Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва інституту)

Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(рівень вищої освіти)
на тему
Розроблення додатку-головоломки на базі середовища Unity

Виконав: студент 2 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)
Лазарчук Т.Е.
(прізвище та ініціали)
Керівник _____ Ляхов О.Л.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Полтава – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Розроблення додатку-головоломки на базі середовища Unity»

Студента групи 601-ТН Лазарчука Тимофія Едуардовича

Керівник роботи
доктор технічних наук,
професор Ляхов О.Л.

Завідуючого кафедри
кандидат технічних наук,
доцент Головко Г.В.

Полтава – 2021

РЕФЕРАТ

Кваліфікаційна робота: 66 с., 25 рисунків, 1 додаток, 23 джерел.

Об’єкт дослідження:додаток-головоломка на базі середовища Unity.

Мета роботи: розробити програмний засіб, представлений у вигляді додатку-головоломки за допомогою багатоплатформового інструменту Unity, а саме реалізувати пов’язані з цим механіки, реалізувати інтерфейс додатку, протестувати додаток та ввести його в експлуатацію.

Методи: проектування та розробка сцен додатку, підготовка графічного матеріалу для інтерфейсу, розробка інтерфейсу, реалізація логіки та механіки додатку, його тестування.

Ключові слова: додаток, головоломка, програмування, C#, Unity, інтерфейс, гра, нода.

ABSTRACT

Masterthesis: 66pages, 25figures, 1appendices, 23sources.

Object of study: a puzzle application based on the Unity environment

Objective: to develop a software tool presented as a puzzle application using the multi-platform Unity tool, namely to implement related mechanics, implement the application interface, test the application and put it into operation.

Methods: design and development of application scenes, preparation of graphic material for the interface, interface development, implementation of logic and mechanics of the application, its testing.

Keywords:application, puzzle, programming, C #, Unity, interface, game.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, СИМВОЛІВ ТА ТЕРМІНІВ

Рендер(Render) – означає процес отримання зображення моделі за допомогою комп’ютерних засобів.

Гекса (Gexa), **Нода** (Node) – шестикутник правильної форми, який знаходиться на ігровому полі та може містити у собі інші ігрові об’єкти.

VR (VirtualReality) – віртуальна реальність

ОС– операційна система

ПЗ – програмне забезпечення

ВСТУП

Актуальність теми: Індустрія комп'ютерних ігор з'явилася відносно недавно – майже 30 років тому, але вже змогла розвинутися у величезну галузь із колосальнимидоходами у кілька мільярдів доларів на рік. Зрозуміти подібне раптове зростання популярності віртуальних розваг дуже просто: все це завдяки широкому поширенню комп'ютерних технологій, у тому числіпояві мережі Інтернет. Завдяки цьому, на відміну від інших видів розваг, ігри більш доступні длякінцевого користувача.

Для того щоб просто пограти гравцеві потрібно мати тільки комп'ютер абоігрову приставку та копію самої гри, а з широким розповсюдженням мережіІнтернет для отримання копії гри не потрібно виходити з дому. Більш того, споживачеві не потрібно мати особливих знань для того, щоб вибратипідходящу йому гру, тоді як для більшісты інших видіврозваг необхідно розбиратися як мінімум у необхідному екіпіруванні.Також варто взяти до уваги, що комп'ютерні ігри останнім часом перестали позиціонуватися як програми лише для відпочинку тарозваг. Наприклад, сьогодні, завдяки використанню ігрових технологій, створюються спеціальні комплекси із симуляції, які служать длянавчання спеціалістів у різних галузях: від лісорубів до пілотів реактивних літаків.

В Україні ж, на жаль, дещо інакше. Ігрова індустрія унас, як і в більшості інших країн пострадянського простору,розвинена вкрай слабо. Пов'язано це з тим, що культура комп'ютерних розваг прийшла до нас пізно і практично не розвивалася. Черезце, навіть на досить високий попит, ми маємо дуже мало компаній-розробників, які можуть скласти конкуренцію зарубіжним компаніям.

Тому розвиток технологій у цьому напрямі можна вважатиодним з найперспективніших, особливо в нашій країні.

Мета:розробити прототипкомп'ютерної гри жанру головоломка засобами середовища Unity.Для досягнення поставленої мети необхідно виконати такізавдання:

1) вивчити особливості та стан комп'ютерної промисловості України;
2) вибрати жанр, вид та платформу для комп'ютерної гри;
3) розробити сценарій, концепцію основних елементів;
4) вибрати та вивчити засіб реалізації;
5) підготувати необхідні для гри елементи;
6) реалізація прототипу гри.

Очікуваний результат:повністю функціонуюча гра жанру головоломка, в якій присутні налаштування, можливість грати після програшу, змінювати рівень складності.

РОЗДІЛ 1ТЕОРЕТИЧНА ЧАСТИНА

1.1 Опис предметної області

Ігрові двигуни– це інструменти розробки програмного забезпечення, призначені для зниження вартості, складності та часу виходу на ринок, необхідних для розробки відеоігор. Ці програмні інструменти створюють шар абстракції поверх найпоширеніших завдань у розробці відеоігор. Шари абстракції об’єднані в інструменти, призначені для функціонування як взаємосумісні компоненти, які можна замінити або розширити додатковими компонентами сторонніх розробників.

Ігрові двигуни забезпечують величезні переваги ефективності за рахунок зменшенняглибини знань, необхідних для створення ігор. Вони можуть бути мінімальними за попередньо створеною функціональністю або повнофункціональними, що дозволяє розробникам ігор повністю зосередитися на написанні ігрового коду. Ігрові движки пропонують неймовірну перевагу перед початком з нуля для одиночних розробників або команд, які просто хочуть зосередитися на створенні найкращої гри [1].

Добре розроблені сучасні ігрові двигуни добре справляються з розділеннямвнутрішніх функціональностей. Ігровий код буде звертатися до чітко визначених інтерфейсів API двигуна, щоб запитувати такі речі, як «намалювати цей спрайт у цьому місці» тощо.Компонентна архітектура добре розробленої гридвигуна дозволяє розширювати функціонал гри, що заохочує прийняття такого двигуна як базу для розробки, тому що команда розробників не прив'язана до заздалегідь визначеного набору можливостей. Ця розширюваність особливо важлива, якщо вихідний код ігрового движка недоступний з відкритим вихідним кодом або його ліцензувати занадто дорого.

Ігровий двигун Unity спеціально створений, щоб дозволяти сторонні плагіни. Це навіть заходить так далеко, що він має у собі Asset Store, що містить

плагіни, доступні через редактор Unity.Багато ігрових двигунів також дозволяють кросплатформну компіляцію,це означає, що код гри не обмежений однією платформою.Двигун робить це, не роблячи припущень щодо основногоархітектури комп'ютера та дозволяє розробнику вказати, яку платформу він використовує. Якщо ви хочете випустити свою гру для консолі, комп'ютера та мобільного пристрою, ігровий двигун дозволяє перемикати кілька перемикачів, щоб налаштувати конфігурацію збірки для цієї платформи.

Але є застереження щодо чудес кросплатформної компіляції. Хоча міжплатформна компіляція є дивовижною функцією та свідченням того, як далеко зайшли ігрові технології, ми повинні мати на увазі, що якщо ми створюємо гру для кількох платформ, то нам потрібно буде надати різні розміри зображення та дозволити читання коду в елементи керування для прийому різних видів периферійних пристроїв, таких як клавіатура.

Можливо, нам знадобиться налаштувати макет нашої гри на екрані, а також багато інших завдань.Насправді, просто перенести гру з однієї платформи на іншу може бути доволі складно, але нам, ймовірно, не доведеться торкатися самого ігрового двигуна. Деякі ігрові двигуни настільки візуально орієнтовані, що дозволяють створювати ігри без написання жодного рядка коду[2].

Unity має можливість спроектувати користувацькі інтерфейси, які можна налаштувати для використання іншими членами команди розробників, які не є програмістами, такими як дизайнери рівнів, аніматори, арт-директори та ігрові дизайнери.

Існує багато різних типів ігрових двигуніві немаєправила щодо того, яка функціональність є абсолютно необхідною, щоб вважатися ігровим двигуном.

Найпопулярніші ігрові двигуни містять деякі або всі наступні функції:

- механізм візуалізації графіки, що підтримує 2D або 3D графіки;
- фізичний механізм, який підтримує виявлення зіткнень;

- аудіосистема для завантаження та відтворення звуків і музичних файлів;
- підтримка сценаріїв для реалізації логіки гри;
- об'єктна модель світу, що визначає зміст і властивості ігрового світу;
- обробка анімації для завантаження кадрів анімації та їх відтворення;
- мережевий код, що дозволяє грати в декількох гравців, вміст, який можна завантажити, і таблиці лідерів;
- багатопоточність, що дозволяє виконувати ігрову логіку одночасно;
- керування пам'яттю, оскільки немає комп'ютера з необмеженоюпам’яттю;
- штучний інтелект для пошуку шляху та моделювання поведінки опонентів.

Для більш точного уявлення про ігровий двигун як платформу для розробки ігор, розглянемо наступну аналогію.Скажіть, що ви хочете побудувати будинок. Для початкуцей будинок буде матибетонний фундамент, гарну дерев’яну підлогу, міцні стіни, захищений від погодних умовдерев'яний дах. Є два способи побудувати цей будинок:

Перший спосіб побудувати будинок: викопайте землю за допомогою ручної лопати, поки не викопаєте достатньо глибоко, щоб закласти фундамент. Зробіть бетон, нагріваючи вапняк і глину при 2640 °F в печі, подрібніть його та змішайте з невеликою кількістю гіпсу. Візьміть порошок бетону, який ви створили, змішайте його з водою, щебенем або дрібним піском і закладіть фундамент.

Одночасно з закладкою фундаменту вам знадобиться сталева арматура щоб зміцнити бетон. Зберіть залізну руду, необхідну для виготовлення сталевої арматури, і виплавіть її в доменній печі для отримання злитків. Розплавіть і гаряче прокатайте ці злитки в міцні арматурні прути для бетонного

фундаменту. Після цього настав час побудувати каркас, на який ви будете вішати стіни. Візьміть сокиру і почніть рубати дерева. Вирубати кілька сотень деревини буде достатньо, щоб забезпечити сировину, але потім вам потрібно буде взяти кожну деревину та подрібнити її на пиломатеріали. Коли ви закінчите, не забудьте обробити пиломатеріали, щоб вони були стійкими до погодних умов, не гнили й не заражалися комахами. Побудуйте свої лаги та прогони, на які ви будете стелити підлогу

Другий спосіб побудувати будинок: придбайте мішки з бетону, сталеві арматури, оброблені пиломатеріалів з млина, десяток ящиків з паперовою стрічкою оцинкованих цвяхів і пневматичний пістолет для цвяхів. Перемішайте і залийте бетон, щоб створити фундамент, покладіть готову сталеву арматуру, дайте бетону застигнути, а потім побудуйте підлогу з обробленої деревини.

Перший спосіб будівництва будинку вимагає колосальних витрат тазнання просто для створення матеріалів, необхідних для початку будівництва. Для цього підходу потрібно знати точне співвідношення сировини,необхідні матеріали та технології виготовлення бетону та сталі. Вам знадобиться знати, як валити дерева, не застрягаючи під ними, і знати, які хімічні речовини потрібні для обробки пиломатеріалів, які ви дуже старалися розрізати на сотні однорідних балок. Навіть якби ви володіли всіма знаннями, необхідними для побудови будинку таким чином, це все одно зайняло б у вас тисячі годин [3].

Цей перший підхід аналогічний тому, щоб сісти написати відеогрубез використання ігрового двигуна. Ми повинні робити все з нуля: писати математичні бібліотеки, графічний код візуалізації, алгоритми виявлення зіткнень, мережевий код, бібліотеки завантаження активів, код аудіоплеєра та багато іншого. Навіть якби ми з самого початку знали, як робити всі ці речі, нам все одно знадобиться багато часу, щоб написати код ігрового двигуна та налагодити його. Якщо ви не знайомі з лінійною алгеброю, технікою візуалізації та оптимізацією алгоритмів відбирання, то слід очікувати, що нам знадобляться роки, перш ніж у нас буде достатньо бази, щоб ми могли почати писати гру та щоб підтримати її. .

Другий спосіб будівництва будинку передбачає, що ми не починаємоповністю з нуля. Для цього не потрібно знати, як працюватидоменної печі, вирубували сотні деревини або фрезерували їх для виготовлення пиломатеріалів. Другий спосіб дозволяє нам повністю зосередитися на будівництві будинку, а не на виготовленні матеріалів, які нам знадобляться для будівництва будинку. Наш будинок буде збудований швидше, в результаті буде коштувати дешевше і, ймовірно, буде якісніше, якщо ми ретельно підберемо матеріали та вміємо їх використовувати.Другий підхід аналогічний тому, щоб сісти написати відеогру з використанням попередньо вбудованого ігрового двигуна. Розробники гри можуть зосередитися на вмісті гри, і їм не потрібно знати, як робити складні розрахунки, щоб з'ясувати, чи зіткнулися два об'єкти, коли вони літали в повітрі, тому що ігровий движок зробить це за них. Немає необхідності створювати систему завантаження активів, писати низькорівневий код для читання введених користувачами даних, розпаковувати звукові файли або аналізувати формати файлів анімації. Немає потреби створювати цю функцію, загальну для всіх відеоігор, оскільки розробники ігрового двигуна вже витратили тисячі годин на написання, тестування, налагодження та оптимізацію коду, щоб зробити це за нас [4].

Неможливо переоцінити перевагу, яку ігрові двигуни дають незалежному розробнику або команді великої студії, що працює над наступною хітовою грою. Деякі розробники хочуть написати власні ігрові движки як вправу з програмування, щоб дізнатися, як усе працює під капотом, і вони здобувають величезну кількість знань. Але якщо ми маємо намір випустити гру, то ми робимо собі ведмежу послугу, використовуючи готову основу ігрового двигуна.

Історично ігрові двигуни були тісно пов'язані зсаміми ігри. У 1987 році Рон Гілберт разом із певною допомогою Чіпа Морнінгстара створив SCUMM, або утиліту створення сценаріїв для ігрового двигуна Maniac Mansion, працюючи в Lucasfilm Games. SCUMM — чудовий приклад ігрового двигуна, створеного на замовлення для певного типу гри. «MM» у SCUMM означає Maniac Mansion, яка була визнана критиками пригодницькою грою та першою,

що використовувала інтерфейс у стилі «накажи й клацни», який також винайшов Гілберт.

Ігровий двигун SCUMM відповідав за перетворення скриптівщо складається з зрозумілих людиною токенізованих слів, таких як «прогулянка від персонажа до дверей» у програми байтового розміру для читання інтерпретатором ігрового движка. Перекладач відповідав за керування акторами ігор на екрані та подання звуку та графіки. Можливість створювати сценарій ігрового процесу замість його кодування сприяла швидкому створенню прототипів і дозволила команді почати будувати і зосереджуватися на ігровому процесі з ранньої стадії. Хоча двигун SCUMM був розроблений спеціально для Maniac Mansion (рис 1.1), він також використовувався для інших популярних ігор, таких як Full Throttle, The Secret of Monkey Island, Indiana Jones and the Last Crusade: The Graphic Adventure тощо.

Рисунок 1.1–Maniac Mansion від Lucasfilm Games використовує двигун SCUMM

Якщо порівнювати з сучасними ігровими двигунами, такими як Unity, toSCUMM Engine не має великої гнучкості, оскільки він був створений на замовлення для ігор у стилі «накажи й клацни». Однак, як і Unity, движок

SCUMM дозволив розробникам ігор зосередитися на ігровому процесі замість постійного переписування графічного та звукового коду для кожної гри, заощаджуючи незліченну кількість часу та зусиль [5].

Іноді ігрові движки можуть мати величезний вплив на промисловості в цілому. У середині 1991 року в компанії id Software стався сейсмічний зрушення в індустрії, коли 21-річний Джон Кармак побудував 3D-ігровий движок для гри під назвою Wolfenstein 3D. До того часу 3D-графіка, як правило, обмежувалася повільними іграми-симуляторами польотів або іграми з простими полігонами, оскільки доступне комп'ютерне обладнання було занадто повільним для обчислення та відображення кількості поверхонь, необхідних для швидко розвивається тривимірної гри. Carmack зміг обійти поточні апаратні обмеження за допомогою графічної техніки, яка називається raycasting. Це дозволило швидко відображати 3D-середовища шляхом обчислення та відображення лише поверхонь, видимих програвачу, а не всієї області навколо програвача. Цей унікальний підхід дозволив Кармаку разом з Джоном Ромеро, дизайнером Томом Холлом та художником Адріаном Кармаком створити жорстоку, швидку гру про те, як знищити нацистів, яка породила жанр відеоігор від першої особи (FPS). Механізм Wolfenstein 3D був ліцензований компанією id Software на кілька інших ігор. На сьогоднішній день вони створили сім ігрових движків, які використовувалися у таких впливових іграх, як Quake III Arena, перезавантаження Doom і Wolfenstein II: The New Colossus.

У наші дні досвідчений розробник гри може створити грубий прототип 3D FPS-ігри за кілька днів, використовуючи потужний ігровий движок, як-от Unity [6].

Сучасні ігрові двигуни.Сучасні студії розробки ігор AAA, такі як Bethesda GameStudios та Blizzard Entertainment часто мають власні,фірмові ігрові двигуни. Внутрішній ігровий движок Bethesda називається:Creation Engine і використовувався для створення The Elder Scrolls V: Skyrim, а також

Fallout 4. Blizzard має власний власний ігровий двгун, який використовується для створення таких ігор, як World of Warcraft і Overwatch.

Власний ігровий двигун може початися як створений для конкретного ігрового проекту. Після виходу цього проекту ігровий двигун часто знаходить нове життя, коли його повторно використовують для наступної гри, що виходить від цієї ігрової студії. Двигун може вимагати оновлення, щоб залишатися актуальним і використовувати новітні технології, але його не потрібно перебудовувати з нуля. Якщо компанія, що займається розробкою ігор, не має власного двигуна, вони зазвичай використовують двигун з відкритим кодом або ліцензують сторонній двигун, такий як Unity. Створити значну тривимірну гру в наші дні без використання ігрового движка було б надзвичайно складним завданням – як у фінансовому, так і в технологічному плані. Фактично, ігрові студії з власними ігровими двигунамивимагають окремих команд програмування, які повністю займаються розробкою функцій двигуна та їх оптимізацією. Сказавши все це, чому ААА-студія вирішила б не використовувати ігровий движок, як Unity, а замість цього вирішила створити власний власний движок? Такі компанії, як Bethesda і Blizzard, мають величезний набір попередньо існуючого коду, з якого можна черпати, фінансові ресурси та велику кількість глибоко талановитих програмістів. Для певних типів проектів вони хочутьповний контроль над кожним аспектом їхньої гри та ігрового движка.Навіть маючи всі ці переваги перед типовою маленькою гроюстудії, Bethesda все ще використовувала Unity для розробки мобільної гри FalloutShelter; і Blizzard використовували Unity для розробки крос-платформиколекційна карткова гра: Hearthstone. Коли час дорівнює грошам, ігровийдвигун, такий як Unity, можна використовувати для швидкого прототипування, розробки та ітерації функціональності. Рівняння час = гроші особливо актуально, якщо ви плануєте випустити гру на кількох платформах. Перенесення внутрішнього двигуна на певні платформи, такі як iOS та Android, може зайняти багато часу. Якщо проект не вимагає такого рівня контролю над

ігровим движком, який вам знадобився б при розробці гри, як-от Overwatch, використовувати кроссумісний ігровий движок, як-от Unity, неважко.

Двигун UNITY. Unity — надзвичайно популярний ігровий двигун, який дає величезну кількість переваг перед іншими ігровими движками, доступними сьогодні на ринку. Unity пропонує візуальний робочий процес з можливостями перетягування та підтримує сценарії за допомогою C#, дуже популярної мови програмування. Unity вже давно підтримує 3D- і 2D-графіку, і набори інструментів для обох стають все більш складними та зручнішими з кожним випуском. Unity має декілька рівнів ліцензій і безкоштовний для проектів із доходом до 100 тисяч доларів. Він пропонує міжплатформну підтримку для 27 різних платформ і використовує графічні API, специфічні для архітектури системи, включаючи Direct3D, OpenGL, Vulkan, Metal та декілька інших. UnityTeams пропонує спільну роботу над проектами в хмарі та безперервну інтеграцію[7].

З моменту свого дебюту в 2005 році Unity використовувався для розробки тисячнастільних, мобільних та консольних ігрор та програм. Невелика вибірка деяких добре відомих ігор, розроблених протягом багатьох років разом із Unity, включає:EscapefromTarkov (2017),ThomasWasAlone (2010), TempleRun (2011), TheRoom (2012), RimWorld (2013), Hearthstone (2014), KerbalSpaceProgram (2015), PokémonGO (2016) і Cuphead (2017), що показано на рисунку 1.2.

Рисунок 1.2– Cuphead, розроблений StudioMDHR, використовує Unity

Для розробників ігор, які хочуть налаштувати свій робочий процес, Unityнадає можливість розширити візуальний редактор за замовчуванням. Це надзвичайнопотужний механізм дозволяє створювати спеціальні інструменти, редактори та інспектори [8]. Уявіть собі, що створюєте візуальний інструмент для ваших ігрових дизайнерів, щоб легко налаштовувати значення для ігрових об’єктів, таких як точки попадання для класу персонажів, дерева навичок, діапазон атаки або випадання предметів, без необхідності вникати в код і змінювати значення або використовувати зовнішній бази даних. Усе це стало можливим і зрозумілим завдяки функціональності розширення редактора, яку надає Unity. Ще одна перевага Unity — це Unity Asset Store. Asset Store — це інтернет-магазин, де художники, розробники та творці контенту можуть завантажувати вміст для покупки та продажу. Asset Store містить тисячі безкоштовних і платних розширень редактора, моделей, скриптів, текстур, шейдерів тощо, які команди можуть використовувати для прискорення розробки тапокращити кінцевий продукт.

1.2Оглядіснуючихпрограмнихрішень

PuzzleSweeper – проект жанру головоломки, який був розроблений компанією ElectricFruit. Випущена гра була у 2015 році. Її видавцем була компанія JohnBurton. PuzzleSweeperдоступна на iOS, завантажити її можна в AppStore. Основна концепція гри полягає в найшвидшому шляху до виходу з печери, знешкодженні пасток на знаходженні скарбів.

Рисунок 1.3 – Інтерфейс гри PuzzleSweeper

Pushy – проект у жанрі освітньої головоломки, який був розроблений компанією medienwerkstatt. Реліз гри стався 11 жовтня 2009 року. Verlagsgesвиступити видавцем. Pushyдоступна на iOS, завантажити її можна в AppStore. Основна ідея гри полягає у врятуванні живої істоти, але щоб достигнути цього гравцеві потрібно задіяти логіку, стратегічне мислення та деяку вдачу, тому що розташовані на ігровому рівні пастки не дадуть дійти до кінця з легкістю.

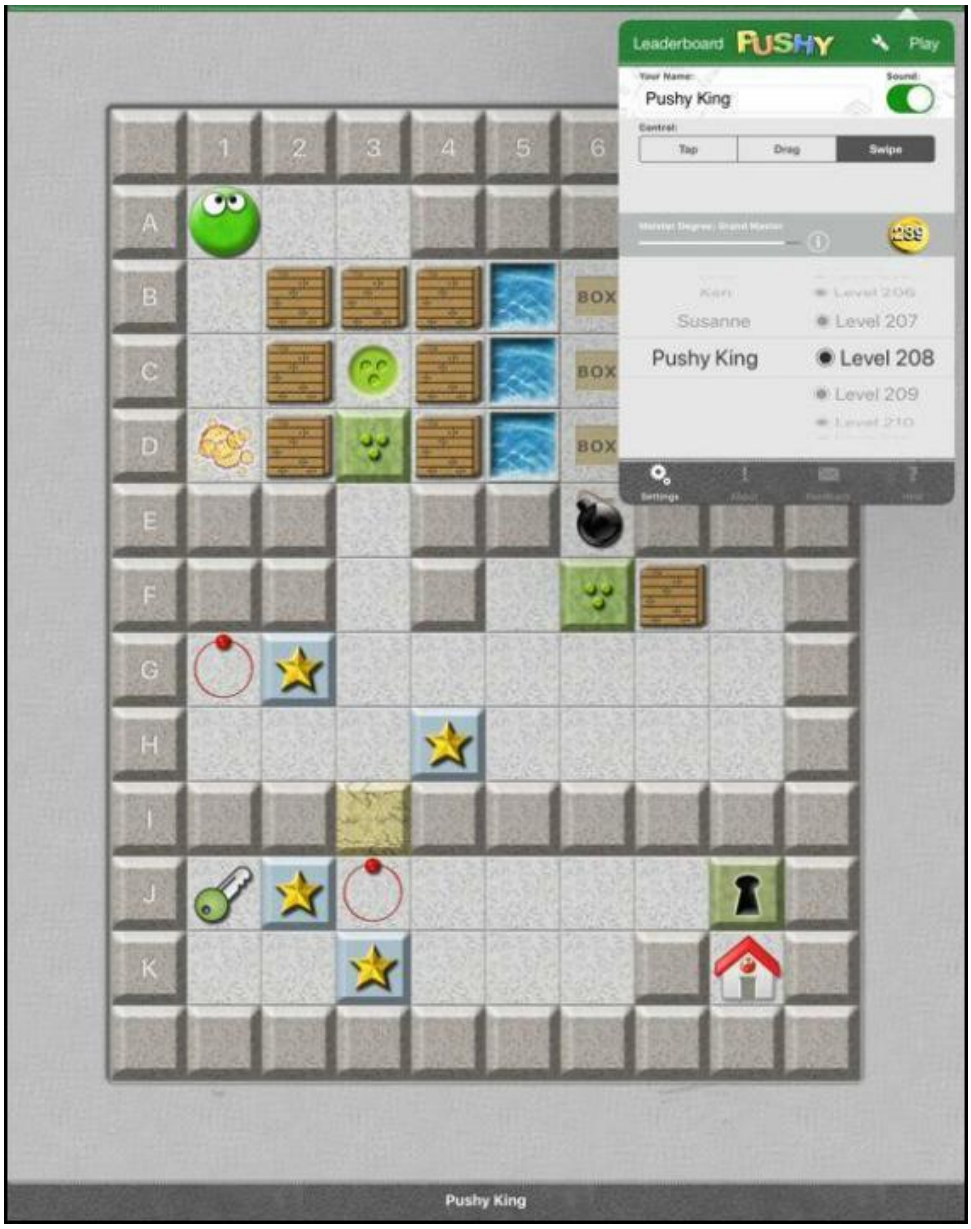

Рисунок 1.4 – Інтерфейс гри Pushy

Aqueduct – гра в жанрі стратегії та головоломки, яку розробила компанія KiefferBros. Вона вийшла 29 травня 2010 року. Видавцем виступила компанія-розробник. Гра доступна на iOS, завантажити її можна в AppStore. Концепція гри полягає в проведенні струму води по акведукам, які розташовані у хаотичному порядку. Гравцеві потрібно використати логіку та стратегію для досягнення цілі.

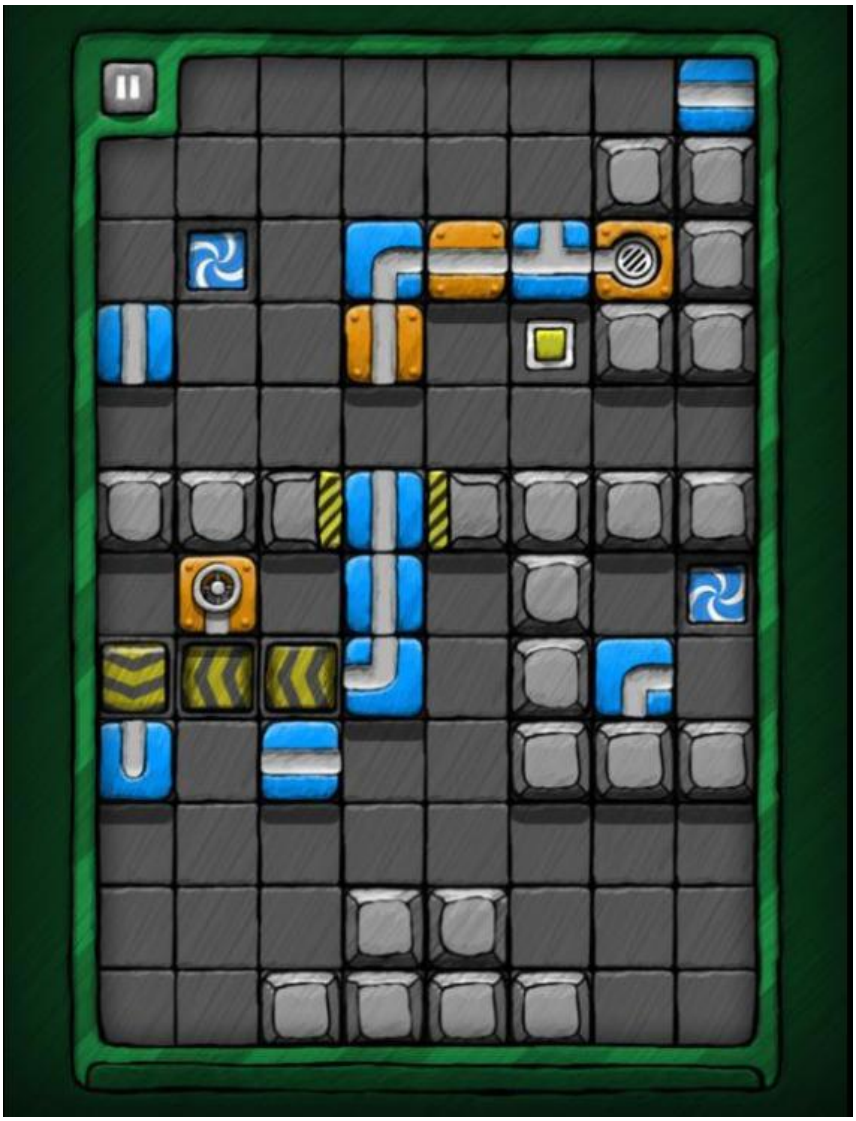

Рисунок 1.5 – Інтерфейс гри Aqueduct

DiamondCaves 3 – гра жанру головоломка, розробником якої є diamondproduction. Вийшла в реліз 18 січня 2014 року. Її видавцем виступила компанія-розробник. Доступна на PC. Основна ідея полягає у зборі ресурсів та втечі з рівня. Всього в грі представлено більш ніж 60 рівнів, на кожному з яких гравця чекають не тільки скарби, але й пастки та ворожі монстри.

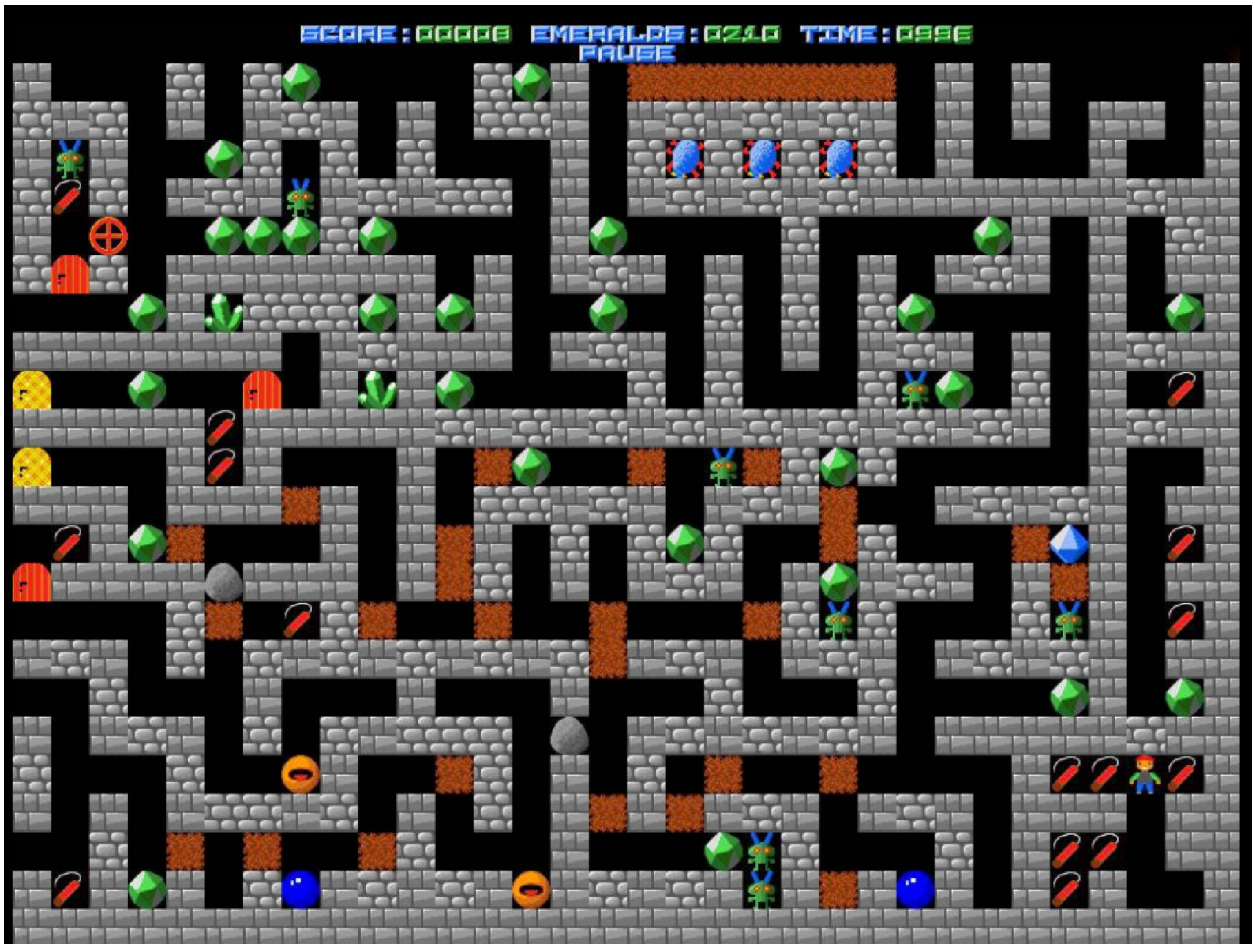

Рисунок 1.6 – Інтерфейс гри DiamondCaves 3

1.3 Постановка задачі

Метою дипломної роботи є розроблення додатку-головоломки на базі середовища Unity.Додаток має бути простим у використанні та нести у собі ціль розважити користувача. Також додаток такого типу може бути інтегрований в більш складні та комплексні проекти, де додатки такого типу часто використовуються заради досягнення деяких цілей.

Розроблене програмне забезпечення не повинне займати багато місця чи задіювати багато ресурсів комп’ютера, не кажучи вже про використання шоломів віртуальної реальності, так як останні значно зменшать кількість користувачів, здатних скористуватися додатком[9, 10].

Для досягнення поставленої метинеобхідно вирішити наступні задачі:

1) Виконати аналіз вимог та розробити концепт додатку;

2) Виконати проектування додатку;
3) Реалізувати інтерфейс додатку;
4) Реалізувати механіки додатку;
5) Виконати тестування розробки.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Функціонал та структура додатку

Так як гра жанру головоломка за визначенням повинна бути націлена на рішення поставленої задачі користувачем, при чому для досягнення мети він повинен використовувати інтуіцію, логіку на вдачу, то і функціонал додатку був спроектований на використання цих якостей.

В самій грі ігрове поле поділено на шестикутники правильної форми (також їх називають гексами або нодами), у поля є початок, тобто шестикутник, на якому гравець починає свій шлях, та кінець, тобто шестикутник, до якого гравцеві потрібно дібратися. Де він знаходиться конкретно гравець не знає, але про це пізніше. Пересування по полю здійснюється шляхом послідовного відкриття одного з прилеглих до гравця шестикутників. При відкритті пустого шестикутника гравець може бачити число, яке показує, за якій віддаленості від конкретного шестикутника знаходиться фінальний гекст, або один з «бафів», який допоможе гравцю при пересуванні.

Всього в додатку реалізовані шестикутники наступних типів:

- закритий – гравець не знає що в ньому міститься, але його можна відкрити при знаходженні на суміжному шестикутнику;
- відкритий –вже пройдений шестикутник, на якому або нічого немає, або знаходиться хороший або поганий ефект;
- посилювач вірусу – відкрита нода, у якій міститься посилювач вірусу, тобто баф, який допоможе гравцеві подолати антивірус;
- антивірус – відкрита нода, у якій міститься антивірус, який негативно впливає на характеристики вірусу гравця та не дає йому змоги переходити на суміжні з антивірусом ноди;

- фінальний вузол – відкрита нода, яка також містить у собі антивірус, але при подоланні якого гра закінчується.

У гравця також є свої параметри життя на сили. Сила показує, скільки поінтів здоров'я гравець може забрати у вірусу за один хід. При досягненні кількості одиниць здоров'я у вірусу або у гравця останній помирає. При смерті гравця гра закінчується. При смерті фінального вузла гра закінчується. Залежно від дій гравця та результату гри, після її закінчення гравцеві присуджуються бали. Чим більше балів набрав гравець, тим більш результативним було його проходження.

В головному меню додатку користувач може обрати наступні дії:

- STARTGAME (РОЗПОЧАТИ ГРУ) – безпосередній перехід до рівня додатку, на якому проходить гра.
- OPTIONS (НАЛАШТУВАННЯ) – перехід в підпункт меню, у якому можна скористатися повзунком налаштування рівня гучності.
- CREDITS (КРЕДИТИ) – перехід до підпункту меню, в якому перераховані усі причасні до роботи люди.
- QUIT (ВИХІД) – закриття додатку.

Також присутнє меню паузи, яке викликається в будь-який момент знаходження на ігровому рівні.

В ньому доступні такі можливості:

- RESUME (ВІДНОВИТИ) – процес зняття з паузи та продовження гри реальному часі.
- MAIN MENU (ГОЛОВНЕ МЕНЮ) – повернення до головного меню додатку.
- QUIT (ВИХІД) – закриття додатку.

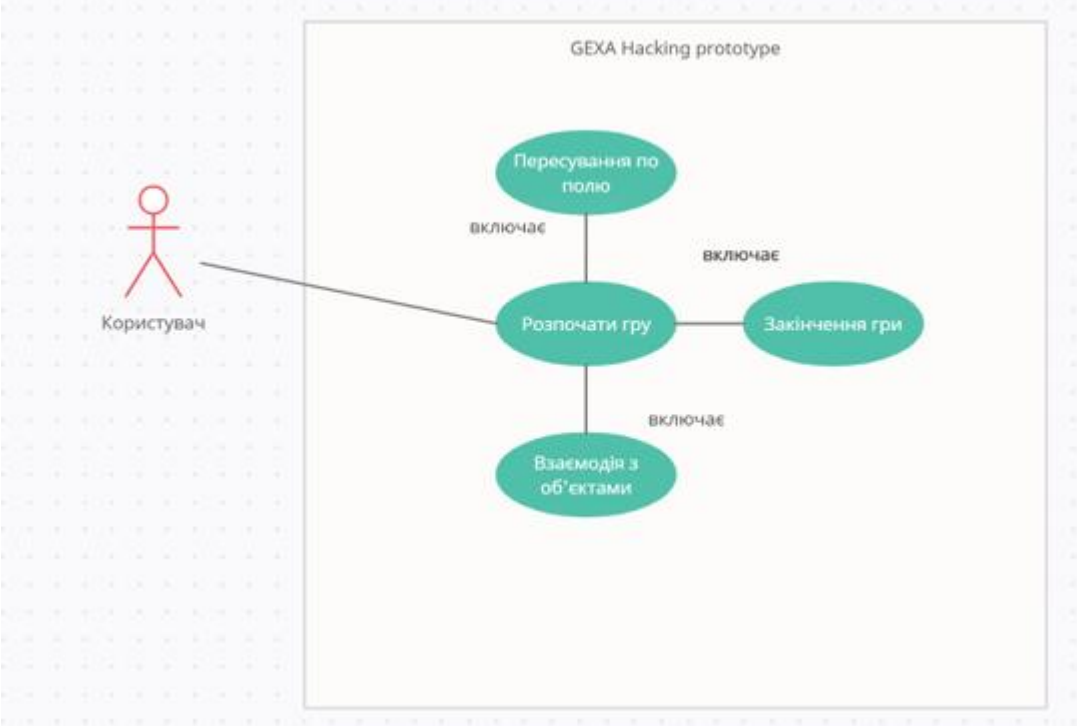

Рисунок 2.1 – Діаграма варіантів використання

2.2 Архітектурадодатку

Загальна архітектура проекту Unity 3D основана на шаблоні Entity-Component-System [4]. Згідно до цього шаблону, додаток складається з базових сутностей, функціональність яких розширяється за допомогою спеціальних компонентів.

Проект Unity 3D складається з декількох сцен (Scene), на яких розташовані ігрові об’єкти (GameObject) з прикріпленим до них компонентами (Component). У кожного об’єкта є обов’язковий компонент Transforn, який відповідає за розташування об’єкту на загальній сцені. Окрім цього, можуть бути підключені як готові компоненти (наприклад Rigidbody, який відповідає за фізичну симуляцію), так і користувацькі компоненти [11].

Безпосередньо програмування в Unity 3D являє собою в першу чергу розробку користувацьких класів, які підключаються до ігрових об’єктів як компоненти та скриптів. Всі класи повинні успадковуватися від класу MonoBehaviour [7].

Були визначені наступні стани додатку [9].

Стан «Меню» – перший стан додатку. При переході в нього відкривається головне меню гри, після команди користувача здійснюється перехід до наступного стану «Початок гри»

Стан «Початок» – другий стан додатку. При переході до цього стану відбувається ініціалізація необхідних для гри об’єктів. Перехід в наступний стан здійснюється автоматично після ініціалізації ігрових об’єктів.

Стан «Гра» – третій стан додатку. Коли додаток знаходиться у цьому стані, користувач здатній пересуватись по ігровому полю, взаємодіяти з об’єктами.

Стан «Результат» – останній стан додатку, при переході до цього стану ігрові об’єкти усуваються і здійснюється перехід до стану «Меню».

Кінцевого стану гри не представлено, так як додаток розрахований на те, що користувач може завершити роботу з ним в будь-який момент.

2.3 Розробка дизайну

В результаті аналізу предметної області був розроблений інтерфейс програмного забезпечення [6]. Він складається з декількох основних екранів – головного меню, меню налаштувань, ігрового рівня та екрану результатів. Також для гри був розроблений логотип (рис. 2.2). Візуальний дизайн був розроблений у Sci-Fістилі, так як він найближче підходить до тематики гри. При розробленні логотипу були використані характерні для цього додатку форми нодів, гексів, або ж просто шестикутників –але в даному випадку один з кутів у фігури відсутній, що додає логотипу загадковий вигляд.

Рисунок 2.2 –Логотип додатку

У головному меню (рис. 2.3) знаходяться наступні елементи: кнопка початку гри, кнопка налаштувань, кнопка кредитів, кнопка виходу з гри.

Рисунок 2.3– Головне меню

У меню налаштувань (рис. 2.4) знаходяться наступні елементи: випадаюче меню з можливістю вибору мови додатку, повзунок зміни рівня звуків гри, повзунок зміни рівня музики гри.

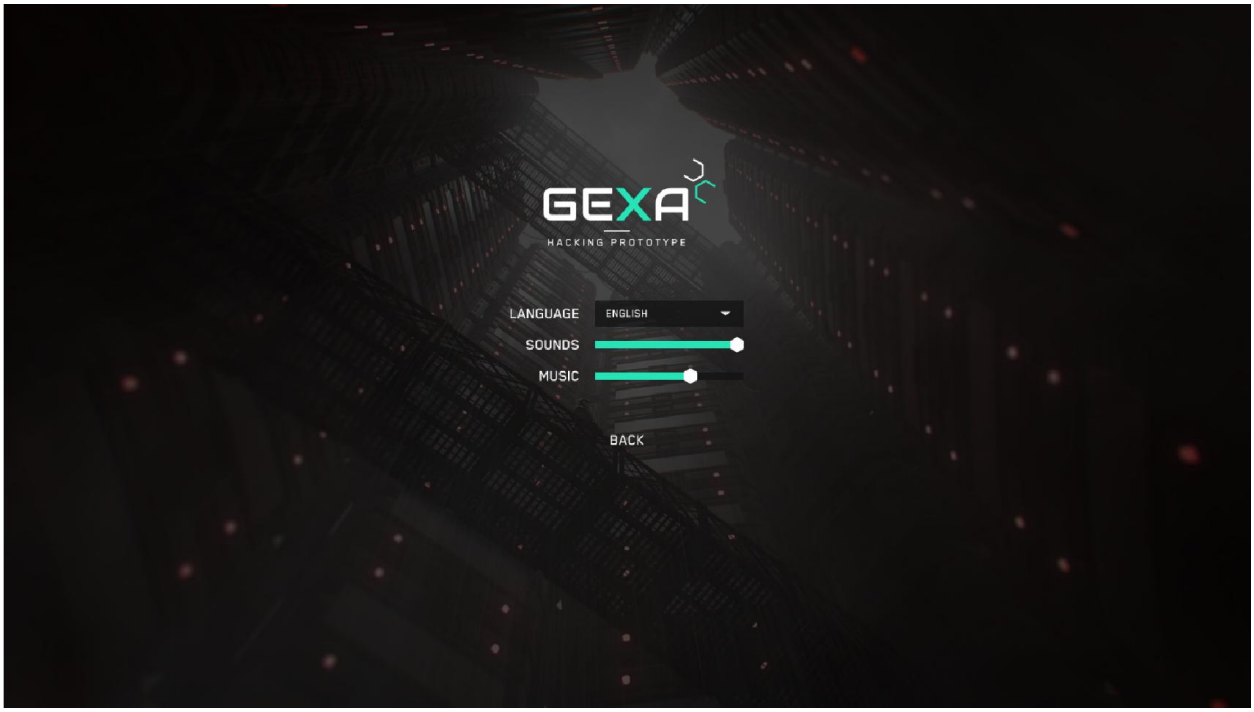

Рисунок 2.4– Меню налаштувань

На сцені з грою (рис. 2.4) присутнє поле з шестикутниками (гексами), шкала рівня здоров’я гравця, наявні у нього усилення (бафи).

Рисунок 2.5– Сцена з гри

У меню з фінальними результатами гри (рис. 2.5) знаходяться кнопки Розпочати заново, виходу у головне меню, кнопка виходу з додатку, кількість зачислених за гру балів.Бали гравець накопичує по ходу гри за подолання пасток, використання посилювачів, а також на фінальний бал впливає кількість ходів, за які гравець дібрався до виходу.

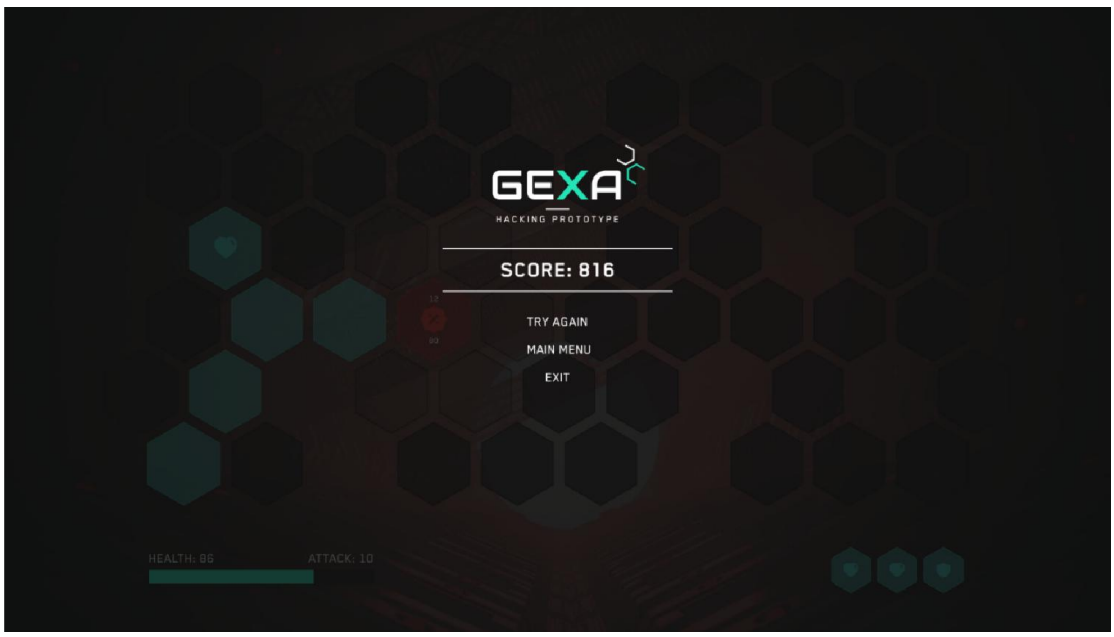

Рисунок 2.6– Меню результату гри

РОЗДІЛ 3
РОЗРОБКА ДОДАТКУ-ГОЛОВОЛОМКИ

3.1 Вибірплатформи

Симулятор розробляється на мові програмування C# за допомогою Visual Studio [1].

Програмування – досить складна технічна наука, яка вимагає досить глибоких знань в області інформатики, фізики, математики, а також синтезу декількох язиків програмування та предметних областей, в рамках яких необхідно працювати. Для поліпшення життя та процесу розробки продукту, була обрана мова програмування C#. Це обумовлено відразу декількома аспектами.

По-перше, ця мова програмування використовує об’єктно-орієнтований підхід. Це означає, що не потрібно описувати абстрактні конструкції на основі предметної області, а потім реалізовувати взаємодію між ними. Даний метод користується популярністю, тому що для роботи не потрібно держати в голові всю інформацію, а працювати по принципу вхідні дані - результат. Також присутні конструкції синтаксичного характеру, які дозволяють перекласти половину роботи на компілятор.

По-друге, все описане вище працює на платформі .NET FRAMEWORK. Що це значить - написаний програмістом код на мові програмування C# буквально транслюється в проміжну мову (IL), яка в свою чергу транслюється в машинний код на автоматі користувача (комп’ютері) відразу при роботі додатку. Така робота дає змогу двом програмістам, які працюють над одним проектом не перевчатися кожен раз, коли потрібно внести якісь правки. Також фінальна компіляція із проміжного коду в машинний відбувається живо та на конкретній машині, що інколи дає змогу покращити продуктивність роботи програми або додатку за рахунок використання специфічних команд процесору.

По-третє, ця мова програмування має досить багато бібліотек та шаблонів, що так би мовити не змушує програміста «винаходити велосипед»раз за разом. Необхідно просто скачати потрібне рішення (в більшості своєї вони безкоштовні) та почати його використовувати. Сюди ж можна віднести велику кількість навчального та довідкового матеріалу.

Розробка велася в середовищі розробки VisualStudio [13].

VS – це серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET CompactFramework та Microsoft Silverlight [13].

Двигуном симулятору було обрано платформу Unity [12].

Unity – це найпопулярніше середовище для розробки ігор. Воно зберігає у собі велику кількість особливостей та гнучка достатньо щоб дозволити зробити будь-яку гру чи симулятор. Дякуючи безпосередніми особливостями свого крос-платформового програмування, Unity дуже популярна як серед новачків, для яких програмування та розробка ігор лише хобі, так і серед AAA студій світового рівня. Так в цьому середовищі були створені такі ігри як Pokemon GO, Hearthstone, Rimworld, Albion та сотні інших. Хоча в офіційній назві присутні символи 3D, Unity також використовується і для розробки 2D ігор. На рисунку 3.1 зображено вікно «AboutUnity».

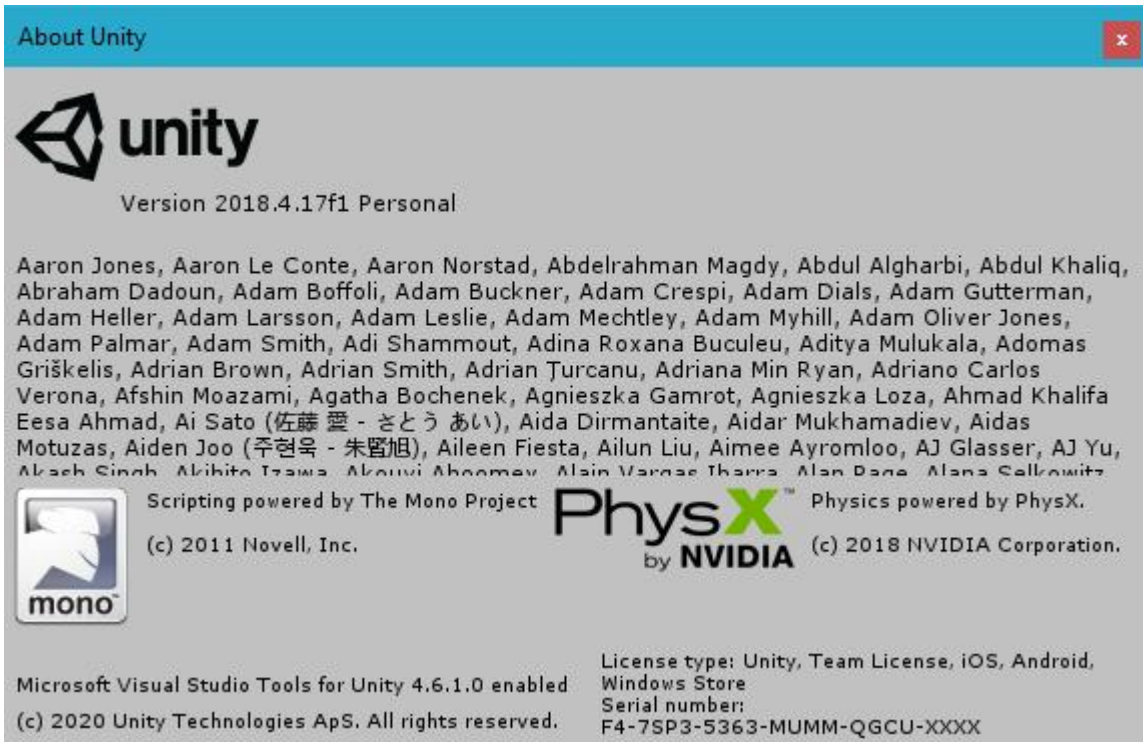

Рисунок 3.1 –Вікно «AboutUnity»

Програмісти та розробники ігор люблять Unity перш за все за API C# скриптинг та вбудований плагінVisualStudio, але також воно надає JavaScript як альтернативну мову в середовищі MonoDevelop IDE для тих, кому одного C# мало.

Також слід відмітити, що Unity користується популярністю серед модельєрів та артистів, які використовують одну з її сильних сторін, а саме багатий на вибір анімаційний інструментарій, для створення 3D сцен або 2D анімацій з начерку.

Unity пропонує безкоштовну версію для всіх бажаючих, що також є одним з рішучих факторів при виборі платформи розробки, але також пропонує платні версії зі своїми особливостями та бонусами [12].

Unity було побудоване на платформі C++ та оптимізоване протягом років для покращення продуктивності. Користувачі Premium навіть отримають доступ до вихідного коду для ще більших можливостей.

Міжплатформова розробка ігор є головною особливістю для сьогоднішніх розробників, і саме у цій галузі Unity вирівається вперед поміж інших

конкурентів. При підтримці кожної ігрової консолі та операційної системи, розробка ігор та додатків у Unity виглядає як ніколи привабливо, а за допомогою інструментів редактора можна одночасно обробляти інформацію з миші, клавіатури та контролерів для ігор.

Існує також досить сильна підтримка «хмарних» рішень для багатокористувацьких ігор з хостингом серверу, що робить його кращим вибором для мультиігрового досвіду. Колективна співпраця також значно покращилася в останніх версіях, вбудований контроль версій та інтеграція в «хмарові» сервіси полегшують роботу з іншими людьми.

Ще однією особливістю Unity є налаштований редактор з повною підтримкою API для створення власних інструментів та сценаріїв. Можна зробити майже будь-який інструмент, який потрібен для Unity, за допомогою самої Unity [12].

І, безумовно, слід згадати магазин асетів, який містить тисячі моделей, сценаріїв, сцен, матеріалів та всього іншого що можна захотіти. Можна навіть продавати власні асети в магазині середовища, такі як, наприклад, моделі або текстури.

Коротко переваги платформи Unity можна описати так:

- Unity – це найкраще середовище для розробки ігор. Воно дуже ефективне при рендері 2D або 3D сцен, надаючи картинку гарної якості порівняно з іншими аналогами.
- Unity ідеально підходить для кросс-платформового програмування, яке набирає обороти з кожним роком. Це зберігає дуже багато часу, коли один скрипт може працювати на декількох платформах одночасно.
- Наявність магазину асетів, в якому можна знайти майже всі необхідні матеріали для розробки, починаючи з маленьких об’єктів та закінчуючи цілими складними анімаціями.
- Графічний інтерфейс Unity досить простий та зрозумілий як для новачка, так і для досвідченого користувача.

- Наявність великої кількості навчальних та довідкових матеріалів робить процес знайомства новачка з Unity досить приємним та не займе багато часу.
- Якщо говорити про професійну розробку, ціна платної версії Unity досить дешева порівняно з аналогами, що робить вибір в його користь довгостроковим вкладенням.

Для розробки користувацького інтерфейсу була обрана програма Figma.

Figma — векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи, що розробляється однойменною компанією. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач. Є безкоштовним для індивідуальних користувачів і платним для фахових команд.[18]

Figma працює на будь-якій операційній системі, яка запускає веб-браузер. З Figma можна використовувати комп'ютери Mac, комп'ютери з Windows, комп'ютери з Linux і навіть Chromebook. Це єдиний інструмент розробки такого типу, який робить це, і в магазинах, які використовують апаратне забезпечення під керуванням різних операційних систем, кожен може ділитися, відкривати та редагувати файли Figma.

У багатьох організаціях дизайнери використовують комп'ютери Mac, а розробники комп'ютери з ОС Windows. Figma допомагає об'єднати ці групи. Універсальна природа Figma також запобігає роздратування PNG-понгу (де оновлені зображення переміщаються між дисциплінами команди дизайнерів). У Figma немає потреби в механізмі посередництва, щоб зробити роботу з дизайну доступною для всіх.

Оскільки Figma базується на браузері, команди можуть співпрацювати так само, як і в Google Docs. Люди, які переглядають і редагують файл, відображаються у верхній частині програми у вигляді круглих аватарів. У кожної людини також є іменований курсор, тому відстежити, хто що робить,

легко. Натискання на чужий аватар збільшує масштаб до того, що вони переглядають у цей час.

Спільна робота з файлами в режимі реального часу допомагає пом’якшити «зміщення дизайну», яке визначається як неправильне тлумачення або відхилення від узгодженого дизайну. Зміна дизайну зазвичай відбувається, коли ідея задумана та швидко реалізована під час виконання проекту. На жаль, це часто призводить до відхилення від встановленої конструкції, що викликає тертя і повторну роботу.

Використовуючи Figma, керівник дизайну може перевірити, що проектує команда в режимі реального часу, просто відкривши спільний файл. Якщо дизайнер якимось чином неправильно інтерпретує коротку або користувацьку історію, ця функція дозволяє дизайнеру втрутитися, виправити курс і заощадити незліченну кількість годин, які інакше були б витрачені даремно. (Для порівняння, команди, які використовують Sketch, не можуть відразу визначити, чи не збиваються дизайнери.)

Figma використовує Slack як канал зв'язку. Коли канал Figma створюється в Slack, будь-які коментарі або правки дизайну, внесені в Figma, передаються команді. Ця функціональність є важливою під час проектування в реальному часі, оскільки зміни до файлу Figma оновлять кожен інший екземпляр, куди вбудований файл (потенційний головний біль для розробників). Зміни в макеті, гарантовані чи ні, негайно перевіряються, і канал зворотнього зв’язку працює в прямому ефірі.

Figma також ділиться фрагментами коду для вставки в реальному часі, щоб вставити iFrame в інструменти сторонніх розробників. Наприклад, якщо Confluence використовується для відображення вбудованих файлів макетів, ці файли не «оновлюються» шляхом збереження файлу Figma — ці вбудовані файли являються файлом Figma.Якщо будь-хто в Figma вніс зміни до макета, цю зміну можна побачити наживо у вбудованому макеті Confluence.

До Figma використовувалися кілька інших інструментів для полегшення обміну макетами дизайну та оновленнями. Цикл ітерації представляв собою

серію оновлень файлів вперед і назад, щоб команди могли переглянути та впровадити поточний дизайн.

Після Figma сторонні інструменти більше не потрібні (але їх можна використовувати за бажанням). Оскільки Figma обробляє функціональні можливості сторонніх інструментів, описаних раніше, є лише один крок у цьому процесі — перехід від ескізів до Figma, і всі групи отримають найновіші макети. «Передачі» в найточнішому значенні цього слова не існує.

3.2 Реалізаціяоб’єктів

Об’єкт Canvasвідповідає за розташування всього користувацького інтерфейсу (рис 3.2). Містить у собі об’єкт background.

Присутні такі елементи:

- Canvas– основний компонент, який відповідає за основніналаштування.
- CanvasScaler – компонент, який відповідає за розширення та адптивність користувацького інтерфейсу до різних моніторів.
- RectTransform– основний компонент, який відповідає за розташування об’єкту по різним осям.

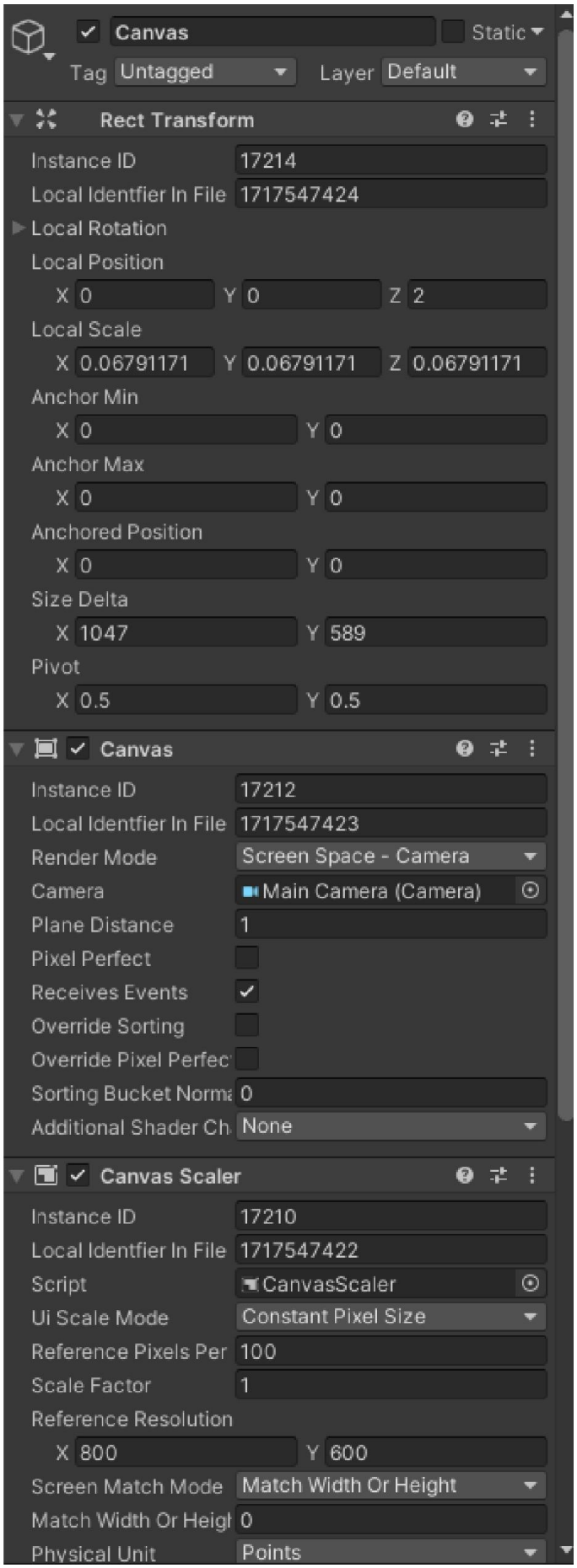

Рисунок3.2 – Об’єкт Canvas

Об’єкт Background (рис 3.5) відповідає за задній фон канвасу, відкритого у конкретній сцені. Має наступні компоненти:

- RectTransform – основний компонент, який відповідає за розташування об’єкту по різним осям.
- CanvasRenderer – основний елемент, який відповідає за параметри рендеру зображення.
- Image – компонент, в якому містяться налаштування графічного зображення.
- DefaultUIMaterial – елемент, який відповідає за присутні у компонента характеристики та властивості матеріалу.
- LayoutProperties – елемент, в якому містяться дані щодо параметрів розширення зображення.

Об’єкт Player (рис 3.4)відповідає за характеристики та параметри гравця, за якого грає користувач, який в нашому випадку виступає в ролі вірусу системи.

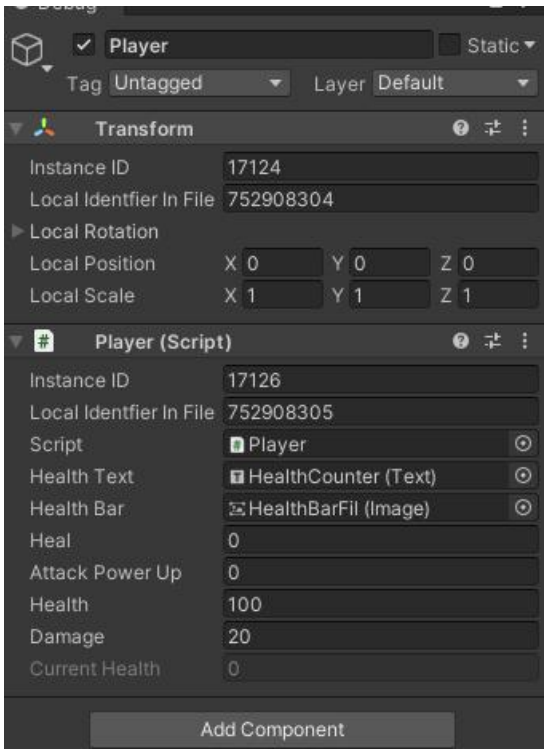

Рисунок – 3.4 Об’єкт Player

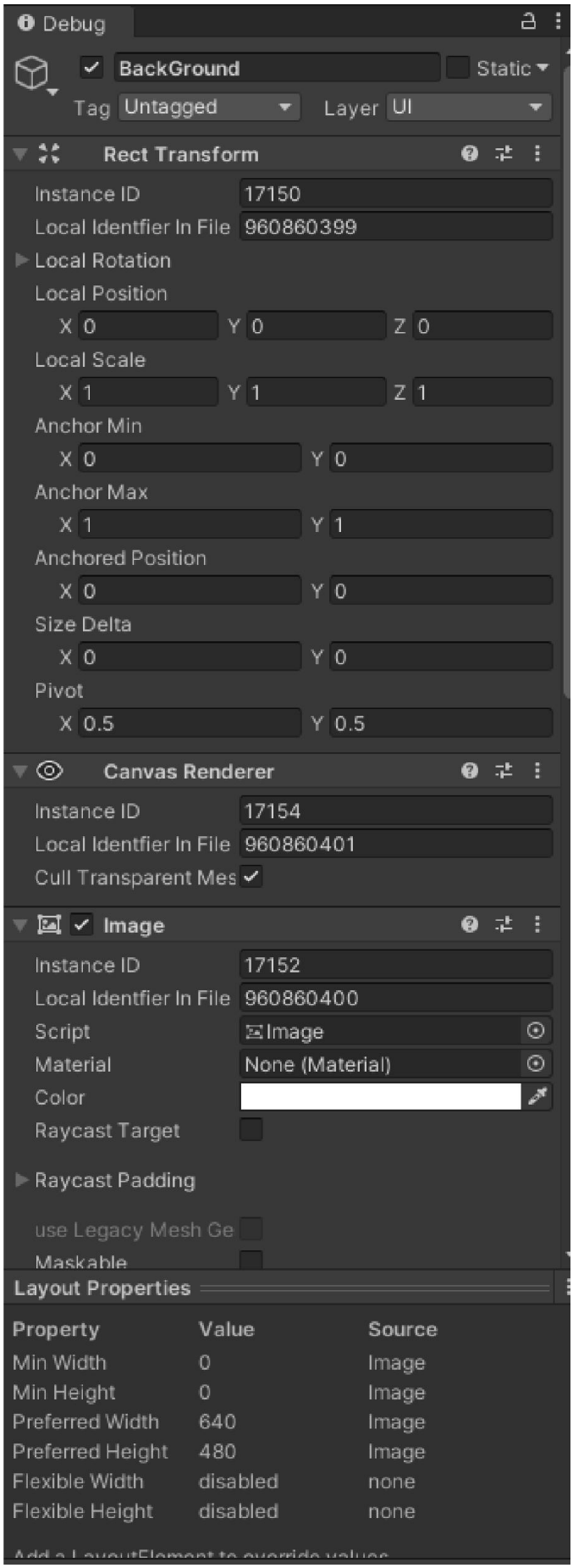

Рисунок 3.5 – Об’єкт Background

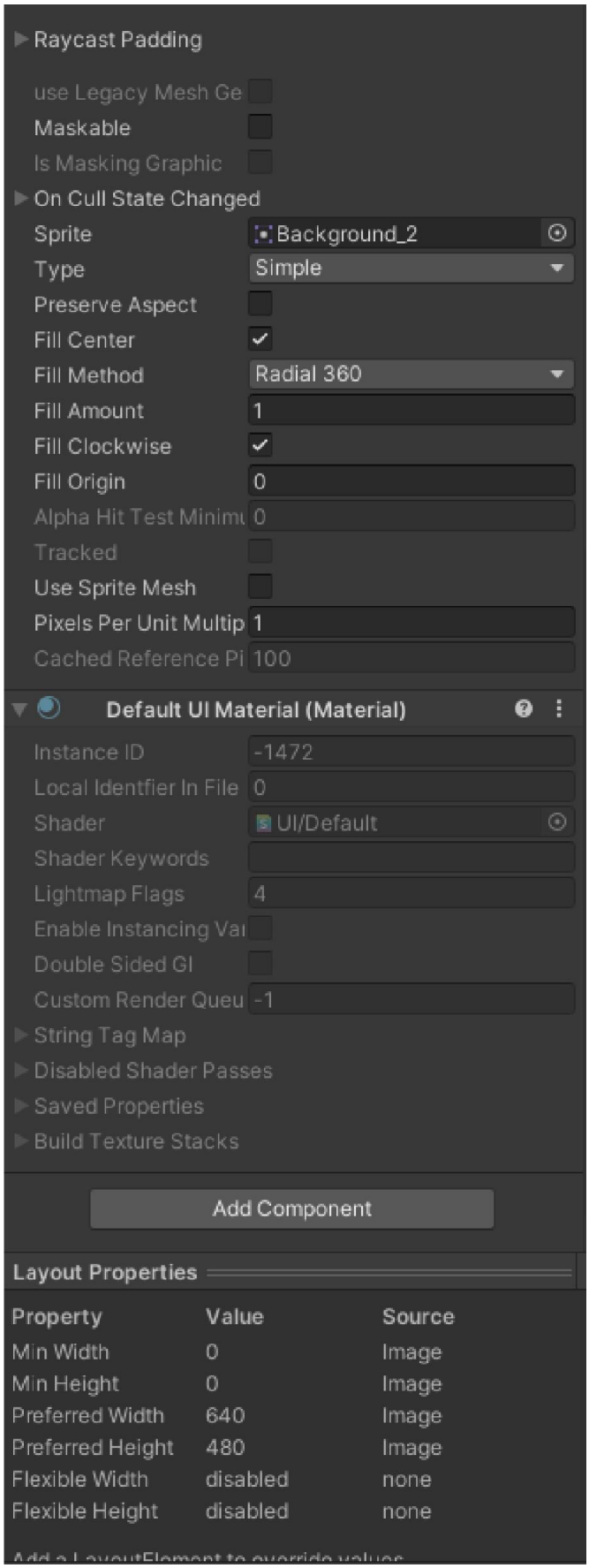

Рисунок 3.5 – Об’єкт Background (продовження)

Об’єкт типу Buttonявляє собою типову кнопку графічного інтерфейсу, натискання на яку запускає деяку закріплену за конкретною кнопкою інтеракцію (здійснення якогось скрипту, запуск процедури чи інш). Має наступні компоненти:

- Button – основний компонент, який відповідає за різні властивості кнопки в залежності від її статусу Кнопка може мати статуси такі як: активна, неактивна, натиснута, наведена курсором користувача та інш. Можлива також зміна анімації при взаємодії з кнопкою.
- Image – компонент, який відповідає за текстуру (зображення) кнопки. Присутні як налаштування самої текстури, так і її матеріалу та параметрів зображення (DefaultUIMaterial).

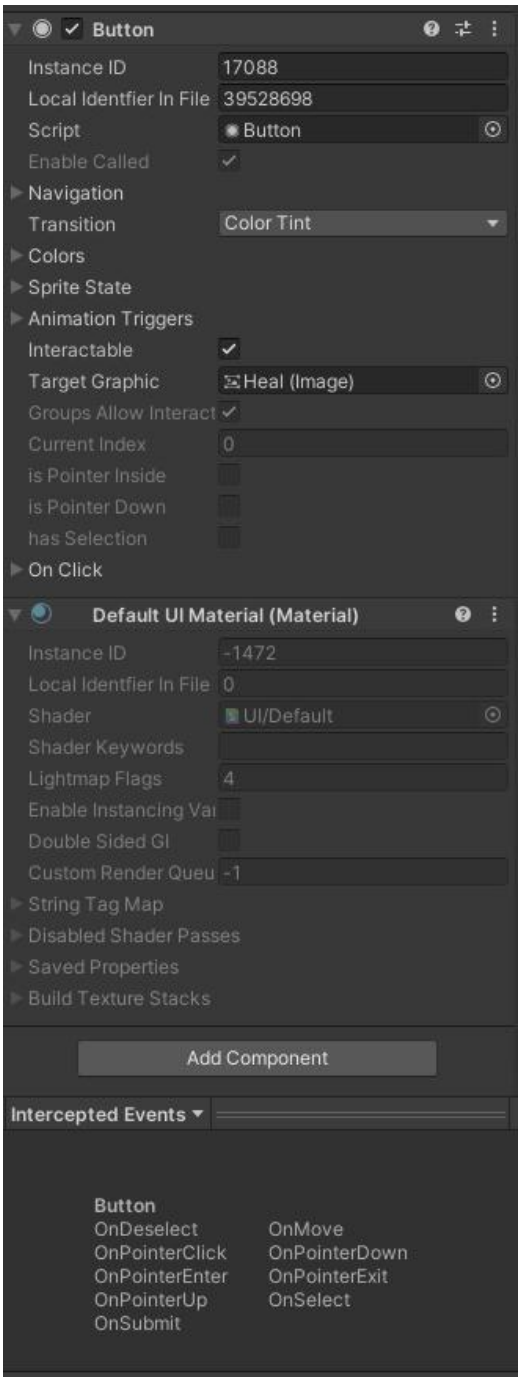

Рисунок 3.6 – Об’єкт типу Button

Об’єкт MainCamera (Рис 3.7)відповідає за налаштування основної камери, яка закріплена за гравцем. Має наступні елементи:

- Transform – елементи, який відповідає за розташування об’єкта на сцені.
- Camera – основний елемент, який відповідає за базові налаштування камери.

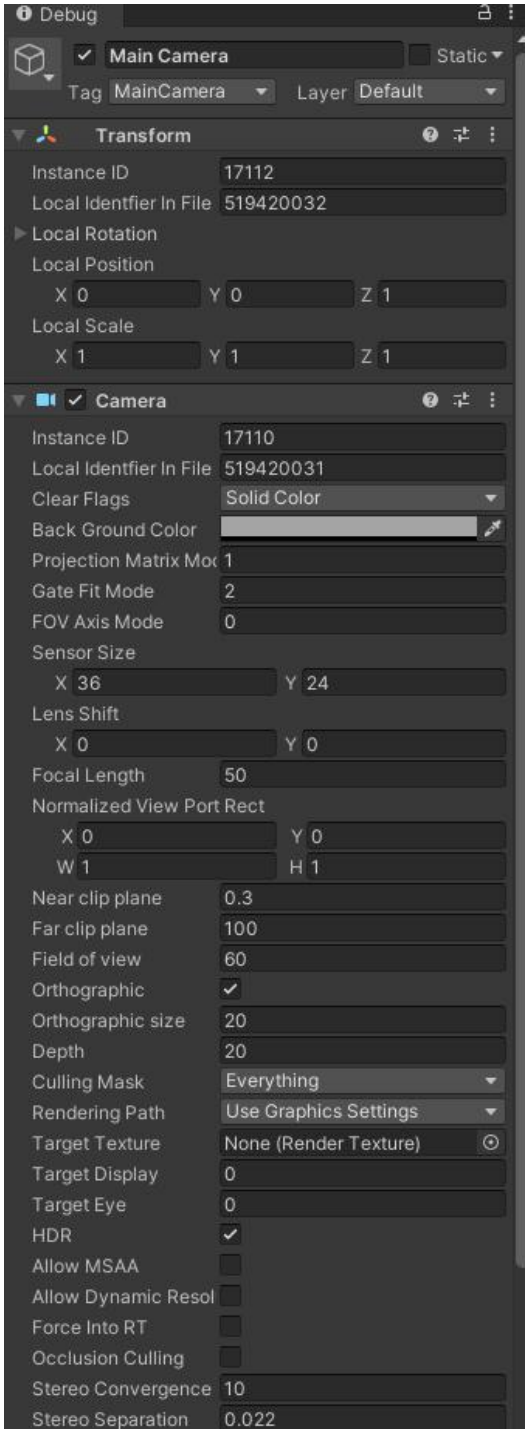

Рисунок 3.7 – Об’єкт MainCamera

Об’єкт типу StartNode (Рис 3.8) відповідає за налаштування одного з декількох типів шестикутників, які містяться на ігровому полі, в даному випадку – початкова нода ігрового поля.

Об’єкт типу StandartEnemyNode (Рис 3.8) відповідає за налаштування одного з декількох типів шестикутників, які містяться на ігровому полі, в даному випадку – типова нода ворогу.

Об’єкт типу MoveNode (Рис 3.9) відповідає за налаштування одного з декількох типів шестикутників, які містяться на ігровому полі, в даному випадку – типова нода пересування.

- Script – елемент, який відповідає за закріплений за конкретним об’єктом скрипт.
- Name – короткий ідентифікатор об’єкта у системі.
- Health – кількість здоров’я даного об’єкту.
- Attack – кількість здоров’я, яку даний об’єкт забирає у гравця при атаці гравця на об’єкт.
- EnemyModel – елемент, який відповідає за модель об’єкту.
- Статуси Isopen, IsDiscovered, IsRegen – елемент налаштування, який відповідає за присвоєння об’єкту різних станів.

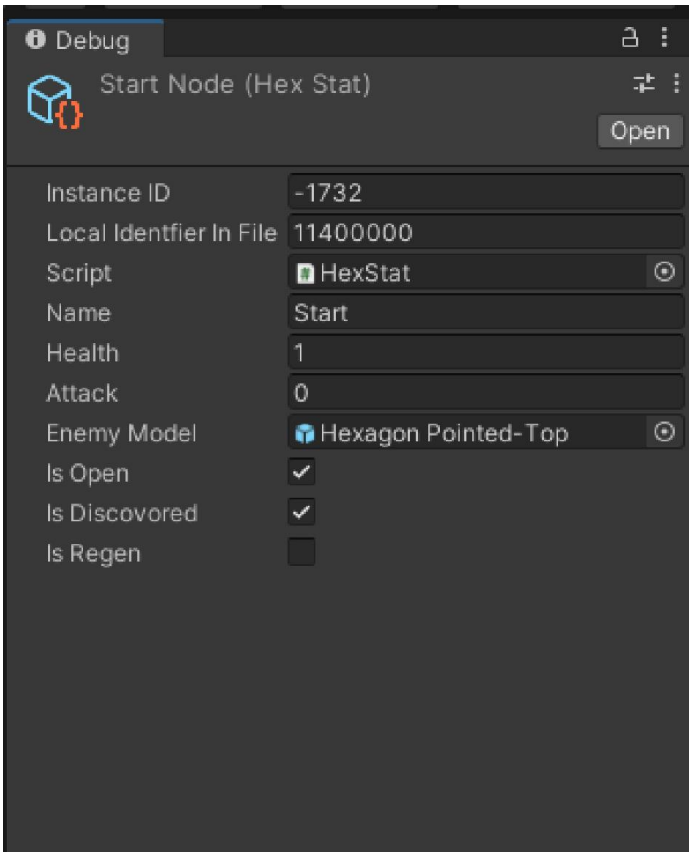

Рисунок 3.8 – Об’єкт Start Node

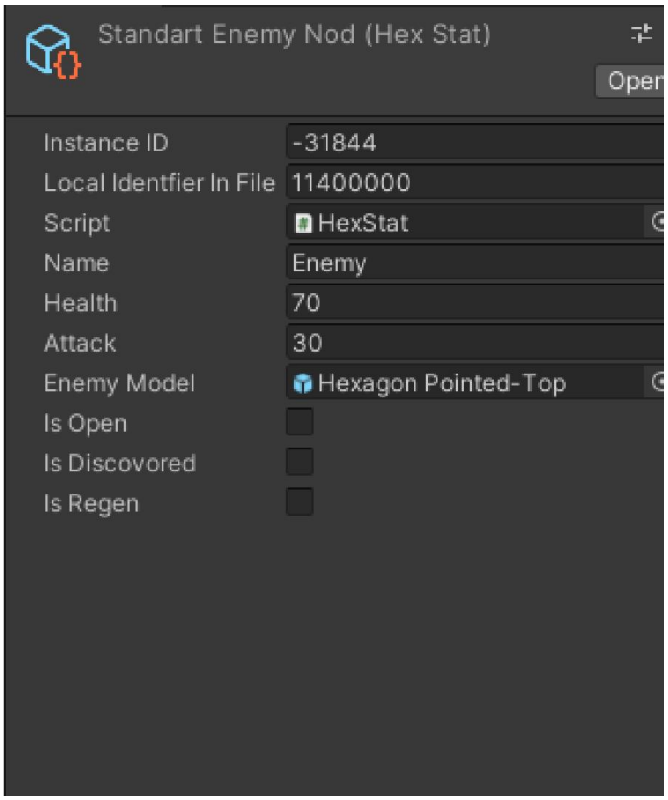

Рисунок 3.9 – Об’єкт Standart Enemy Node

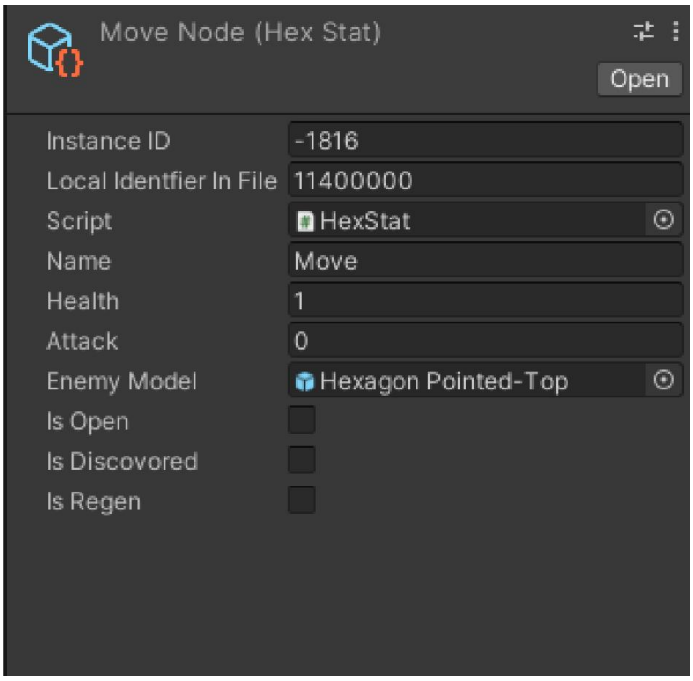

Рисунок 3.10 – Об’єкт MoveNode

Об’єкт EventSystemвідповідає за реалізацію та функціонал взаємодіїгравця з об’єктами на ігровому полі. Має наступні параметри:

- Transform – елемент, який відповідає за розташування об’єкта на сцені.
- EventSystem – основний елемент, який відповідає за налаштування взаємодії об’єктів з гравцем. Містить у собі скрипт Event System.
- InputModule – основний елемент, який відповідає за налаштування моделі вводу та імітує роботу курсора або контролера. Містить у собі однойменний скрипт.

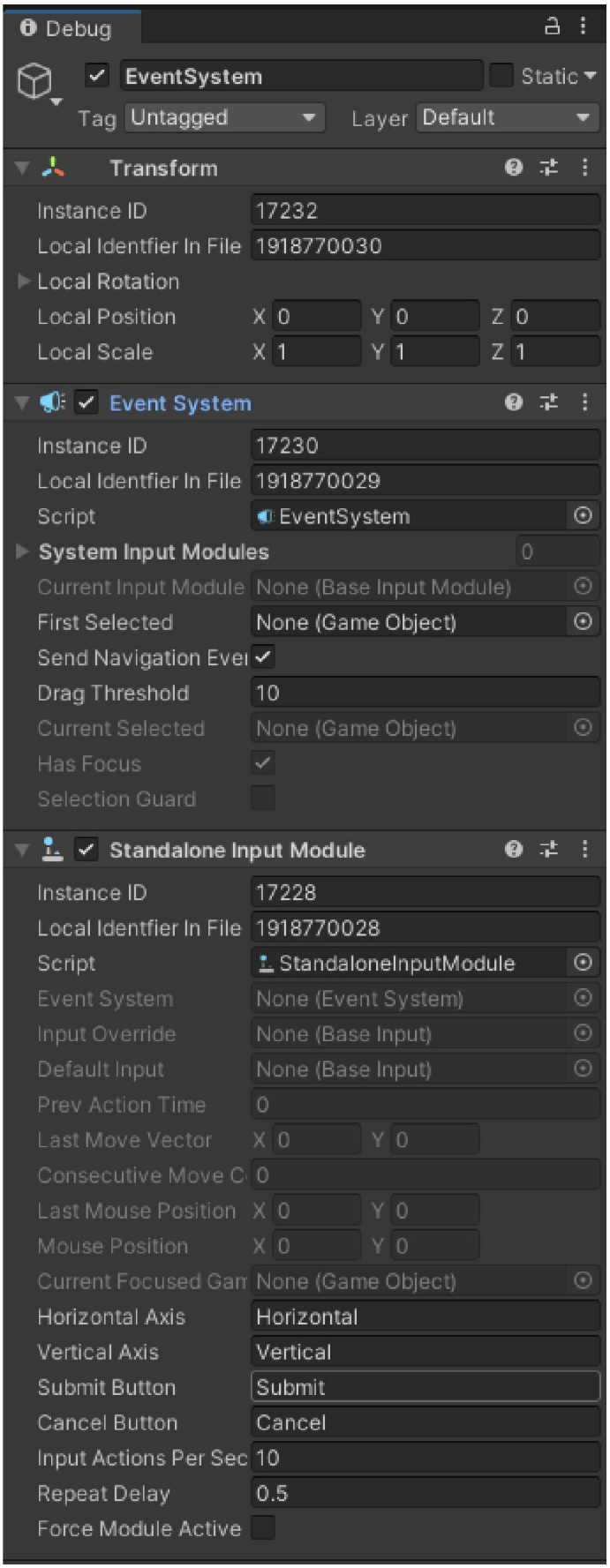

Рисунок 3.11 – Об’єкт EventSystem

Об’єкт GameManager (рис. 3.12)відповідає за налаштування самої гри, в тому числі за розмір поля та кількість наявних на полі нодів того чи іншого типу. Має у собі скрипт HexSpawner, який і несе відповідальність за вказані вище функції.

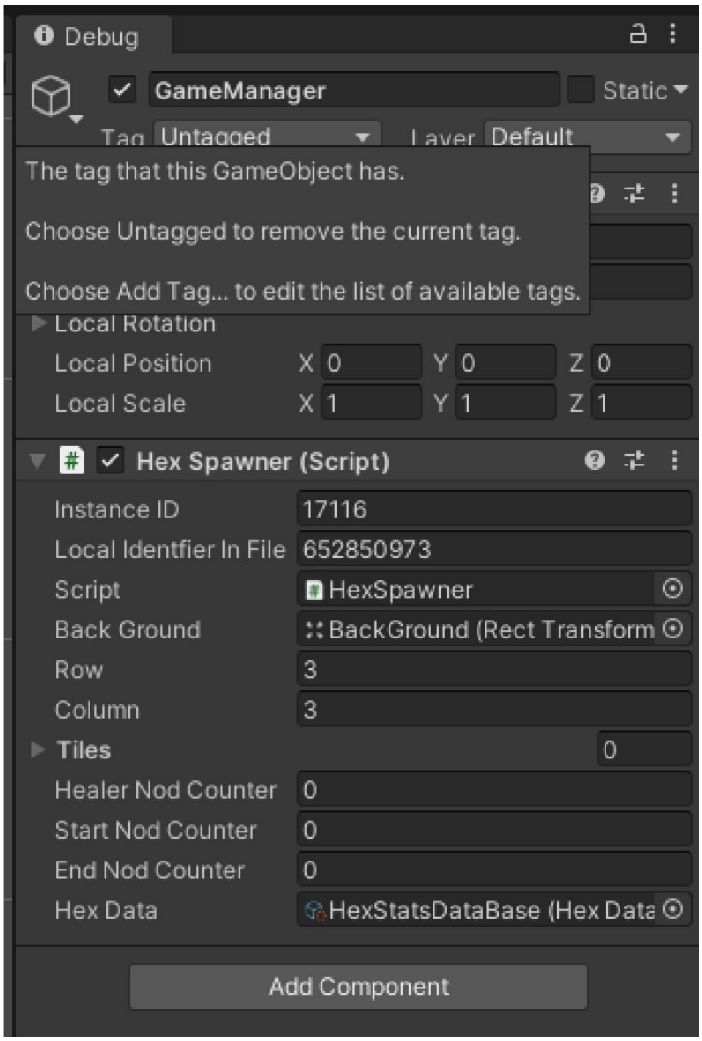

Рисунок 3.12 – Об’єкт GameManager

3.3 Використання штучного інтелекту

Ігри - це не лише розвага. Навчання агента, щоб перевершити гравців-людей та досягти високого рахунку, може навчити нас, як оптимізувати різні процеси у найрізноманітніших та найцікавіших областях. Для регулювання

складності та балансу гри ми використаємо штучний інтелект на базі Q-Learning. Для цього ми реалізуємо глибоке навчання алгоритмом використовуючи Keras поверх Tensorflow. Цей підхід полягає у наданні системним параметрам, пов'язаним з її станом, і позитивній або негативній винагороді, заснованому на її діях. Правила гри відсутні, і спочатку мережа не має інформації про те, що їй потрібно робити. Ціль системи полягає в тому, щоб з'ясувати це і розробити стратегію, щоб максимізувати рахунок або винагороду.

Зміцнення навчання являє собою підхід, заснований на процесі прийняття рішень Маркова для прийняття рішень.У реалізації було використаноглибоке Q-Learning замість традиційного підходу машинного навчання під наглядом. Яка різниця? Традиційні алгоритми ML повинні бути навчені із введенням та «правильною відповіддю», званою метою. Потім система спробує навчитися прогнозувати мету відповідно до нових даних. У цьому прикладі ми не знаємо, які дії найкраще вжити у кожному стані гри, тому традиційний підхід не буде ефективним.

У зміцненні навчання ми передаємо винагороду позитивний чи негативний залежно від дій, вжитих системою, і алгоритм повинен дізнатися, які дії можуть максимізувати винагороду, які слід уникати.

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q-Value — Current Q-Value — Learning rate — Reward — Discount rate — Maximum predicted reward, given new state and all possible actions

Рисунок 3.11 – Демонстративна формула Q-Learning

На загальному рівні алгоритм працює наступним чином:

- Гра починається, і значення Q ініціалізується випадковим чином.
- Система отримує поточний стан,

- На основі s він виконує дію, випадково або на основі своєї нейронної мережі. На першому етапі навчання система часто вибирає випадкові дії, щоб максимізувати дослідження. Пізніше система все більше покладається на свою нейронну мережу.
- Коли II вибирає і виконує дію, система збирає винагороду, Тепер він отримує новий стан штату і він оновлює своє значення Q за допомогою рівняння Беллмана, як згадано вище. Крім того, для кожного ходу він зберігає вихідний стан, дію, стан, досягнутий після виконання цієї дії, отриману винагороду і чи закінчилася гра чи ні. Ці дані пізніше відбираються на навчання нейронної мережі. Ця операція називається Відтворити пам'ять.
- Ці дві останні операції повторюються доти, доки не буде виконано певну умову.

Глибока нейронна мережа оптимізує відповідь (дія) на конкретний вхід (стан), намагаючись максимізувати винагороду. Значення висловлення того, наскільки хороший прогноз, називається втратою. Завдання нейронної мережі полягає в тому, щоб мінімізувати втрати, зменшуючи тоді різницю між реальною метою та прогнозованою.

3.4 Фінальний продукт

Фінальний продукт представляє собою розроблений додаток-головоломку під назвою GEXAHackingPrototype. Були реалізовані наступні сцени гри: головне меню, меню налаштувань, ігровий рівень, меню результатів, меню credits. У головному меню доступні наступні дії:

- Запуск гри;
- Перехід в меню налаштувань;
- Перехід до меню credits.;
- Вихід із додатку.

В процесі самої гри користувачеві потрібно провести злом системи захисту модуля. Для цього у нього є свій вірус з параметрами сили та атаки. Пересування по ігровому полю здійснюється за допомогою «гексів», «нодів», - правильних шестикутників, які можуть приймати наступні стани:

- Початковий – нода, на якій гравець починає свій рух по системі.
- Фінальний вузол – нода, яка містить у собі антивірус, при подоланні якого гра завершується перемогою.
- Вузол посилення вірусу – нода, при відкритті якої вірус гравця може використати одне з представлених в грі посилень.
- Вузол антивірусу – нода, при відкритті якої пересування на суміжні ноди становиться неможливим до повного подолання вузлу антивірусу. Сам вузол має параметри атаки за здоровья.
- Також всі ноди діляться на «відкриті» та «закриті». Закриті ноди – це ті, до яких гравець ще не дібрався, відповідно він не знає, що саме міститься у ноді. Відкриті – це ті, які гравець вже відкрив, та зміст якої вже відомий.

Для регулювання та балансу складності гри був залучений штучний інтелект на базі Q-Learning, але останній все ще знаходиться на стані розвитку, відповідно, складність гри ще може бути відрегульована, але вже на даному етапі агент штучного інтелекту може завершити гру з результатом близько 600 балів, а на старті не міг набрати навіть 200.

РОЗДІЛ 4
ТЕСТУВАННЯ

4.1 Вибір типу тестування

Тестування – це останній на черзі етап процесу розроблення програмного забезпечення перед вводом його в експлуатацію, воно необхідно для контролю якості кінцевого продукту. Так як повне тестування навіть малого за масштабом проекту буде складатися з майже безкінечних тестів, тому протестовані будуть лише основні функції продукту. Так для перевірки працездатності додатку було проведено функціональне тестування, а для перевірки на проблеми в використанні – юзабіліті-тестування [5].

4.2 Тест план

Для проведення успішного тестування складемо план тестування та визначимо функції, які потрібно протестувати:

- Зміна налаштувань додатку через головне меню;
- Переміщення користувача по ігровому полю;
- Взаємодія користувача з нодами різних типів;
- Вплив посилювачів на характеристики гравця.

4.3Функціональне тестування

Функціональне тестування – це тестування ПЗ на предмет реалізованих вимог, тобто здатності ПЗ вирішувати поставлені задачі.

Тест №1. Зміна налаштувань додатку через головне меню.

Вхідні дані: користувач рухає повзунок корегування гучності.

Очікуваний результат: зміна рівня гучності.

Отриманий результат: відповідає очікуваному.

Тест пройдений успішно.

Тест № 2. Переміщення гравця по полю

Вхідні дані: користувач натискає на суміжну з ним ноду

Очікуваний результат: суміжна нода, на яку натиснув користувач, змінює свій статус на «відкрита», після чого суміжні до неї ноди стають можливими для подальшого пересування.

Отриманий результат: відповідає очікуваному.

Тест № 3. Взаємодія з нодою типу «посилювач вірусу».

Вхідні дані: користувач відкриває суміжну з ним ноду, у якій знаходиться посилювач, натискає на цей посилювач.

Очікуваний результат: посилювач зникає з ігрового поля та додається у список наявних у користувача посилювачів.

Одержаний результат: відповідає очікуваному.

Тест № 4. Взаємодія з нодою типу «антивірусний вузол».

Вхідні дані: користувач відкриває суміжну з ним ноду, у якій знаходиться антивірус.

Очікуваний результат: суміжні з антивірусним вузлом ноди стають неможливими для пересування, ефект антивірусної ноди впливає на характеристики вірусу гравця.

Одержаний результат: відповідає очікуваному.

Тест № 5. Проведення атаки на ноду типу «антивірусний вузол».

Вхідні дані: після знаходження ноди типу «антивірусний вузол» гравець натискає на неї, тим самим атакуючи її.

Очікуваний результат: кількість здоров’я ноди зменшується на число, яке дорівнює силі вірусу гравця, а кількість здоров’я гравця зменшується на число, яке дорівнює силі антивірусного вузла.

Одержаний результат: відповідає очікуваному.

Тест № 6. Використання наявного у гравця посилювача.

Вхідні дані: користувач натискає на наявний у нього посилювач типу «бомба», після чого проводить атаку на антивірусний вузол.

Очікуваний результат: атака не коштує користувачеві здоров’я, а антивірусний вузол знешкоджується моментально.

Одержаний результат: відповідає очікуваному.

4.4 Юзабіліті-тестування

Юзабіліті-тестування – це метод оцінки зручності продукту при використанні, який заснований на залученні користувачів в якості випробувачів та сумуванні отриманих від них висновків [10].

До проходження тесту були залучені шість користувачів. Їм було запропоновано вирішити наступні задачі:

- Змінити налаштування додатку
- Запустити рівень гри
- Пройти гру
- Вийти з гри за допомогою меню паузи

Всі поставлені задачі були виконані тестувальниками без проблем. Користувачі відмітили інтуїтивність управління та приємний дизайн. Юзабіліті-тестування пройшло успішно.

4.5 Інструкція впровадження

Після розробки та тестування продукту його можна розгорнути на будь-якому комп’ютері з упакованого файлу RAR/7z. чи розповсюдити за допомогою мережі інтернет через посередників, а саме середовищ для продажу ігор та симуляторівабо інтернет-сайтів.

ВИСНОВКИ

Мета дипломної роботи полягала в розробленні додатку-головоломки на базі середовища Unity. Під час аналізу доступних джерел було проведено дослідження наявних програмних засобів рішення поставленої задачі.

Був проведений огляд існуючих ігрових середовищ та популярних програмних засобів розробки. В ході аналізу було проведено їх порівняння та вибір найкращого для даної дипломної роботи. Вибір засобів розробки проходив за наступними параметрами: доступність та функціонал.

При аналізі існуючих програмних рішень було проведене їх порівняння виділення недоліків та переваг. В ході аналізу стало ясно, що подібні додатки повинні бути простими в своїх механіках та складними в плані покарання за помилку при неправильному виборі. Простота допомагає користувачеві швидко розібратися в правилах гри, а покарання за помилку змушує його думати про кожен наступний крок. Висока складність також покращує інтерес до гри з боку аудиторії.

На основіотриманої в ході дослідження інформації, було вирішено розробити прототип двомірного додатку-головоломки для одного гравця на ігровому движку Unity. Таке рішення було ухвалено з кількох причин:

1)двовимірна графіка, на відміну від тривимірної легше у створенні;

2) з ігрової механіки, додаток жанру головоломка простіше реалізується;

3) ігровий движок Unity поширюється безкоштовно і дозволяє розробляти програми мовою програмування C#. Після вибору засобів розробки було розпочато вивчення Unity, а також розробка самого проекту. У ході розробки було вивчено ігровий двигун Unity і були придбані необхідні знання та вміння, а саме:

- створення сцен;
- створення та написання скриптів;
- налаштування об'єктів;
- створення UI;

• компіляція проекту.

Освоєння середовища розробки Unity несе не маловажний характер, такяк у світі індустрія розробки ігор дедалі більше поширюється у суспільстві. Ігри перестали бути лише предметом для розваг, і тепер використовуються і в інших областях, наприклад, у науці чинавчання користувачів. Тому розвиток у цьому напрямі можнавважати одним із найважливіших у суспільстві.

У ході реалізації проекту було виконано такі завдання:

1) вивчені особливості та стан комп'ютерної промисловості України;

2) обрані жанр, вид та платформа для комп'ютерної гри;

3) розроблено сценарій та концепцію основних елементів;

4) обрано та вивчено засіб реалізації;

5) підготовлено необхідні для гри елементи;

6) реалізовано прототип гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sherwani N.A. Algorithms for VLSI Physical Design Automation. Third Edition, Kluwer Academic Publisher, USA, 2013.

2. Scolastici C. – Unity 2D Game Development Cookbook, USA, 2014. – P. 6-30.

3. Навчання машин та штучний інтелект: конспект лекцій з дисципліни «Технлогії програмування» ч. 2 «Навчання машин та штучний інтелект» для студентів напряму підготовки 163 «Біомедична інженерія» всіх форм навчання / уклад. Верескун М.В.. – Маріуполь : ДВНЗ «ПДТУ», 2019. – 84 с.

4. Stephen Marsland. Machine Learning: An AlgorithmicPerspective / Stephen Marsland. – 2015. – 452 p.

5. Kuliev E.V. Dukkardt A.N, Kureychik V.V. Legebokov A.A. Neighborhood Research Approach in Swarm Intelligence for Solving the Optimization Problems // Proceedings of IEEE EastWest Design & Test Symposium – (EWDTS'2014) Kiev, Ukraine, September 26–29, 2014. – P. 112-115.

6. Martello, S. Knapsack Problems – Algorithms and Computer Implementations / S. Martello, P. Toth. – Chichester: John Wiley & Sons, 1990.

7. Milenkovic, V.J. Translational polygon containment and minimal enclosure using mathematical programming / V.J. Milenkovic, K. Daniels // International Transactions in Operational Research. – 1999. – Vol. 6, № 5. – P. 525-554.

8. Генетичні алгоритми / Л.А. Гладков, В.В. Курейчик, В.М. Курейчик; під ред.В.М. Курейчика. - 2-е изд., Испр. і доп. - М .: ФИЗМАТЛИТ, 2006. - 320 с.

9. Vincent C., Darzi A. Simulation based training. London, Department of Surgical Oncology and Technology, Imperial College, 2012.

10. C# Programming Guide. [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide

11. Flower M. UML Distilled: A BriefGuidetothe Standard ObjectModelingLanguage. – USA: Addison-WesleyPublishingCompany, 2011. – 152 с.

12. Gold J. Object-Oriented Game Development. – UK: Pearson Education Limited, 2014. – 104 с.

13. Gregory J. Game Engine Architecture. – USA: A K Peters/CRC Press, 2010. – 64 с.

14. Kaner C., Bach J., Pettichord B. LessonsLearnedinSoftwareTesting. USA: Wiley, 2001. – 120 с.

15. Tidwell J. DesigningInterfaces. – USA: O'ReillyMedia, 2004. – 278 с.

16. Unity Manual. [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://docs.unity3d.com/Manual/index.html

17. Офіційний сайт Unity 3D. [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://unity.com

18. Вігенс К., Бітті Д. Розробка вимог до програмного забезпечення. – М.: Російська редакція, 2014. – 736 с.

19. Гама Э., Хелм Р., Джонсон Р., Влиссидс Д. Прийоми об'єктно-орієнтованого програмування. Патерни проектування. – СПб, 1994. – 395 с.

20. Ларман К. Застосування UML 2.0 та шаблонів проектування. Введення в об'єктно-орієнтований аналіз, проектування та ітеративну розробку. – М.: Віль'ямс, 2013. – 736 с.

21. Майерс Г., Баджет Т. Мистецтво тестування програм. – М.: Віль'ямс, 2012. – 272 с.

22. Офіційний сайт Visual Studio. [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://visualstudio.microsoft.com/ru/vs/

23. Рамбо Дж. UML 2.0. Об'єктно-орієнтоване програмування та розробка. – СПб, 2007. – 244 с.

ДОДАТОК А

ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

1. Компонент ResourceBinding (ResourceBinding.cs)

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


                    public class ResuerceBinding : MonoBehaviour
                    {
                    public HexSpawner HexSpawner;
                    public Player player;
                    public HexDataBase hexData;
                    public HexEnemy enemy;


                    public void SetEnemyComponent(List<GameObject> objects)
                    {
                    foreach (GameObject gameObjects in HexSpawner.tiles)
                    {
                    var hexObjName = gameObjects.name;
                    enemy = gameObjects.GetComponent<HexEnemy>();
                    var hexFound = hexData.GetHexStat(hexObjName);
                    enemy.health = hexFound.health;
                    enemy.attack = hexFound.attack;
                    enemy.isOpen = hexFound.isOpen;
                    enemy.isDiscovored = hexFound.isDiscovored;
                    enemy.isRegen = hexFound.isRegen;
                    }
                    }
                    }
```

2. КомпонентStat (Stat.cs)

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


[Serializable]
public class Stat
{
```

```
    public int baseValue;


    public int GetValue()
    {
        return baseValue;
    }
}
```

3. Компонент HexEnemy (HexEnemy.cs)

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


                    public class HexEnemy : MonoBehaviour
                    {
                        public int health;
                        public int attack;
                        public bool isOpen;
                        public bool isDiscovored;
                        public bool isRegen;


                    }
```

4. Компонент HexSpawner (HexSpawner.cs)

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


          public class HexSpawner : MonoBehaviour
          {
              public Transform BackGround;
              public int row;
              public int column;
              public List<GameObject> tiles;
              public static HexSpawner instance;
              public int HealerNodCounter;
              public int StartNodCounter;
              public int EndNodCounter;
```

```
public HexDataBase hexData;
public ResuerceBinding resuerce;
int EntranceCounter = 0;


void Start()
{
    instance = this;
    CreateHexTileMap(hexData);
    resuerce.SetEnemyComponent(tiles);
}


void CreateHexTileMap(HexDataBase hexData)
{
    for (int x = 0; x <= row-1; x++)
    {
        for (int y = 0; y <= column-1; y++)
        {
            var hexStat = SpawnChose(hexData);
            var visual = Instantiate(hexStat.enemyModel);
            visual.name = hexStat.name;
            visual.transform.SetParent(BackGround, true);
            visual.transform.localScale = 20 * Vector2.one;
            Vector2 pos;
            if (y % 2 == 0)
            {
                pos = new Vector2(x*20,y*20 * 3 / 4);
            }
            else
            {
                pos = new Vector2(x * 20 + 10 + 1, y * 20 * 3 / 4);
            }
            visual.transform.localPosition = pos;
            tiles.Add(visual);
        }
    }
Debug.Log(tiles.Count);
}
HexStat SpawnChose(HexDataBase hexData)
{
    var lastEntrance = row * column-1;
    System.Random rnd = new System.Random();
    if (EntranceCounter == 0)
    {
        var hexStat = hexData.allHexTypes.Find(x=>x.isStart==true);
        EntranceCounter++;
        return hexStat;
```

```
            }
            else if (EntranceCounter == lastEntrance)
            {
                var hexStat = hexData.allHexTypes.Find(x => x.isEnd == true);
                EntranceCounter++;
                return hexStat;


            }
            else
            {
                var hexToSpawn = hexData.allHexTypes.FindAll(x => x.isEnd == false &&
        x.isStart == false);
                var hexDataToSpawn = rnd.Next(0, hexToSpawn.Count);
                var hexStat = hexToSpawn[hexDataToSpawn];
                EntranceCounter++;
                return hexStat;
            }
        }


        }
```

5. Компонент HexStat (HexStat.cs)

```
using
System;
        using UnityEngine;
        using UnityEngine.UI;


        [CreateAssetMenu (fileName = "NewHexStat", menuName = "HexStat")]
        public class HexStat : ScriptableObject
        {
            public int health;
            public int attack;
            public GameObject enemyModel;
            public bool isOpen;
            public bool isDiscovored;
            public bool isRegen;
            public bool isStart;
            public bool isEnd;


            public int Health => health;
            public int Attack => attack;
            public bool IsOpen => IsOpen;
```

```
            public bool IsDiscovored => IsDiscovored;
            public bool IsRegen => isRegen;
            public bool IsStart => isStart;
            public bool IsEnd => isEnd;
        }
```

6. Компонент HexUI (HexUI.cs)

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


                public class HexUI : MonoBehaviour
                {
                    public Text healthText;
                    public Text attackText;
                }
  using System.Collections;
                                                    using System.Collections.Generic;
                                                    using UnityEngine;
                                                    using UnityEngine.UI;


                                                    public class IneractionManager :
                                                    MonoBehaviour
                                                    {




                                                        public void Attack()
                                                        {
                                                        }


                                                    }
```

7. Компонент InteractionManager (InteractionManager.cs)

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

```
public class IneractionManager : MonoBehaviour
{



    public void Attack()
    {
    }


}
```

8. Компонент HexStatDataBase (HexStateDataBase.meta)

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!114 &11400000
MonoBehaviour:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 0}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: d165c5c758e6c8e4f91f3aac22043aa9, type: 3}
  m_Name: HexStatsDataBase
  m_EditorClassIdentifier:
  allHexTypes:
  - {fileID: 11400000, guid: 5f3bb2db026649347bff75c6407e56ac, type: 2}
  - {fileID: 11400000, guid: 1415d62031ba38340b709e0427c6dc3d, type: 2}
  - {fileID: 11400000, guid: bf535a5afd32c924d9b28866ddc76e6c, type: 2}
  - {fileID: 11400000, guid: 1d341db255a9f0d4c9586bfbf2abe47f, type: 2}
  - {fileID: 11400000, guid: fae8efc0a2ed6cc4492bc8f59a3df615, type: 2}
  - {fileID: 11400000, guid: 34d51ec6603ac1f4e9deb81e9eb77ba2, type: 2}
```

9. Компонент EndNode (EndNode.asset)

```
%YAML
1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!114 &11400000
MonoBehaviour:
  m_ObjectHideFlags: 0
```

```
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 0}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: 232582ac8a2fa164c97d2e9bb59862fd, type: 3}
  m_Name: EndNode
  m_EditorClassIdentifier:
  name: End
  health: 90
  attack: 20
  enemyModel: {fileID: 5310724791910437545, guid: 10e24e14035f5a846bf0f07d76b72ea1, type:
3}
  isOpen: 0
  isDiscovored: 0
  isRegen: 0
```

10.Компонент StartNode (StartNode.asset)

```
%YAML
1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!114 &11400000
MonoBehaviour:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 0}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: 232582ac8a2fa164c97d2e9bb59862fd, type: 3}
  m_Name: StartNode
  m_EditorClassIdentifier:
  name: Start
  health: 0
  attack: 0
  enemyModel: {fileID: 5310724791910437545, guid: 10e24e14035f5a846bf0f07d76b72ea1, type:
3}
  isOpen: 1
  isDiscovored: 1
  isRegen: 0
```

11.Компонент MoveNode (MoveNode.asset)

```
%YAML
```

```
1.1
        %TAG !u! tag:unity3d.com,2011:
        --- !u!114 &11400000
        MonoBehaviour:
          m_ObjectHideFlags: 0
          m_CorrespondingSourceObject: {fileID: 0}
          m_PrefabInstance: {fileID: 0}
          m_PrefabAsset: {fileID: 0}
          m_GameObject: {fileID: 0}
          m_Enabled: 1
          m_EditorHideFlags: 0
          m_Script: {fileID: 11500000, guid: 232582ac8a2fa164c97d2e9bb59862fd, type: 3}
          m_Name: MoveNode
          m_EditorClassIdentifier:
          name: Move
          health: 1
          attack: 0
          enemyModel: {fileID: 5310724791910437545, guid: 10e24e14035f5a846bf0f07d76b72ea1, type:
        3}
          isOpen: 0
          isDiscovored: 0
          isRegen: 0
```

12. Компонент SecurityNode (SecurityNode.asset)

```
%YAML
1.1
        %TAG !u! tag:unity3d.com,2011:
        --- !u!114 &11400000
        MonoBehaviour:
          m_ObjectHideFlags: 0
          m_CorrespondingSourceObject: {fileID: 0}
          m_PrefabInstance: {fileID: 0}
          m_PrefabAsset: {fileID: 0}
          m_GameObject: {fileID: 0}
          m_Enabled: 1
          m_EditorHideFlags: 0
          m_Script: {fileID: 11500000, guid: 232582ac8a2fa164c97d2e9bb59862fd, type: 3}
          m_Name: SecurityNod
          m_EditorClassIdentifier:
          name: Def
          health: 90
          attack: 20
          enemyModel: {fileID: 5310724791910437545, guid: 10e24e14035f5a846bf0f07d76b72ea1, type:
        3}
          isOpen: 0
          isDiscovored: 0
```

```
  isRegen: 0
```

13.Компонент OnClickEnemy (OnClickEnemy.asset)

```
%YAML
1.1
        %TAG !u! tag:unity3d.com,2011:
        --- !u!114 &11400000
        MonoBehaviour:
          m_ObjectHideFlags: 0
          m_CorrespondingSourceObject: {fileID: 0}
          m_PrefabInstance: {fileID: 0}
          m_PrefabAsset: {fileID: 0}
          m_GameObject: {fileID: 0}
          m_Enabled: 1
          m_EditorHideFlags: 0
          m_Script: {fileID: 0}
          m_Name: onClickEnemy
          m_EditorClassIdentifier: Assembly-CSharp::GameEvent
```