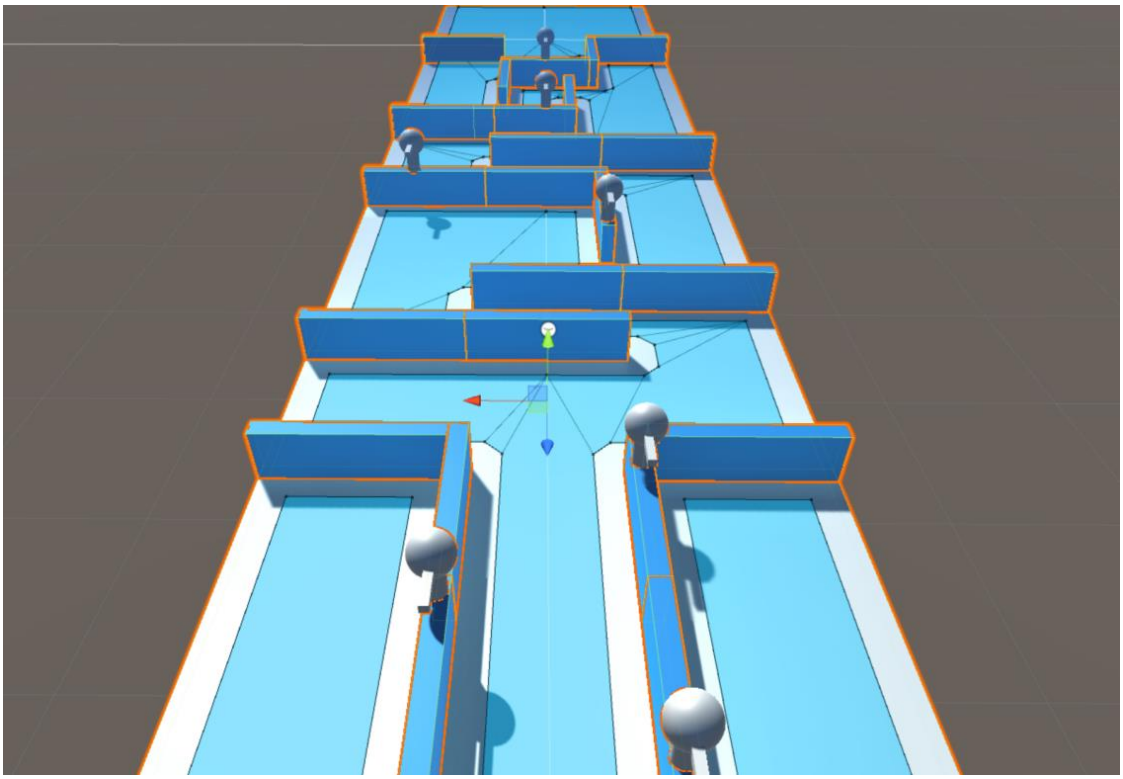


Рис. 3.17 — Сцена першого рівня після запікання *NavMesh*

3.7 Тестування ігрового рівня

На даному етапі приклад сцени готовий до відтворення. Оскільки зараз немає потреби змінювати якісь параметри за замовчуванням, просто натискаємо кнопку відтворення та спостерігаємо за танком. На екран було додано полотно з текстовою міткою, що пояснює гравцеві елементи керування. Це проста інструкція типу «*Клавіша «X» – дія*» (рис. 3.18):

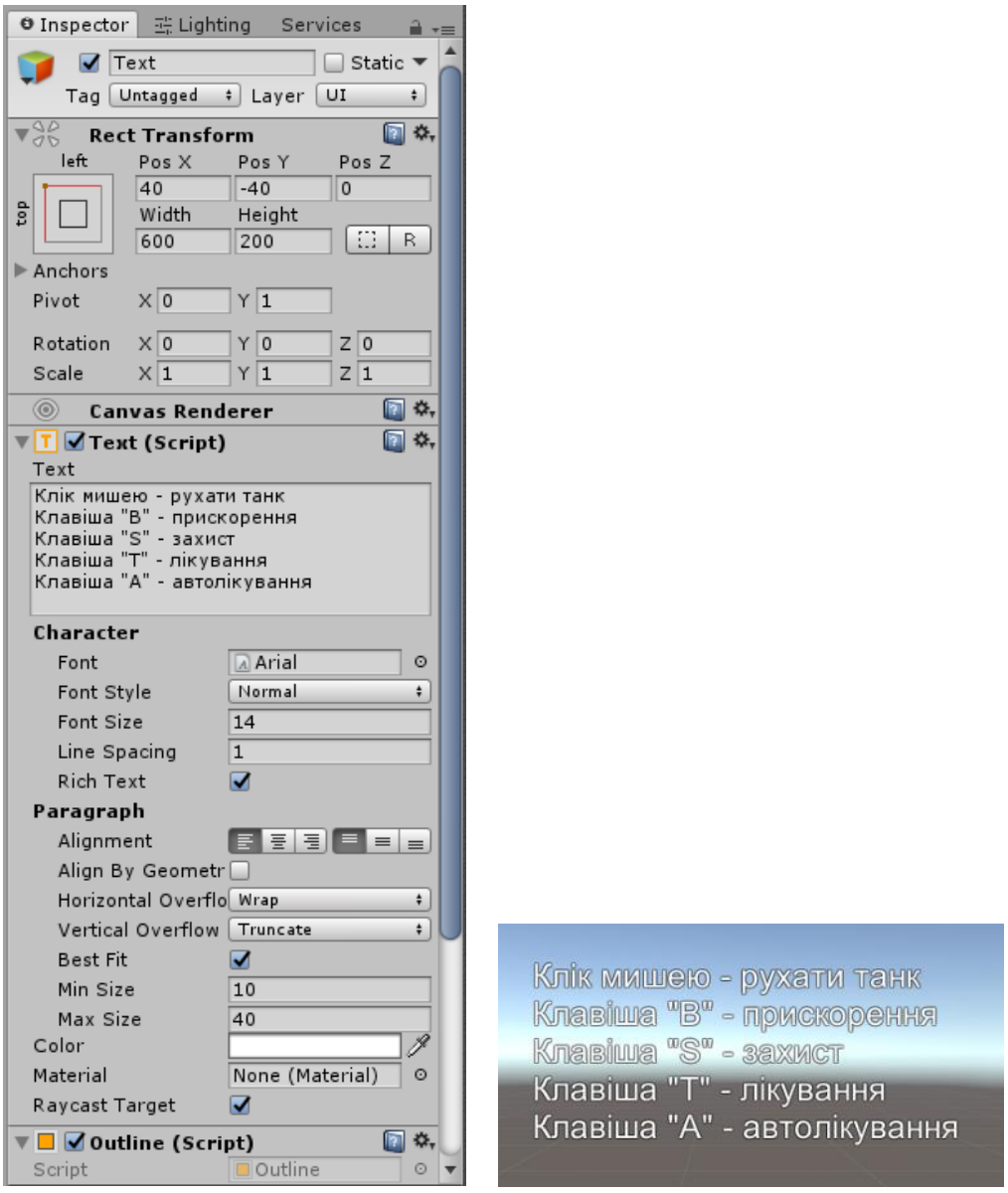


Рис. 3.18 — Міні-інструкція для гравця

На рис. 3.19 показана гра в дії.

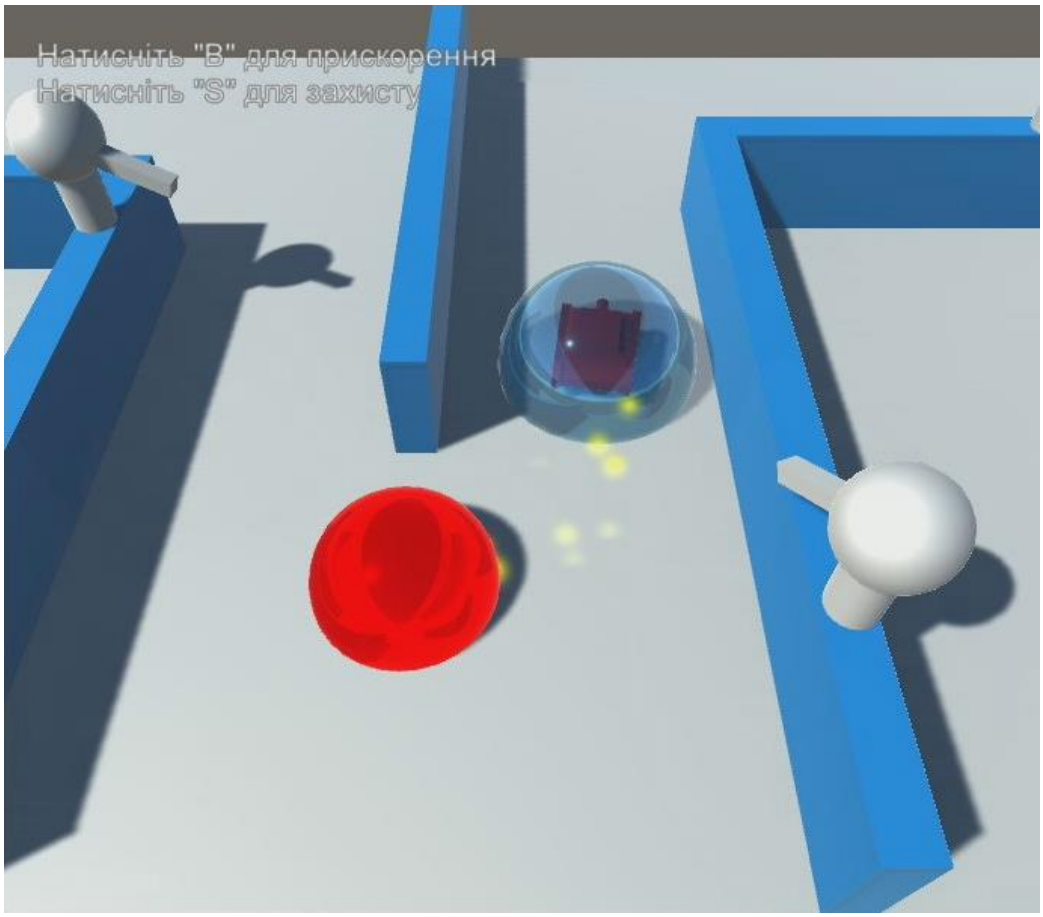


Рис. 3.19 — Перший рівень гри «Танковий лабіринт» в дії

Таким чином, у проекті була використана концепція кінцевих автоматів для створення штучного інтелекту, який керує поведінкою веж. За допомогою функції Unity *NavMesh* реалізована навігація, яка дозволяє танку з простим штучним інтелектом переміщатися по рівню, схожому на лабіринт, крізь низку веж з автономним штучним інтелектом, які намагаються його знищити.

Перша демо-версія гри передбачає розширення. В першу чергу планується додати другий рівень. В майбутньому можна додати ще рівні, нові типи веж, розширити здібності танка, ввести додаткові правила, наприклад, надати танку більш складну, нюансовану поведінку, тощо.

Висновки по третьому розділу

В даному розділі була виконана реалізація першого (прототипного) варіанту гри «Захист танку» за поставленим завданням. У наступних релізах у гру можна додати:

- звуки і фонова музика;
- ігровий рахунок;
- карта рівня;
- нові рівні;
- різні типи поверхонь лабіринта з різними властивостями;
- нові здібності танка;
- автоконфігуратор рівнів;
- приборна панель управління захисними функціями танку.

Також у грі можна поліпшити графіку: додати текстури для «землі» і стін, використати більш деталізовану модель танку, ефект підняття пилу з землі тощо. Можна також додати анімацію перемоги, коли гравець дістається до кінця лабіринту тощо.

РОЗДІЛ 4
ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

4.1 Організаційне і маркетингове обґрунтування проекту

Метою даної дипломної роботи є створення програмного застосунку комп'ютерної гри для платформи Windows. Об'єктом безпосередньої розробки є програмний продукт – комп'ютерна гра з використанням технологій нечіткої логіки, кінцевого автомату і штучного інтелекту. При виконанні дипломної роботи програмний застосунок був створений за допомогою ігрового рушія і середовища розробки Unity.

Область використання даного проекту є сфера розваг, комп'ютерні ігри.

По масштабу проект, що розробляється, відноситься до малих проектів, які невеликі по масштабу, прості і обмежені об'ємами даних і ресурсів.

По термінах реалізації проект короткостроковий (термін реалізації до 3 років). По складності проект, що розробляється, є проектом середньої складності.

По характеру цільового завдання – комбінований проект. По характеру проекту – розважальний проект.

За типом комп'ютерна гра, яка розробляється в даному проекті, є казуальною. Тобто, вона не потребує спеціальних пристроїв для грання, має прості правила і має короткий цикл проходження рівнів.

Розробка казуальних відеоігор є однією з потужних тенденцій у сучасній геймінг-індустрії. Взагалі геймінг-індустрія постійно зростає і стає все більш значимою в сучасному світі, а казуальні ігри – все більш популярними серед геймерської спільноти. За даними дослідження Newzoo [3], обсяг ринку казуальних ігор у 2022 році склав 2,8 мільярда доларів, і очікується його подальше зростання до 6,1 мільярда доларів до 2025 року. Цей ріст

обумовлений зростанням кількості смартфонів та мобільних пристроїв, що використовуються для геймінгу.

Казуальні ігри вже давно знаходяться в полі зору розробників та інвесторів через свій комерційний успіх. Вони вирізняються високою монетизацією та вдалим використанням рекламних можливостей. Крім того, вони є одними з найприбутковіших жанрів мобільних ігор, забезпечуючи значні доходи через внутрішньоігрові покупки та рекламу.

Казуальні ігри привертають різноманітну і дуже широку аудиторію будь-якого віку, що є перевагою для їх успіху. Аналітичне агентство DFC Intelligence повідомило, що на даний момент у всьому світі в ігри грають близько 3,7 мільярдів людей. Одним з прикладів надпопулярних казуальних ігрових проектів є Candy Crush. Генеральний директор проекту Тодд Грін нещодавно заявив, що гру завантажили понад 3 мільярди разів, що зробило її рекордсменом за кількістю завантажень за всю історію ігрової індустрії [4].

На основі наведених статистичних даних можна зробити висновок, що розробка даної казуальної гри має потенціал комерційного успіху.

Ринком збуту даного продукту є світовий ринок. Потенційний споживач – будь-який користувач, який хоча б іноді грає у комп'ютерні ігри. Очікуваною конкурентною перевагою являється низька ціна продукту.

Для досягнення мети були проаналізовані сучасні тенденції у розробці програмних застосунків, досліджені засоби розробки програмних застосунків і деякі проекти конкурентів.

Передбачається поширення продукту за допомогою продажу його компанії замовнику, або самостійне розміщення на платформах – ігрових магазинах, таких як Steam, itch.io, Kongregate. Так як гра не відрізняється технологічністю, та виконана переважно однією людиною за короткий строк, планується виставлення мінімально можливої ціни на усіх платформах. Це 0,49$ для Steam, та 1$ для itch.io. Kongregate працює за моделлю розділення прибутку від реклами. Реклама продукту не передбачена.

4.2 Розрахунок науково-технічної ефективності проекту

Умови відкритої ринкової економіки сприяють розширенню спектра оцінки ефективності науково-технічних розробок. Це призводить до збільшення числа основних категорій ефективності НДДКР, які потрібно визначити з метою проведення цієї оцінки.

«До них належать:

- науково-технічний ефект, який проявляється у підвищенні науково-технічного рівня, поліпшенні параметрів техніки і технологій, що випливає з відкриття нових законів та закономірностей у природі, а отже, і нових технологічних засобів виробництва речовин, матеріалів та видів продукції;
- економічний ефект розробки та впровадження комп'ютерної ігри полягає в забезпеченні економічних результатів для кожного виробничого суб'єкта та в цілому для економіки країни. Економічна ефективність проекту відображає вплив його результативності на розвиток галузей, регіонів та підприємств, що беруть участь у впровадженні новітніх технологічних рішень;
- соціальний ефект, що відображає зміни умов діяльності людини в суспільстві. Його прояв спостерігається в змінах характеру та умов праці, підвищенні життєвого рівня населення, поліпшенні побутових його умов, розширенні можливостей духовного розвитку особистості, у змінах стану довкілля;
- маркетинговий ефект відображає потреби ринку в наукових дослідженнях і розробках та можливість їх реалізації.

Науково-технічну ефективність (НТЕ) результатів прикладних робіт визначаємо на основі показників науково-технічного рівня. Оцінка науково-технічної ефективності НДДКР відбувається на основі показника ($О_{НТЕ}$), який представляє собою ступінь досягнення максимально можливого рівня, значення якого дорівнює 1 (одиниці)» [12]:

$$O_{НТЕ} = K^{Ф}_{НТЕ} / K^{П}_{НТЕ}, \quad (4.1)$$

де $K^{Ф}_{НТЕ}$ – показник (коефіцієнт) фактичного рівня науково-технічної ефективності;

$K^{П}_{НТЕ}$ – показник (коефіцієнт) потенційно можливого рівня науково-технічної ефективності (дорівнює одиниці).

Значення показника $K^{Ф}_{НТЕ}$ визначаємо на основі шкали експертних оцінок.

Таблиця 4.1

Шкала експертних оцінок для виміру рівня науково-технічної ефективності проектів

№	Групи показників	Характеристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
1	Науково-технічний рівень	Перевищує кращі світові аналоги	10	0,30
		Відповідає світовому рівню	7 – 9	
		Нижче кращих світових аналогів	5 – 6	
		Перевищує кращі вітчизняні аналоги	3 – 4	
		Відповідає вітчизняному рівню	1 – 2	
		Нижче вітчизняного рівня	0	
2	Перспективність	Першочергова значущість	8 – 10	0,25
		Значущий	5 – 7	
		Корисний	1 – 4	
3	Потенційний масштаб практичного використання	Світовий ринок	10	0,20
		Галузі національної економіки	7 – 9	
		Галузь (регіон)	3 – 6	
		Окремі підприємства (об’єднання)	1 – 2	
4	Ступінь вірогідності досягнення позитивних результатів	Великий	10	0,25
		Середній	5 – 9	
		Малий	1 – 4	

«Об’єкт оцінки і аналоги, які порівнюють за однаковими показниками, наведеними у співставленому вигляді відхилення в значеннях кожного з показників, мають бути однаковими для варіантів, що порівнюються.» [12]

4.3 Проведення оцінки науково-технічного рівня розробки

Визначаємо $K^{Ф}_{НТЕ}$ на основі експертної оцінки науково-технічного рівня розробки.

«З цією метою:

- розробляємо перелік специфічних показників, необхідних для виміру науково-технічного рівня розробки;
- формуємо групу аналогів, які реалізовані на світовому і вітчизняному ринках;
- здійснюємо відповідні розрахунки для співставлення показників і визначення балів по таблиці 4.1.

До числа специфічних показників відносять:

- для нової техніки: продуктивність, споживання інженерних ресурсів на виробітку одиниці продукції, потреба в робочих, які обслуговують обладнання, експлуатаційні витрати на одиницю продукції;
- для нових матеріалів і речовин: вміст корисних речовин для виробітки готової продукції, питома вага відходів у загальному обсязі переробленої сировини, вартість одиниці нового матеріалу, додаткові витрати на екологічну компенсацію;
- для нових технологій: якість виробленої продукції, енергоємність і трудомісткість продукції, собівартість одиниці продукції.» [12]

На основі співставлення даних таблиці 4.2 встановлюємо бали по характеристиках чотирьох груп і на цій основі розраховуємо значення інтегрального показника *НТЕ*:

$$НТЕ = \Sigma\, Б_i \cdot К_i^3\,, \qquad (4.2)$$

де $i = 1 \div 4$,

$Б_i$ – бали (рейтингове число),

$К$ – коефіцієнт значущості показників.

Таблиця 4.2

Порівняльні показники для виконання оцінки НТЕ

ПОКАЗНИКИ	Варіанти технології	
	Розробленої	співвідносної (аналога)
Рівень новізни	Високий	Середній
Якість продукції	Висока	Середня
Споживання на 1 версію продукту – тепла, Гкал – електроенергії, кВт·годину – води, $м^3$	 6 800 3	 8 1200 4
Трудомісткість виробництва, людино-годин	90	60

Рівень науково-технічної ефективності НДДКР розраховано на основі наведених даних прикладу (таблиця 4.3).

Таблиця 4.3

Експертна оцінка і розрахунок величини інтегрального показника *НТЕ*

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	НТЕ
		1	2	3		
1	Науково-технічний рівень	6	7	8	7	2,45 (7 · 0,35)
2	Перспективність	7	6	6	6,33	2,22 (6,33 · 0,35)
3	Потенційний масштаб практичного використання	7	8	6	7	1,4 (7 х 0,20)
4	Ступінь вирогідності досягнення позитивних результатів	8	9	9	8,66	0,87 (8,66 · 0,1)
Всього						6,94

$$НТЕ = 7 \cdot 0{,}35 + 6{,}33 \cdot 0{,}35 + 7 \cdot 0{,}20 + 8{,}66 \cdot 0{,}1 = 2{,}45 + 2{,}22 + 1{,}4 + 0{,}87 = 6{,}94.$$

Отриманий результат слід порівняти з максимально можливим значенням, яке дорівнює 10 балам (10·0,35 + 10·0,35 + 10·0,20 + 10·0,1).

Отже, оцінка рівня НТЕ може бути зроблена за допомогою інтегрального коефіцієнта оцінки НТЕ ($К_{НТЕ}$):

$$К_{НТЕ} = \frac{НТЕ}{10} \cdot 100\ \% \qquad (4.3)$$

На основі даних таблиці 4.3, можна дійти до висновку, що $К_{НТЕ}$ відповідає 69,4 %, тобто:

$$\frac{6{,}94}{10} \cdot 100 = 69{,}4.$$

В тому випадку, коли значення $К_{НТЕ}$ перевищує середнє значення, яке дорівнює 5,0, має бути зроблено висновок про достатній рівень НТЕ:

- цілком достатній 5,0 – 6,0;
- достатній 6,1 – 8,0;
- достатньо високий 8,1 – 9,0;
- високий 9,1 – 10.

Таким чином, рівень НТЕ технології можна визнати достатнім. Отже, розроблений ПП пропонується випускати на ринок.

4.4 Розрахунок економічної ефективності проекту

Розрахунок економічної ефективності проекту – це процес визначення потенційного доходу та витрат проекту, а також оцінювання того, наскільки доцільно вкладати гроші у реалізацію проекту.

Для розрахунку економічної ефективності проекту необхідно визначити наступні показники:

6. Вартість проекту: це загальна вартість всіх витрат, пов'язаних з реалізацією проекту, включаючи затрати на розробку, впровадження та експлуатацію.
7. Прибуток від проекту: це дохід, отриманий від реалізації продукту або послуги, яку реалізовує проект.
8. Термін окупності: це період, за який витрати на реалізацію проекту повернуться від прибутку.
9. Рентабельність інвестицій: це відношення чистого дисконтованого доходу до вартості інвестицій.

Для розрахунку економічної ефективності проекту, потрібно мати інформацію про витрати та прибуток проекту.

4.5 Розрахунок витрат на матеріали

Таблиця 4.4

Витрати на матеріали

Найменування матеріальних витрат	Одиниця виміру	Кіль-кість	Ціна за одиницю, грн.	Вартість, грн.
Папір	Пачка	1	240	240
Фарба (картридж)	шт.	1	350	350
Флеш-карта, 4 Gb	шт.	1	130	130
Разом				720
Транспортно-заготівельні витрати – 10%				$720 \cdot 0{,}10 = 72$
Усього:				792

4.6 Розрахунок основної заробітної плати

Таблиця 4.5

Витрати на матеріали

Найменування робіт	Трудомісткість робіт у днях	Місячний оклад	Денна заробітна плата	Заробітна плата
1. Програмування	86	7800	354	30444
2. Керівництво і контроль	20	10000	455	9100
3. Розробка дизайну	35	8000	363	12705
Усього:		--	--	52249

Додаткова заробітна плата враховує оплату чергових відпусток, премії, інші доплати. Приймається в розрахунок 10% від основної.

$$ЗП_{дод} = ЗП_{осн} \cdot 0{,}10. \quad (4.4)$$

$$ЗП_{дод} = 52249 \cdot 0{,}10 = 5224 \text{ грн.}$$

Єдиний соціальний внесок ($Є_{св}$) приймається в розмірі 22 % від суми основної і додаткової заробітної плати.

$$Є_{св} = (ЗП_{осн} + ЗП_{дод}) \cdot 0{,}22. \quad (4.5)$$

$$Є_{св} = (52\,249 + 5\,224) \cdot 0{,}22 = 12\,644 \text{ грн.}$$

«Витрати, пов’язані з використанням обчислювальної техніки, визначаються за формулою:

$$С_{еом} = Т_{еом} \cdot К_{еом} \cdot Ц_{еом} \cdot К_{еом}; \quad (4.6)$$

де $Т_{еом}$ – час використання ЕОМ для розробки даного ПП, години; $Т_{еом}$ = = 564 год;

$К_{еом}$ – поправочний коефіцієнт обліку часу використання ЕОМ (1,08);

$Ц_{еом}$ – ціна однієї години роботи на ЕОМ, грн, (5 грн.);

$К_{еом}$ – коефіцієнт обліку швидкодії ЕОМ (1,0 – швидкодія ЕОМ більш 20х10 опер./с; 1,2 – швидкодія ЕОМ менш 20х10 опер./с,).» [4]

$$С_{еом} = 564 \cdot 1,08 \cdot 5 \cdot 1,0 = 3045 \text{ грн.}$$

Накладні витрати враховують адміністративні, загальновиробничі витрати, витрати на збут. Приймаються в розмірі 50% від основної заробітної плати.

$$Н_{в} = 0,50 \cdot ЗП_{осн} \quad (4.7)$$

$$Н_{в} = 0,50 \cdot 52249 = 26\,125 \text{ грн.}$$

На підставі здійснених розрахунків складається калькуляція планової собівартості ПП.

4.7 Калькуляція собівартості продукту

Таблиця 4.6

Витрати для розрахунку собівартості ПП

Найменування статей витрат	Сума витрат, грн.	Питома вага, %
1. Матеріали	792	0,79
2. Основна заробітна плата	52 249	52,6
3. Додаткова заробітна плата	5 224	5
4. Єдиний соціальний внесок	12 644	12,6
5. Витрати, зв’язані з обчислювальною технікою	3 045	3
6. Накладні витрати	26 125	26
Разом:	100 079	100

Ціна ПП визначається по формулі:

$$Ц = С \cdot Пр, \tag{4.8}$$

де $С$ – витрати на розробку програмної продукції (планова собівартість), грн;

$Пр$ – розмір прибутку, розрахований по формулі:

$$Пр = (С - С_м) \cdot \%Рн / 100, \tag{4.9}$$

де: $Рн$ – плановий рівень рентабельності (25%);

$С_м$ – матеріальні витрати.

Прибуток у ціні ПП становить: $П = (100\,079 - 792) \cdot 0{,}25 = 24\,822$ грн.

Ціна ПП становить: $Ц = 100\,079 \cdot 1{,}1 + 24\,822 = 134\,909$ грн.

4.8 Розрахунок капітальних витрат

Для розрахунку економічної ефективності проекту визначають капітальні і поточні витрати, зв’язані з використанням програмного продукту.

«Розрахунок капітальних витрат, пов’язаних із впровадженням програмного продукту здійснюється по формулі:

$$К_2 = К_{пп} + К_п + К_{ко} + К_{во}, \tag{4.10}$$

де $К_{пп}$ – ціна програмного продукту;

$К_п$ – попередвиробничі витрати;

$К_{ко}$ – вартість комп'ютерного устаткування;

$К_{во}$ – вартість допоміжного устаткування, необхідного для надійної роботи продукту.» [4]

Ціна програмного продукту складає 134 909 грн.

Попередвиробничі витрати містять у собі усі витрати, пов'язані з налагодженням і впровадженням гри – постановка задач та їхня алгоритмізація, розробка, налагодження і впровадження програмного продукту (ПП).

Приймаються $К_п$ у розмірі 100% від вартості розробленого ПП:

$$К_п = 134\,909 \text{ грн.}$$

Вартість комп'ютера ($К_{ко}$) становить 30000 грн. Вартість допоміжного устаткування визначається укрупнено в розмірі 10 % від вартості комп'ютера.

$$К_{во} = 30\,000 \cdot 0{,}10 = 3000 \text{ грн.}$$

$$К_2 = 134\,909 + 30\,000 + 3000 = 167\,909 \text{ грн.}$$

4.9 Розрахунок поточних експлуатаційних витрат

«Розрахунок поточних (експлуатаційних) витрат, пов'язаних з використанням ПП ($С_i$), здійснюється по формулі:

$$С_i = С_{опл} + С_а + С_{ел} + С_р + С_{доп} \qquad (4.11)$$

де $С_{опл}$ – річний фонд основної і додаткової оплати праці персоналу, що обслуговує програмний продукт з нарахуваннями;

$С_а$ – сума річних амортизаційних відрахувань від вартості основного і допоміжного устаткування програмного продукту;

$С_{ел}$ – вартість витрат на електроенергію на рік;

$С_р$ – вартість річного ремонту основного і допоміжного устаткування;

$С_{доп}$ – річна вартість допоміжних матеріалів, зв'язаних з впровадженням ПП.» [4]

Весь процес контролю виконує один співробітник, заробітна плата якого складає 7800 грн. Розрахунок річного фонду основної і додаткової оплати праці персоналу з нарахуванням ($С_{опл}$).

Річний фонд основної заробітної плати персоналу, що обслуговує ПП:

$$ЗП_{осн} = (ЗП_{окі} \cdot Ч_i) \cdot 12, \quad (4.12)$$

де $Ч_i$ – чисельність фахівців *і*-ї категорії, що обслуговують ПП;

$ЗП_{окі}$ – місячний оклад фахівця *і*-ї категорії;

$$ЗП_{осн} = (7800 \cdot 1) \cdot 12 = 93\,600 \text{ грн.}$$

Фонд додаткової заробітної плати;

$$З_{дод} = ЗП_{осн} \cdot К_{дод}, \quad (4.13)$$

де $К_{дод}$ – коефіцієнт додаткової заробітної плати, приймається в розмірі $К_{дод} = 0{,}2$.

$$З_{дод} = 93\,600 \cdot 0{,}2 = 18\,720 \text{ грн.}$$

Єдиний соціальний внесок приймається в розмірі 22% від суми основної і додаткової заробітної плати.

$$Є_{св} = (ЗП_{осн} + ЗП_{дод}) \cdot 0{,}22,$$

$$Є_{св} = (93\,600 + 18\,720) \cdot 0{,}22 = 24\,710 \text{ грн.}$$

Загальні витрати на оплату праці;

$$С_{опл} = ЗП_{осн} + ЗП_{дод} + Є_{св} = 93\,600 + 18\,720 + 24\,710 = 137\,030 \text{ грн.}$$

Розрахунок амортизаційних відрахувань визначається по формулі;

$$C_а = (K_{ко} \cdot K_{во}) \cdot H_а / 100, \quad (4.14)$$

де $H_а$ – норма амортизаційних відрахувань (для комп'ютерних устаткувань $H_а$ = 50%).

$$C_а = (30\,000 + 3000) \cdot 0{,}5 = 16\,500 \text{ грн.}$$

Річна вартість споживаної електроенергії $C_{ел}$, визначається по формулі:

$$C_{ел} = M_у \cdot T_{ко} \cdot Ц_е \cdot K_н, \quad (4.15)$$

де $M_у$ – установлена сумарна потужність комп'ютерного устаткування, $M_у$ = 0,45 кВт;

$T_{ко}$ – річний фонд часу роботи ЕОМ, який визначається виходячи з кількості робочих днів в році ($Д_р$), тривалості робочого дня (T) і з урахуванням часу на профілактичні огляди за рік ($T_{огл}$);

$$T_{ко} = Д_р \cdot T - T_{огл},$$

$$T_{ко} = 250 \cdot 8 - 300 = 1700 \text{ год.}$$

$Ц_е$ – вартість 1 кВт-години електричної енергії (3,25 грн.);

$K_н$ – коефіцієнт інтенсивного використання потужності ($K_{н\ реком}$, = 0,9).

$$C_{ел} = 0{,}45 \cdot 1700 \cdot 3{,}25 \cdot 0{,}9 = 2237 \text{ грн.}$$

Витрати на допоміжні матеріали ($C_{доп}$) приймаються в розмірі 2 % від вартості комп'ютерного устаткування:

$$C_{доп} = 30\,000 \cdot 0{,}02 = 600 \text{ грн.}$$

Вартість річного ремонту основного й допоміжного устаткування (6 % $K_{ко}$);

$$C_р = 30\,000 \cdot 0{,}06 = 1800 \text{ грн.}$$

Загальні витрати:

$$C_2 = 137\,030 + 16\,500 + 2\,237 + 600 + 1800 = 158\,167 \text{ грн.}$$

4.10 Розрахунок показників економічної ефективності проекту

«Очікуваний економічний ефект визначається по формулі:

$$E_o = (C_1 - C_2) - E_н \cdot (K_2 - K_1), \qquad (4.16)$$

де C_1, C_2 – поточні витрати відповідно до і після впровадження проекту; $(C_1 - C_2)$ – річна економія на поточних витратах, грн;

K_2 – капітальні витрати на впровадження ПП, грн;

K_1 – капітальні витрати до впровадження ПП, грн;

$E_н$ – нормативний коефіцієнт ефективності одноразових витрат (рекомендовано $E_н = 0{,}25$).» [4]

Капітальні вкладення до впровадження застосунка були відсутні, тому $K_1 = 0$.

$$E_o = (158167 - 100079) - 0{,}25 \cdot (167909 - 0) = 58088 - 41977 = 16111 \text{грн.}$$

Коефіцієнт ефективності капітальних витрат по формулі:

$$E = (C_1 - C_2) / K_2 - K_1 , \qquad (4.17)$$

$$E = 58088 / 41977 = 1{,}38.$$

Так, як $E > E_н$, то проект ефективний.

Рентабельність проекту можна розрахувати за формулою:

$$R = \textit{Прибуток} / \textit{Витрати} \cdot 100\ \% \qquad (4.18)$$

Припустимо, що прибуток складає 210 000 грн.

$$R = 210000 / (158167 + 100079) \cdot 100\ \% = 210000 / 258246 \cdot 100\ \% = 81\ \%.$$

Розраховуємо строк окупності капітальних витрат на впровадження проекту:

$$T = 1 / R \cdot 100\% = 1 / 81 \cdot 100\% = 1{,}23 \text{ року} = 14{,}8 \text{ міс.}$$

Результати економічних розрахунків відображаються в підсумковій таблиці 4.7.

Таблиця 4.7

Техніко-економічні показники програмного продукту

Найменування показників	Одиниця виміру	Значення показника	
		до впровадження проекту	після впровадження проекту
Трудомісткість розробки проекту	дні		86
Ціна ПП	грн		134 909
Капітальні витрати	грн	–	167 909
Поточні витрати	грн/рік	100 079	158 167
Економічний ефект від реалізації проекту	грн/рік		16 111
Строк окупності	роки		1,23
Економічна ефективність			1,38

Висновки по четвертому розділу

Порівняння коефіцієнту ефективності та одноразових витрат дозволяє зробити висновок що впровадження даного продукту є економічно вигідним, тому розробка цього програмного продукту є доцільною і актуальною.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ НА РОБОЧОМУ МІСЦІ

5.1 Гігієнічні норми до організації і обладнання робочих місць користувачів персональних комп'ютерів

Широке промислове та побутове використання персональних комп'ютерів (ПК) актуалізувало питання охорони праці користувачів ПК. Найбільш повним нормативним документом щодо забезпечення охорони праці користувачів ПК є «Державні санітарні норми і правила роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин» ДСанПІН 3.3.2.007-98.

При розташуванні елементів робочого місця користувача ВДТ слід враховувати:

- робочу позу користувача;
- простір для розміщення користувача;
- можливість огляду елементів робочого місця;
- можливість введення захистів;
- розміщення документації і матеріалів, які використовуються користувачем.

Конструкція робочого місця користувача ВДТ має забезпечити підтримання оптимальної робочої пози. Робочі місця з ВДТ слід так розташувати відносно вікон, щоб природне світло падало збоку, переважно зліва.

Робочі місця з ВДТ повинні бути розташовані від стіни з вікнами на відстані не менше 1,5 м, від інших стін – на відстані 1 м, відстань між собою – не менше ніж 1,5 м.

Для забезпечення точного та швидкого зчитування інформації в зоні найкращого бачення площина екрана монітора повинна бути перпендикуля-

рною нормальній лінії зору. При цьому має бути передбачена можливість переміщення монітора навколо вертикальної осі в межах ±30° (справа наліво) та нахилу вперед до 85° і назад до 105° з фіксацією в цьому положенні.

Таблиця 5.1

Допустимі рівні звуку, еквівалентні рівні звуку і рівні звукового тиску в октавних смугах частот

Вид трудової діяльності	Рівні звукового тиску в дБ в октавних смугах із середньогсометричними частотами, Гц									
	31,5	63	125	250	500	1000	2000	4000	8000	Рівні звуку, еквівалентні рівні звуку, дБА/дБАеквв.
Програмісти ЕОМ	86	7	61	54	49	45	42	40	38	50
Оператори в залах обробки інформації на ЕОМ та оператори комп'ютерного набору	96	83	74	68	63	60	57	55	54	65
В приміщеннях для розташування шумних агрегатів ЕОМ	103	91	83	77	73	70	68	66	64	75

Клавіатура має бути розташована так, щоб на ній було зручно працювати двома руками. Клавіатуру слід розміщати на поверхні столу на відстані 100...300 мм від краю. Кут нахилу клавіатури до столу повинен бути в межах

від 5 до 15°, зап’ястя на долонях рук повинні розташовуватись горизонтально до площини столу.

Принтер має бути розміщений у зручному для користувача положенні, так, щоб максимальна відстань від користувача до клавіш управління принтером не перевищувала довжину витягнутої руки користувача.

Конструкція робочого стола має забезпечувати можливість оптимального розміщення на робочій поверхні обладнання, що використовується, з врахуванням його кількості та конструктивних особливостей (розмір монітору, клавіатури, принтера, ПК тощо) і документів, а також враховувати характер роботи, що виконується.

5.2 Пожежна профілактика

Будівлі і ті їх частини, в яких розміщуються ПК, повинні бути не нижче другого ступеня вогнестійкості. Приміщення для обслуговування, ремонту та налагодження ПК повинні належати за пожежо-вибухонебезпечністю до категорії В відповідно до НАПБ Б.03.002-2007, а за класом приміщення – до П-2а за ПУЕ. Ми розглядаємо приміщення категорії Д.

Приміщення з ПК, крім приміщень, в яких розташовуються великі комп'ютери спеціального призначення (сервери, мейнфрейми), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до вимог Переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства внутрішніх справ України від 20.11.97 № 779 та зареєстрованого в Міністерстві юстиції України 28.11.97 за № 567/2371, та СНіП 2.04.09-84 «Пожежна автоматика будинків і споруд» з димовими пожежними оповіщувачами та переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 кв.м площі приміщення з урахуванням гранично допустимих концентрацій вогнегасної рідини відповідно до вимог Правил пожежної безпеки в України. В інших приміщеннях допускається встановлювати теплові пожежні оповіщувачі. Підходи до засобів пожежогасіння повинні бути вільними.

Виконаємо розрахунок об'єму недоторканого запасу води для зовнішнього пожежогасіння за такими вхідними даними:

– будівля ступеня вогнестійкості I, категорія А;

– площа території S = 250 га;

– час гасіння пожежі τ = 3 год;

– кількість одночасних пожеж для даного підприємства $n_{пож}$ = 2;

– витрата води на зовнішнє пожежогасіння g = 80 л/с.

Витрата води на зовнішнє пожежогасіння протягом розрахункового часу:

$$V_{В} = \frac{k * g * \tau}{1000} * 3600 \text{ м}^3,$$

$$V_{В} = \frac{1{,}1 * 80 * 2 * 3}{1000} * 3600 \text{ м}^3$$

Визначимо необхідну кількість водойм, якщо об'єкт квадратної форми площею 250 га зі стороною: $L = \sqrt{250*10^4}$ = 1581 м та радіусом дії насосу R = 200 м:

$$n_{вод} = \frac{L}{2R} = \frac{1581}{2 * 200} = 3{,}95 \text{ шт}$$

Приймаємо $n_{вод}$ = 4 шт.

Розрахуємо кількість балонів з вуглекислотою для внутрішнього пожежогасіння приміщення об'ємом 576 м^3.

Кількість вогнегасного газового складу:

$$G_{г} = 1{,}25 * \left(G_{в} * V_{п} * K_{у}\right), \text{кг}$$

де $G_{в}$ – вогнегасна концентрація газового складу для вуглекислоти ($G_{в}$ = 0,7 кг/м^3);

$V_{п}$ – об'єм приміщення, м^3;

$K_{у}$ – коефіцієнт, що враховує особливості процесу газообміну, витік вуглекислоти крізь нещільності приміщення, приймаємо 1,1.

$$G_{г} = 1{,}25 * (0{,}7 * 576 * 1{,}1) = 554{,}4 \text{ кг}$$

Потрібна кількість робочих балонів $N_б$ (од.) із вуглекислотою визначається наступним чином:

$$N_б = \frac{G_г}{V_б * \rho * \alpha_n} \text{ од.}$$

де $V_б$ = 25 л – об'єм балону (при 25 л у балоні міститься 15,6 кг вуглекислоти);

ρ = 0,625 кг/л – щільність засобу гасіння;

$\alpha_п$ = 1 – коефіцієнт наповнення.

$$N_б = \frac{554,4}{25 * 0,625 * 1} = 35,48 \approx 36 \text{ од.}$$

Кількість резервних балонів приймаємо такою ж, як і робочих:

$$N_б = N_р = 36 \text{ од.}$$

$$N_{заг.} = 36 + 36 = 72 \text{ од.}$$

Висновки п'ятого розділу

У даній частині дипломного проекту були розглянуті питання щодо гігієнічних норм організації і обладнання робочих місць користувачів персональних комп'ютерів, питання пожежної профілактики приміщень, виконані необхідні розрахунки об'єму недоторканого запасу води для зовнішнього пожежогасіння. Отримані результати дозволяють обладнати приміщення для роботи користувачів обчислювальної техніки з дотриманням вимог безпеки їх праці.

ЗАГАЛЬНІ ВИСНОВКИ

В результаті виконаної роботи була проаналізована предметна область комп'ютерних логічних ігор взагалі і реалізацій гри судоку зокрема, а також принципи і засоби їх створення. Проведений аналіз жанру дозволив ґрунтовно підійти до розробки концепції гри, подальшого її проектування і реалізації.

Узагальнена постановка завдання фіксує вимоги до проекту гри як з боку розробника, так і з боку користувача. На основі поставленого завдання був розроблений проект гри, який ліг в основу технічної реалізації демо-версії гри «Танковий лабіринт» для платформи Windows. На основі вимог технічного завдання були закладені основи концепту і правил гри, принципів геймплею. За допомогою обраного інструменту реалізації – ігрового рушія Unity – гра отримала своє програмне втілення.

Також були досліджені суміжні питання охорони праці і виконано техніко-економічне обґрунтування проекту.

Хоча перший рівень реалізації гри ще не дозволяє зробити її доступною для широкої аудиторії, ідеї та механізми, які були вкладені в гру, дозволяють оцінити її як потенційно привабливу.

Гра має перспективи розвитку. Для конкуренції з аналогами можна додати:

- звуки і фонову музику;
- ігровий рахунок;
- карту рівня;
- нові рівні;
- нові здібності танка;
- автоконфігуратор рівнів;
- приборну панель управління захисними функціями танку тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agafonov E. Mastering C# Concurrency / E. Agafonov, A. Koryavchenko. – New York: Packt Publishing, 2015. – 284 p.
2. Barrera, Ray. Unity 2017 Game AI Programming: Leverage the power of Artificial Intelligence to program smart entities for your games / Ray Barrera, Kyaw Aung Sithu, Swe Thet Naing. 3rd Revised Ed. – Packt Publishing, 2017. – 243 p.
3. Бородкіна І. Інженерія програмного забезпечення. Навчальний посібник / І. Бородкіна, Г. Бородкін. – Центр учбової літератури, 2021. – 204 с.
4. Вігуржинська С. Ю, Колесник В. І. Дипломне проектування економічної частини проекту: Методичні вказівки для студентів, що навчаються за комп'ютерними спеціальностями "Інформаційні управляючі системи та технології", "Інформаційні технології проектування", "Комп'ютерні системи та мережі" та "Спеціалізовані комп'ютерні системи". – Одеса: ОНАХТ, 2016 р. – 22 с.
5. Відеогра // Вікіпедія [Електронний ресурс]. – Режим доступу: *https://uk.wikipedia.org/wiki/Відеогра/*.
6. Вільямс, Річард. Анімація. Посібник з виживання. — ArtHuss, 2019. — 384 с.
7. Геннессі, Джонатан. Історія відеоігор в коміксах. – Yakaboo Publishing, 2020 р. – 192 с.
8. Касьянов М.А., Васильчук М.В. Охорона праці користувачів ПК. Навчальний посібник. – Луганськ : видавництво СНУ ім. В. Даля, 2009. – 101 с.
9. Катренко Л. А., Катренко А. В. Охорона праці в галузі комп'ютинґу. — Львів : Магнолія – 2006, 2012. – 544 с.
10. Князєва Н. О. Дипломне проектування: Методичні вказівки до дипломної роботи спеціаліста / Н. О. Князєва, С. В. Шестопалов, С. Л. Жуковецька. — Одеса : ОНАХТ, 2016. — 46 с.

11. Мэннинг Д. Unity для разработчика. Мобильные мультиплатформенные игры // Д. Мэннинг, П. Батфилд-Эддисон – СПб. : Питер, 2018. – 304 с.
12. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. Укладачі Басюркіна Н. Й., Свистун Т. В. – Одеса : ОНАТУ, 2022. – 18 с.
13. Развитие сегмента инди-игр / *Alconost* // Хабр [Електронний ресурс]. — Режим доступу: *https://habr.com/ru/company/alconost/blog/555550/*.
14. Шраєр, Джейсон. Кров, піт і пікселі. Тріумфальні та бурхливі історії по той бік створення відеоігор. – К. : Book Chef, 2020. – 336 с.
15. Boyes, Emma. GDC ‘08: Are casual games the future?– 19.02.2008. [Електронний ресурс]. — Режим доступу: *https://www.cnet.com/tech/gaming/gdc-08-are-casual-games-the-future/*.
16. Candy Crush has been downloaded over 3 billion times // Gamereactor. – 2023-04-11. – *https://www.gamereactor.eu/candy-crush-has-been-downloaded-over-3-billion-times-1257023/*.
17. Casual Game Genres on Big Fish Games [Електронний ресурс] // Big Fish Games. – 1 March 2002. — Режим доступу: *http://www.bigfishgames.com/ download-games/top-pc-games.html*.
18. Casual Gaming Worth $2.25 Billion, and Growing Fast [Електронний ресурс] // VentureBeat. 29 October 2007. — Режим доступу: *https://venturebeat.com/ business/casual-gaming-worth-225-billion-and-growing-fast/*.
19. Gamezebo staff. Casual Game Genres on Gamezebo [Електронний ресурс]. – 1 March 2006. — Режим доступу: *http://www.gamezebo.com/games/find*.
20. Hyper-casual: Mobile gaming’s newest genre *[Електронний ресурс]* // AppLovin Blog. 3 August 2018. — Режим доступу: *https://blog.applovin.com/ hyper-casual-mobile-gamings-newest-genre/*.

21. IR Information : Financial Data – Top Selling Title Sales Units – Wii Software [Електронний ресурс] // Nintendo Co., Ltd., 2019. — Режим доступу: *http://www.nintendo.co.jp/ir/en/finance/software/wiiu.html*. [IR]
22. Kent, Steve L. The ultimate history of video games: from Pong to Pokémon and beyond : the story behind the craze that touched our lives and changed the world // Prima, 2001. – p. 143. [K]
23. Kharif, Olga. Tetris' Maker Has His "A" Game [Електронний ресурс] // Bloomberg. – 23 Nov 2005. — Режим доступу: *https://www.bloomberg.com/ news/ articles/2005-11-22/tetris-maker-has-his-a-game*. [Kh]
24. Kohler, Chris. Q&A: Pac-Man Creator Reflects on 30 Years of Dot-Eating [Електронний ресурс] // Wired. – 21 May 2010. — Режим доступу: *https://www.wired.com/gamelife/2010/05/pac-man-30-years/*. [Ko]
25. McDonald, Emma. Newzoo's video games market size estimates and forecasts for 2022. – May 17 2023. – *https://newzoo.com/resources/blog/the-latest-games-market-size-estimates-and-forecasts*.