Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ГРИ НА UNITY/C# З ІМПЛЕМЕНТАЦІЄЮ ЕЛЕМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
с.в. Вовк Н.Є.
(прізвище та ініціали)
_____
(підпис)
“__”_____ 2023 р.

Виконав студент
Погодічев І.Д.
(прізвище та ініціали)
“__”_____ 2023 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____ О. П. Жежерун (підпис)
“___”_____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Погодічеву Івану Дмитровичу факультету інформатики 4-го курсу

ТЕМА Розробка гри на UNITY/C# з імплементацією елементів штучного інтелекту

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Огляд ігрового рушія Unity
2 Штучний інтелект в відеоіграх
3 Розробка проєкту
Висновки
Список використаних джерел
Додатки (за необхідністю)

Дата видачі “__”_____ 2022 р. Вовк Н.Є. _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка гри на UNITY/C# з імплементацією елементів штучного інтелекту

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	01.10.2022	
2.	Аналіз матеріалів за темою	14.01.2023	
3.	Розрозбка та програмування алгоритму	01.03.2023	
4.	Написання текстової частини до курсової роботи	14.04.2023	
5.	Створення слайдів для доповіді та написання доповіді	11.05.2023	
6.	Остаточне оформлення роботи та слайдів	20.05.2023	
7.	Захист курсової роботи	31.05.2023	

Погодічев І. Д. _____

Вовк Н. Є. _____

“_____”_____ 2023 р.

Зміст

Анотація

Під час виконання роботи було розроблено гру на ігровому рушії Unity та за допомогою мови програмування C# а також розроблено штучний інтелект. Робота є гарним прикладом використання штучного інтелекту, який все більше набирає обертів у відеоіграх.

Ключові слова: Unity, C#, штучний інтелект, поведінкові дерева.

Вступ

Постановка завдання

В сьогоднішньому світі відеоігри є найбільшим ринком серед цифрових медіа та продовжує відриватися від своїх конкурентів.[11] Ігри стають дедалі розвинутішими, цікавішими, реалістичнішими і, найголовніше, розумнішими.

Об’єктом даної роботи є штучний інтелект, який наразі застосовується майже у всіх цифрових сферах і ігри не є винятком. Саме за допомогою ШІ відеоігри набувать реалістичності. Відповідно керуючись актуальністю цього напряму, за предмет роботи була поставлена розробка гри з імплементацією штучного інтелекту на основі поведінкових дерев.

Основна частина роботи складається з 3 розділів.

Перший розділ містить загальну інформацію про ігровий рушій, який використовувався при розробці проєкту. Розглядаються також переваги та недоліки даного рушія.

В другому розділі розглядається тема штучного інтелекту у відеоіграх та які існують його типи та споосби використання. Окрема увага приділяється графам, а також докладно розглядається такий тип ШІ, як поведінкові дерева.

Останній, третій розділ, приділяє увагу безпосередньо процесу розробки практичної частини роботи. В ньому присутня інформація про дизайн проєкту та головні функції, які були розроблені. Окремо приділяється увага ШІ, який був реалізований у практичній частині роботи.

Основна частина

Розділ 1 : Огляд ігрового рушія Unity

1.1 Основна характеристика

Unity Engine – це рушій для створення ігор, такий собі фреймворк, який допомагає розробникам створювати відеоігри. Наразі це найпоплуряніший ігровий рушій серед усіх інших на ринку. Більш того, він є кросплатформенним, а отже надає можливість створювати гру одразу як для персональних комп'ютерів, так і для мобільний пристроїв або консолей. Для системи скриптування використовується мова програмування C#.

Unity забезпечує розробників всіма інструментами та технологіями, які можуть знадобитися. Наприклад, рендерінг графіки, симуляція фізики, освітлення, візуальні ефекти, звукові ефекти тощо. Також існує підтримка технологій доповненої та віртуальної реальностей.

Зважаючи на широкий функціонал ігрового рушія, його використовують не тільки для створення ігор. Дуже часто Unity використовується в кінематографі, для створення 3D презентацій, браузерних додатків, освітніх додатків для школярів та навіть симуляцій злочинів для їх кращого розслідування.

1.2 Переваги

Висока популярність Unity Engine на ринку зумовлена рядом унікальних переваг, які він має. Першою і, на мою думку, основною перевагою є кросплатформенність ігрового рушія і її відносно легка підтримка, оскільки це дуже знижує час і, відповідно, вартість розробки продукту. Другою перевагою є низький поріг входження у роботу з рушієм в порівнянні з іншими рушіями, що дає змогу спробувати розробку відеоігор для всіх бажаючих. Це зумовленно досить зрозумілим та інтуїтивним

дизайном Unity Engine а також мовою програмування C# для системи скриптингу. Додатковою перевагою можна виділити свій власний магазин асетів, який пропонує дуже широкий асортимент як від окремих розробників, так і від компаній.[1] Це дає змогу ще більше заощадити час на розробці, використавши вже якесь готве рішення для окремого модуля програми, наприклад інпуту користувача.

1.3 Недоліки

Звичайно окрім переваг присутні також і недоліки. Першим і головним недоліком, на мою думку, є погана масштабованість проєктів, а саме симуляція фізики і відмалювання графіки. Наприклад, якщо потрібно щоб одночасно на сцені було дуже багато фізичних об’єктів зі складними графічними 3D моделями (великою кількість трикутників та вершин), то можуть виникнути великі проблеми, особливо на слабких девайсах. Втім, за останні роки Unity випустили і продовжують розвивати Data-Oriented Technology Stack (DOTS) – новий для рушія підхід до розробки який дає можливість вирішити проблему поганого масштабування.

Ще одна проблема пов’язана з попередньою це відсутність повного контролю пам’яті, оскільки C# не дає такої змоги. З однієї сторони це перевага, тому що новачкам буде легше почати розбиратися в сфері розробки відеоігор. З іншої, дуже часто при роботі з AAA проектами цього не хватає для того, щоб краще оптимізувати роботу вашої гри.

Підсумовуючи недоліки, можна сказати, що це та ціна, яку платять користувачі для того, щоб швидше розібратися у ігровому рушії та почати розробляти свої власні проєкти або працевлаштуватися у сфері відеоігор.

Розділ 2 : Штучний Інтелект в відеоіграх

2.1 Що таке ШІ в відеоіграх?

Для того щоб зрозуміти що таке штучний інтелект в відеоіграх треба спочатку взагалі розібратися зі значеням терміну «Ігровий ШІ». Більш того, варто почати з терміну «гра». В контексті відеоігор, які розглядаються в даній роботі «гра» означає в першу чергу ігровий досвід гравця, який він може отримати в рамках дизайну гри. Дизайн гри, в свою чергу, означає набір налаштувань та правил штучного «ігрового світу» де відбувається гра.

Головна відмінність відеоігор від інших видів цифрових продуктів розважальної та медіа індустрій це інтерактивність. Споживач здатний сам формувати свій власний ігровий досвід. Для прикладу, під час перегляду фільму або прослуховування музики такої опції не надається.

Інтеракції, які зазвичай трапляються в відеоіграх можуть набувати зовсім різних форм, але в той час як в одних іграх інтеракція полягає в натисканні якоїсь кнопки, інші дизайнери намагаються «оживити» їхню гру, зробити більш переконливою. З’являється потреба створити для гравця відчуття, що по іншу сторону екрану існує щось майже живе, що здатне думати та приймати рішення в залежності від дій гравця. Саме тут на допомогу приходить штучний інтелект.

2.2 Типи ШІ в відеоіграх

Розібравшись для чого потрібен ШІ в відеоіграх і яку функцію він виконує, варто зазначити які є різновиди застосування ШІ[2]. Розглянемо найбільш популярні:

1. NPC’s Control – non-player character control – контроль негральних персонажів. NPC - це персонажі, які керуються безпосередньо комп’ютером. Це найбільш розповсюджений приклад ШІ, оскільки у

наш час складно уявити гру без NPC. Алгоритмів для визначення їхньої поведінки дуже багато. Дуже часто такі алгоритмі побудовані на теорії графів, один з яких і імплементовано в практичній частині даної роботи.

2. PCG – procedural content generation – процедурна генерація контенту – створення будь-якого контенту або даних комп'ютером самостійно або при взаємодії з розробником, ігровим дизайнером або гравцем. [3] Зазвичай використовується для створення ігрового світу – мапи та її наповнення. Втім, також можна зустріти його застосування у створенні ігрових квестів, сюжетних історій та навіть музики.
3. Pathfinding – пошук шляху – пошук оптимального, коротшого та вигіднішого шляху з однієї точки до іншої. Ігровий штучний інтелект повинен вміти точно прораховувати свій маршрут в залежності від різних перешкод які можуть трапитися йому на шляху.
4. Data mining – збір ігрових даних – надважливий і розповсюджений вид штучного інтелекту у будь-якій сфері інформаційних технологій. Game Data mining дозволяє компаніям розробникам краще зрозуміти поведінку користувача і зрозуміти як покращити їхній продукт. Цей тип ШІ дозволяє відповісти на такі питання, як: «Як користувачі грають у вашу гру?», «Чому користувачі припиняють грати?», «Чи можемо ми передбачити поведінку гравця?» тощо.
5. PEM – player experience modelling – моделювання досвіду гравця. Доволі новий тип ШІ у геймінгу, який прийшов сюди з великим ростом кількості користувачів. [4] PEM бере до уваги вхідні дані (те, з чого робиться передбачення поведінки: психологія, параметри дизайну, стиль гри гравця тощо) та вихідні дані (те, що очікується від гравця: задоволення, розчарування, концентрація тощо) і безпосередньо методологія моделювання.

2.3 Графи в відеоіграх.

Розібравшись з теорією застосування ШІ в відеоіграх, перейдемо ближче до варіантів його імплементації. Одним з таких варіантів є імплементація за допомогою теорії графів. Взагалі, дискретна математика і зокрема теорія графів дуже часто використовуються в дизайні та розробці відеоігор. Майже в кожному проєкті знайдеться хоч якесь застосування графів. Отже, які існують способи використання теорії графів у відеоіграх:

1. Репрезентація ігрового світу. Насправді це дуже всеосяжний спосіб, але для прикладу вершини графу можуть визначати якісь точки у ігровому світі, а його ребра – шляхи між цими точками. Напевно всі чули про алгоритм Дейкстри або A* алгоритм які базуються на графах і використовуються в іграх для пошуку оптимального шляху.
2. Побудова ігрового світу. Багато ігор використовують процедурну генерацію для створення своїх світів. І найчастіше використовується ніщо інше, як тріангуляції Делоне та діаграми Вороного, які також базуються на теорії графів. Для прикладу, такі відомі ігри як Minecraft, No Man’s Sky або Civilization використовують процедурну генерацію для генерування своїх світів.
3. Ігрові механіки. В звичайних ігрових механіках також може використовуватися граф просто для зручного моделювання якихось процесів або структурування даних. Наприклад, дерево прокачки здібностей у персонажа, де вершини відображають здібності, а ребра – передумови для розблокування здібності. Іншим прикладом може бути система діалогів, де в залежності від відповіді NPC відповідає відповідним чином.
4. Штучний Інтелект. Ну і звичайно ж штучний інтелект, на якому я зупинюся і розповім в деталях про один із підходів використання графів. Загалом, зазвичай графи для штучного інтелекту

використовуються в контексті decision-making – прийняття рішень для NPC, часто в комбінації з Minimax або Alpha-beta pruning алгоритмами, для прийняття максимально логічних рішень.

2.4 Поведінкові дерева

2.4.1 Загальний опис

Behaviour Tree – поведінкове дерево – це структура даних в якій задається набір правил і поведінок та їхні умови та порядок виконання. Поведінкові дерева це дуже розповсюджений для сучасний комп'ютерних ігор дизайн паттерн, за допомогою якого можна визначити поведінку та логіку прийняття рішень для ігрової сутності. [10]

Тепер треба розібратися що з себе представляє поведінкове дерево. По-перше це дерево, тобто ніщо інше як напрямлений ациклічний граф. Граф, який має набір нод (вершин), які пов'язані між собою де кожен зв'язок має чіткий напрям і зв'язує ноди таким чином, що ніколи не може утворитися цикл. Як наслідок, маємо ієрархічну структуру нодів, тобто вершин, які будують загальну поведіну ігрової сутності. Дерево виконується зверху вниз та зліва на право. На кінцях дерева розташовані ноди, які відповідають за фактичні команди, а гілки дерева скаладються з різних утилітний нод, які контролюють як ШІ буде рухатися по дереву і до якої кінцевої команди йому треба прийти.

2.4.2 Основні елементи

У типовому поведінковому дереві можуть бути різні типи нод. Проте, у них є одна важлива спільна риса – вони всі повертаються значення. В класичній імплементації ноди можуть повертати одне з трьох значень: Success, Failure, Running. Перші два, відповідно до їхньої назви, інформують батьківську ноду про результат їхньої операції. Running – означає, що поки що неможливо визначитися з результатом і операція не є

закінченою. Саме цей функціонал є ключовим при роботі з поведінковим деревом та дозволяє зручно і легко контролювати поведінку ШІ.

Відтак з таким спільним функціоналом ми маємо 4 основні архітипи нодів поведінкового дерева:

1. Root – корінь дерева – єдина нода, у якої немає батьківської ноди, тільки children – дочірні ноди. Корінь дерева слугує його початком, звідки і починається виконання всієї логіки.
2. Leaf – листок – кінцева, найнижча нода, яка має тільки батьківські ноди і не може мати дочірні ноди. Тим не менш, листки це найбільш потужний тип, оскільки вони відпоідають за фактичну дію ШІ. Наприклад, якщо розглядати логіку типового NPC, то це могло б бути «піти до якогось місця», «атакувати гравця» або «взаємодіяти з якимось предметом» тощо. Також листкам можна подавати на вхід параметри. Якщо порівнювати кінцеві ноди з структурами коду, то їхні батьки це функції, оператори if, цикли while та інші конструкції, а листки це код який виконується всередині цих конструкцій.
3. Composite or Control – композитні або іноді їх ще називать контрольними нодами. Ці ноди можуть мати одну або більше дочірніх нод, які будуть виконуватися в залежності від логіки композитної ноди. Саме вони відповідають за контроль нашого руху по дереву. В класичному визначенні існують 2 типи композитних нод.
    i. Sequence – послідовність – найпростіша композитна нода, назва якої вичерпно описує її функціонал. Ця нода по черзі виконує кожну дочірню ноду. У разі, якщо дочірня нода повертає Failure, то і Sequence поверне таке саме значення. Якщо ж усі дочірні ноди по черзі повернули Success, то після останнього Success, Sequence також поверне це значення.

Функціонал послідовності можна прирівняти до логічного оператора AND.

ii. Selector – Селектор – це нода яку можна вважати антагоністом до Sequence ноди. Якщо Sequence нода це оператор AND, то Selector – OR. Відповідно селектор поверне Success тоді, коли перша-ліпша дочірня нода поверне Success. В іншому ж випадку, якщо ніхто не поверне Success, то селектор поверне Failure. Найпростіший приклад, якщо гравець достатньо близько він обере ноду яка відповідає за ближній бій, якщо далеко – то за дальній, а якщо занадто далеко то перейде в якусь третю ноду.

Також існує окрема категорія нодів, яка називається Decorator. Він також може мати батьківську та дочірню ноди. Зазвичай, їх функція полягає у тому, щоб перетворювати результат дочірньої ноди на якийсь інший, переривати роботу дочірньої ноди або зациклювати її роботу. Їх існує доволі багато, але наведу пару показових прикладів:

1. Inverter – дуже проста декоративна нода, яка обертає повернуте дочірньою нодою значення.
2. Repeater – повторювач – повторює виконувати дочірню ноду, поки вона повертає значення. Також за потреби є опція зазначити скільки разів він виконає дочірню ноду перед тим, як повернеться до батьківської ноди.
3. Repeat Until Failure – аналогічно до Repeater, але виконує дочірню ноду до тих пір, поки вона не поверне Failure.

2.4.3 Алгоритм роботи.

Розібравшись з основними компонентами поведінкового дерево можна перейти до теми його роботи. В класичному варіанті, коли такий

підхід тільки винайшли, це дерево виконувалося на кожний кадр нашої гри починаючи з Root ноди. Тобто кожний кадр, наприклад 60 разів на секунду, дерево пробігало від рута, визначаючи на кожному рівні ієрархії в яку сторону піти і так аж до кінцевої ноди. Очевидно, що такий підхід дуже дорогий в контексті використання ресурсів наших девайсів, особливо коли дерево стає дійсно великим, з дуже розгалуженою логікою.

Тому сьогодні алгоритм вже більш оптимізований, дерево зберігає референс на активну кінцеву ноду та шлях до неї. Коли ця нода в якийсь момент закінчить свою роботу та поверне значення, дерево перерахує що йому робити.

В додаток до цього була придумана така технологія, як Blackboard – тобто дошка. Дошка це майже буквально «дошка», яка зберігає корисну інформацію, яку дерево використовує для прийняття рішень при русі по дереву. Це можуть бути координати якихось найближчих точок інтересу (наприклад авто або якийсь об’єкт який потрібно захопити), відстань до гравця або навіть статусні булеві змінні, такі як «чи гравець живий» або «чи ведеться по нам вогонь».

Окрім того, що дошки дозволяють нам мінімізувати потребу у переобрахунках, вони також можуть бути одночасно використовувані багатьма поведінковими деревами. Тому що наприклад координати гравця зазвичай потрібні майже всім NPC які є у вашій грі, незалежно від унікальності їхньої логіки.

2.4.4 Приклади використання та причини популярності

На останок, варто зазначити популярність поведінкових дерев при розробці відеоігор, оскільки з часом все частіше зустрічається ШІ побудований саме на основі цього паттерну. Здебільшого такий підхід використовується у ААА відеоіграх (ігри з дуже великим бюджетом

відносно інших). Його можна зустріти у таких франшизах як: Tom Clancy’s, Far Cry, Halo, Bioshock та інших.

Це дуже зручний і гнучкий інструмент який можна підлаштувати майже під будь-які ігрові сценарії та задачі. Також дуже сильно приваблює фактор “reusability”, тобто перевикористання з мінімальними змінами, що пришвидшує темпи розробки і зменшує її вартість. Ще один фактор великої популярності є те, що в порівнянні з іншими системами, як Finite State Machines, значно легше виконувати процес дебагу, тому що дуже легко прослідкувати на якому кроці виникає помилка, та і в цілому така структура значно оптимізованіша.

Розділ 3 : Розробка Проєкту

3.1 Ігровий Дизайн

Практичною частиною роботи є розробка 3D гри в жанрі top down stealth action – тобто гра, з камерою розташованою зверху та ігровим процесом, який вимагає від гравця скритності.

За game setting – тематику гри взято японський напрямок. Для пришвидшення розробки, графічна складова була взята з магазину ассетів [1], а саме: 3D моделі персонажів та їхні анімації, моделі споруд та декорацій. Дії відбуваються вночі.

Гравець грає за ніндзя, задача якого полягає в викраденні якогось коштовного предмету. На шляху в нього є ряд перешкод, а саме: самураї, які або стоять на посту, або патрулюють якусь територію і власне сама мапа рівня, яка побудована спеціальним чином, щоб ускладнити алгоритм та подовжити шлях проходження рівня. Також гравець має у своєму інвентарі певну кількість відволікаючих предметів, які він може кидати на землю поруч з ворогом для того щоб його відволікти на певний час. Після підбирання головного предмету рівня гравець повинен завершити рівень, покинувши його через точку виходу, яка також змінює свій вигляд з рівня в рівень. У випадку, якщо гравця зловлять, то рівень одразу перезапускається спочатку.

3.2 Player Loop

Player Loop – це цикл дій які відбуваються у грі кожного рівня. У даній грі player loop буде виглядати наступним чином:

1. Гравець з’являється на мапі.
2. Гравець долаючи перешкоди доходить до головної «точки інтересу» – предмету який йому треба вкрасти на даному рівні.

3. Гравець підбирає цей предмет.
4. Гравець доходить до точки виходу з рівня знову долаючи перешкоди на своєму шляху.
5. Гравець покидає рівень і одразу переходить на наступний.

3.3 Технології розробки

Як вже зазначалося в першому розділі роботи, Unity Engine використовувався в якості ігрового рушія разом з мовою програмування C#.

Для розробки було обрану сервісну архітектуру проєкту, оскільки в проєкті такої складності це дозволяє дуже зручно контролювати залежності в коді. Також була розроблена система ініціалізації об’єктів рівня, «прогрівання» сервісів та завантаження і зберігання ігрових даних по типу розташування гравця та ворогів.

Щодо паттернів [5], які були використані можна виділити наступні:

1. Factory – фабрика – паттерн, який потрібен для того щоб зробити зручне створення об’єктів. У грі присутні дві фабрики: одна для звичайних об’єктів та інша для створення об’єктів інтерфейсу користувача.
2. Singleton – сінглтон – паттерн з сумнівною репутацією, який зручно використовувати, поки проєкт не набрав великих обертів. Проте, в цьому проекті він використовується виключно для зручного доступу до сервісів застосунку.
3. State – паттерн станів – паттерн, який дозволяє контролювати поведінку об’єкту в залежності від його стану. Використовується для керування камерою.

Щодо графічної частини застосунку, то можна виділити роботу з 3D моделями, які були оптимізовані для кращої продуктивності – однакові моделі, наприклад декорації, промальовуються одночасно. Також для

покращення графіки та досягнення більш реалістичного відчуття ночі у гравця був використаний Post-Processing – пост-обробка, яка дозволила виділити штучне світло, затемнити зовнішнє средовище та зробити корекцію кольору.

3.4 Основні Механіки

Механік у грі присутньо не дуже багато, оскільки основний акцент був зроблений саме на штучному інтелекті. Проте, є декілька речей вартих уваги.

По-перше, це ввід користувача, а саме рух головного героя, його взаємодія з зовнішнім світом та здібності. Оскільки гра зроблена під ПК, то весь ввід користувача відбувається за допомогою клавіатури, але в гру вже закладена логіка для мобільного керування або керування контроллером, оскільки в майбутньому, користуючись перевагою кросплатформеності рушія, буде дуже легко адаптувати гру під інши платформи. Також для зручності саме рух персонажа продубльований як на клавіші WASD так і на стрілочки.

По-друге, слідкування камерою за гравцем. Камера – це один з найважливіших елементів, які впливають на досвід користувача, тому для даного проєкту була використана спеціальна технологія від Unity, яка називається Cinemachine [6]. Вона значно розширює функціонал камери за замовчуванням та дозволяє в повній мірі налаштувати всі параметри, які потребуються для якісного керування камерою.

Остання механіка, яку можна виділити це здібність персонажа. Як вже згадувалося раніше, гравець може кинути відволікаючий предмет зі свого інвентаря для того, щоб на певний час відволікти ворога. Також було вирішено спочатку малювати траєкторію польоту цього об’єкту, щоб гравець міг влучно прицілитися і кинути його саме туди, куди хоче. Для

обрахунку траєкторії прицілювання та безпосередньо руху нашого об’єкта було використано перше з чотирьох основних кінематичних рівнянь[9]:

$$d = v_i * t + \frac{1}{2} * a * t^2$$

Рисунок 1 Перше кінематичне рівняння

Де d – displacement – переміщення, v_i – initial velocity – початкова швидкість об’єкту, a – acceleration – прискорення об’єкту, t – час, протягом якого рухався об’єкт. Відповідно, гравець може влучно прицілитися та запустити предмет відволікання, що показано на рисунках:

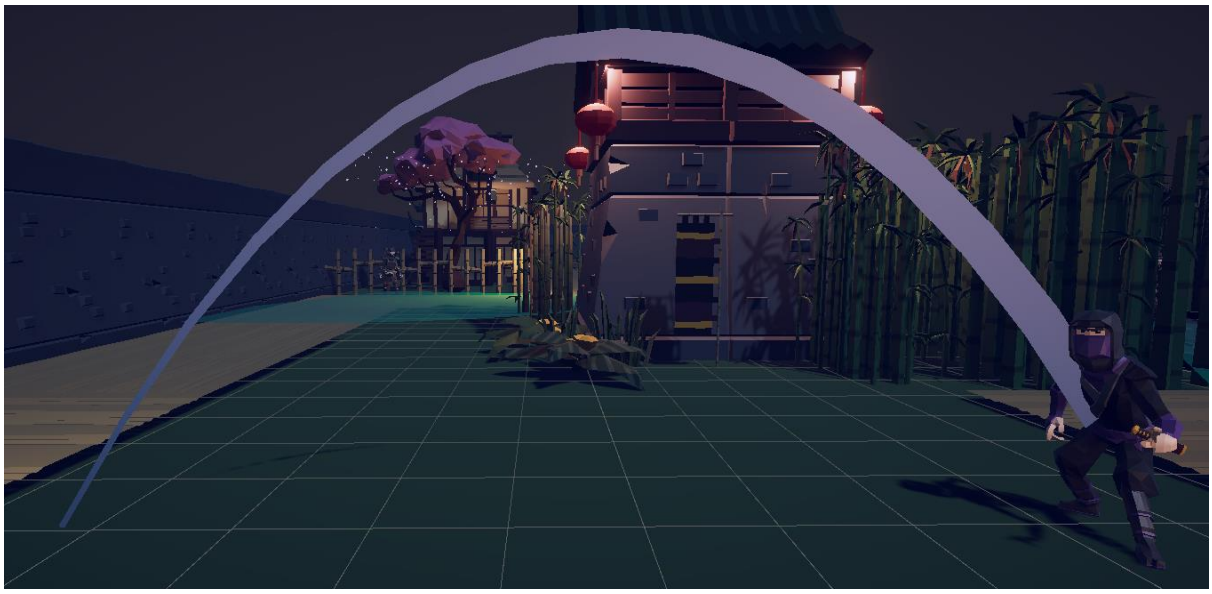

Рисунок 2 Візуалізація траєкторії польоту предмету

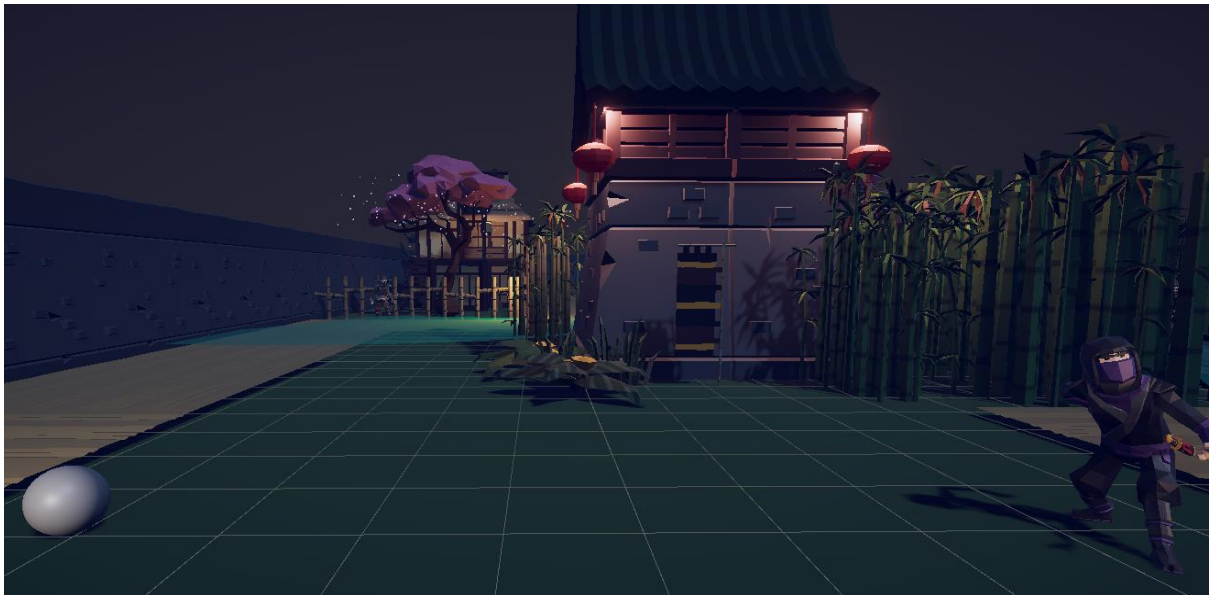

Рисунок 3 Результат запустку предмету по траєкторії

3.5 Реалізація Штучного Інтелекту

Перед декомпозицією ШІ у даному проекті варто нагадати про його дизайн. У даній грі штучним інтелектом наділені вороги, які повинні вміти робити наступні дії: пересуватися по рівню, мати зір обмежений конусом а також мати поведінкове дерево, яке власне і буде повністю контролювати дії ворога.

3.5.1 Unity Navigation System

Для того, щоб вміти реалістично пересуватися в просторі та виконувати заплановані дизайном дії, а саме: патрулювання, переслідування гравця тощо, вороги мають слідувати якійсь навігаційній системі, яка використовує певний алгоритм пошуку шляху. Існує дуже багато різних алгоритмів пошуку шляху: всім відомий A*, Minimum Spanning Tree та інші. Але оскільки я роблю проєкт на ігровому рушії Unity, то він нам пропонує засоби та технології, які можуть полегшити розробку такого алгоритму – Navigation System. [7] Насправді, «під капотом» навігаційна система цього рушія використовує свою власну імплементацію алгоритму пошуку шляху A*, який вони назвали – “NavMesh A* Algorithm.

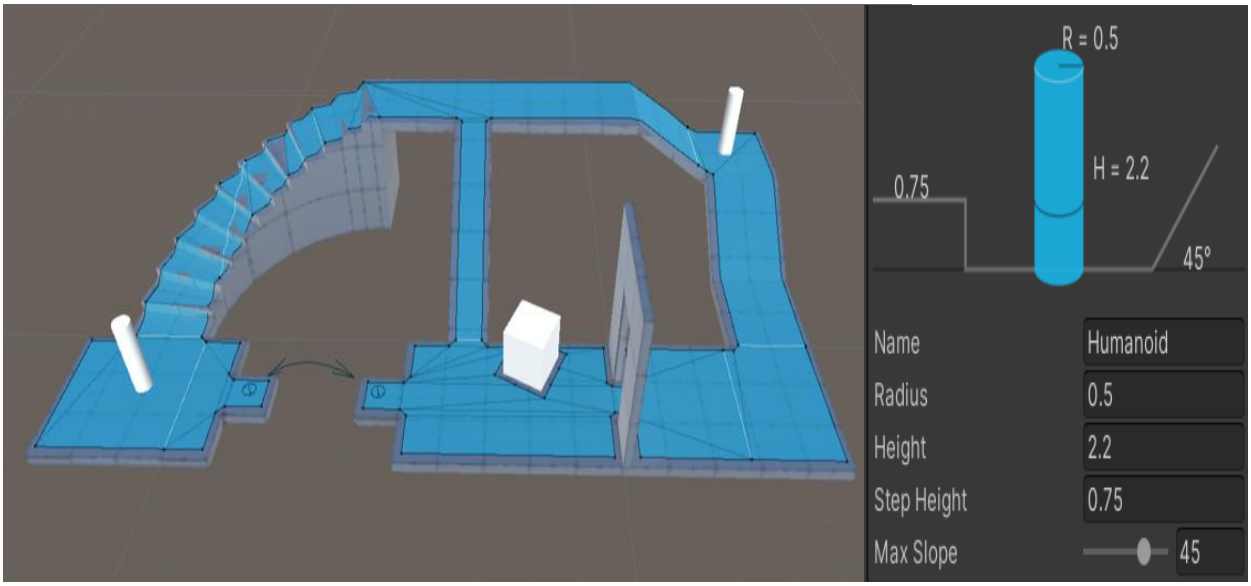

Рисунок 4 Навігаційна система Unity

На рисунку 4 зображено те як працює система навігації в Unity. Потрібно обрати що є поверхнею по якій можна ходити, бігати, або стрибати – тобто створити NavMesh [8], а також обрати поверхню по якій заборонено пересуватися. Як результат ми отримуємо спрощену репрезентацію світу, в якій зазначено які властивості яка поверхня набуває. В додаток до налаштування поверхні потрібно зазначити параметри нашого агента або ж, в даному випадку, ворога – висоту, його радіус, кут на який він може підійматися та дальність стрибка, що також зображено на рисунку 4. Синім зображена поверхня, по якій може ходити агент, сірим – яка є для нього перешкодою.

Таким чином, для побудови такої системи в моєму проєкті треба обрати будинки, стіни, паркани та якісь декорації як те, що вороги повинні оминати, а доріжки нашого рівня як поверхня по якій можна пересуватися.

3.5.2 AI Vision Cone

Для того, щоб зробити ворогів більш цікавими та реалістичними, було прийняте рішення обмежити їхній зір конусом. Це дуже популярний та дієвий прийом, який можна зустріти майже у кожній грі подібного жанру.

Сам зір наших ворогів реалізован через Physics.Raycast – метод, який запускає промінь за заданими параметрами (точка пуску, вектор направлення, дистанція та маска) і повертає те, у що він влучив або null. Відповідно, якщо він влучає у гравця, то значить ворог його побачив. Для налаштування рівнів можна змінюючи вхідні параметри кожного ворога для того щоб варіювати як далеко він бачить, який в нього кут огляду тощо.

Конусна форма зору реалізована за допомогою великої кількості цих променів, вектор направлення змінюється в залежності від кута. Для того, щоб з кута отримати вектор треба скористатися тригонометричною функцією: *vector = (cos(angle), 0, sin(angle))*.

За аналогією можемо побудувати графічне відображення зору ворогів зображене зеленим кольором, для того щоб гравець теж розумів їхні спроможності. Зібравши це до купи ми отримаємо конусний зір ШІ:

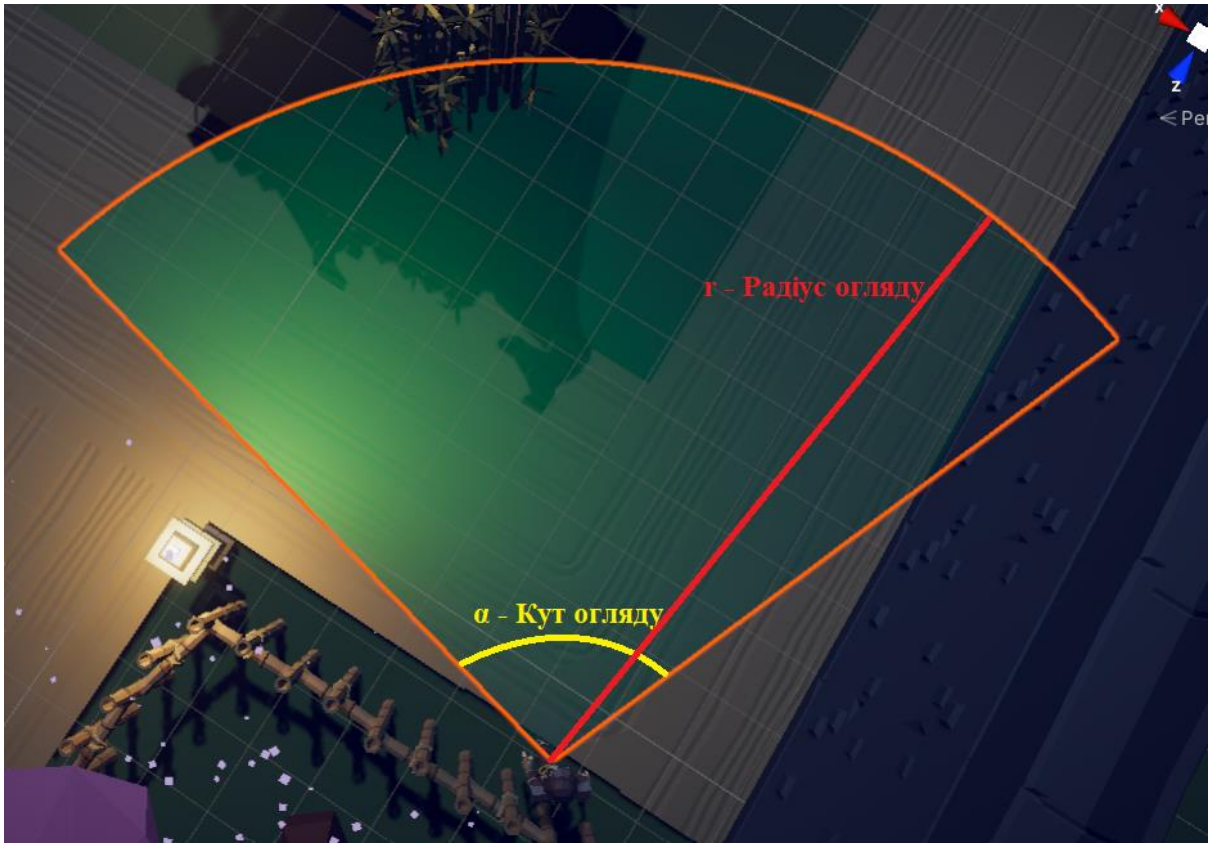

Рисунок 5 Конусний зір ШІ

3.5.3 Behaviour Tree

Останньою і найважливішою складовою ШІ ворога це його поведінкове дерево. Оскільки основи логіки роботи та основні типи нод вже були докладно описані у розділі 2.4, то знову на них зупинятися не буду. Повна структура дерева зображена на рисунку 6.

Як і належить, починається дерево з базової Root ноди, яка в даному випадку називається Entry. Як вже було описано раніше, дуже часто після кореневої ноди ставлять Repeater, для того щоб дерево оновлювалося постійно, а не тільки один раз. Це дерево – не виняток, тут також після кореневої ноди іде повторювач. Дочірньою нодою повторювача стоїть

селектор. Саме він буде обирати одну з наступних трьох гілок, кожна з яких має в собі деяку послідовність.

Перша гілка складається з послідовності трьох нод – однієї conditional ноди та двох action нод. Перша нода і я перевіркою на те, чи є гравець у полі зору ворога. Якщо вона повертає Success, то дерево переходить до наступної ноди. Для релаістичності, було прийняте рішення робити коротку паузу між виявленням і погонею за гравцем, тому друга нода чекає якийсь час, дивлячись чи гравець все ще знаходиться в полі зору. Після того, як час сплив, то дерево переходить до третьої ноди, яка відповідає за погоню за гравцем.

Друга гілка відповідає за логіку дій ворога у випадку якщо він почув якийсь звук. Це може бути спричинено тим, що гравець застосував свою здіність і кинув поруч з ворогом відволікаючий предмет. Структура цієї гілки дуже схожа на попередню. Спочатку іде нода умови, яка перевіряє чи був ворог зачеплений «звуковою хвилею» від предмета. У разі повернення Success, дерево переходить до наступної Action ноди, яка заставляє ворога повернутися в сторону, звідки він почув звук. Остання нода просто чекає певний час, перед тим як повернути Success.

Третя гілка з поведінкового дерева ворогів відповідає за патрулювання. Саме вона буде виконуватися більшу частину часу. Вона складається з повторювача, який постійно повторює послідовність двух нод. Ця послідовність складається з двух Action нод – Patrol та LookAround. Відповідно спочатку ворог іде до наступної точки патрулювання. Якщо ж він до неї дійшов, то Patrol повертає Success і послідовність переходить до LookAround, яка заставляє ворога подивитися по сторонам, після чого також повернути Success.

Також кожне дерево використовує свій набір змінних, а саме координати точок, по яким відбувається патрулювання. В додаток до цього, всі дерева ворогів використовують blackboard, про яку було сказано в пункті 2.4.3, яка зберігає референс на гравця, який використовується всередині імплементованих нод.

Таким чином можна отримати комплексну логіку для NPC у грі. Таке дерево покриває головні вимоги дизайну, його можна перевикористовувати та розширяти, за наявності нових NPC у грі.

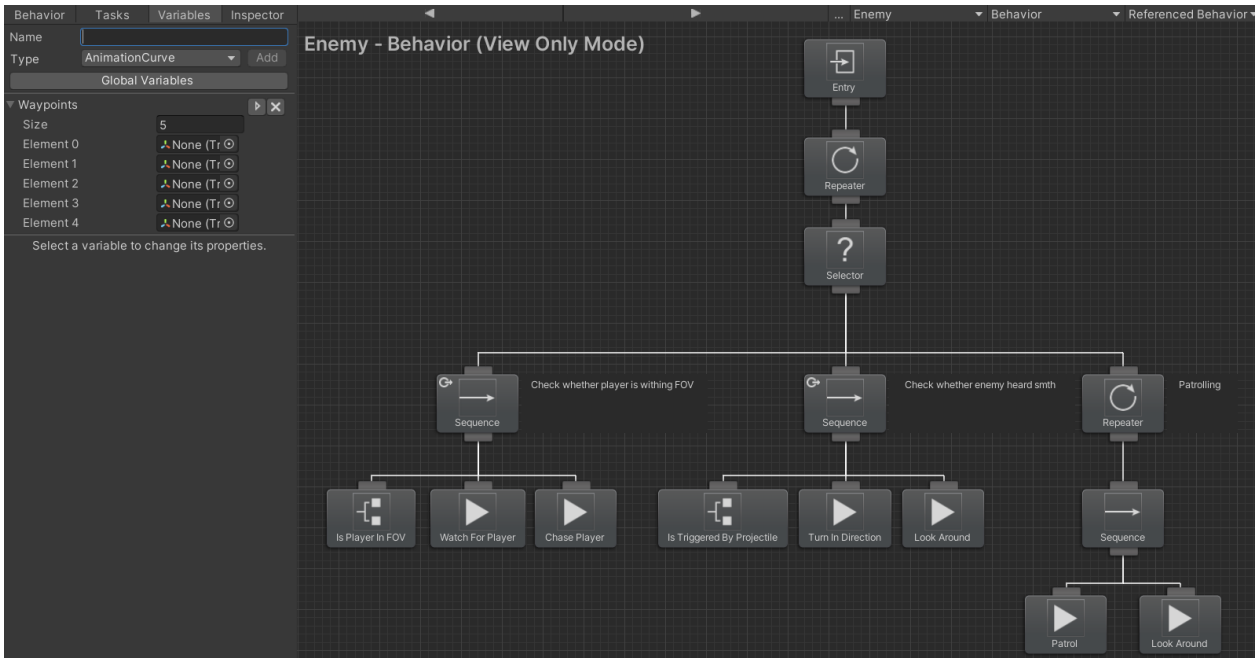

Рисунок 6 Поведінкове дерево ШІ

Висновки

В процесі виконання роботи було досліджено процес розробки відеоігор за допомогою ігрового рушія Unity та мови програмування C#. Були розглянуті основні характеристики, переваги та недоліки цього рушія.

Також була досліджена тема штучного інтелекту у відеограх, а саме які існують його види та підходи до реалізації. Окремо була розглянута тема графів у відеоіграх разом з поведінковими деревами, оскільки цей тип ШІ є основним в контексті даної роботи.

В останній частині роботи було описано розробку практичної частини. Був розроблений ігровий дизайн, розглянуті технології, які були використані, а також проведена декомпозиція реалізації штутчного інтелекту, який було імплементовано. Окремо були розглянуті складові ШІ, а саме: алгоритм пошуку шляху, реалізацію конусного зору агентів та побудову поведінкового дерева.

Список термінів та скорочень

3D – 3-dimensional – щось, що має 3 виміри. В даному випадку мова іде про простір.

DOTS – Data-Oriented Technology Stack – стек орієнтованих на дані технологій.

ШІ – штучний інтелект.

ААА проєкт – проєкт, націлений на досягнення найвищих показників якості, масштабу, бюджету тощо в певній сфері. В даній роботі мова іде про сферу відеоігор.

NPC – Non-Player Character – персонаж, керований комп’ютером.

PCG – procedural content generation – процедурна генерація контенту.

Pathfinding – пошук шляху з однієї точки до іншої.

PEM – player experience modelling – моделювання досвіду гравця.

NavMesh – Navigation Mesh – навігаційна поверхня.

Agent – агент, ігрова сутність, яка взаємодіє зі світом.

AI Vision Cone – artificial intelligence vision cone – конусний зір ШІ.

BT – Behaviour Tree – поведінкове дерево.

Список використаних джерел

1. [Електронний ресурс] – https://assetstore.unity.com/
2. Game AI Revisited: Georgios N. Yannakakis, 2012 – https://yannakakis.net/wp-content/uploads/2012/03/gameAI.pdf
3. [Електронний ресурс] – http://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html
4. J. Juul. A Casual Revolution: Reinventing Video Games and Their Players. MIT Press, 2009.
5. Game Programming Patterns: Robert Nystrom, 2011.
6. [Електронний ресурс] – https://unity.com/unity/features/editor/art-and-design/cinemachine
7. [Електронний ресурс] – https://docs.unity3d.com/Manual/Navigation.html
8. [Електронний ресурс] – https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html
9. [Електронний ресурс] – https://www.physicsclassroom.com/class/1DKin/Lesson-6/Kinematic-Equations
10. Daniel Hilburn: Simulating Behaviour Trees – http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter08_Simulating_Behavior_Trees.pdf
11. [Електронний ресурс] – https://www.statista.com/outlook/dmo/digital-media/worldwide