Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

Розробка ігрового проекту на базі Unreal Engine 4

Студента 4 курсу, 7 групи,
напрям підготовки
6.050103 «Програмна інженерія» _____ Коломійця Івана Олеговича

підпис керівника

Науковий керівник
доктор технічних наук,
професор _____ Цюцюра Світлана Володимирівна

підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович

підпис керівника

КИЇВ – 2019

					КНТЕУ 6.050103 07-07.БР			
					Розробка ігрового проекту на базі Unreal Engine 4	Стадія	Аркуш	Аркушів
Зм.	Аркуш	№ докум.	Підпис	Дата		ТС	1	25
Зав. каф.		Криворучко О.В.						
Керівник		Цюцюра. С.В.						
Гарант		Цензура М. О.			Титульна сторінка	Факультет обліку, аудиту та інформаційних систем, 4 курс, 7 група		
Розроб		Коломієць І.О.						

ЗМІСТ

					КНТЕУ 6.050103 07-07.БР		
					Розробка ігрового проекту на базі Unreal Engine 4	Стадія	Аркуш	Аркушів
Зм.	Аркуш	№ докум.	Підпис	Дата		Зміст	2	25
Зав. каф.		Криворучко О.В.						
Керівник		Цюцюра. С.В.						
Гарант		Цензура М. О.			Зміст	Факультет обліку, аудиту та інформаційних систем, 4 курс, 7 група		
Розроб		Коломієць І.О.						

ВСТУП

Сьогодні світ ігрової індустрії дуже різноманітний та захоплюючий. У нас є можливість створювати додатки різних напрямів. Як на мене, найскладнішою та най захоплюючою є сфера гейм розробки. В даному напрямку можна використовувати і не тільки можна, а й потрібно розділи математики, такі як: лінійна алгебра, аналітична геометрія, фізика, розділи механіки та динаміки, складні патерни об’єктно-орієнтованого програмування, алгоритми створення штучного інтелекту. І все це не тільки за для створення розваги, яка може впливати на настрій, психологічне задоволення і на вільний час людини, яка потім буде використовувати ігровий додаток, але й для того, щоб вивчати ігрову розробку і покращувати свої навички програмування. В нашому проекті автор не буде поглиблюватися у такі advanced теми, його задача - на легкому проекті показати всю ту красу розробки ігрового додатку, написання ігрової логіки, використовуючи ігровий рушій Unreal Engine 4.

Основна мета: написання ігрової логіки, використовуючи Blueprints та рушій UE4.

Об’єкт дослідження: етапи розробки ігрового проекту з використанням рушію UE 4.

Предмет дослідження: методи та алгоритми створення програмного забезпечення.

Завдання дослідження: проектування та створення ігрового додатку.

					КНТЕУ 6.050103 07-07.БР		
					Розробка ігрового проекту на базі Unreal Engine 4	Стадія	Аркуш	Аркушів
Зм.	Аркуш	№ докум.	Підпис	Дата		В	3	25
Зав. каф.		Криворучко О.В.						
Керівник		Цюцюра. С.В.						
Гарант		Цензура М. О.			Вступ	Факультет обліку, аудиту та інформаційних систем, 4 курс, 7 група		
Розроб		Коломієць І.О.						

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ РОЗРОБКИ ІГРОВОГО ЗАБЕЗПЕЧЕННЯ

1.1. *Сутність та значення розробки ігрових додатків у програмуванні в цілому*

В наші дні величезна кількість самих різних за інтересами людей частенько грає в комп'ютерні ігри - і не тільки нудьгуючі школярі або прогульники-студенти, зовсім ні! Серед гравців зустрічаються і бізнесмени, і політики, і домогосподарки, і інженери, і художники - в цілому абсолютно різні люди. Всіх їх об'єднує одне - бажання випробувати в віртуальних світах щось нове, досі незвідане, спробувати удачу і отримати насолоду як від ігрового процесу, так і від досягнутих в грі результатів. Комп'ютерні ігри стали справжнім культурним феноменом - виникнувши як нехитрий плід творчої думки програмістів, вони з кожним роком набували все більшої популярності - і розвинулися до того, що стали окремою специфічною спортивною дисципліною - кіберспорт. Попит народжує пропозицію - і ось по всьому світу зросли компанії з розробки ігор, а робота в геймдеві (gamedevelopment'i) стала рожевою мрією для багатьох юних умів, які бажають захоплено створювати улюблені комп'ютерні іграшки. Деякі ігрові серії стали культовими - наприклад, DOOM, Quake, Civilization, HoMM, Fallout, Metal Gear, Драгон Квест, Legend of Zelda, Final Fantasy, TES, CoD, Half-Life, Контра, WoW, Starcraft, Diablo, NFS, GTA . Як мінімум про одну з них напевно чув будь-яка людина, яка хоч раз стикався з комп'ютером.

Спроби створити простенькі ігри на цифрових пристроях робилися ще до початку Другої Світової війни (а в 1947 вже була запрограмована перша електронна гра, монітором для якої служив екран військового радара - це був симулятор ворожих ракет) - проте вважається, що першою комп'ютерною грою стала "ОХО "(" Хрестики нулики "), в поодинці зроблена А.С. Дугласом в далекому 1952 році.

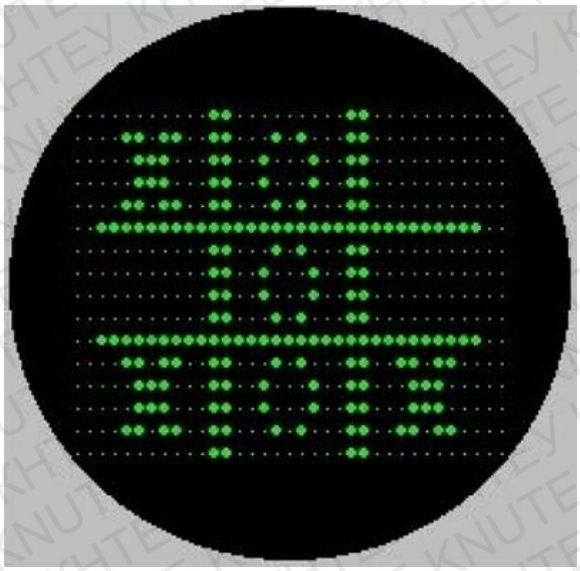

Рис.1.1. - Перша в світі комп'ютерна гра ХОХ (Хрестики-нулики)

Коли на ринку з'явилися IBM-комп'ютери, ігри знову стали ставати популярними. Нові технології, в числі яких - нові звукові чіпи, 16-колірної (а згодом - і 256-кольорової VGA) стандарт дозволили ігроробів створювати більш складні і красиві ігри на персональні комп'ютери. У 1983 з'явився перший інтерактивний мультфільм на ігровому автоматі Dragon's Lair, який працював на великому оптичному диску.

Рис.1.2. - Гра-мультфільм Dragons Lair 1983 року

Ігри стають все більш вимогливим до заліза, з'являються нові консолі, багатоядерні процесори, Інтернет є практично у всіх геймерів, продажі ігор стрімко переходять в цифровий формат, хоч і виходять об'ємні Blu-ray диски, які, проте, не змогли виграти боротьбу зі стрімко дешевшають і збільшуються в розмірі флеш-картами. Під кінець в конструкцію багатьох нових ноутбуків навіть перестали додавати оптичний привід - заради економії місця, зменшення і полегшення корпусу. Приблизно в 2014-2015 році на ринок віртуальних розваг показала ніс технологія VR. Visual Reality почала пробиватися відразу по всіх фронтах - і на ринок портативних мобільних пристроїв, і на ринок стаціонарних пк і ігрових консолей. Але у віртуальній реальності виникло безліч проблем, серед яких недостатня кількість гідних ігрових проектів, що підтримують технологію VR, висока вартість пристроїв для неї і сирі технології взаємодії з віртуальним світом (різні джойстики, сенсори і камери або забезпечували незручне або просто погана взаємодія з віртуальністю, або коштували стільки, що "гра не варта свічок"). Серед VR-шоломів можна виділити: Oculus Rift, HTC Vive, Sony PlayStation VR, Samsung Gear VR і картонний Google Cardboard (два останні - для смартфонів).

1.2. *Основні алгоритми які використовуються у розробці ігор*

В програмуванні ігор використовується багато патернів, шаблонів та алгоритмів програмування. В цьому розділі ми розглянемо найголовніший з них, а саме алгоритм ігрового циклу - Game loop. Ігровий цикл - це квінтесенція прикладу "шаблону в ігровому програмуванні". Він є практично в кожній грі і двох однакових практично немає. І при цьому в НЕ іграх він зустрічається вкрай рідко. Щоб побачити наскільки він корисний, давайте згадаємо минуле. У старі часи комп'ютерного програмування у всіх були бороди, а програми працювали як посудомийні машини: ви завантажували в неї код, натискали кнопку, чекали і забирали результат. Готово. Це називалося пакетним режимом (batch mode): як

тільки робота була зроблена, програма зупинялася. Сучасні програми з графічним призначеним для користувача інтерфейсом якщо зняти з них оболонку разюче нагадують старі текстові квести. Ваш текстовий процесор зазвичай просто сидить і нічого не робить доти, поки ви не натисните будь-яку клавішу. Єдина відмінність тут в тому що замість текстових команд, програма очікує призначеного для користувача введення - натискання миші і клавіш. Але в основі лежить те ж саме що і в текстових квестах, де програма блокується в очікуванні призначеного для користувача введення, що насправді є проблемою. На відміну від інших програм, ігри продовжують працювати навіть коли користувач не надає ніякого введення. Якщо ви будете просто дивитися на екран, гра не зупиниться. Анімації продовжать відтворюватись. Візуальні ефекти танцюють і блищать. А якщо вам не пощастить, монстр продовжить потроху гризти вашого героя. Якщо цей цикл не блокується при введенні - виникає резонне питання: наскільки швидко він крутиться? На кожному кроці ігрового циклу стану гри трохи просувається вперед. З точки зору мешканців ігрового світу це змушує їх час йти вперед.

При цьому у гравця час теж цокає. Якщо ми виміряємо швидкість виконання циклів в одиницях реального часу, ми отримаємо одиницю виміру "кадри в секунду" або FPS. Якщо ігрові цикли змінюються швидко, FPS високий, а гра працює швидко і плавно. Якщо повільно - гра пригальмовує і стає схожою на кіно в уповільненому відтворенні. У нашому примітивному ігровому циклі, який намагається змінюватися якомога частіше, частоту кадрів визначають два чинники. Перший - це скільки роботи потрібно виконувати на кожному кадрі. Складна фізика, купа ігрових об'єктів і деталізована графіка можуть настільки навантажити ваші процесор і відеокарту, що на обробку кадру знадобиться дуже багато часу. Другий фактор - це продуктивність платформи. Швидкі чіпи можуть перемелювати набагато більше коду за той же час. Кількість ядер, відеокарта, дискретний аудіо чіп і

планувальник ОС - все це впливає на кількість дій, які можна встигнути виконати за один тик. У кожній грі другий фактор завжди фіксований. Якщо ви пишете гру для NES або Apple II, ви точно знаєте який процесор у вас буде і можете на нього розраховувати. Все про що вам потрібно піклуватися - це яким обсягом роботи ви його навантажуєте. Старі ігри розроблялися таким чином, щоб виконувана робота дозволяла грі працювати з потрібною швидкістю. Але якби ви спробували пограти в ту ж саму гру на більш швидкому або повільному комп'ютері, сама гра стала б працювати швидше або повільніше. Зараз рідко хто з розробників може дозволити собі розкіш знати на якому залозі буде працювати їх гра. Замість цього ігор доводиться адаптуватися під велику різноманітність конфігурацій.

Ігровий цикл працює протягом всієї гри. На кожному своєму циклі ігровий цикл обробляє користувача введення, оновлює стан гри і рендерить гру. А ще він стежить за ходом часу і управляє швидкістю ігрового процесу. Використовувати невідповідний шаблон - гірше ніж не використовувати ніяких шаблонів зовсім, так що зазвичай цей розділ покликаний застерегти вас від зайвого ентузіазму. Завданням шаблонів проектування не є проникнення в вашу кодову базу в максимальній кількості. Але цей шаблон не такий як інші. Я з впевненістю можу сказати що ви будете його використовувати. Якщо ви використовуєте готовий рушій, ви його не пишете самі, але він все одно присутній.

1.3. *Теоретичні засади комп'ютерної графіки та фізики в ігрових додатках*

Перші 3D ігри залежали від реалізації їхнього рендерингу в програмному забезпеченні. Це означало, що навіть щось настільки базове, як малювання лінії, повинно бути реалізовано графічним програмістом. Набір алгоритмів, необхідних для малювання 3D-об'єктів в 2D-буфер кольорів, загальновідомо як програмне забезпечення растеризації, і більшість курсів з комп'ютерної графіки витрачають деяку кількість часу

обговорення цього аспекту 3D-графіки. Але сучасні комп'ютери мають спеціальну графічну апаратуру, відому як графічний процесор (GPU), який вміє малювати основні побудови такі блоки, як точки, лінії та трикутники. Через це сучасні ігри не потребують реалізації алгоритмів програмної ростеризації. Натомість увага зосереджується на наданні відеокарті даних, які вона потребує для візуалізації 3D-сцени в бажаного способу, використовуючи такі бібліотеки, як OpenGL і DirectX. І якщо для подальшого налаштування є необхідні митні програми, відомі як шейдери, також можуть бути застосовані до цих даних. Але як тільки ці дані будуть зібрані, відеокарта займе її і намалює на екрані. Для кодування алгоритму лінійного малювання Bresenham, на щастя, пройшли. Слід зазначити, що в 3D-графіці часто необхідно використовувати наближення. Це тому що просто не вистачає часу для обчислення фотореалістичного світла. Ігри не схожі на CG фільми, де годинник можна витратити на обчислення одного кадру. Гра повинна бути намальована 30 або
60 разів на секунду, тому точність повинна бути порушена на користь продуктивності. Графічна помилка, що є результатом наближення, відома як графічний артефакт, і жодна гра не може уникнути артефактів повністю.

Найпростіший спосіб представити колір у 3D-сцені - використовувати колірний простір RGB, що означає окреме значення червоного, зеленого та синього кольорів. Це тому, що RGB безпосередньо відповідає як 2D монітор і малює кольорові зображення. Кожен піксель на екрані має окремий червоний, зелений і сині субпікселі, які об'єднуються для створення остаточного кольору для цього пікселя. Якщо ви уважно подивитеся на екран комп'ютера, ви повинні мати можливість розпізнавати ці різні кольори. Після вибору RGB наступним рішенням є бітова глибина або кількість бітів, виділених кожному

основний колір. Сьогодні найбільш поширеним є 8 біт на колір, а значить, є 256 можливих значень для червоного, зеленого та синього кольорів. Це дає в цілому приблизно 16 мільйонів унікальних кольорів.

У сучасних 3D-іграх дуже звично для людських персонажів складатися з 15000 і більше багатокутників. Коли в грі потрібно визначити, чи зіткнуться два символи чи ні, це є надзвичайно неефективним, щоб перевіряти всі ці трикутники. З цієї причини більшість ігор використовують спрощену геометрію зіткнення, наприклад сфери і коробки, для перевірки на зіткнення. Це колізія зіткнення не намальована на екрані, але використовується лише для ефективної перевірки зіткнень. Варто також зауважити, що для ігор не часто трапляється декілька рівнів зіткнення на один об'єкт. Таким чином, можна виконати просту перевірку зіткнення (наприклад, зі сферами) спочатку перевірити, чи існує будь-яка можливість зіткнення двох об'єктів. Якщо проста перевірка зіткнення говорить про можливе зіткнення, ми можемо виконувати обчислення з більшою кількістю складних зіткнень.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі автор розглянув загальну історію розробки ігор та основні алгоритми для розробки ігрових додатків . Дізнався про розвиток ігрової індустрії і про те, як це вплинуло на подальший розвиток комп’ютерної техніки та техніки в цілому. Також автор ознайомився з теоретичними засадами комп’ютерної графіки та фізики в ігрових додатках. Занотував основні теоретичні поняття та визначився з програмним забезпеченням для подальшої роботи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІГРОВОГО ДОДАТКУ

2.1. *Розробка гейм дизайну*

Що розпочати роботу над своїм проектом ми маємо розробити гейм дизайн та спроектувати модель нашого ігрового додатку. Ігровий дизайн - процес створення форми і змісту ігрового процесу (геймплея) розробляємої гри. Робота з гейм дизайном може відбуватися як через відповідний документ, так і існувати тільки в свідомості розробників гри. Ігровий дизайн визначає: набір можливих варіантів, з яких гравець може вибирати під час гри; умови перемоги і поразки; як гравець контролює те, що відбувається в грі; як взаємодіє з ігровим світом; складність гри і ін.

Існує багато спеціалізацій гейм дизайну, наприклад:

- Системний дизайн - створення правил і супутніх розрахунків для гри. Це - єдине завдання з області гейм дизайн, актуальна для будь-якої гри, тому що правила є у всіх ігор.
- Контент-дизайн - створення персонажів, предметів, загадок і місій. Хоча він і більш поширений у відеоіграх, рольові та колекційні карткові ігри також задіють значну кількість контенту.
- Дизайн рівнів - створення рівнів гри, що включає ландшафт карти і розташування на цій карті об'єктів. Хоча дизайн рівнів і є широко поширеним - майстри в настільних рольових іграх складають карти підземель починаючи з 1970-х років - кажучи «дизайнер рівнів», найчастіше мають на увазі дизайнера рівнів для відеоігри.

- Дизайн світу (в ММО і відкритих світах) - продумування просторів, локацій, як вони диктують призначений для користувача досвід і пов'язуються із загальною задумкою гри.

Автор вирішив розробити невеликий проект в якому нам потрібно з одного кінця дістатися іншого і заробити якомога більше очок. Складність гри полягає в тому, що наша швидкість пересування з часом збільшується, тому грати стає дуже непросто. Виграти в цій грі неможливо виграти, але цей додаток - чудова можливість, щоб позмагатися зі своїми рідними та близькими.

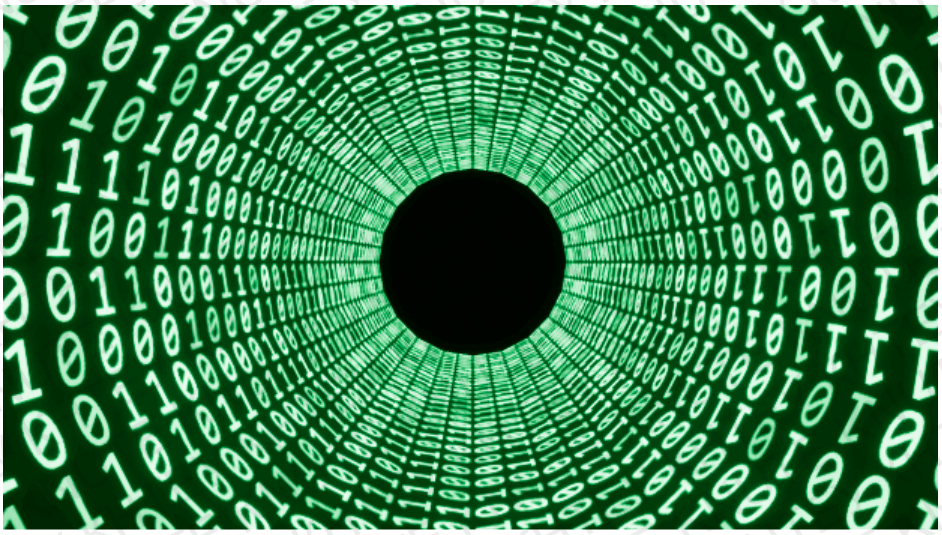

Рис.2.1. - Прототип ігрового додатку в UE4

2.2.*Способи реалізації ігрового додатку*

Unreal Engine — ігровий рушій, розроблюваний і підтримуваний компанією Epic Games. Перша гра, створена на цьому рушію — Unreal — з'явилася 1998 року. З тих пір різні версії цього ігрового рушія використали в більш ніж сотні ігор, серед яких Deus Ex, Lineage II, Thief: Deadly Shadows, Postal 2, серіях ігор Brothers in Arms, серія ігор Splinter Cell, Tom Clancy's Rainbow Six, а також у відомих ігрових серіях Unreal і Unreal Tournament від самих Epic Games. Пристосований у першу чергу для шутерів від першої особи, рушій використовувався й при створенні ігор інших жанрів. Написаний мовою C++, рушій дозволяє створювати

ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, Wii, Dreamcast і Nintendo GameCube.

Рис.2.2 - Логотип додатку Unreal Engine 4

Для спрощення портування рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic; раніше підтримувалися Glide API, S3 Metal, PowerVR SGL), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше підтримувалися A3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows, також планувалося для Linux і Mac), модулі для роботи з мережею й підтримка різних пристроїв вводу. Для гри у мережі підтримуються технології Windows Live, Xbox Live, і GameSpy, включаючи до 64 гравців (клієнтів) одночасно. Попри те, що офіційно засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

Усі елементи ігрового рушія представлені у вигляді об'єктів, що мають набір характеристик, і клас, який визначає доступні характеристики. У свою чергу будь-який клас є «дочірнім» класом **object**. Серед основних класів і об'єктів можна виділити наступні:

- Актор (**actor**) — базовий клас, що містить усі об'єкти, які мають відношення до ігрового процесу й мають просторові координати.
- Павн, пішак (**pawn**) — фізична модель гравця або об'єкта, керованого штучним інтелектом. Назва походить від англ. *pawn* — той, ким *маніпулюють* (*pawn* можна перевести також як *пішак*, тому такий об'єкт без якої-небудь моделі виглядає як пішак). Метод керування описаний спеціальним об'єктом, такий об'єкт називається *контролером*. Контролер штучного інтелекту описує лише загальну поведінку пішака під час ігрового процесу, а такі параметри як «здоров'я» (кількість пошкоджень, після яких пішак перестає функціонувати) або, наприклад, відстань, на якій пішак звертає увагу на звуки, задаються для кожного об'єкта окремо.
- Світ, рівень (**world**, **game level**) — об'єкт, що характеризує загальні властивості «простору», наприклад, силу тяжіння й туман, у якому розташовуються всі актори. Також може містити в собі параметри ігрового процесу, як, наприклад, ігровий режим, для якого призначений рівень.

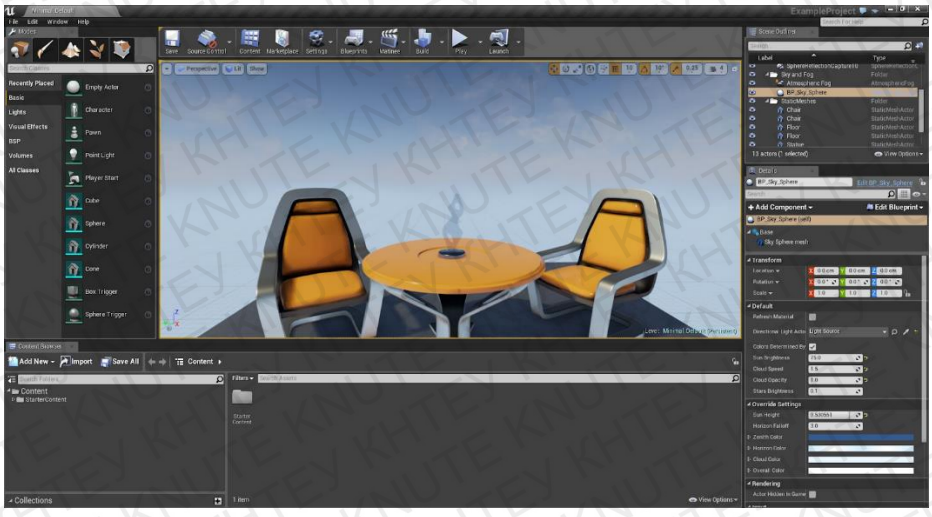

Рис.2.3 - Приклад акторів в Unreal Engine 4

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі автор розглянув загальні відомості про гейм дизайн та ігровий рушій Unreal Engine 4. Дізнався для чого нам потрібні актори в Unreal Engine 4 та як їх правильно застосовувати під час виконання випускної кваліфікаційної роботи. Також автор ознайомився ігровим рушієм UE4 , визначив його переваги та недоліки. Занотував основні теоретичні поняття та підготувався до виконання практичної частини свого ігрового додатку.

РОЗДІЛ 3
РОЗРОБКА ІГРОВОГО ДОДАТКУ

3.1. *Реалізація левел дизайну*

Дизайн рівня починається зі створення концептуального дизайну. Для цього найчастіше залучають художника, який здатний створити ескізи рівня, але не володіє професійними навичками дизайнера рівнів. У сучасних іграх дизайн рівня включає в себе документацію по дизайну, моделювання, заповнення ландшафту і установка на нього різних об'єктів. Як правило, все це здійснюється за допомогою редактора рівнів. Іноді редактори рівнів являють собою багатофункціональні пакети інструментів, здатні конкурувати навіть з комерційним 3D-моделюванням. Існує кілька основних етапів у процесі створення рівнів. В іграх різних жанрів ці етапи можуть відрізнятися відповідно до особливостей гри.

Основні етапи включають:

- Розбивку території карти на сектора - гори, міста, тунелі, площі для можливості переміщення гравця і противників;
- Визначення окремих регіонів на карті, де повинна відбуватися будь-яка діяльність, наприклад, видобуток ресурсів, будівництво бази і т. п ;
- Визначення нестатичних об'єктів на карті, наприклад, двері, ключі, кнопки, які взаємодіють з різними механізмами, приховані проходи і т. п .;

- Визначення місць організацій, наприклад, точки відродження ворогів, гравця, розміщення сходів, монет, скупчень ресурсів, зброї, точок збереження і т. п .;
- Визначення місць старту і кінця для одного або декількох гравців;
- Додавання певних деталей, наприклад, текстур, звуків, анімацій, освітлення і музичного супроводу і т. п .;
- Додавання скриптів і тригерів на рівень;
- Додавання скриптів пошуку шляху для мобів, області, в яких вони можуть знаходитися, дії, які будуть відбуватися після перетину певного тригера і діалоги з гравцем і між мобами.

Дизайн рівня може складатися з декількох ітерацій для досягнення бажаного результату. Створення кат-сцен в більшості випадків вимагає додаткових навичок, і тому вони створюються окремими співробітниками або навіть командою. Дизайнери рівнів, а також художники, іноді потрібні для проведення попереднього рендеринга рівня або цілого ігрового світу. Існує велика кількість помилок на рівнях, які дизайнери рівнів намагаються уникати. Так як ці помилки не завжди бувають поміченими відразу, вони деякий час залишаються на рівні. Гравець може застрягати в геометрії карти, без можливості знайти з такої ситуації вихід або ж померти. Гравець може знаходити різні місця, на яких можна заробляти велику кількість очок, вбиваючи відроджуються там монстрів або інших ворогів. У багатокористувацьких картах гравець може якимось чином потрапити на позицію, яка повинна бути недоступна для гравців, наприклад, на дах будинку, яка є прекрасною позицією для кемпінгу. У гіршому випадку гравець може випасти за межі ігрової зони, звідки його не зможуть дістати інші гравці. У деяких випадках спеціальні інструменти для створення карт можуть мати передбачену функцію для виявлення подібних помилок на картах. Ретельні дизайнери рівнів завжди використовують подібні інструменти як завершальний етап у створенні карти. У більшості випадків найкращим способом для визначення

помилок на карті є тестування карти досвідченими гравцями, які мають можливість доповісти про помилку.

Опрацювавши усі важливі етапи будування level дизайну для свого проекту, автор вирішив зробити ігровий рівень максимально простим. Для цього автор використав велику зелену тубу з додаванням матеріалів та асетів, за допомогою яких рівень стає виглядати ще краще та цікавіше.

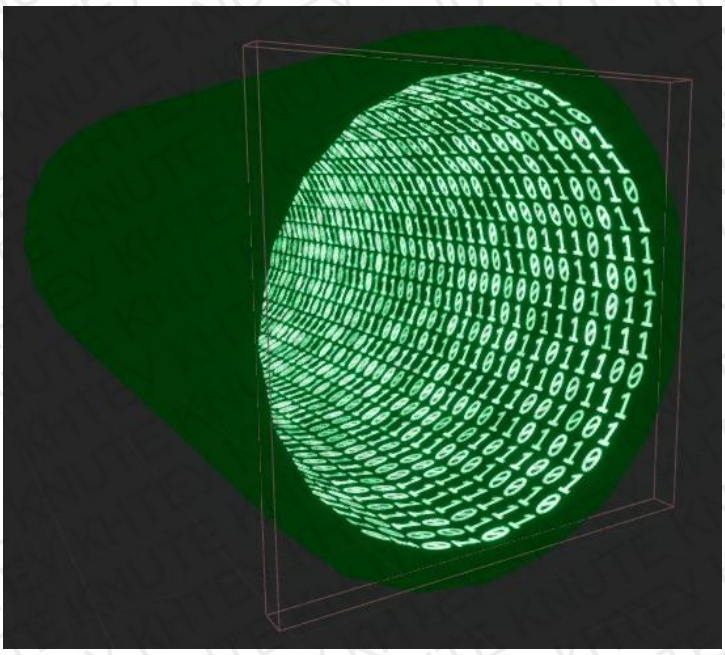

Рис.3.1 - Приклад level дизайну в Unreal Engine 4

3.2. *Додавання ігрової логіки до проекту*

Перш за все нам потрібно зробити так, щоб ми могли пересуватися, а саме падати вниз. Для цього ми створюємо змінну для визначення швидкості руху вперед гравця. Ми створили змінну типу Float з ім'ям ForwardSpeed і встановили значення за замовчуванням до 2000.

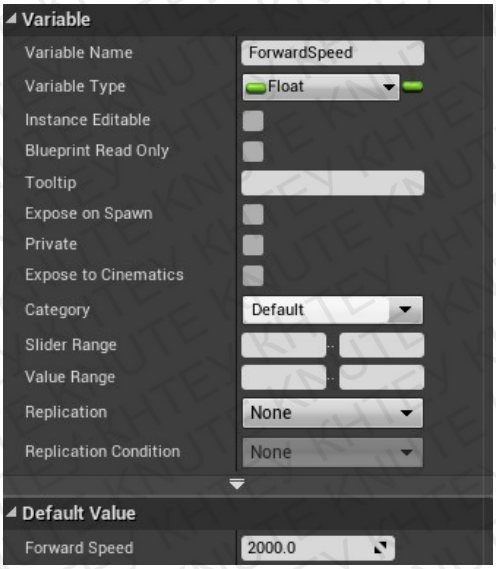

Рис.3.2 - Приклад роботи в Unreal Engine 4

Помноживши ForwardSpeed з Delta Seconds, ми отримуємо незалежний результат від частоти кадрів.

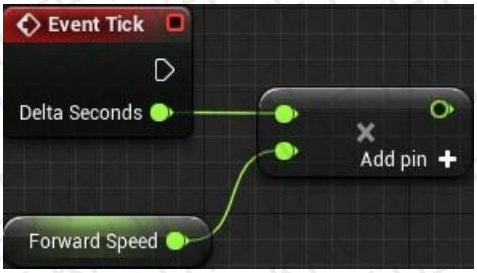

Рис.3.3 - Приклад роботи в Unreal Engine 4

Щоб перемістити гравця, ми створили вузол AddActorWorldOffset і встановили параметр Sweep до істини.

Рис.3.4 - Приклад роботи в Unreal Engine 4

Якщо ви спробуєте підключити результат Float до входу Delta Location, Unreal автоматично перетворить його на вектор. Однак, це

дозволить покласти значення Float у компоненти X, Y і Z вектора. Для цієї гри рух вперед повинен здійснюватися тільки по осі X. На щастя, можна розділити вектор на три компоненти Float. В результаті ми маємо таку картину:

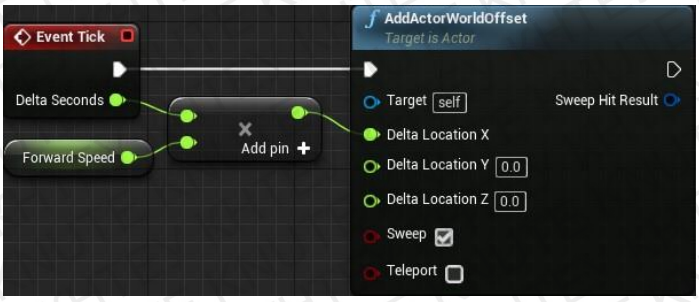

Рис.3.5 - Приклад роботи в Unreal Engine 4

Оскільки гра буде постійно спавнити тунелі, потрібно створити функцію спавну. Метою цієї функції буде створення тунелю на заданому місці. Щоб передати місце розташування функції, функція потребує вхідного параметра. Вони з'являться у вигляді вхідних контактів під час виклику функції.

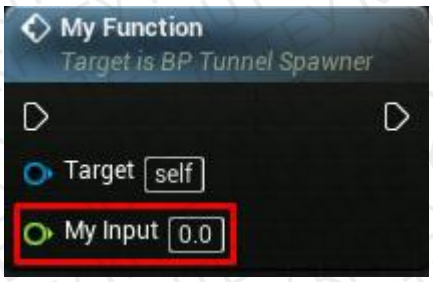

Рис.3.6 - Приклад роботи в Unreal Engine 4

Щоб встановити місце спавну, ми клацаємо правою кнопкою миші на значку Spawn Transform і вибераємо пункт Split Struct Pin. Після цього з'єднуємо вузол Spawn Actor From Class з вхідним вузлом.

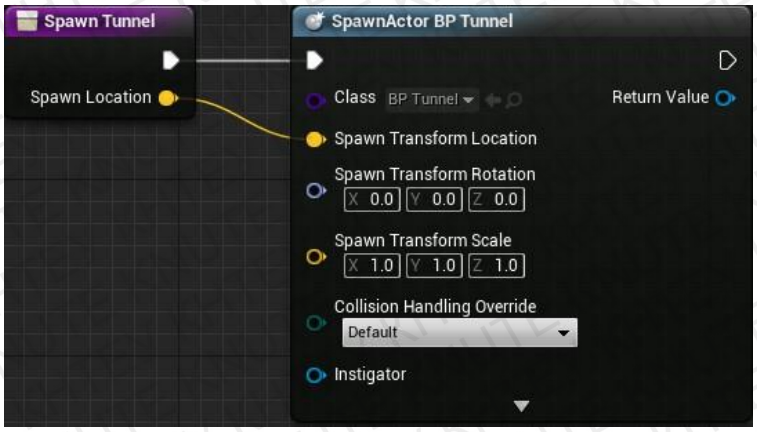

Рис.3.7 - Приклад роботи в Unreal Engine 4

Щоб зробити гру більш цікавою, ми створили стіни які стануть нам на заваді і також будуть обертатися. Додаємо нову змінну Float і називаємо її RotateSpeed. Встановлюємо значення за промовчанням до 30.

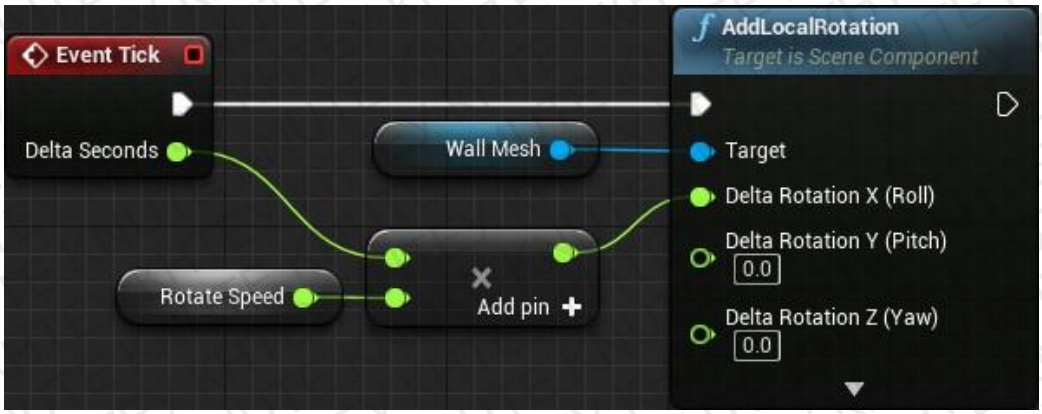

Рис.3.8 - Приклад роботи в Unreal Engine 4

Для перезапуску гри нам потрібно створити ще дві важливі речі. Перша з них це скидання гравця. Для цього ми створюємо нову функцію RestartGame.

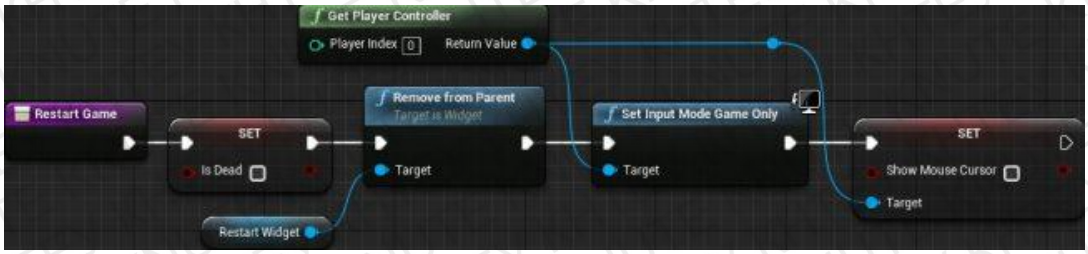

Рис.3.9- Приклад роботи в Unreal Engine 4

Друга важлива річ це спавн нових тунелів. По-перше, потрібно видалити існуючі тунелі, перш ніж заспавняться нові. Додаємо вузол послідовності після вхідного вузлу. Підключаємо контакт 1 на вузол ForLoop.

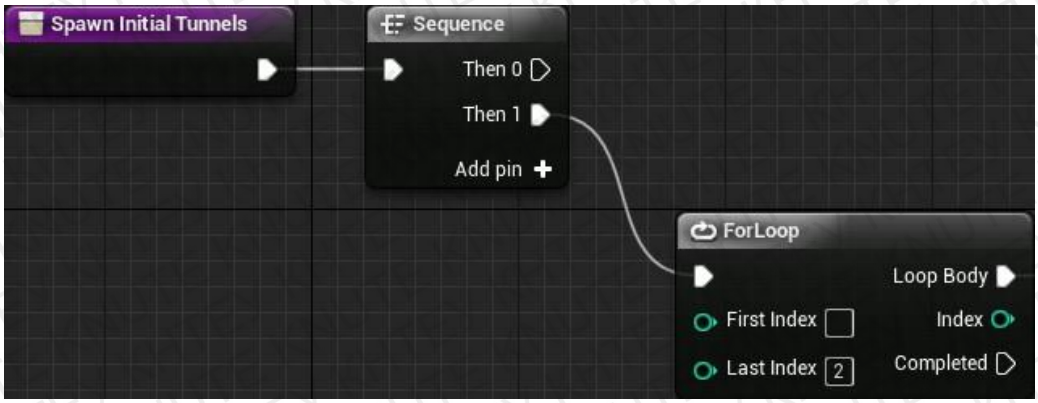

Рис.3.10- Приклад роботи в Unreal Engine 4

Ця установка дозволить отримати всі існуючі тунелі і видалити їх з гри. Нарешті, підключіть тоді Pin 0 вузла послідовності до вузла Get All Actors of Class. Це забезпечить видалення тунелів перед процесом спавну.

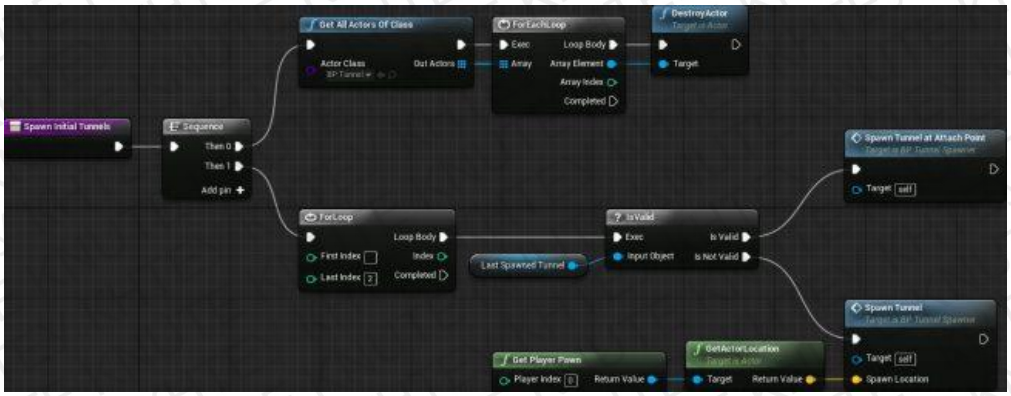

Рис.3.11- Приклад роботи в Unreal Engine 4

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі автор розглянув загальні відомості level дизайну та проектування ігрового додатку. Дізнався про особливості створення level дизайну для ігрових проектів та їх правильному застосуванні. Також автор використав безкоштовні асети для побудування прототипу свого проекту. Йому вдалося розписати усі найважливіші моменти створення ігрової логіки для свого проекту та продемонстрував фінальний вигляд своєї гри.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В випускній кваліфікаційній роботі автору вдалося змоделювати та розробити власний ігровий додаток, використовуючи рушій Unreal Engine 4 та блупрінти. Дати визначення та описати інструменти, які застосовуються сьогодні для проектування та створення ігрових проектів. Привести докази того, що програміст може не обмежувати себе, а створювати програмний продукт різної ступені важкості та масштабу, використовуючи при цьому різноманітне програмне забезпечення. Від невеличких проектів до потужних та дуже великих програмних продуктів. Також, в моїй роботі я описав практичну частину створення власного ігрового додатку. В подальшому він планує розвивати свій проект та поширювати його для інших користувачів. Автор вибрав найдоступніші для себе інструменти, так як у програмуванні дуже важливо вміти використовувати відкриту інформацію, аналізувати вихідний код схожих додатків та користуватися інструментами, які були зроблені для досягнення своїх потреб.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Joanna Lee, Learning Unreal Engine Game Development, 2016р. - 274 с.
2. Nicola Valcasara, Unreal Engine Game Development Blueprints, 2015р. - 352 с.
3. Benjamin Carnall, Unreal Engine 4.X By Example, 2016. - 506 с.
4. Joanna Lee, Unreal Engine: Game Development from A to Z, 2016р. - 837 с.
5. John.P Doran, Unreal Engine Game Development Cookbook, 2015р. - 326 с.
6. Aram Cookson, Unreal Engine 4 Game Development in 24 hours, 2016р. - 497с.
7. Alireza Tavakkoli, Game development and Simulation with Unreal Technology, 2015. - 742 с.

Інтернет ресурси

1. Level design – Режим доступу: https://en.wikipedia.org/wiki/Level_design
2. Game design – Режим доступу: https://en.wikipedia.org/wiki/Game_design
3. Unreal Engine 4 Blueprints tutorial – Режим доступу: https://www.youtube.com/watch?v=BVpy7kvz_yg&list=PLL0cLF8gjBpqRUy7r0DtVY3Fcdgq5Wk-h
4. Unreal Engine 4 Documentation - Режим доступу: https://docs.unrealengine.com/en-us/

					КНТЕУ 6.050103 07-07.БР		
					Розробка ігрового проекту на базі Unreal Engine 4	Стадія	Аркуш	Аркушів
Зм.	Аркуш	№ докум.	Підпис	Дата		СД	25	25
Зав. каф.		Криворучко О.В.						
Керівник		Цюцюра. С.В.						
Гарант		Цензура М. О.			Список використаної літератури	Факультет обліку, аудиту та інформаційних систем, 4 курс, 7 група		
Розроб		Коломієць І.О.						

ДОДАТКИ

Додаток А