МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ»
Фізико-математичний факультет
Кафедра інформатики та прикладної математики

«Допущено до захисту»
Завідувач кафедри
_____ Соловйов В.М.
«___» _____ 2021 р.

Реєстраційний № ____
«___» _____ 2021 р.

РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПОВЕДІНКИ НЕКЕРОВАНИХ ГРАВЦЕМ ПЕРСОНАЖІВ У 3D-ШУТЕРІ

Кваліфікаційна робота студента
фізико-математичного факультету
групи Ім-16
ступінь вищої освіти «магістр»
спеціальності
014.09 Середня освіта (Інформатика)
Ростального Богдана Альбертовича

Керівник к.ф.-м.н., доцент
Моісеєнко Н.В.

Оцінка:
Національна шкала _____
Шкала ECTS __ Кількість балів _____

Голова ЕК: _____ _____
Члени ЕК: _____ _____
_____ _____
_____ _____

Кривий Ріг - 2021

Я, Ростальний Богдан Альбертович,
розумію і підтримую політику Криворізького державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело.
Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1 ШТУЧНИЙ ІНТЕЛЕКТ В КОМП'ЮТЕРНИХ ІГРАХ 6

1.1. Історія штучного інтелекту та його роль у сучасному світі 6

1.2. Використання штучного інтелекту в комп'ютерних іграх 9

1.3. Перспективи розвитку штучного інтелекту у ігровій індустрії 17

Висновки до розділу 1 18

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ 20

2.1. Порівняння ігрових рушіїв 20

2.2. Unreal Engine 4 28

2.3. Модель поведінки некерованих гравцем персонажів 36

Висновки до розділу 2 40

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПОВЕДІНКИ НЕКЕРОВАНИХ ГРАВЦЕМ ПЕРСОНАЖІВ У 3D-ШУТЕРІ 41

3.1. Програмна реалізація поведінки некерованих гравцем персонажів у 3D-шутері 41

3.2. Опис ігрового рівня для тестування поведінки некерованих гравцем персонажів 50

3.3. Тестування програмної реалізації поведінки некерованих гравцем персонажів у 3D-шутері 53

Висновки до розділу 3 56

ВИСНОВКИ 57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58

ВСТУП

Актуальність. Багато років поспіль кіберспорт набирає популярність. І це не дивно, адже призові фонди на турнірах по певним іграм можуть сягати декількох мільйонів доларів. Турнір з гри «Dota 2» під назвою «The International 2021» поставив рекорд, коли його призовий фонд перейшов відмітку у 40 мільйонів доларів. Корейська кіберспортивна асоціація (KeSPA) навіть добилась визнання кіберспорту олімпійським видом спорту другого рівня.

Майже всі ігри мають певний ігровий штучний інтелект. Це можуть бути міньйони у іграх жанру MOBA, звичайні неігрові персонажі в іграх жанру RPG, усі непідконтрольні гравцю юніти і супротивники у стратегіях, і звісно супротивники у іграх жанру шутер.

В таких іграх у розробників є багато можливостей зробити так, щоб гравець не мав навіть теоретичної можливості обіграти бота, навіть без занадто детально розроблених сценаріїв руху. Можна дати боту інформацію про те, де знаходиться гравець, наділити його миттєвою реакцією, навчити стріляти прямо в ціль найпотужнішими типами зброї, дати команду на постійне зближення. З таким супротивником вправитися буде вкрай важко.

В зв’язку з цим для реалістичності віртуальних суперників авторам ігрового штучного інтелекту доводиться користуватись концепцією відкритого світу: комп’ютерний гравець має тільки ту інформацію, яку міг би мати на його місці гравець реальний, тобто «бачити», «чути» та «знати» те саме, що й живий супротивник.

Все вищезазначене визначило м**ету роботи:** розробити модель та програмно реалізувати поведінку некерованих гравцем персонажів у 3D шутері.

Відповідно до мети дослідження визначено такі **завдання**:

- вивчити відповідну літературу з розробки ігор та застосування штучного інтелекту в іграх;

- проаналізувати переваги та недоліки ігрових рушіїв, що існують, які використовуються при розробці комп'ютерних ігор та обрати інструментарій для розробки ігри;

- дослідити сценарії поведінки некерованих гравцем персонажів, що існують та створити на їх основі модель поведінки та розробити необхідні алгоритми;

- програмно реалізувати та протестувати функції, що реалізують модель поведінки некерованих гравцем персонажів.

Об'єктом є комп'ютерні ігри на базі ігрових рушіїв.

Предметом є модель поведінки некерованих гравцем персонажів в комп'ютерній грі.

Практична цінність роботи полягає в розроблених функціях, що реалізують поведінку некерованих гравцем персонажів в комп'ютерній грі та можливості використовувати їх як за прямим призначенням, так і в якості навчального матеріалу в курсі «Розробка комп'ютерних ігор».

Структура роботи. Пояснювальна записка складається з вступу, трьох розділів, висновків та списку використаних джерел з 25 найменувань. Робота містить 57 сторінок тексту, 58 рисунків. Загальний обсяг 60 сторінок.

РОЗДІЛ 1
ШТУЧНИЙ ІНТЕЛЕКТ В КОМП'ЮТЕРНИХ ІГРАХ

1.1. Історія штучного інтелекту та його роль у сучасному світі

За означенням наведеним на сайті Oracle, штучний інтелект – це система, або машина, що здатна імітувати людську поведінку для виконання певних задач та може поступово навчатись, використовуючи отриману інформацію. Прикладами штучного інтелекту можуть бути: розпізнавання зображень, розумний пошук у пошукових системах, персоналізовані рекламні рекомендації, голосові помічники, автопілоти, тощо [16].

Можна стверджувати, що протягом останнього десятиріччя штучний інтелект стрімко набрав популярність у різних сферах, через що став чимось повсякденним, але це було не завжди. Якщо звертатися до висловлювання дійсного члену Європейської асоціації штучного інтелекту, професора Жана-Габріеля Ганасія «штучний інтелект, як галузь науки, офіційно побачила світ у 1956 році на літньому семінарі в Дартмут-коледжі (Хановер, США), який організували четверо американських учених: Джон Мак-Карті, Марвін Мінськ, Натаніель Рочестер і Клод Шеннон. З того часу термін «штучний інтелект», придуманий, найімовірніше, з метою залучення загальної уваги, став настільки популярним, що сьогодні навряд чи можна зустріти людину, яка ніколи її не чула. З часом цей розділ інформатики розвивався дедалі більше, а інтелектуальні технології останні шістдесят років зіграли значної ролі у зміні образу світу» [3].

Для Джона Мак-Карті та Марвіна Мінські, як і для інших організаторів літнього семінару в Дартмут-колледжі, штучний інтелект від самого початку був областю науки, котра займалась комп'ютерним моделюванням різноманітних навичок інтелекту, не звертаючи увагу на те який саме був інтелект: людський, тваринний, рослинний соціальний чи філогенетичний. В

основі цієї науковою дисципліни лежить припущення про те що усі когнітивні функції, такі як навчання, мислення, розрахунок, сприйняття, пам’ять, навіть наукове відкриття чи художня творчість, можуть бути описані с точністю, що дає змогу запрограмувати комп’ютер на їх відтворення.

За час свого існування штучний інтелект зазнав багато змін, в історії його розвитку можна виділити шість етапів.

Період пророцтв. Початок вивчення штучного інтелекту протягом якого вчені робили дещо необачні висловлювання, котрими пізніше їх дорікали. Це було зумовлено успіхами у перший час вивчення штучного інтелекту [3].

Похмурі часи. Сповільнення вивчення штучного інтелекту в середині 1960-х років. В 1965 році десятирічних хлопчик отримав перемогу в шаховому матчі над комп’ютером. 1966 року в доповіді, підготовленій на замовлення Сенату Сполучених Штатів Америки, йшлося про внутрішні обмеження, властиві машинному перекладу. Близько десяти років преса відгукувалася про штучний інтелект несхвально [3].

Семантичний штучний інтелект. Дослідження не зупинились, але пішли в нових напрямах. Вчені зацікавилися психологією пам'яті, механізмами розуміння, які намагалися імітувати на комп'ютері, і роллю знань у розумовому процесі. Це призвело до появи методів семантичного представлення знань, що значно розвинулися в середині 1970-х років, а також до створення експертних систем, названих так тому, що для відтворення розумових процесів у них використовувалися знання кваліфікованих фахівців. На початку 1980-х років на експертні системи покладалися великі надії через широкі можливості їх застосування, наприклад, для медичної діагностики [3].

Неоконекціонізм та машинне навчання. Технічні удосконалення дозволили вченим розробити алгоритми машинного навчання, завдяки яким комп’ютери отримали змогу накопичувати знання і автоматично перепрограмовуватися на основі власного досвіду. Такі інтелектуальні системи широко розповсюджені і у сучасному світі, наприклад: розпізнавання

відбитків пальців, тексту, мови. Також вчені почали комбінувати методи з різних дисциплін для створення гібридних систем [3].

Від штучного інтелекту до інтерфейсів «людина – машина». Наприкінці 1990-х років штучний інтелект почали об'єднувати з робототехнікою та інтерфейсом «людина – машина», з метою створення автономних агентів, котрі мали би почуття та емоції. На основі цих досліджень з'явився новий напрям дослідницького напрямку – афективних досліджень. Ці дослідження направлені на аналіз реакцій суб'єкта здатного відчувати емоції і відтворенні їх на машині. Це дозволило удосконалити діалогові системи [3].

Відродження штучного інтелекту. Завдяки обчислювальним потужностям сучасних комп'ютерів дослідники отримали можливість комбінувати великі данні з методами глибокого навчання, котрі базуються на використанні нейронних мереж. Це призвело до того що розробки на основі штучного інтелекту стали для людей чимось повсякденним і свідчить про відродження штучного інтелекту.

Технології на основі штучного інтелекту дуже сильно допомагають на підприємствах. Вони поліпшують продуктивність завдяки автоматизації процесів і задач котрі раніше виконували люди. Також штучний інтелект здатен обчислювати та інтерпретувати великий обсяг даних. Це дозволяє компаніям створювати персональні рекомендації для кожного користувача. Штучний інтелект використовується і у сфері безпеки, адже технологія здатна виявляти надзвичайні ситуації, або запобігати шахрайству.

Ще одною сферою в якій розповсюджений штучний інтелект є розробка відеоігор. Розробники використовують штучний інтелект для генерації ландшафту, будівель, дерев, тощо. Ну і звісно розробники використовують його для створення супротивників для гравців. Хоча і штучний інтелект в комп'ютерних іграх дещо примітивний, там він має зовсім інші цілі.

1.2. Використання штучного інтелекту в комп’ютерних іграх

Штучний інтелект широко розповсюджений в ігровій індустрії. Його використовують як в процесі розробки, так і в процесі гри. Штучний інтелект у іграх може виконувати різноманітні задачі: від обробки загального набору правил, відповідальних за поведінку базових об’єктів, до управління персонажами. Найпростішою системою штучного інтелекту є система, що базується на певному наборі правил [8]. Яскравим прикладом такої системи є гра Pac-Man (Рис. 1.1). Кожен привид діє за певним алгоритмом правил. Один з привидів завжди повертав вліво, друге йде вправо, третє може повертатися в будь-яку сторону і четверте завжди переслідує гравця. Для більшості ігор подібні алгоритми занадто прості, адже з часом гравець зможе визначити основні правила, а значить зможе легко здолати супротивника.

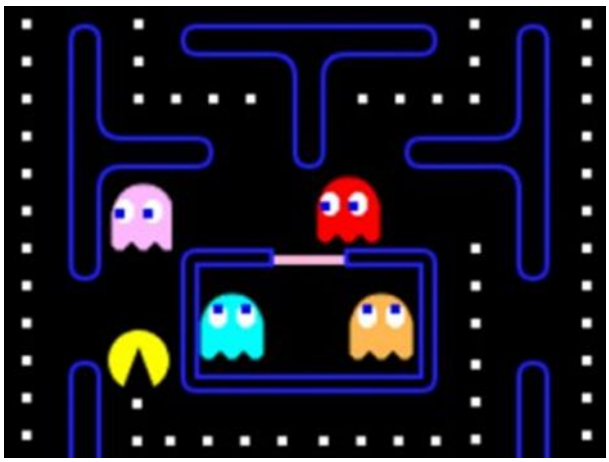


Рис. 1.1. Pac-Man

Також розробники не ставлять ціль створити ворога, котрого неможливо буде здолати, адже гравцям не буде цікаво грати в гру, в яку неможливо виграти. Як казав Тімур Бухараєв: «Головна задача штучного інтелекту – не виграти у гравця, а красиво йому віддатися» [2].

Ще один з засобів що широко використовується при розробці ігрового штучного інтелекту є кінцевий автомат. При його реалізації розробники описують усі ймовірні ситуації з якими може зіткнутися штучний інтелект, а

після цього реакцію на кожну з них. Приклад штучний інтелект на кінцевому автоматі наведено на рис. 1.2.

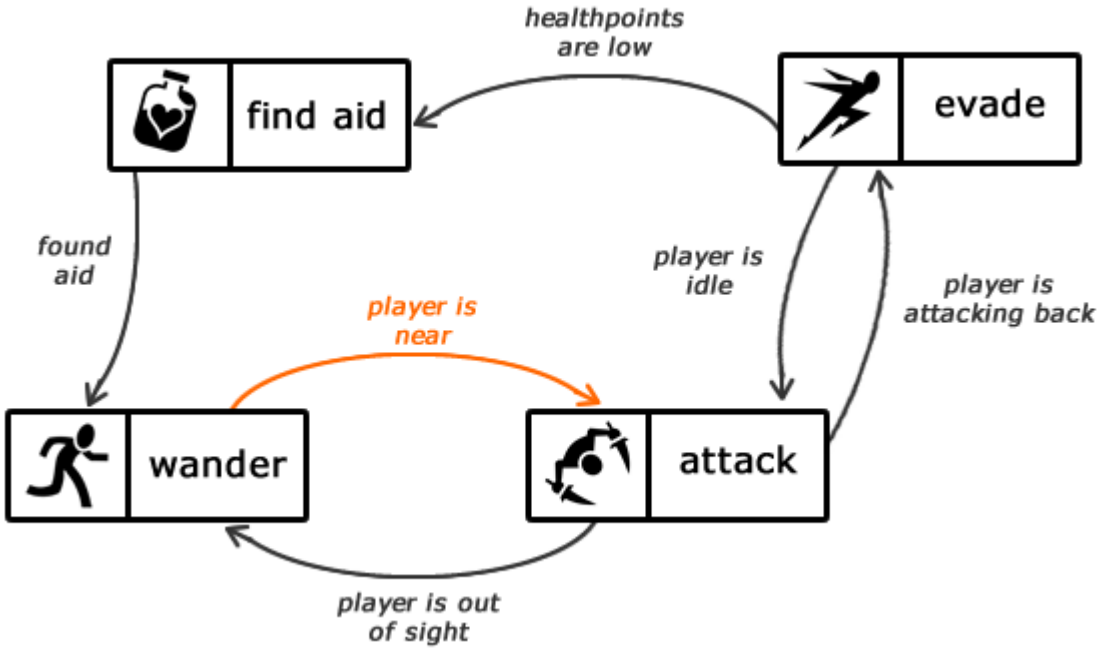


Рис. 1.2. Приклад схеми штучного інтелекту на базі кінцевого автомату

Інший більш складний метод – це використання алгоритму дерева пошуку Монте-Карло (Рис. 1.3). Він був розроблений для того щоб зменшити повторюваність, що притаманна кінцевому автомату. Алгоритм дерева пошуку Монте-Карло спочатку аналізує усі можливі ходи, котрі доступні автономним агентам в конкретний момент часу, а потім перевіряє усі можливі дії якими гравець може відповісти. Після цього алгоритм знову повертається до оцінки автономного агента, але вже на основі дій гравця [11].

Сучасні розробники намагаються створити скоріше не максимально складний штучний інтелект, а скоріше якісно ввести його в систему гри, аби досягнути так званого емерджентного геймплею. Емерджентність – властивість системи, в якій всі елементи взаємопов'язані та взаємозалежні. Це означає, що з впливом на одну підсистему, реагувати будуть і інші [11].

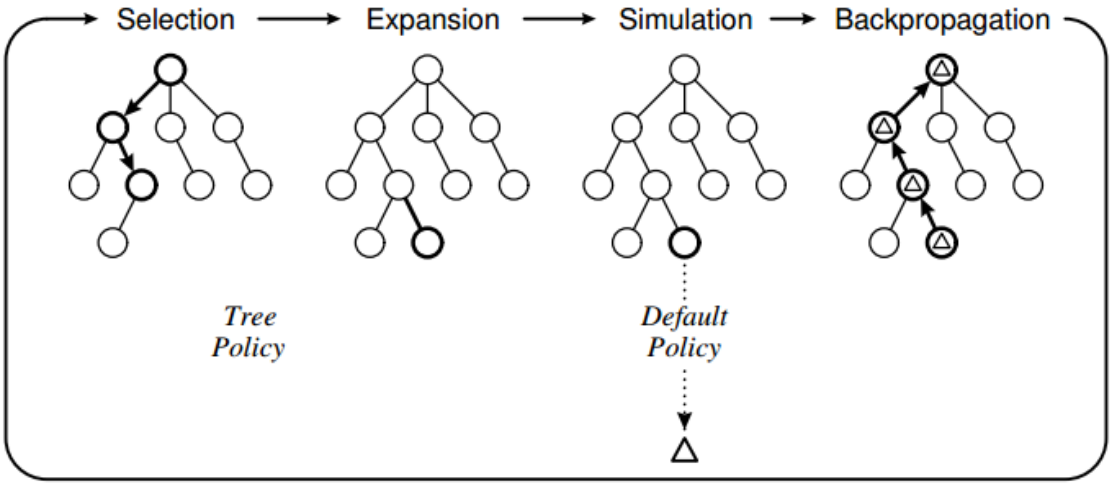


Рис. 1.3. Приклад алгоритму на базі дерева пошуку Монте-Карло

Гарним прикладом може слугувати гра Red Dead Redemption 2. Гравець може взаємодіяти з оточуючим світом багатьма способами. Наприклад: якщо зайти у місто будучи заляпаним кров'ю, жителі покличуть шерифа, якщо попасти противнику у зброю він може її упустити, або якщо стояти поряд із жителями вони спочатку перервуть розмову, а потім або розійдуться, або нападуть на гравця (Рис. 1.4.). І таких моментів у грі неймовірно багато, через що різні гравці можуть переживати зовсім різні ситуації.

Рис. 1.4. Напад на гравця у грі Red Dead Redemption 2

Якщо брати приклад сучасного, адаптивного штучного інтелекту то згадується гра 2015 року випуску студії Kojima Production – «Metal Gear Solid V: The Phantom Pain». У ній гравець повинен переміщуватись між різними точками ігрового світу, якомога скритніше, ховаючись від супротивників. Гравець може підкрадатися до ворогів, допитувати їх (Рис 1.5), отримуючи цікаву інформацію про оточення, або персонажів. Але основною перевагою цієї гри є реакція штучного інтелекту на дії гравця. На початку гри головного зустрічають звичайні супротивники, але штучний інтелект поступово адаптується і змінює оточення підстроюючись під дії гравця. Якщо часто нейтралізує супротивників в пострілом в голову, вороги починаю носити каски, якщо гравець часто нападає під покровом ночі, серед ворогів з'являються патрульні з приборами нічного бачення. Іноді доходить до того що проти гравця виводять важко-броньованих ворогів.

Рис 1.5. Допит ворожого солдата у грі «Metal Gear Solid V: The Phantom Pain»

Іншим прикладом може слугувати «Alien: Isolation» – гра в жанрі survivalhorror студії Creative Assembly, де головним ворогом є ксеноморф (Рис 1.6) з відомого фільму Рідлі Скота. Чужий, мабуть найрозумніший ворог

на сьогодення, ніколи не знаєш що він зробить у наступну хвилину. Команда розробників виконала велику роботу на пропис його інтелекту. Розробники вирішили зробити лише одного прибульця на кораблі, що дозволило зменшити затрачені ресурси, і зробити акцент не на кількості ворогів а на якості. Гравцю доводиться обережно красться оскільки чужий може почути гравця і вилізти з найближчого повороту. Пізніше у грі з'являються засоби для залякування чужого, але це не заважає йому втекти та обійти гравця позаду. Він майстерно користується вентиляцією, використовуючи її як для засад, так і для швидкого переміщення при обходжені гравця. Поведінка прибульця створена на основі прописаного заздалегідь дерева. Це дерево складається більш ніж зі 100 вузлів. Розробники прийшли до цікавого рішення і на початку гри прибулець використовує лише 30 цих вузлів, а вже протягом проходження певних умов протягом всієї гри, система розблокує і інші. Це створює ілюзію того що ксеноморф постійно вчиться, через що гравець завжди знаходиться в напрузі [5].

Рис 1.6. Ксеноморф з гри Alien Isolation

Ще одним цікавим прикладом штучного інтелекту є система заклятих ворогів під назвою «Nemesis» з гри Middle-earth: Shadow of War, створена командою Monolith Productions. В грі головний герой повинен захоплювати ворожі фортеці. Для того щоб зменшити бойовий потенціал захисників, герою доведеться вбивати, або вербувати ворожих капітанів (Рис 1.7). І в цих капітанах криються основні можливості Nemesis. Кожен капітан має свої переваги та недоліки (Рис 1.8), свій загін, свої і кровні брати. Капітани воюють між собою і можуть навіть без втручання гравця, повбивати один-одного. Shadow Of War створює штучну соціальну пам’ять. Ця штучна пам’ять пригортає увагу гравця, адже вороги починають думати про гравця на дещо іншому, особистому рівні. Вони запам’ятовують різні ситуації, в яких вони зіштовхувались з гравцем, і зазвичай намагаються помститися. Вони можуть напасти під час бою з іншими капітанами, під час дослідження локацій, під час штурму фортець, тощо. Але сама погана ситуація це коли один з підлеглих гравцю капітанів зраджує його, і наносить удар у саму відповідальну мить, а все через те що гравець вбив його кровного брата. І з таких моментів складається сама гра чим сильно пригортає увагу гравців [5].

Рис 1.7. Ворожі капітани

Рис 1.8. Характеристики капітана

Неможливо забути гру «F.E.A.R.», Monolith Productions. Штучний інтелект «страху» ще не перевершила жодна гра жанру шутер. Розробники зробили супротивників найбільш наближеними до реальних гравців. Вони ховаються за укриттями, ведуть вогонь всліпу, влітають у вікна, кидають гранати, спілкуються один з одним. Один з розробників розповів що вся поведінка розкривається у формулі трьох станів, показаній на рис 1.9.

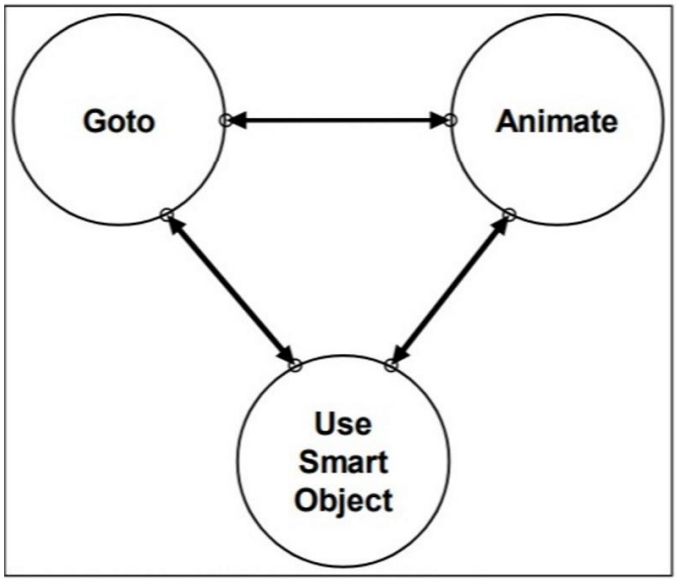


Рис 1.9. Формула «Three States».

Джефф Оркін називає цю формулу «Three States». Замість окремого вказування анімацій, необхідна анімація виконувалась через Smart Object в базі даних. Отже усе чим займається штучний інтелект в F.E.A.R. це рух и відтворення анімації. Коли супротивник рухається к укриттю, він просто рухається в необхідну позицію, після чого відтворює анімацію пригинання або повзання [9].

Головне завдання при моделюванні поведінки персонажа – визначити, коли потрібно переключатись між станами Goto та Animate, а також які параметри необхідно встановити. Розробники ввели в гру систему планування, котра дозволяє штучному інтелекту самостійно вирішувати коли переходити з одного стану в інший.

В F.E.A.R. штучний інтелект використовує укриття і координується між собою для організації вогню на пригнічення, що дозволяє іншим членам загону обходити гравця. Вороги покидають укриття лише якщо їх притиснуть, а стріляють всліпу лише в крайньому випадку.

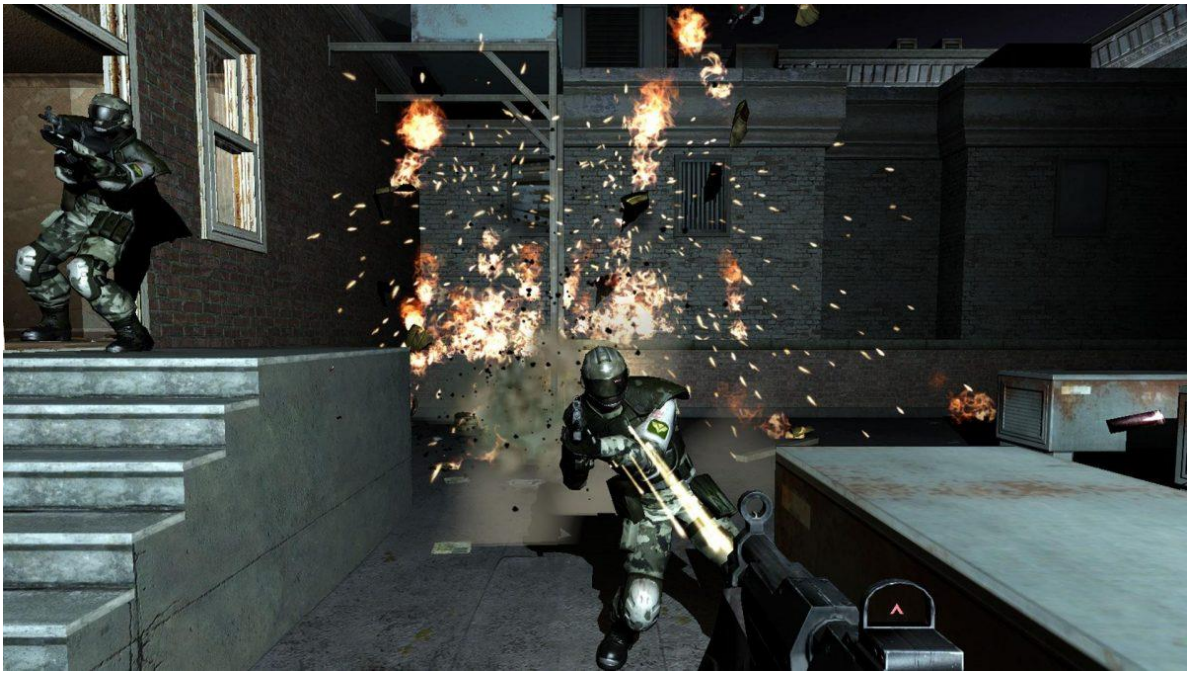


Рис 1.10 Координація автономних агентів у F.E.A.R.

1.3. Перспективи розвитку штучного інтелекту у ігровій індустрії

Не звертаючи уваги на старання розробників рано чи пізно гравець починає передбачати дії неігрових персонажів, тому зараз набирають популярність експериментальні концепції, а саме – ланцюги Маркова.

Ланцюг Маркова – це марківський процес з дискретним часом та дискретним простором станів [23]. Отже, ланцюг Маркова – це дискретна послідовність станів, кожен із яких береться з дискретного простору станів (кінцевого чи нескінченного), що задовольняє марківській властивості [23]. Ланцюги маркова використовуються у самих різних областях, від генерації тексту до фінансового моделювання.

Найвідоміший приклад використання ланцюгів Маркова є Subreddit Simulator. Це форум який повністю ведуть боти. Вони створюють пости, спілкуються, обговорюють новини, відправляють один-одному відео та зображення, тощо. Іноді їхні діалоги важко відрізнити від діалогу реальних людей, а іноді виходить нісенітниця.

Ланцюги Маркова є імовірнісним автоматом. Зазвичай розподіл вірогідностей представлений у вигляді матриці

$$A=\begin{vmatrix} 0.4 & 0.25 & 0.35 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.6 & 0.35 \end{vmatrix}$$

Матриця буде мати вигляд NxN, де N – кількість станів автомата, а значення (I, J) – вірогідність перейти із стану I у стан J.

Ланцюг Маркова також має вектор початкового стану, представлений матрицею N x 1. Він описує вірогідність початку в кожному з N ймовірних станів:

$$X(0)=\begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix}$$

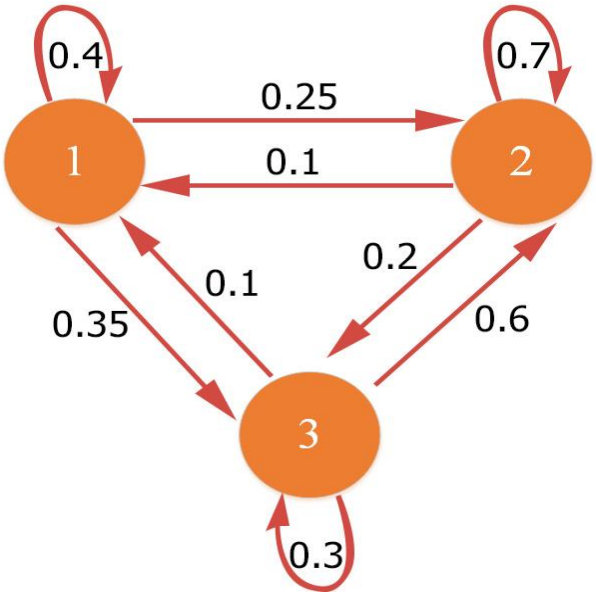


Рис 1.11. Приклад графу створеного на основі ланцюга Маркова

Ще один напрямок у якому розробники можуть розвиватися для поліпшення ігрового штучного інтелекту – це штучний інтелект що навчається. Зараз це дуже важка в реалізації технологія, що базується на використанні нейронних мереж. Вона допомагає створювати ігрових персонажів, котрі зможуть запам’ятовувати інформацію, класифікувати її і на основі отриманих даних еволюціонувати. Нажаль зараз ця технологія непопулярна і потребує занадто багато ресурсів, як людських так і фінансових. Окрім розробки такого штучного інтелекту потрібно буде його ще й навчити, бо інакше на початку гри на гравця будуть випускати ворогів чий інтелект буде знаходитися у кращому випадку на рівні дітей.

Висновки до розділу 1

Штучний інтелект вже давно перестав бути частиною наукової фантастики і прижився у нашому повсякденному житті. Його почали використовувати повсюди, від смартфонів, які є майже у кожного, до великих виробництв. Щодо штучного інтелекту в ігровій індустрії, там штучний інтелект ще далеко від ідеалу. Технології котрі дозволили би зробити його

більш непередбачуваним, або здатнім до еволюції ще вивчають, і не відомо коли ми побачимо подібний штучний інтелект. Натомість розробники придумують дуже цікаві рішення, такі як поступове відкриття нових гілок у дереві поведінки, що дозволяє створити ілюзію розвитку, неймовірно складна система«Nemesis» з гри Middle-earth: Shadowof War, та в цілому живе оточення з Red Dead Redemption 2.

Дослідження багатьох ігор приводить до думки, що розробники зараз створюють не нездоланний штучний інтелект, а той що дозволяє гравцям поринути у ігровий світ. Навіть зараз гравці в першу чергу звертають на деталі: графіку, стиль і саме головне, інтерактивність. Виходячи з усього вищесказаного, можна припустити, що в майбутньому ігровий штучний інтелект буде розвиватися у напрямку емерджентної системи, тому саме це в першу чергу привертаю увагу більшості гравців.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Порівняння ігрових рушіїв

Для розробки моделі поведінки автономних агентів у грі шутері необхідно обрати інструментарій, для цього необхідно проаналізувати існуючі ігрові рушії.

Cocos 2d перший ігровий рушій який ми роздивимось. В першу чергу це рушій створений для розробки ігор на мобільні пристрої (iOS, Android), але також в ньому є можливість створювати ігри для персональних комп'ютерів (Windows, Mac). У чистому вигляді являє собою скоріше фреймворк, але при доповненні його Cocos Creator'ом (Рис. 2.1) стає достатньо зручним ігровим рушієм для створення простих 2D чи 3D ігор [20].

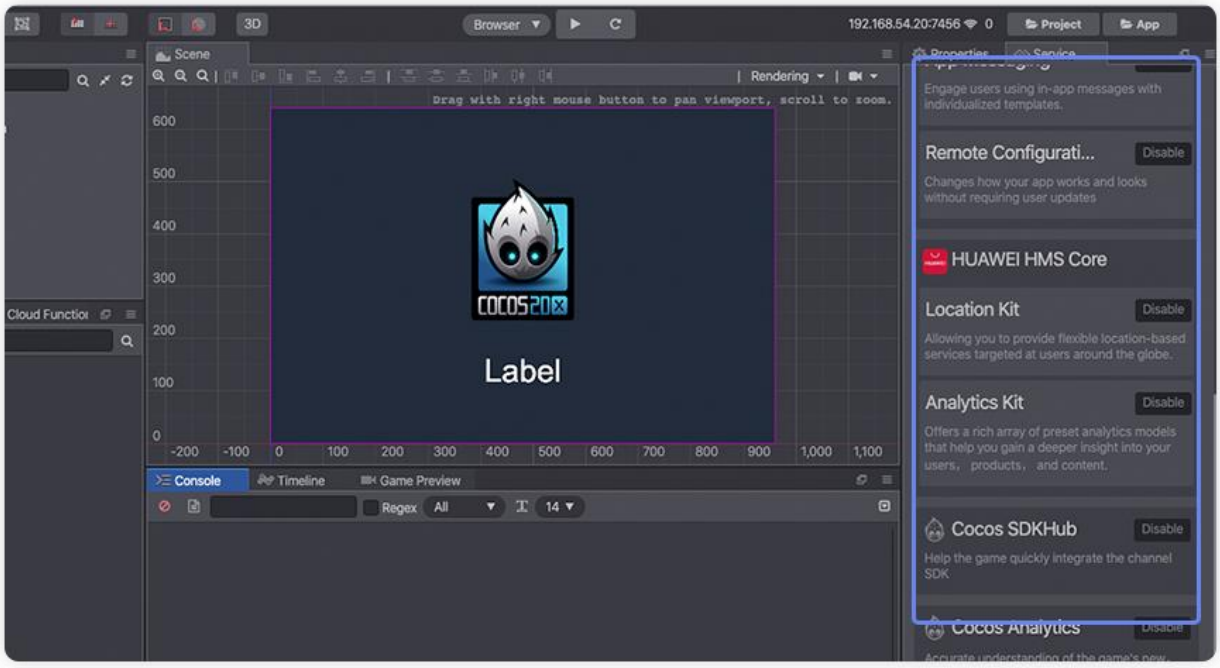


Рис. 2.1. Інтерфейс ігрового рушія Cocos Creator

Наступний рушій який ми роздивимось – це Source. Це відносно старий ігровий, на якому були розроблені такі культові ігри як Half Life 2, Portalta

Portal 2, Dota 2. Першою перевагою Source є доступність. Його інструментарій доступний усім користувачам платформи Steam. Другою перевагою є можливість створювати як повноцінні, самостійні ігри так і модифікації до вже існуючих. Недоліком Source є застарілість технологій. Ігри на Source не можуть конкурувати з іграми створеними на популярних ігрових рушіях [18].

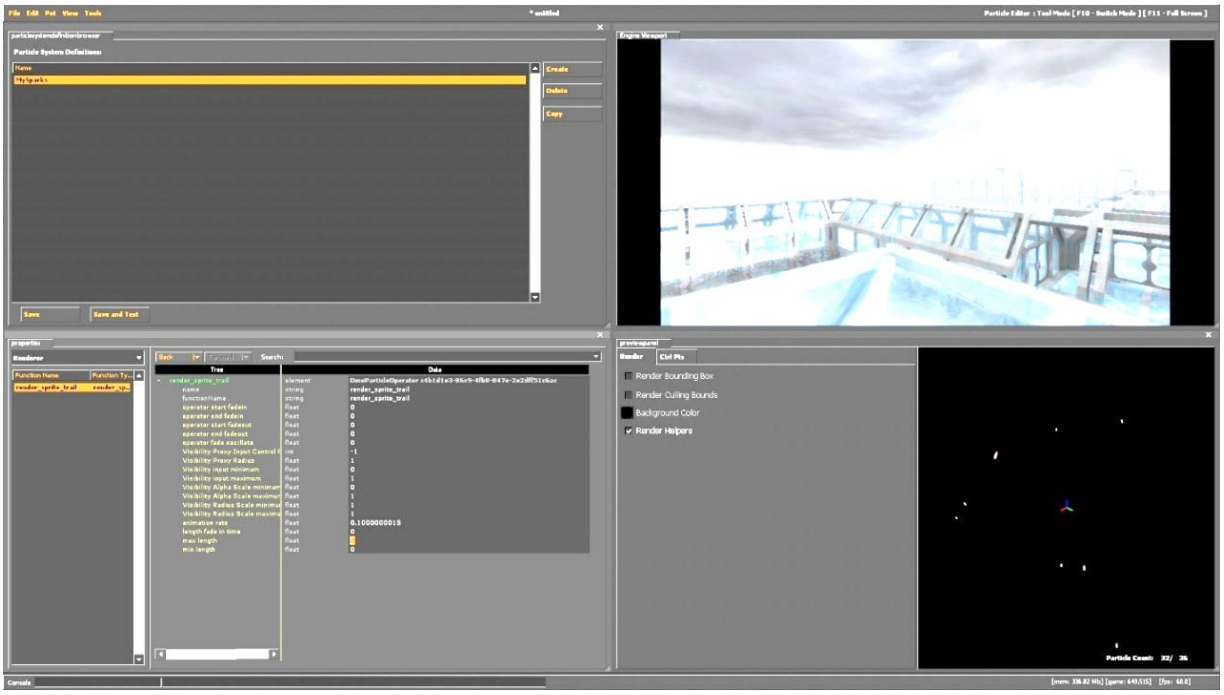


Рис. 2.2. Інтерфейс ігрового рушія Source

Panda 3D не дуже відомий ігровий рушій. Він був створений командою Disney, але пізніше був переданий у розпорядження університету CMU (Carnegie Mellon University). Його перевагою є мова програмування яку він використовує, а саме Python. Недоліком є те що це знову фреймворк, а не ігровий рушій, як і Cocos 2d. До того ж Disney та університет CMU майже не підтримують його, тому дороблennям цього ігрового рушія займається спільнота [14].

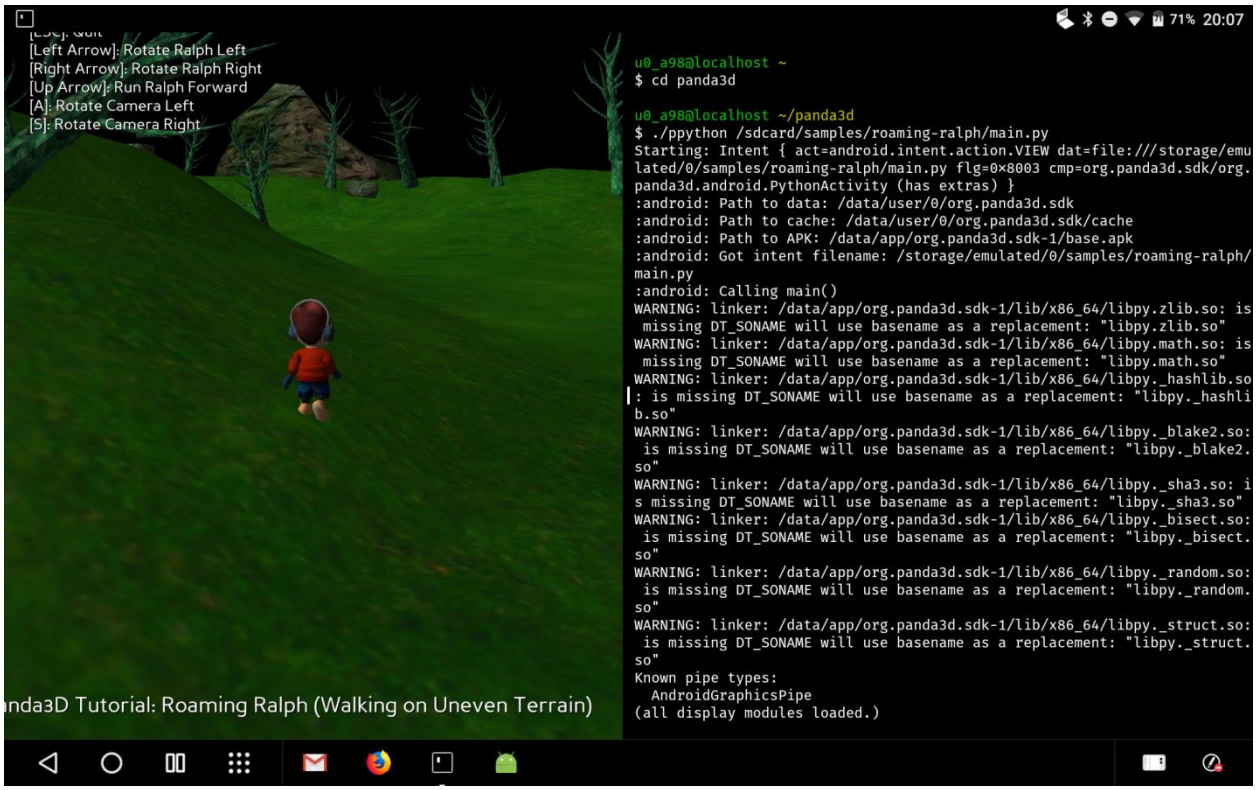


Рис. 2.3. Приклад гри на Panda3D

CryEngine – наступний ігровий рушій який ми роздивимось. Головною перевагою цього рушія є можливість створювати фотореалістичне зображення, завдяки підтримці передових технологій. Також в ньому є власна технологія трасування променів, котра не потребує потужності графічних чипів RTX. Також у CryEngine є інструмент Game SDK, на якому можна швидко створювати свої ігри використовуючи ассети з офіційного сайту Crytek. Але цей рушій не зміг уникнути й недоліків. В першу чергу це велика кількість багів котра заважає зборці проекту. Також до недоліків відносять невелику кількість ассетів, відсутність якісної технічної підтримки і активного ком’юніті [6].

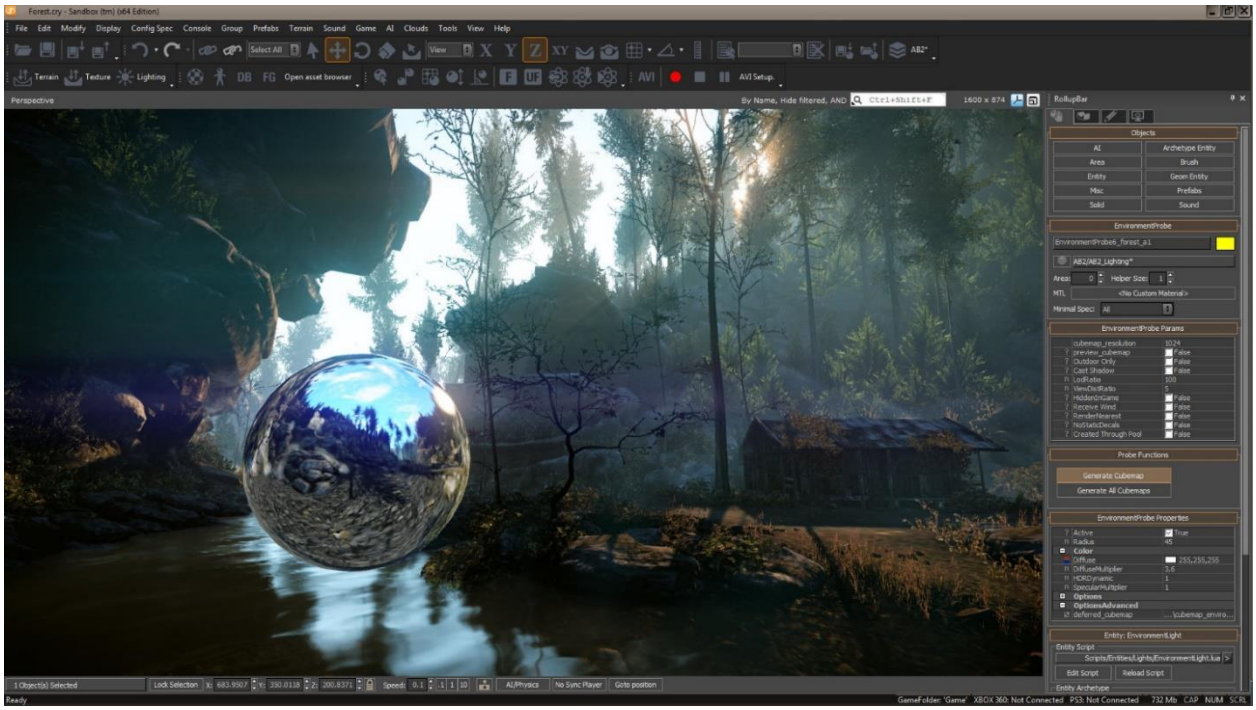


Рис. 2.4. Інтерфейс CryEngine

Godot – один з найпопулярніших ігрових рушіїв. Він має декілька значних відмінностей від інших рушіїв, а саме:

Архітектура рушія, яка заснована на дереві наслідуваних сцен. Кожен елемент сцени може сам легко стати повноцінною сценою.

Робота з ресурсами проекту. Всі ігрові ресурси, від скриптів до графічних ассетів та ігрових сцен, зберігаються в папці проекту як звичайні файли, і не є складною базою даних проекту.

Gogot підтримує різні мови програмування:

- C++
- GDScript (власна мова ігрового рушія)
- C#
- Візуальне програмування

На Godot можна створювати як 2D так і 3D ігри і завдяки його простоті він може стати гарним ігровим рушієм для новачка розробника [13].

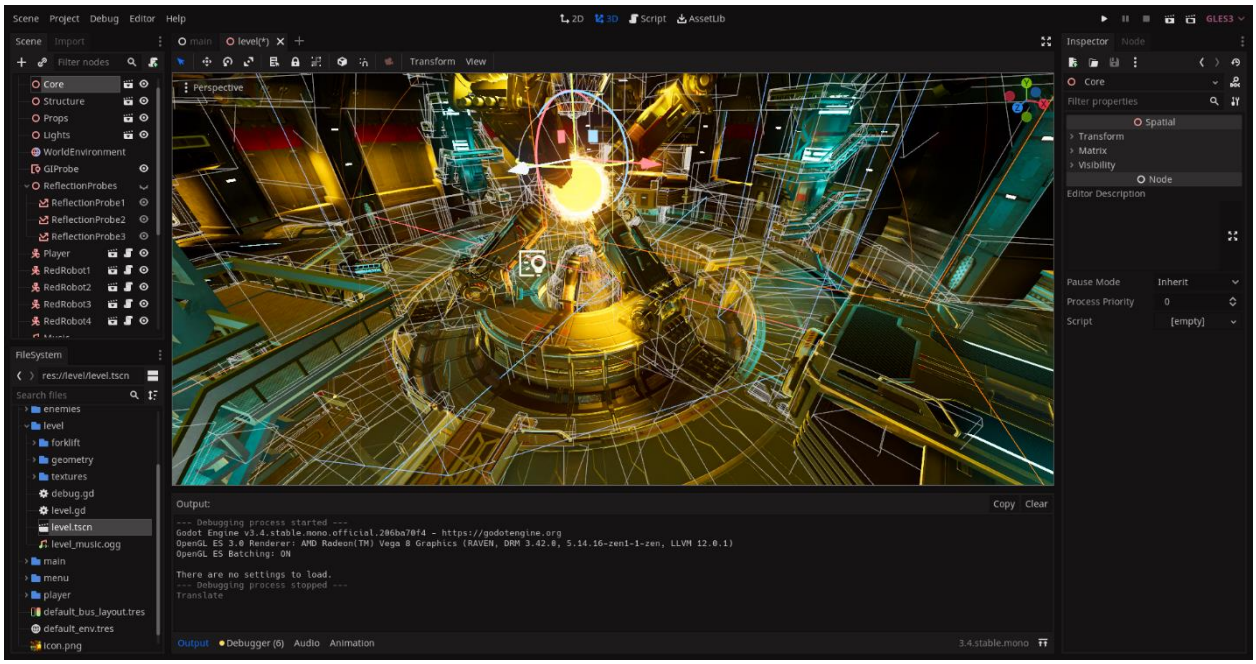


Рис. 2.5 Інтерфейс ігрового рушія Godot

Особисту увагу привертають два найпопулярніших ігрових рушія: Unity3D та UnrealEngine 4 тому зупинимось на них детальніше. У першу чергу слід звернути увагу на причину популярності цих ігрових рушіїв, а саме обидва вони безкоштовні. Звісно якщо гра починає приносити великий прибуток ви повинні виплачувати певні суми розробникам цих рушіїв. Unity3D можна використовувати доки ваша гра не почне приносити прибуток більше ніж 100 тис. $ за останні 12 місяців. Якщо оборот або обсяг залучених інвестицій не перевищує 200 тис. доларів за останні 12 місяців то мінімальна ціна буде становити 400 $ в рік. Якщо оборот або обсяг залучених інвестицій перевищує 200 тис. доларів за останні 12 місяців, то необхідно придбати Pro, абоEnterpriseверсію котрі коштують 1800 $ і 4000 $ в рік відповідно. Звісно кожна з платних версій дає доступ до певних можливостей. З Unreal Engine 4 ситуація дещо інша. Ви повинні виплачувати 5% від доходу якщо він перевищує 1 млн $. При цьому на відмінну від Unityрозробники Unreal Engine 4 не урізають функціонал рушія, щоб продавати його у платних версіях [7].

Перейдемо до інтерфейсу. У обох ігрових рушіях він схожий і в цілому дуже зручний для розробника.

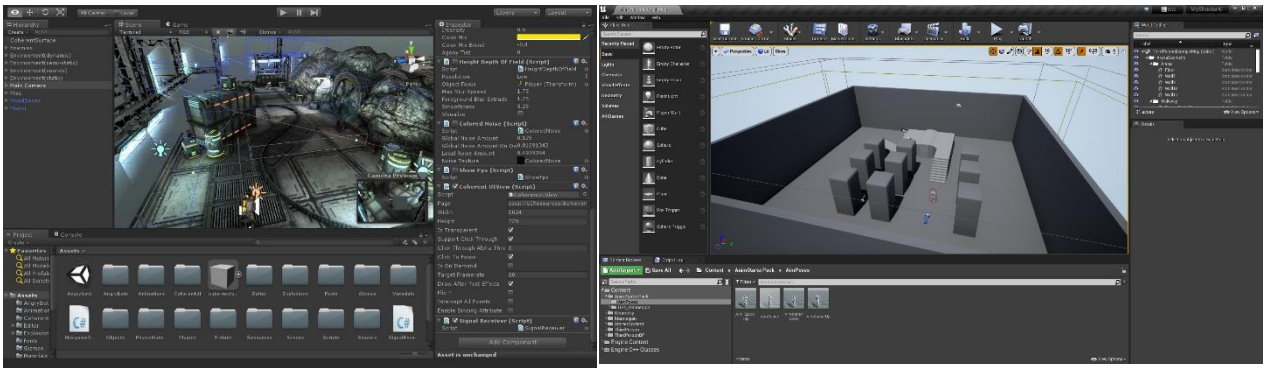


Рис. 2.6 Інтерфейси Unity3D та Unreal Engine 4

Наступне про що слід згадати – це мови програмування що використовуються у ігрових рушіях. В Unity3Dосновною мовою програмування є C#. Це доволі проста і продуктивна мова програмування, котра активно використовується і за межами ігрового рушія. У Unreal Engine 4 основною мовою програмування є C++. Це не дивно адже C++ вважається найпродуктивнішою мовою програмування. Також C++ використовується у багатьох інших ігрових рушіях. Крім C++ в Unreal Engine 4 присутній і Blueprints. В своїй основі це той самий C++, але трохи обмежений і зроблений у вигляді візуального скриптінгу (Рис. 2.7).

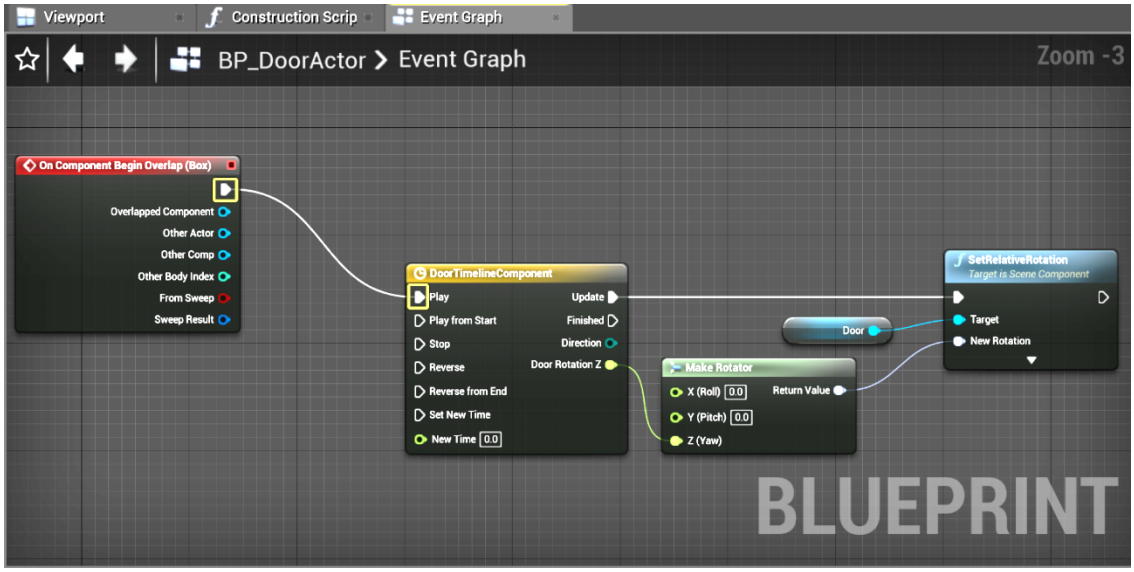
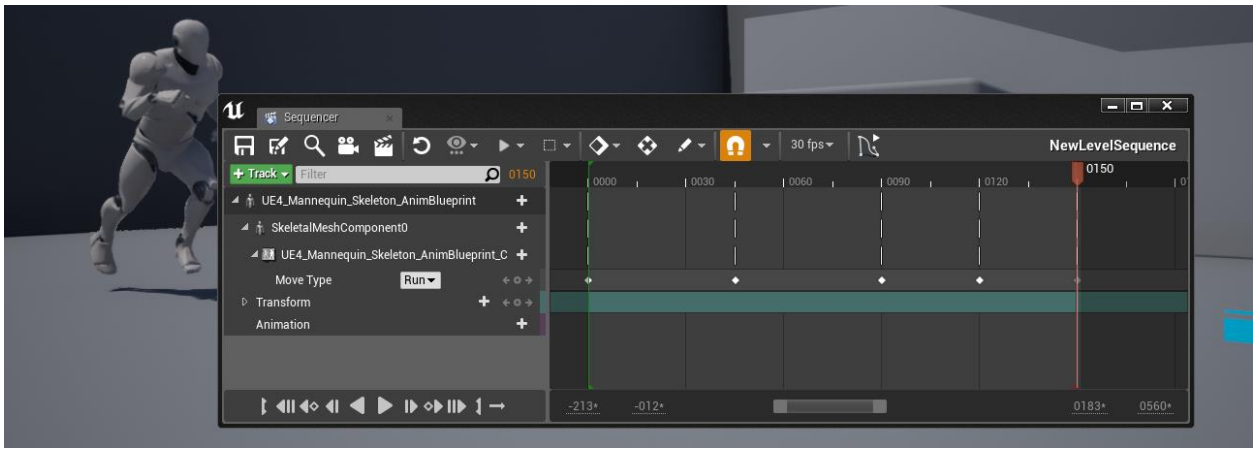
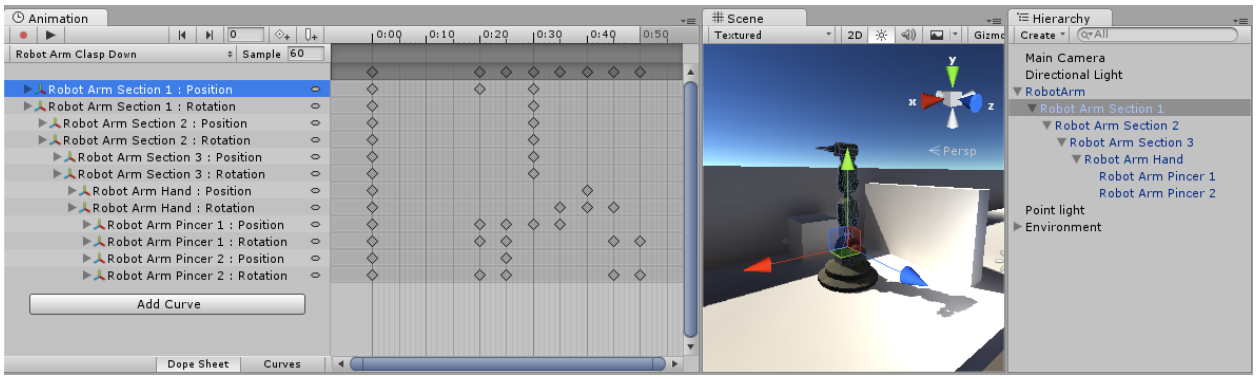


Рис. 2.7. Інтерфейс редактора Blueprint

Наступними порівняємо анімації. У Unity3D анімації прикріплюються до об’єктів за допомогою кліпів анімацій (animation clips) і керуються контролерами анімацій (animation controllers), в Unreal Engine 4 вони представлені послідовностями анімацій (animation sequences) і керуються за допомогою Blueprints. В обох рушія є стейт-машини, що визначають переходи з одного стану ассета до іншого. У Unreal Engine 4 система називається Persona, а в Unity – Mecanim. Також можливе застосування скелетних мешів одного скелета до інших, але в Unity це в основному використовується для анімації гуманоїдів.



а)



б)

Рис. 2.8. Редактори анімацій у а) Unity3D та б) Unreal Engine 4

В Unreal Engine 4 є вбудований постпроцесінг, завдяки чому можливо використовувати ефекти як глобально, так і до різних частин сцени. В Unity3Dє стек постпроцесінга, але його потрібно завантажувати з магазину

ассетів і він використовується лише у якості стеку або скриптів прикріплених до камери.

В обох рушіях є магазини ассетів котрі переповнені різними розробками користувачів. Оскільки Unreal Engine 4 став безкоштовним лише у 2015 році, перелік товарів у Unity asset store, набагато ширший. Обидва магазини постійно поповнюються новими розробками.

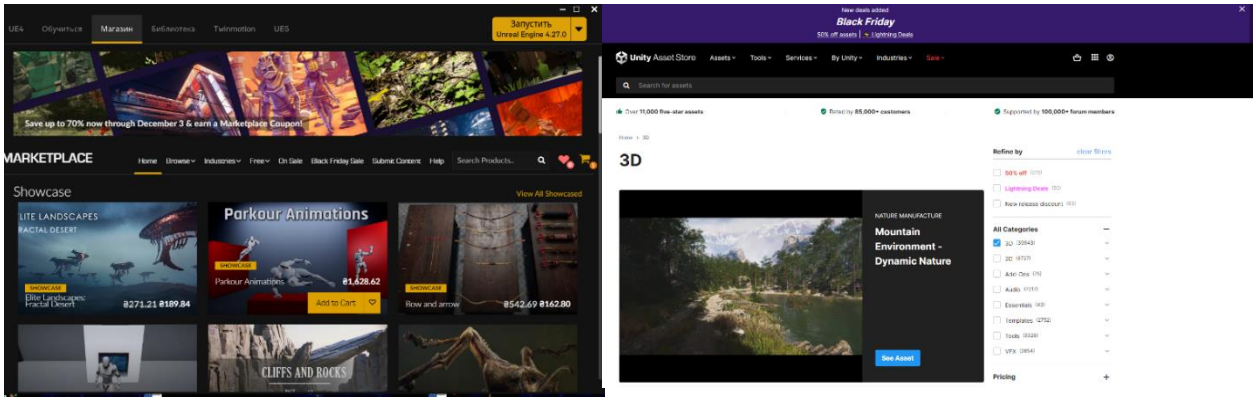


Рис 2.9. Магазини ассетів Unity3D та Unreal Engine 4

Останній пункт який варто розібрати – наявність навчального матеріалу для рушія. У Unity3D матеріалів набагато більше, з тієї ж самої причини що і в минулому пункті. Але Unreal Engine 4 зараз дуже швидко набирає популярність завдяки деяким причинам [25].

- Unreal Engine 4 підтримує велику кількість функцій, завдяки чому можна створити будь-яку гру.
- Unreal Engine 4 має вбудовану систему візуального скриптінгу, яка дозволяє без особливих перешкод вибудовувати ігрову логіку навіть новачкам.

Оцінивши переваги обох рушіїв було вирішено зупинитися на Unreal Engine 4.

2.2. Unreal Engine 4

Unreal Engine 4 – потужний кросплатформений ігровий рушій, який використовується при створенні ААА-ігор, при розробці комп'ютерних ефектів для фільмів, та комп'ютерних моделей.

При запуску ігрового рушія нас зустрічає вікно з вибором проекту, на якому ми можемо створити новий, чи продовжити роботу над старим проектом. На вибір нам даються 4 категорії проектів:

- Ігри;
- фільми, телебачення та живі заходи;
- архітектура, інжиніринг, будівництво;
- автотранспорт, дизайн продукту, виготовлення.

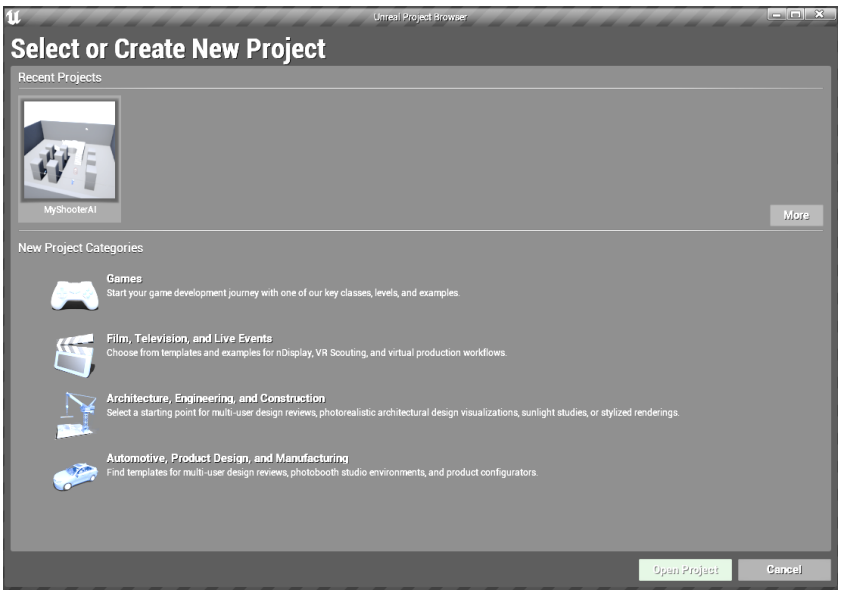


Рис 2.10. Вікно створення проекту в Unreal Engine 4

При виборі категорії «Ігри» ми можемо обрати одну із заготовок гри, після чого налаштувати майбутній проект, і завантажити обрану заготовку (Рис. 2.11).

Рис 2.11. Вікно вибору шаблону гри

Після завантаження проекту з’являється головне вікно Unreal Engine 4 (Рис 2.12.). Воно містить:

- Панель вкладок, рядок меню
- Панель інструментів
- Панель об’єктів/моделей
- Вікно сцени
- Контент браузер
- World Outliner
- Деталі

Рис 2.12. Інтерфейс Unreal Engine 4

Панель вкладок

У редакторі рівнів у верхній частині є вкладка з назвою поточного рівня. Вкладки з інших вікон редактора можуть бути прикріплені поруч із цією вкладкою для швидкої та легкої навігації, подібно до веб-браузера.

Кнопки меню

Рядок меню в редакторі повинен бути знайомий кожному, хто раніше використовував програми Windows. Він надає доступ до загальних інструментів і команд, які використовуються під час роботи з рівнями в редакторі.

Панель інструментів

Панель інструментів відображає групу команд, що забезпечує швидкий доступ до часто використовуваних інструментів та операцій.

Панель об'єктів/моделей

Панель на якій знаходяться різні об'єкти, які можна додати до сцени.

Вікно сцени

Вікно зі сценою яку ви створюєте.Ця панель містить набір вікон для перегляду, кожне з яких можна розширити, щоб заповнити всю панель, і надати можливість відображати світ з одного з трьох орфографічних видів (зверху, збоку, спереду) або в перспективі, що дає вам повний контроль над тим, що ви бачите і як ви це бачите.

Контент Браузер

Вікно з усіма файлами вашого проекту

World Outliner

На панелі World Outliner відображаються всі актори сцени в ієрархічному дереві. Ви можете вибирати та змінювати акторів безпосередньо з World Outliner. Використовуйте спадне меню «Інформація», щоб відобразити додатковий стовпець із зазначенням рівнів, шарів або назв ідентифікаторів.

Деталі

Панель «Деталі» містить інформацію, утиліти та функції для поточного вибору у вікні перегляду. Вона містить поля редагування для переміщення, обертання та масштабування акторів, відображає всі редаговані властивості для вибраних акторів і забезпечує швидкий доступ до додаткових функцій редагування залежно від типу акторів, вибраних у вікні перегляду. Наприклад, вибрані актори можна експортувати до FBX і конвертувати в інший сумісний тип. Деталі вибору дозволяють переглядати матеріали, використані вибраними акторами, якщо такі є, і швидко відкривати їх для редагування [22].

Як вище згадувалось в UnrealEngine 4 присутня система візуального скриптінгу Blueprints. Система Blueprint в UnrealEngine – це самостійна система сценаріїв ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів, для створення елементів ігрового процесу з UnrealEditor. Як і у багатьох поширених скриптових мовах, вона використовується для визначення об’єктно-орієнтованих (ОО) класів або об’єктів у рушії.

У Blueprintє декілька типів, кожен з яких має власне специфічне використання, від створення нових типів, до програмування заскриптованих подій на рівні, створення інтерфейсів, або макросів, для використання в інших Blueprint’ів.

BlueprintClass

Класи створені за допомогою Blueprint, часто називаються просто «Blueprint». Вони є ассетами завдяки яким розробники можуть легко додавати функціональні можливості поверх існуючих класів. Вони зберігаються як ассети в папках проекту. Вони можуть визначати новий клас, або тип об’єкту, який потім можна розміщувати на рівні.

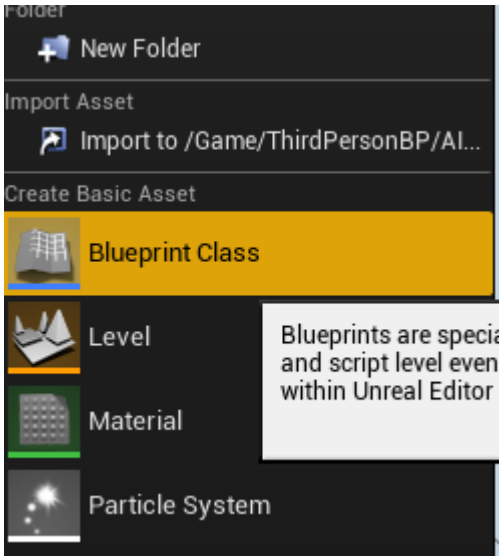


Рис. 2.13. Створення класу Blueprint

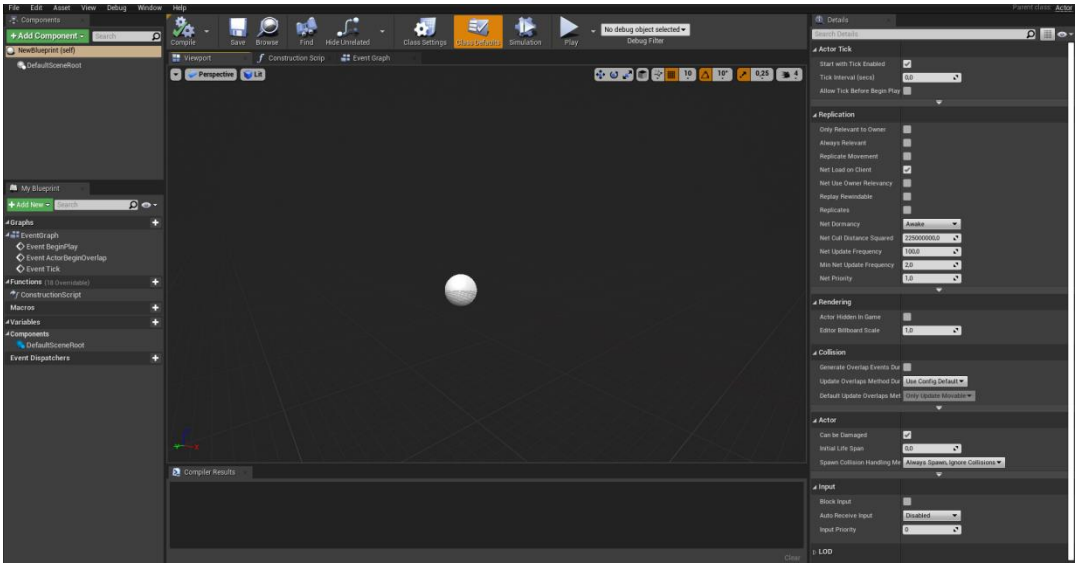


Рис. 2.14. Редагування класу Blueprint

Data-Only Blueprint

Це ті самі класи Blueprint, які мають лише код, змінні, і компоненти успадковані від батьківського класу. Вони дозволяють успадковувати налаштування які можна підігнати і налаштувати під себе, але можливість додавати нові елементи відсутня. Взагалі це зміна архетипів, яка може використовуватися дизайнерами для підгону налаштування у об’єктів зі

змінними. Ці Blueprint редагуються у невеликому вікні налаштувань, але можуть бути конвертовані і відредаговані як і звичайні.

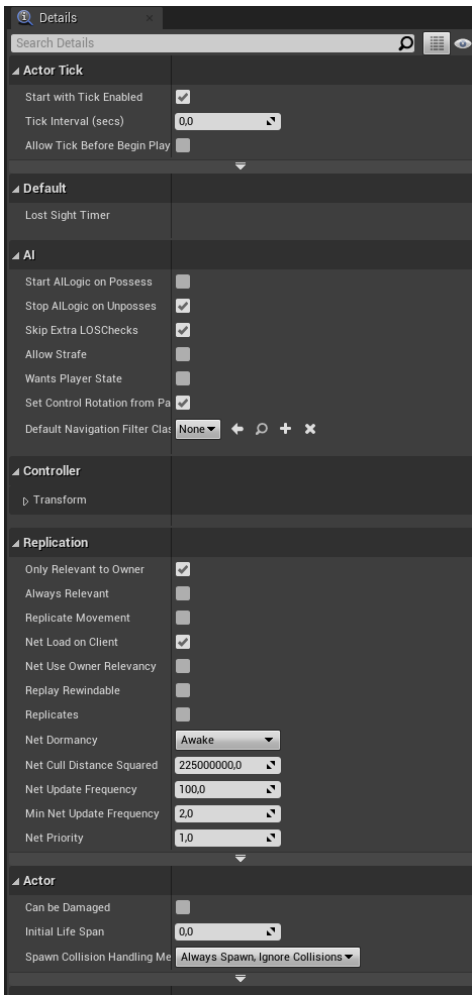


Рис. 2.15. Редактор Data-Only Blueprint

Level Blueprint

Це спеціальний тип Blueprint який діє як глобальний граф подій. Кожен рівень у проекті має власний Level Blueprint який стандартно створює Unreal Editor, но цей Blueprint неможливо створити за допомогою інтерфейсу редактора. Події, що стосуються рівня в цілому, або окремих екземплярів класів у межах рівня, використовуються для запуску послідовностей дій у формі викликів функцій або операцій керування потоком.

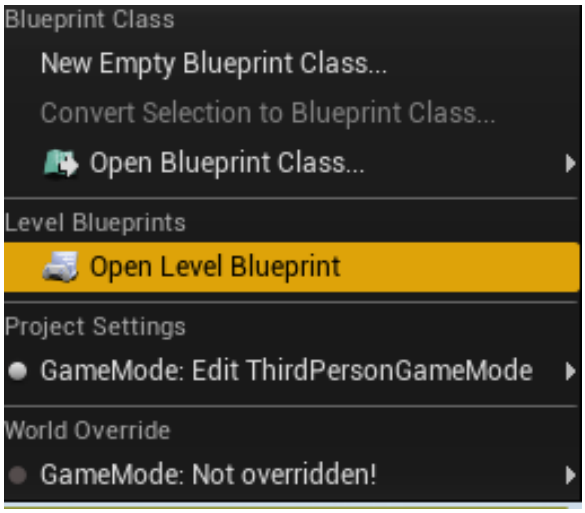


Рис. 2.16. Редагування Level Blueprint

Blueprint Interface

Це набір з однієї або більшої кількості функцій у вигляді імен, без реалізації, які можна додати до інших схем. Кожен Blueprint до якого доданий інтерфейс, отримує ці функції. Роботу цих функцій можна описати у Blueprint, до яких доданий інтерфейс. Таку концепцію мають інтерфейси у загальному програмуванні. Вони дозволяють отримувати доступ до багатьох різних типів об’єктів за допомогою використання спільного інтерфейсу.

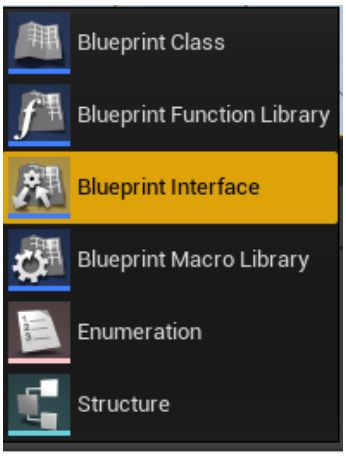


Рис 2.17. Створення Blueprint Interface

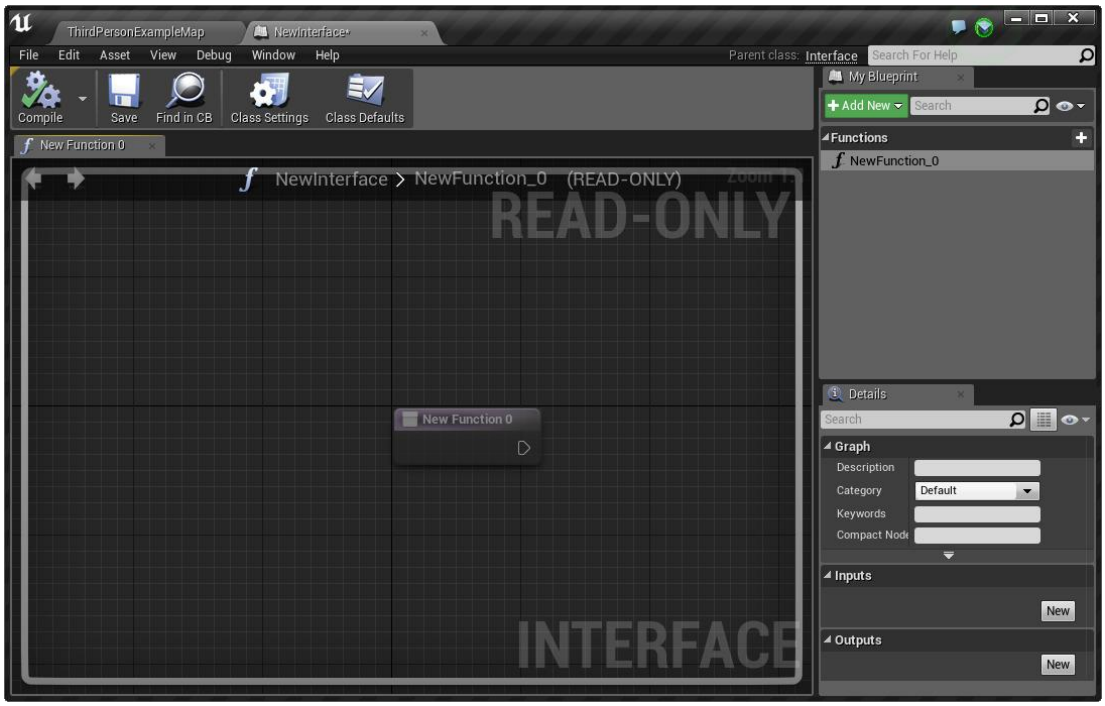


Рис. 2.18. Редагування Blueprint Interface

Blueprint Macro Library

Це вмістилище що зберігає в собі набір макросів, або автономних графів, які можна додати до інших Blueprint. Вони дозволяють зменшити час розробки, оскільки вони можуть зберігати часто використовувані послідовності вузлів разом з входами і виходами [19].

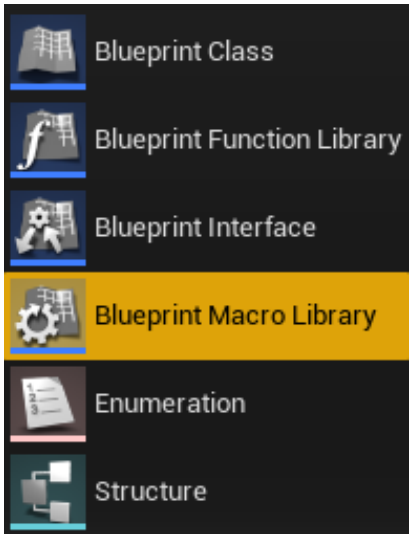


Рис. 2.19. Створення Blueprint Macro Library

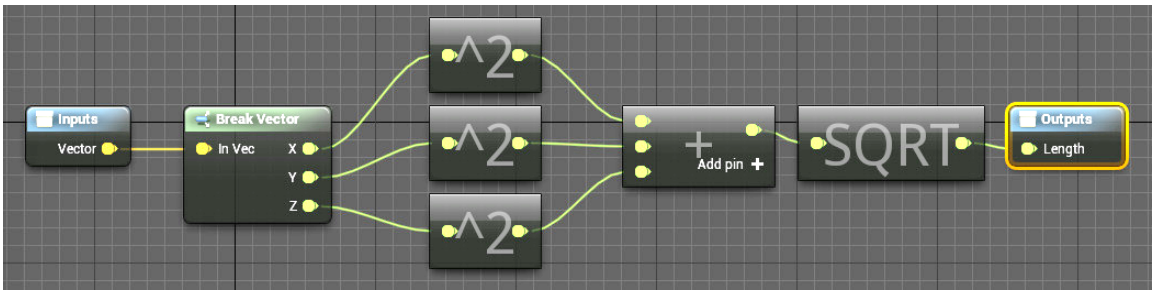


Рис. 2.20. Редагування Blueprint Macro Library

2.3. Модель поведінки некерованих гравцем персонажів

Зазвичай на старті гри некеровані гравцем персонажі (автономні агенти) знаходяться у стані спокою. У грі це виглядає як охорона однієї будівлі чи цілої локації, або патрулювання за певним маршрутом.

Рис. 2.21. Приклад патрулювання вулиці у грі Thief (2014)

Маршрут патрулювання може бути заданий заздалегідь, на етапі розробки, але й може генеруватися протягом гри. Генерація гри проходить в декілька етапів (Рис. 2.22.).

Рис. 2.22. Алгоритм патрулювання

Алгоритм патрулювання переривається якщо автономний агент помічає гравця і переходить до іншого стану – стану бою. У стані бою автономний агент починає переслідувати гравця та намагається влучити у нього зі своєї зброї (Рис. 2.23.). Це продовжується доки агент не загубить гравця на тривалий час.

Рис .2.23. Алгоритм бою

При виході з цього алгоритму автономний агент знову переходить до патрулювання. Саме так працює штучний інтелект у більшості 3D-шутерів. Все це повторюється доки автономний агент не загине. В деяких іграх автономні агенти зовсім не переходять до патрулювання і весь час знаходяться у стані бою. Це не завжди погано, яскравим прикладом слугує оновлена серія «Doom». Її штучний інтелект завжди знаходиться у стані бою що дозволяю постійно підтримувати так званий «спинномозковий» геймплей.

У обох алгоритмах є перевірка: «чи бачить автономний агент гравця?». В першу чергу необхідно побудувати сектор котрий буде відповідати полю зору автономного агента. Після побудови зору ми визначаємо, який з об’єктів

на рівні є гравцем. Визначивши обидва ці значення, ми можемо працювати за алгоритмом, схема якого показана на рис. 2.24.

Рис .2.24. Алгоритм зору автономного агента

Висновки до розділу 2

Шляхом порівняльного аналізу декількох популярних ігрових рушіїв, у якості інструментарію для реалізації моделі поведінки некерованого гравцем персонажа було обрано Unreal Engine 4. Зроблений огляд його основних функцій, а також візуальної мови програмування Blueprint. Unreal Engine 4 вважається найпотужнішим ігровим рушієм, який використовується у створенні AAA-ігор. Виходячи з цього його потужності цілком задовольняють нашим потреба та цей рушій може бути обраний для програмної реалізації моделі поведінки некерованого гравцем персонажа в 3D-шутері.

Проаналізовані основні сценарії поведінки автономних агентів, а саме патрулювання і бій. Створена модель поведінки некерованого гравцем персонажа, розроблені алгоритми.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПОВЕДІНКИ НЕКЕРОВАНИХ ГРАВЦЕМ ПЕРСОНАЖІВ У 3D-ШУТЕРІ

3.1. Програмна реалізація поведінки некерованих гравцем персонажів у 3D-шутері

Програмно реалізовувати некерованих агентів ми будемо засобами Unreal Engine 4. В першу чергу нам необхідно створити проект. Для цього ми обираємо його тип, а саме «Game». Після обрання типу проекту ми можемо обрати один з шаблонів, а також зробити первинні налаштування (Рис. 3.1).

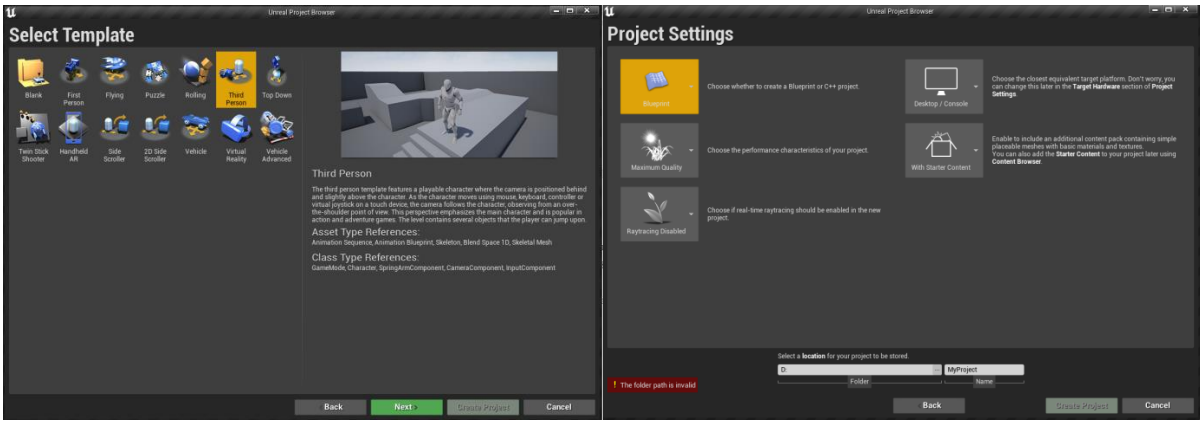


Рис 3.1. Створення нового проекту

У нашому проекті створюємо нову папку в яку ми помістимо файли для роботи з автономним агентом. Назвемо її AI. У першу чергу слід помістити у цю папку клас ThirdPersonCharacter котрий буде виступати нашим майбутнім ворогом. Наступним кроком ми створюємо дерево поведінки (EnemyAIBT) нашого автономного агента, та дошку (EnemyAIBB). Дошка зберігає у собі ключі – змінні, які можна передавати у вузли нашого дерева поведінки (Рис. 3.2).

Тепер нам потрібно створити новий клас котрий буде успадковуватись від класу AIController, назвемо його EnemyAIController. Це буде наш основний клас у якому ми будемо підключати наше дерево поведінки, а також

налаштовувати AIPerception. Повертаємось до дошки. У ній створюємо 4 ключі:

- SelfActor (Об'єкт)
- TargetActor (Об’єкт)
- MoveToLocation (Вектор)
- HasLineOfSight(Логічна величина)

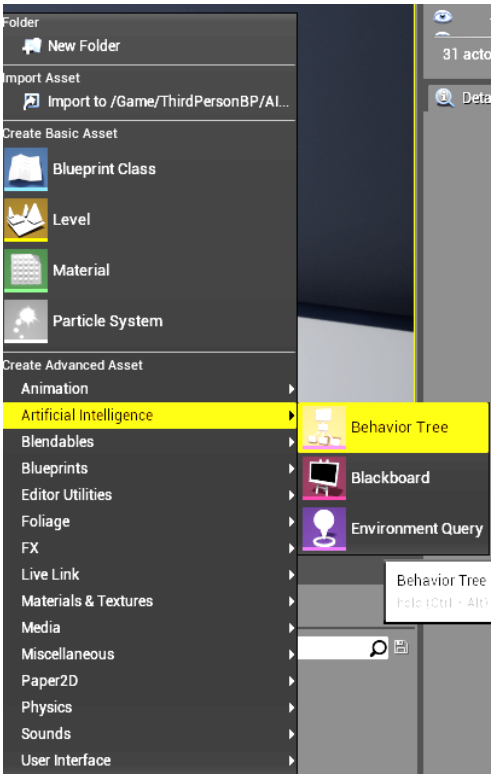


Рис. 3.2. Створення дерева поведінки

У Дереві поведінки виставляємо вузол Selector, який буде обирати гілку з певною моделлю поведінки. Створюємо ще Selector та Sequence які будуть відповідати за стан бою та стан патрулювання,. Так їх і назвемо InCombat та Patrolling.

Тепер реалізуємо алгоритм патрулювання (Рис. 3.4). Для цього ми створюємо 3 задачі. Перша RandomLocationBTT має бути описана власноруч. У ній ми будемо знаходити випадкову точку. Друга Move To вже описана, тому нам залишається передати у неї обрану точку за допомогою ключа

MoveToLocation. Третій вузол Wait буде відповідати за очікування після переміщення у обрану точку. Він також описаний у Unreal Engine 4.

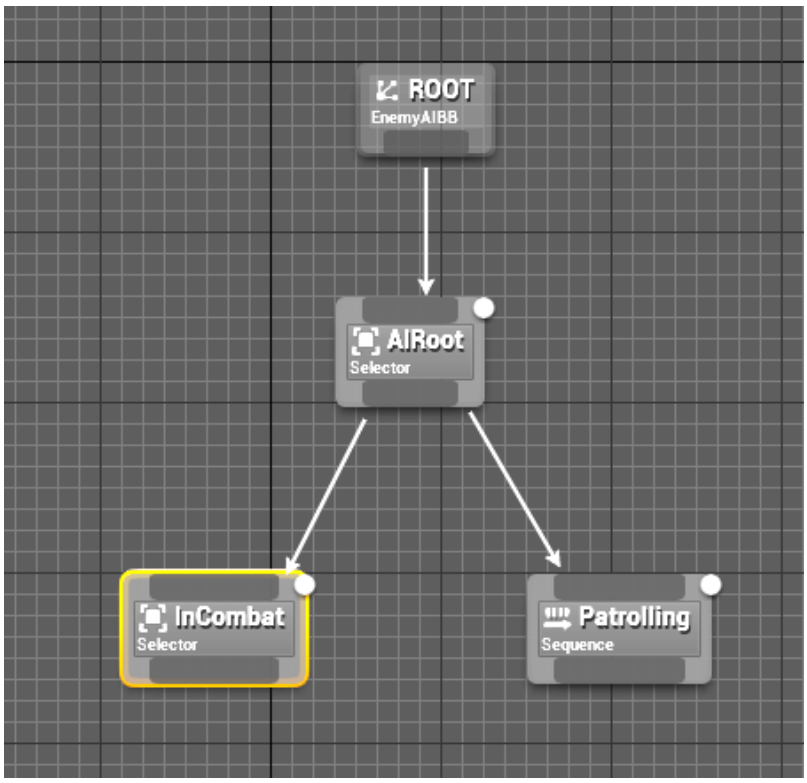


Рис. 3.3. Створення вузлів InCombat та Patrolling

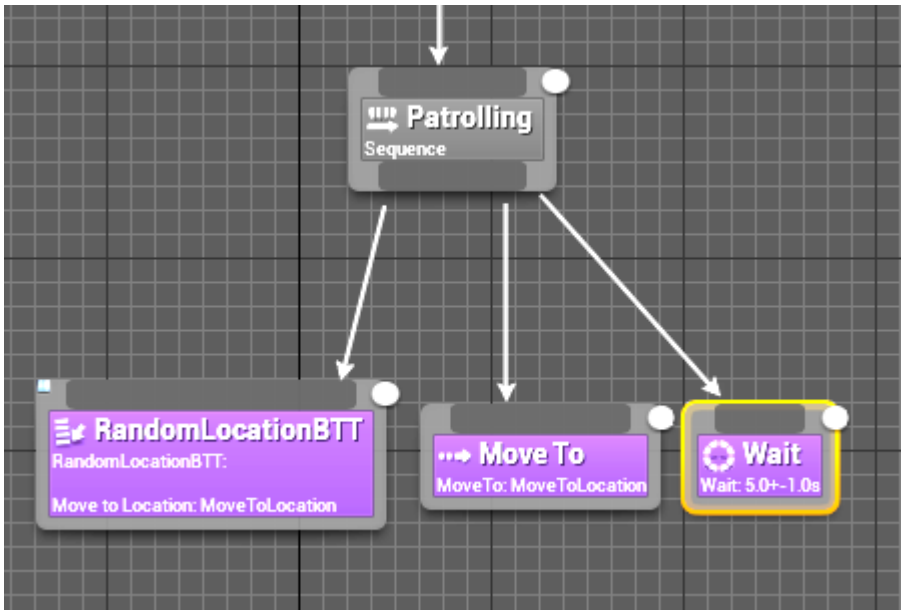


Рис. 3.4. Підключення вузлів для реалізації алгоритму патрулювання

Перейдемо до реалізації RandomLocationBTT (Рис. 3.5). В першу чергу створюємо вузол Event Receive Execute AI. Ця подія буде викликатися якщо дерево поведінки обере цей вузол. З цього вузла ми передаємо данні про нашого агента у вузол GetActorLocation, для того щоб отримати його координати. З цього ми переходимо у вузол Get Random Reachable Point In Radius. Цей генерує точку до якої ми можемо дістатися в межах заданого радіуса. Це значення ми передаємо у ключ MoveToLocation на нашій дошці, за допомогою вузлів Set Blackboard Value as Vector та завершуємо виконання команд цього вузла.

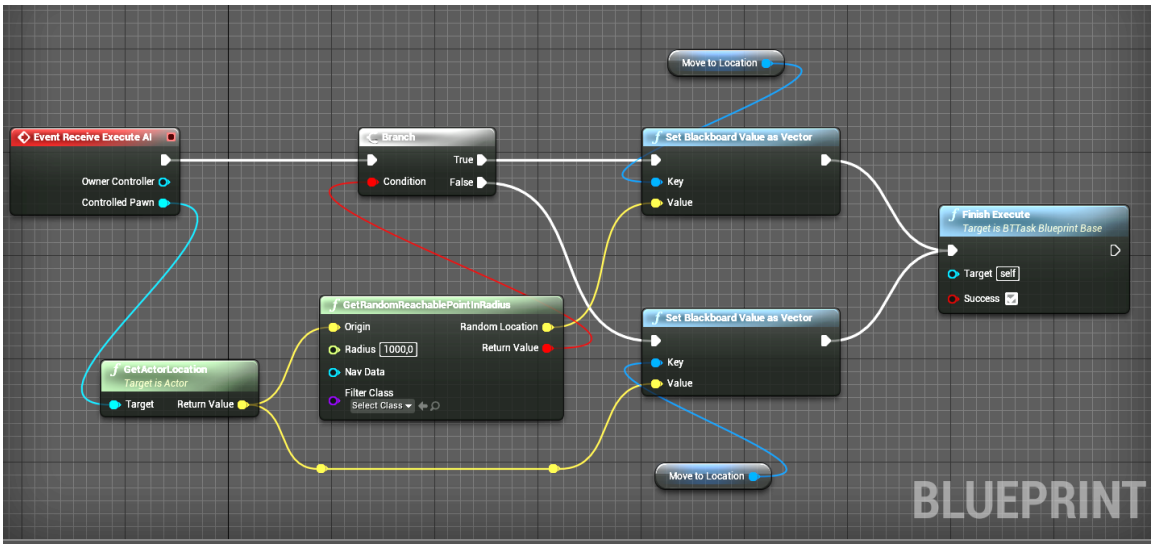


Рис. 3.5. Реалізація RandomLocationBTT

Для правильного налаштування пошуку шляху та виходу на позицію нам необхідно додати на сцену навігаційну сітку. Для цього на сцену ми додаємо об’єкт Nav Mesh Bounds Volume, та налаштовуємо його розміри.

Отже реалізувавши патрулювання перейдемо до реалізації бою (Рис. 3.6). Переходимо до вузла InCombat. До цього вузла ми додаємо Decorator у якому обираємо параметр Blackboard. У ньому ми вказуємо умови переходу до цього вузла. У полях Flow Control ми обираємо параметри On Result Change та Lower Priority. У полях Blackboardми обираємо ключ який будемо перевіряти, а саме TargetActor. З вузла InCombat ми виводимо дав вузли. Перший Attack буде відповідати за атаку гравця, а другий

MoveIntoPosition за вихід на позицію. До вузла Attackми знову додаємо Decorator. У полях Flow Control ми обираємо параметри On Value Change та Lower Priority. У полях Blackboardми обираємо Is Set та ключ HasLineOfSight.3 цього вузла йдуть 2 завдання – Rotate to face BB entry та EnemyShootingBTT, які відповідають за поворот у сторону гравця та стрільбу.

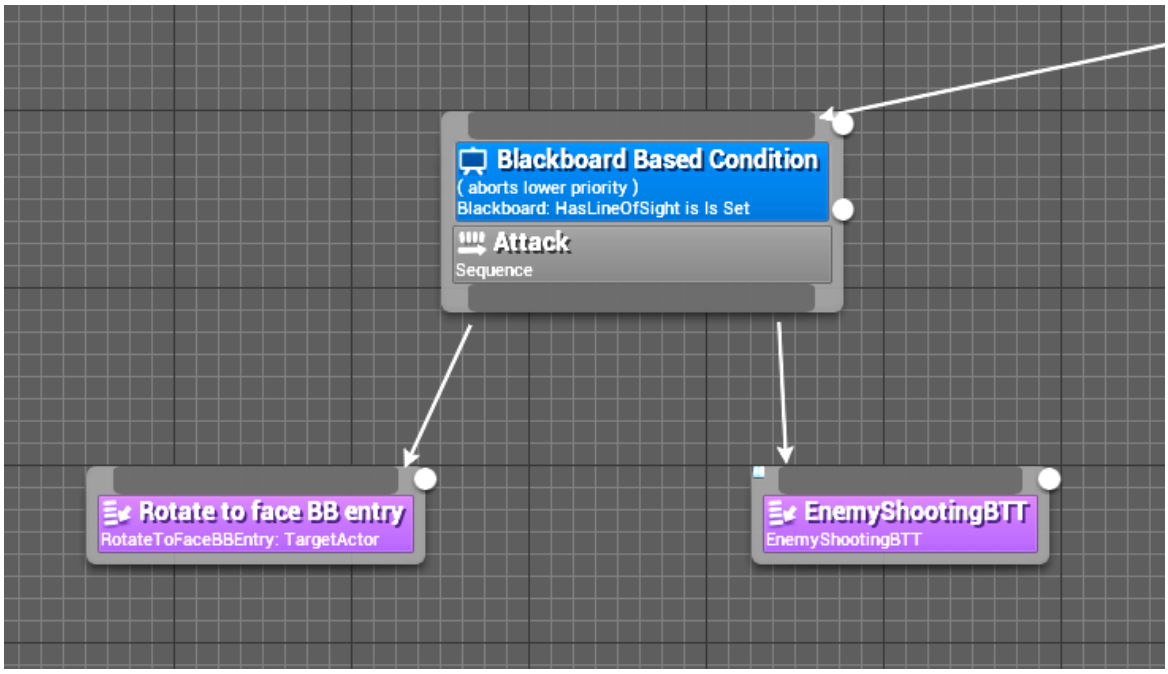


Рис. 3.6. Підключення вузлів для реалізації алгоритму атаки

Для реалізації алгоритму виходу на позицію нам необхідно підключити Environment Query System. Це один з інструментів для роботи з штучним інтелектом в Unreal Engine 4, який дозволяє збирати інформацію про оточення (Рис. 3.7). Воно використовує визначені користувачем тести, повертаючи найкращий елемент що відповідає запиту. При налаштуванні EQS ми знову бачимо граф. Нам необхідно згенерувати спеціальну сітку навколо нашого автономного агента завдяки якій він буде перевіряти перешкоди між собою та гравцем. В налаштуваннях сітки ми трохи збільшуємо її загальний розмір, та розмір окремих елементів. До цієї сітки ми підключаємо два тести:

- Trace, який перевіряє чи бачимо ми гравця з певної точки на сітці;
- Distance, використовується для оцінки найкращих точок з яких агент може побачити гравця.

У Trace нам необхідно поставити параметри TestPurpose – FilterOnly, Context – PlayerContextEQS та BoolMatch – Вимкнено. У Distance ми обираємо параметри Test Purpose – Score Only та Scoring Factor – -1.0. Значення Scoring Factor -1.0 повертає нам точки які знаходяться ближче до гравця.

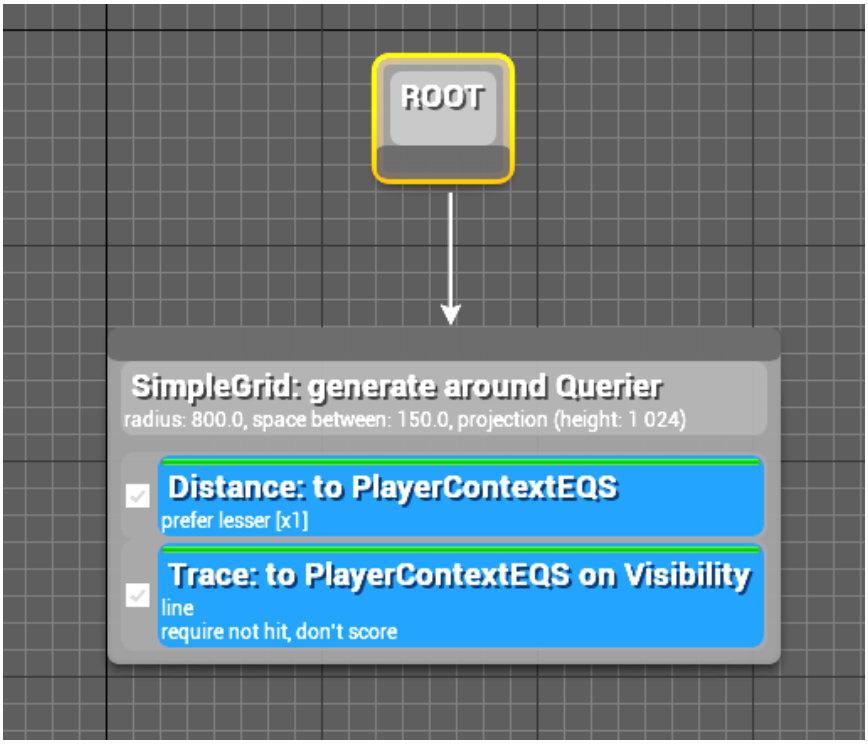


Рис. 3.7. Налаштування тестів у EQS

Повернувшись до дерева поведінки нам необхідно від вузлу MoveIntoPosition перейти у 3 ймовірні завдання. Після підключення EQS ми можемо запустити обрані нами тести. Для цього ми створюємо вузол Run EQS Query та обираємо необхідний нам ключ в який ми будемо повертати результат, а саме MoveToLocation. Після цього обираємо створені нами тести які ми назвали FindPlayerEQS. Інші вузли які підуть з MoveIntoPosition це Move to, який буде приймати ключ MoveToLocation і переміщувати нашого агента, та Rotate to face BB entry, котрий буде отримувати ключ TargetActor і повертати ворога у сторону гравця (Рис. 3.8).

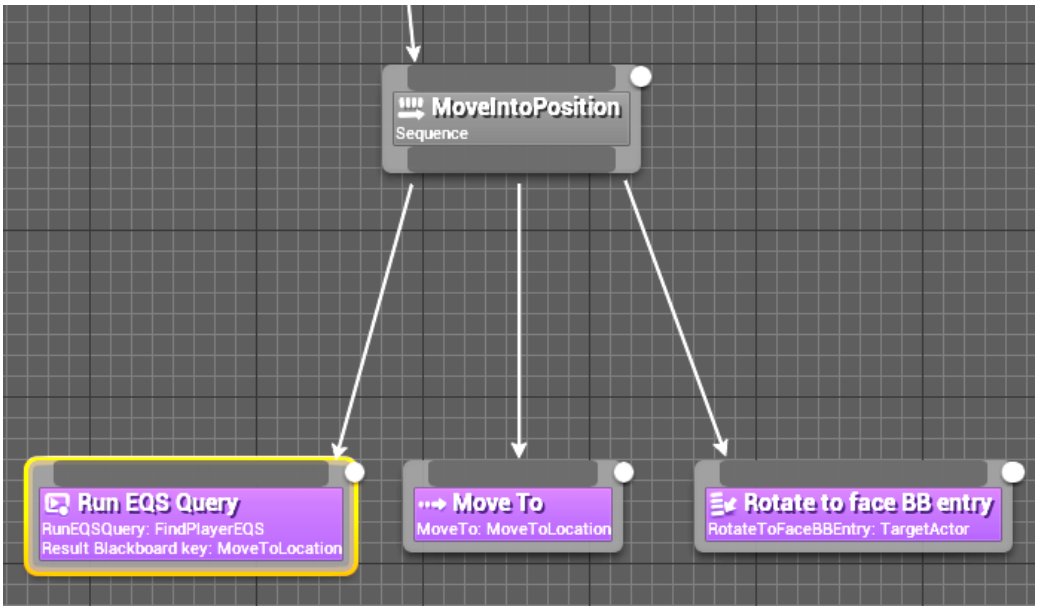


Рис. 3.8. Підключення вузлів для реалізації пошуку позиції

Отже ми налаштували наше дерево поведінки, і тепер нам слід перейти до налаштування EnemyAIController. У першу чергу він повинен підключати дерево поведінки. Для цього ми створюємо подію Event On Possess до якої підключаємо вузол Run Behavior Tree, в якому обираємо наше дерево (Рис. 3.9).

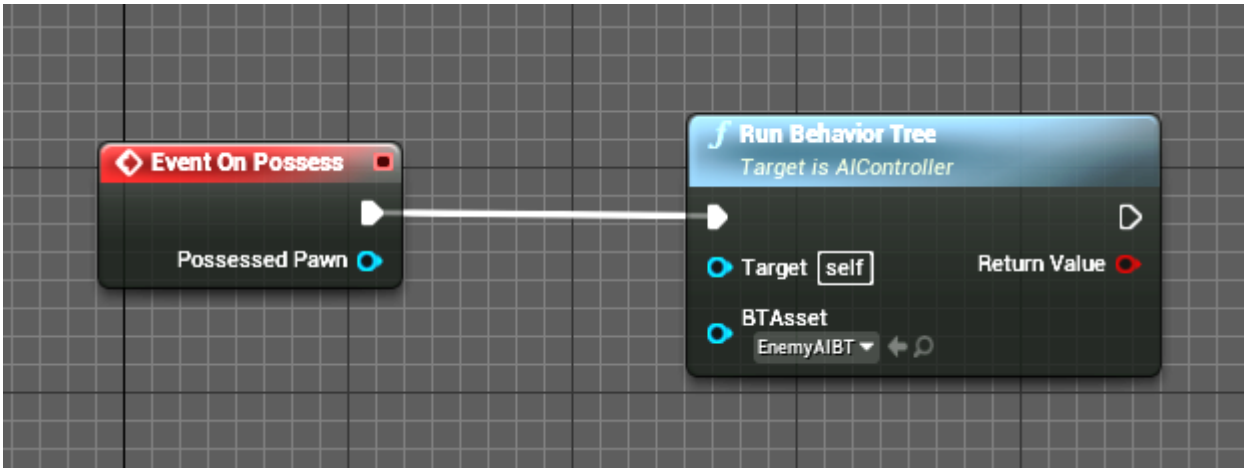


Рис. 3.9. Підключення дерева поведінки

Далі, до нашого контролера ми підключаємо компонент AIPerception. Цей компонент відповідає за систему сприйняття. Ми будемо його використовувати для створення зору нашого автономного агента. Ми можемо

налаштувати радіус зору, кут. Після налаштувань ми створюємо подію On Target Perception Updated (AIPerception). До цієї події ми підключаємо перевірку тегу Player (Actor Has Tag), а також перевіряємо чи не покинув гравець наше поле зору (Рис. 3.10).

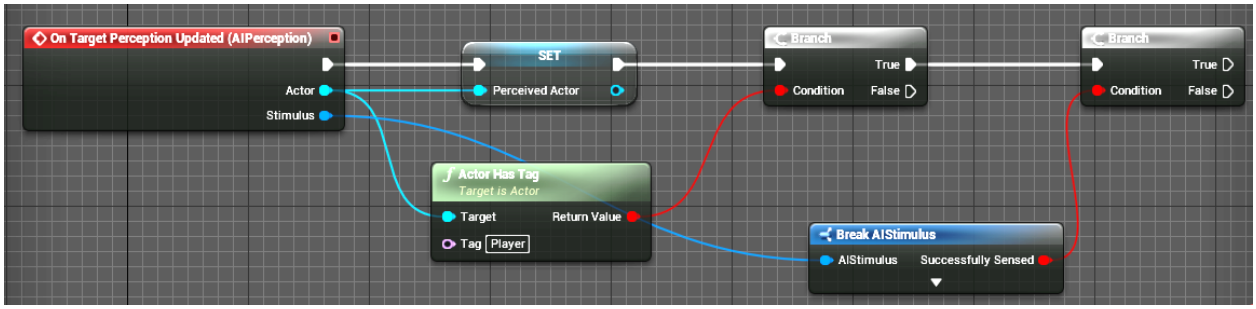


Рис. 3.10. Програмна реалізація зору автономного агента

Далі необхідно створити дві власні функції, для оновлення ключів (Рис. 3.11). Назвемо їх Update Sight Key та Update Target Key. У функції Update Sight Key ми будемо оновлювати ключ HasLineOfSight, а в Update Target Key – TargetActor.

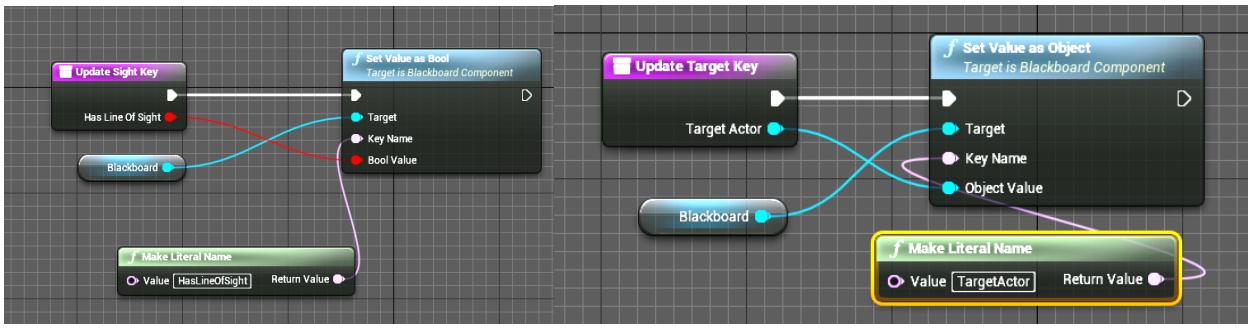


Рис. 3.11. Програмна реалізація функцій оновлення ключів

Необхідно створити 2 додаткові змінні, назвемо їх LostSightTimer та PerceivedActor. Перша буде зберігати в собі таймер загублення гравця, а друга данні про нашого гравця. Після перевірки наявності гравця в полі зору ми повинні розписати дії агента на обидва результати перевірки. Якщо результат перевірки True, то ми повинні скинути таймер у змінній LostSightTimer. Після цього ми викликаємо створені нами функції і передаємо данні у ключі (Рис. 3.12).

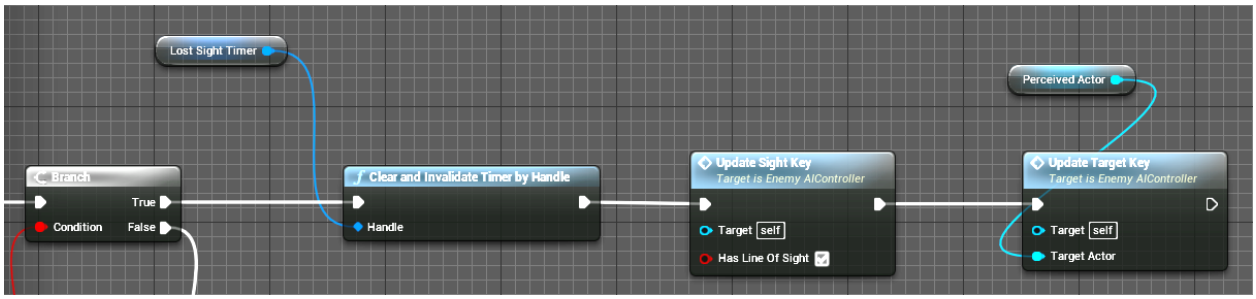


Рис. 3.12. Підключення функцій оновлення ключів у випадку коли гравець знаходиться у полі зору

У випадку коли результатом перевірки стає False нам необхідно запускати таймер в LostSightTimer. Для цього нам необхідно створити власну подію, яку ми називаємо LostSight. Після неї ми оновлюємо ключ TargetActor видаляючи звідти данні про гравця, створюємо вузол Set Timer by Event, передаємо з нього значення у змінну LostSightTimer, а також оновлюємо значення ключа HasLineOfSight (Рис. 3.13).

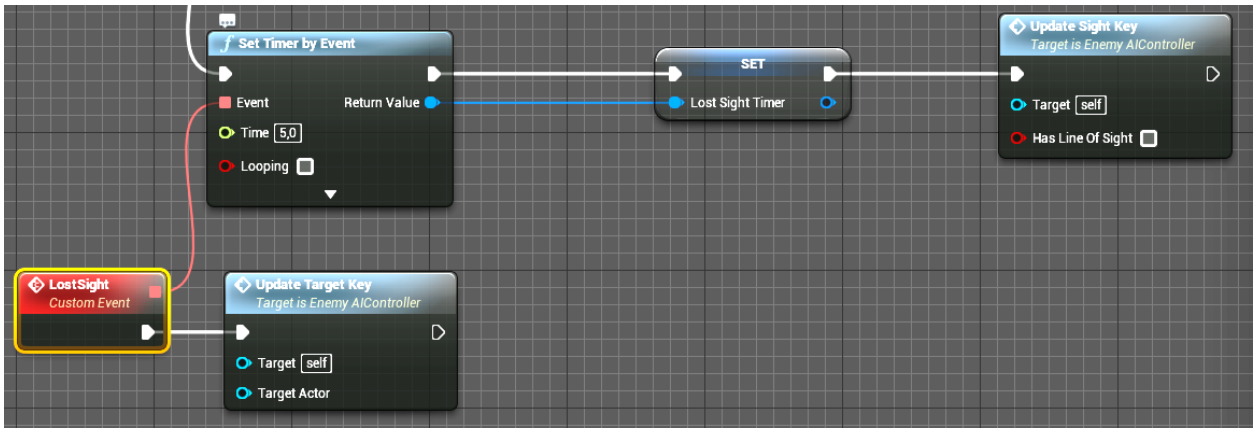


Рис. 3.13. Підключення функцій оновлення ключів коли гравець покинув поле зору

Тепер, коли ми завершили створення нашого дерева поведінки, та налаштування контролера ми можемо перейти саме до нашого автономного агента. Ми перейменовуємо наш клас ThirdPersonCharacter у EnemyBP. Відкривши цей клас у панелі Detail знаходимо параметр Use Controller Rotation Yaw та вмикаємо його. Також у параметрі AI Controller Class ми обираємо наш

клас EnemyAIController. Після цього ми переходимо до підконтрольного гравцю манекена та додаємо до нього тег «Player». Тепер розміщуємо об’єкти EnemyBP на ігровому рівні і перевіряємо його працездатність (Рис. 3.14.).

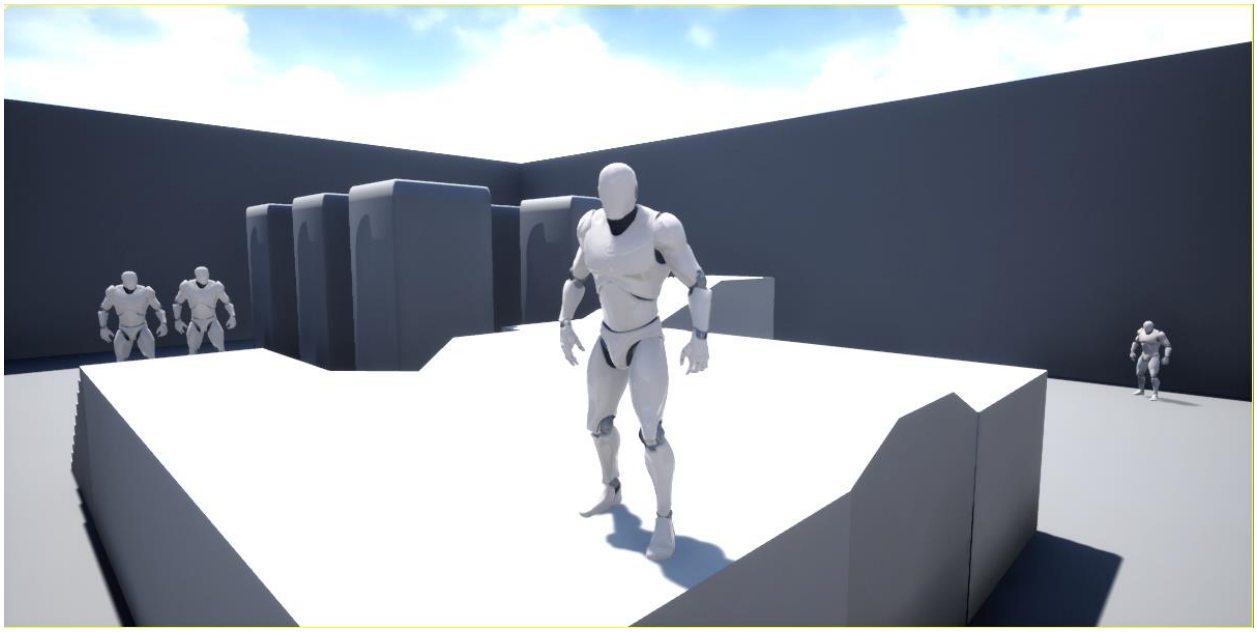


Рис. 3.14. Розміщені автономні агенти на сцені

3.2. Опис ігрового рівня для тестування поведінки некерованих гравцем персонажів

Для організації тестування був підготовлений спеціальний рівень з різними особливостями ландшафту, які дають можливість перевірки різноманітних ситуацій, з якими може зіштовхнутися некерований гравцем персонаж (Рис. 3.15).

Перше випробування призначене для перевірки дій агента на нерівностях та елементах ландшафта. Для цього був створений об’єкт Landscape та внесені зміни за допомогою інструменту Sculpt (Рис. 3.16).

Друге випробування перевіряє дії автономного агента всередині приміщень. Для цього створена будівля в якій знаходиться багато невеликих об’єктів, які можуть заважати автономному агенту під час пошуку позиції (Рис. 3.17).

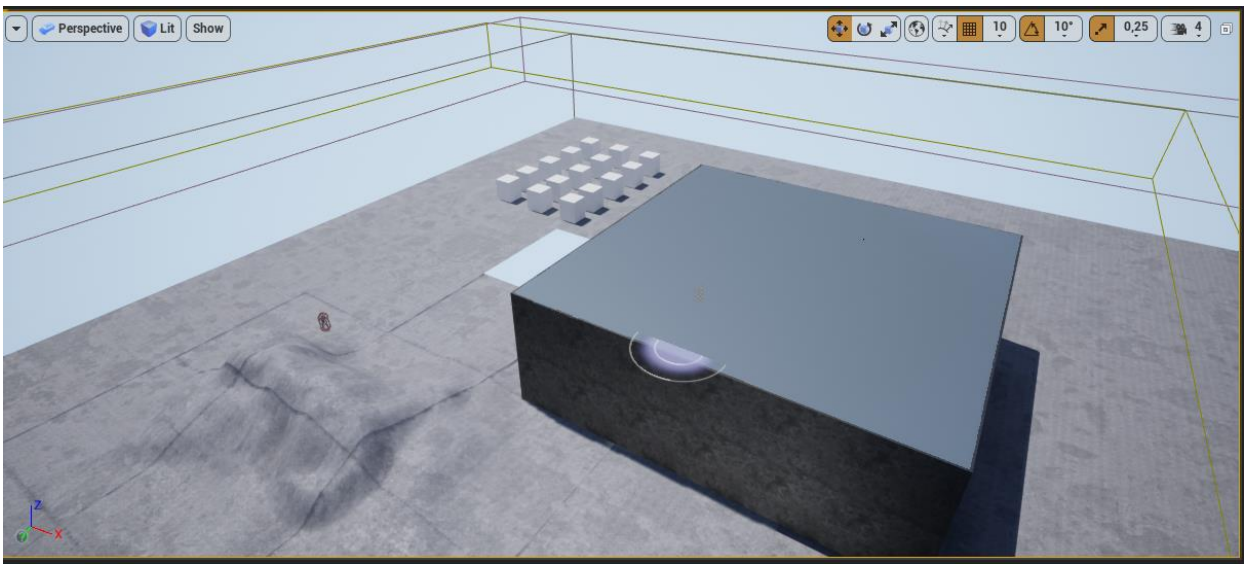


Рис. 3.15. Рівень для тестування некерованих гравцем персонажів

Рис. 3.16. Елемент ландшафту для першого випробування

Третє випробування призначене для перевірки реакції автономних агентів на провалля у ландшафті. Для цього були видалені частини ландшафту (Рис. 3.18).

Рис. 3.17. Закрите приміщення з перешкодами для другого випробування

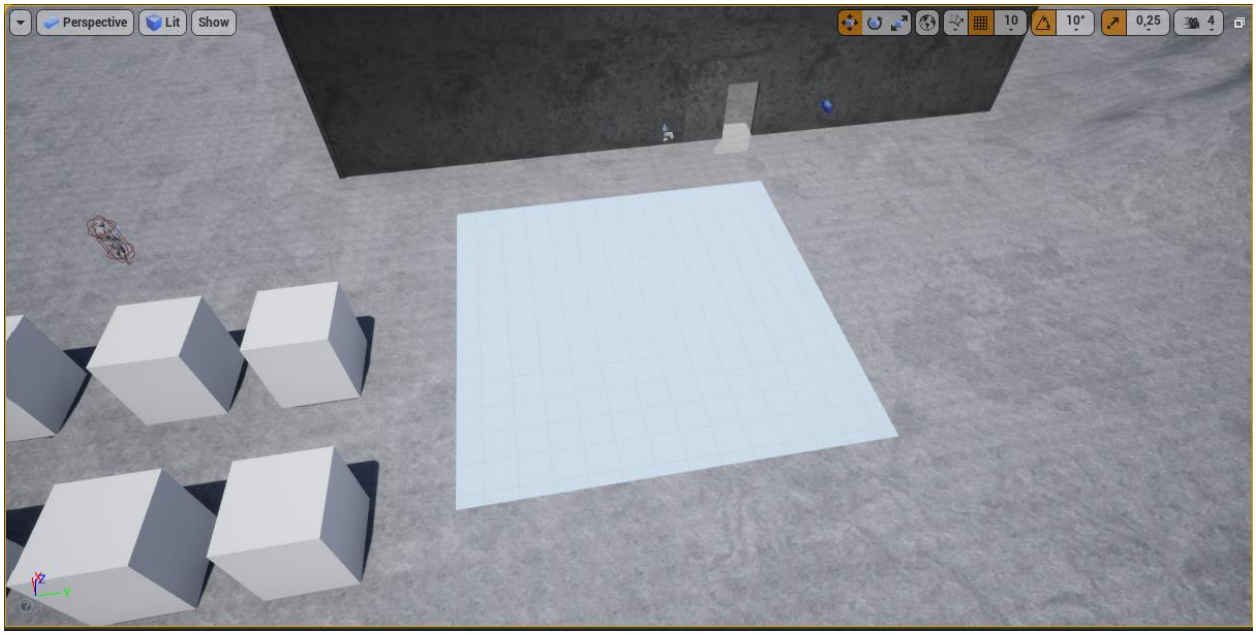


Рис. 3.18. Елемент ландшафту «Провалля»

Останнє випробування перевіряє дії автономного агента при наявності великої кількості великих об’єктів-перешкод. Для цього був створений масив об’єктів у вигляді прямокутного паралелепіпеда .

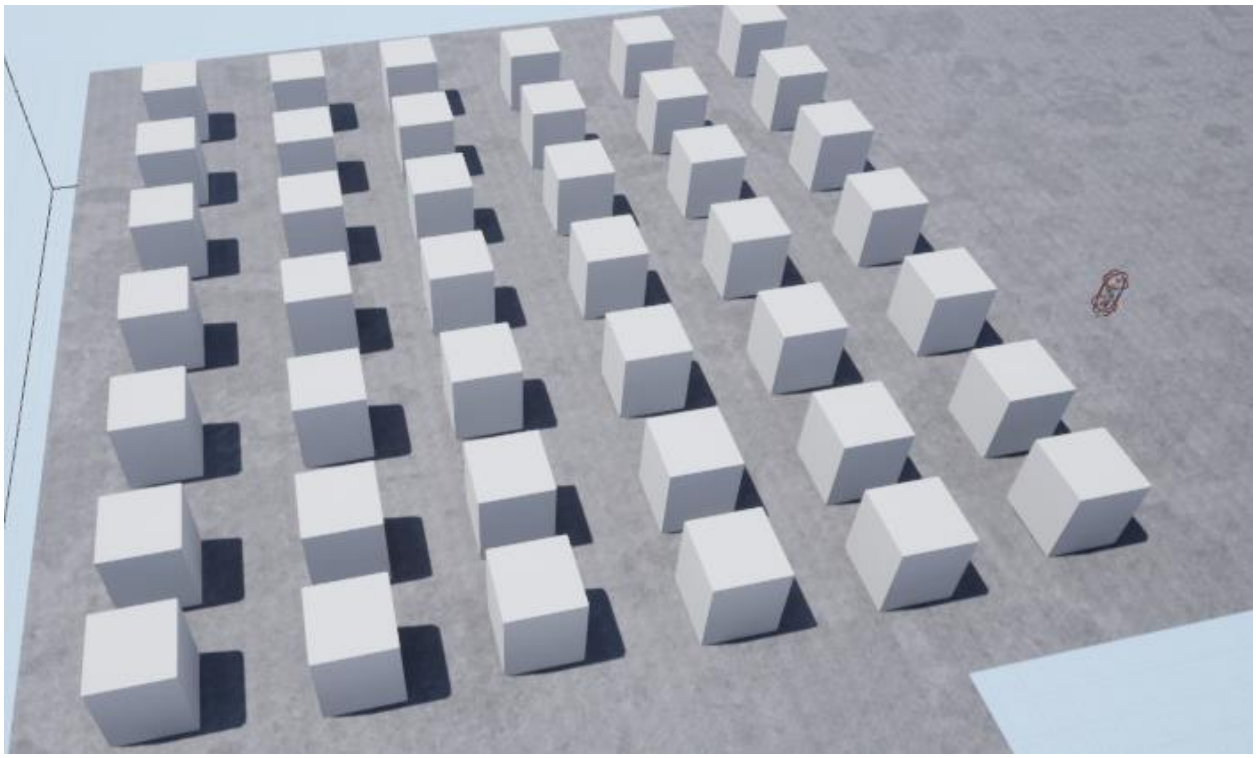


Рис. 3.19. Масив великих перешкод

3.3. Тестування програмної реалізації поведінки некерованих гравцем персонажів у 3D-шутері

З першим тестом автономний агент вдало впорався (Рис. 3.20). Агент постійно використовував нерівності ландшафту для того щоб вийти на позицію з якої можна вести вогонь. Отже нерівності ландшафту мало впливають на роботу автономних агентів.

З другим випробуванням автономні агенти також впорались без жодних проблем (Рис. 3.21). Невеликі об’єкти лише трохи заважали під час переходу в іншу позицію, але часу заданого для загублення гравця достатньо для знаходження необхідної позиції та виходу на неї.

Рис. 3.20. Проходження першого випробування

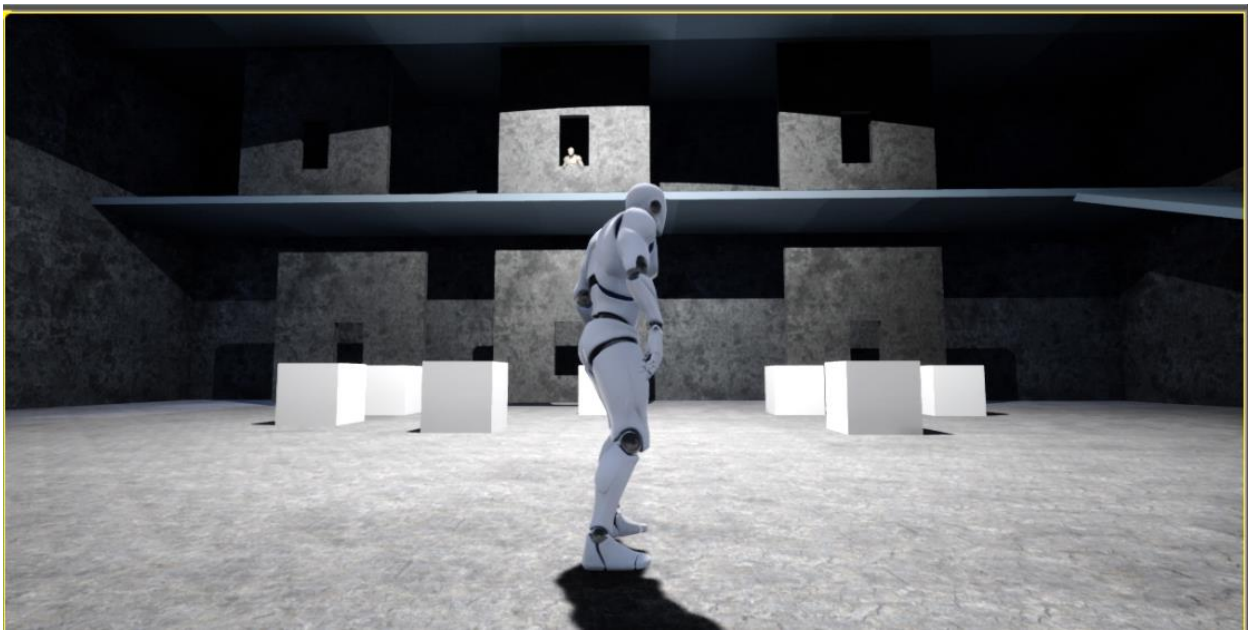


Рис. 3.21. Дії автономного агента під час проходження другого випробування

З третім тестом автономні агенти впорались бездоганно (Рис. 3.22) . Разом з видаленням частини ландшафту, видалилась частина навігаційної сітки що їй відповідала. Після видалення частини навігаційної сітки автономні агенти навіть не намагалися строїти маршрут через прірву і тому успішно впорались з цим випробуванням.

Рис. 3.22. Дії автономного агента під час проходження третього випробування

Фінальне випробування виявилось найскладнішим для автономних агентів (Рис 3.23.)

Рис. 3.23. Дії автономного агента під час проходження останнього випробування

При постійному переміщенні гравця між перешкодами через деякий час автономний агент його втрачав. Великі об’єкти у такій кількості постійно заважали автономному агенту тримати гравця у полі зору.

Висновки до розділу 3

Програмну реалізацію моделі поведінки автономних агентів було розроблено шляхом створення ігрових об’єктів, та програмного коду для них засобами Unreal Engine 4 та вбудованої у нього мови програмування Blueprint. Також був розроблений рівень з переліком тестів для перевірки дій автономних агентів у різних ситуаціях. В ході цих тестів були виявлені сильні та слабкі сторони автономних агентів. Усі несправності виявлені в ході тестування були виправлені.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено та програмно реалізовано модель поведінки некерованих гравцем персонажів за допомогою ігрового рушія Unreal Engine 4 та отримані наступні результати та висновки.

1. Проаналізовано сучасний стан використання штучного інтелекту в галузі розробки комп'ютерних ігор та розглянуто актуальні підходи до створення ігрового штучного інтелекту. Переглянуто приклади впровадження сучасних зразків штучного інтелекту у крупні ігрові проекти, проаналізовано прогрес їх розвитку.

2. Шляхом порівняльного аналізу наявного на ринку програмного забезпечення в секторі інструментарію для створення комп'ютерних ігор, для реалізації моделі було обрано ігровий рушій Unreal Engine 4.

3. Проаналізовані основні сценарії поведінки автономних агентів. Створена модель поведінки некерованого гравцем персонажа, розроблені алгоритми для її реалізації.

4. Програмно реалізовано та протестовано функції, що реалізують модель поведінки некерованих гравцем персонажів.

Практична цінність роботи полягає в розроблених функціях, що реалізують поведінку некерованих гравцем персонажів в комп'ютерній грі та можливості використовувати їх як за прямим призначенням, так і в якості навчального матеріалу в курсі «Розробка комп'ютерних ігор».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буковшин В. А., Воскобойников С. Г. Интеллектуальные системы в компьютерных играх. Перспективы развития искусственного интеллекта в игровой индустрии. *Современные материалы, техника и технологии*. 2017. № 3 (11). С. 21–36. URL : https://cyberleninka.ru/article/n/intellektualnye-sistemy-v-kompyuternyh-igrah-perspektivy-razvitiya-iskusstvennogo-intellekta-v-igrovoy-industrii8.

2. Бухараев Т. Искусственный интеллект в Heroes of Might and Magic V. URL : http://www.might-and-magic.ru/blog/1/entry-50-iskusstvennyj-intellekt-v-heroes-of-might-and-magic-v/

3. Ганасия Жан-Габриэль. Искусственный интеллект: между мифом и реальностью. *Курьер ЮНЕСКО*. 2018. № 3. URL : https://ru.unesco.org/courier/2018-3/iskusstvennyy-intellekt-mezhdu-mifom-i-realnostyu

4. Гаранін О. М., Кацко О. О., Моісеєнко Н. В. Інструментарій розробника в курсі «Розробка комп'ютерних ігор». *Новітні комп'ютерні технології*. Кривий Ріг : Видавничий центр ДВНЗ «Криворізький національний університет», 2017. Том XV. С. 161–164.

5. Голуб О.І., Моісеєнко Н. В., Хомінятич А. В. Проекти на базі ігрового рушія UnrealEngine в курсі «Розробка комп'ютерних ігор». *Новітні комп'ютерні технології*, Кривий Ріг : Видавничий центр ДВНЗ «Криворізький національний університет», 2018. Том XVI. С. 255–260.

6. Движок CryEngine – особенности, преимущества и недостатки. URL : https://cubiq.ru/dvizhok-cryengine/#__CryEngine

7. Диденко Г. Орёл или решка: сравнениею. URL : Unity и Unreal Enginehttps://dtf.ru/gamedev/7227-orel-ili-reshka-sravnenie-unity-i-unreal-engine

8. Как работает искусственный интеллект в играх? *itProger*: веб-сайт. URL :

https://itproger.com/news/kak-rabotaet-iskusstvenniy-intellekt-v-igrah

9. Как создавался «живой» ИИ для F.E.A.R.? *PlayGround.ru* : веб-сайт. URL : (https://www.playground.ru/fear/news/kak_sozdavalsya_zhivoj_ii_dlya_f_e_a_r-350024)

10. Латыпова А. Р. Игровой искусственный интеллект как медиум социального мира. *Международный журнал исследований культуры*. 2019. № 1 (34). URL : https://cyberleninka.ru/article/n/igrovoy-iskusstvennyy-intellekt-kak-medium-sotsialnogo-mira/viewer

11. Лебедев В. Не совсем человек: искусственный интеллект в играх. *Skillbox / Геймдев* : веб-сайт. URL : https://skillbox.ru/media/gamedev/iskusstvennyy-intellekt-v-igrakh/

12. Ножов И. М. Практичне застосування штучного інтелекту в комп'ютерних іграх. Москва : РГГУ, 2008. 140 с.

13. Обзор игрового движка Godot Engine. URL : https://teletype.in/@capslock/rJFupc5gE

14. Обзор игрового движка Panda3D и его особенностей. URL : https://dtf.ru/gamedev/108285-obzor-igrovogo-dvizhka-panda3d-i-ego-osobennostey

15. Паласиос Х. Unity 5.x. Программирование искусственного интеллекта в играх. Москва : ДМК Пресс, 2016. 272 с.

16. Что такое искусственный интеллект – (ИИ)? Oracle : веб-сайт. URL : https://www.oracle.com/ru/artificial-intelligence/what-is-ai/

17. Шампандар А. Д. Искусственный интеллект в компьютерных играх: как обучить виртуальные персонажи реагировать на внешние воздействия. Москва : Вильямс, 2007. 768 с.

18. An introduction into source engine particles. URL : https://steamcommunity.com/sharedfiles/filedetails/?id=530493383

19. Blueprint Overview. URL : https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/

20. Cocos Service. URL : https://www.cocos.com/en/service

21. Godot Engine. *Вікіпедія*. URL : https://ru.wikipedia.org/wiki/Godot

22. Level Editor. URL : https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelEditor/)

23. Rocca J., Rocca B. Introduction to Markov chains. *Towards data science.* URL : https://towardsdatascience.com/brief-introduction-to-markov-chains-2c8cab9c98ab

24. Unity3D или Unreal Engine 4 - Режим доступу: https://stfalcon.com/ru/blog/post/unity3d-vs-unreal-engine-4

25. Unreal Engine 4 – один из самых популярных движков. Почему? URL : https://habr.com/ru/company/netologyru/blog/561006/