Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Створення додатку з підтримкою бази даних на Unity Sengine

Студента 4 курсу, 7 групи спеціальності 121 «Інженерія програмного забезпечення»

Освальда Владислава Дмитровича

_____
підпис студента

Науковий керівник доктор технічних наук,
професор

Криворучко Олена Володимірівна

_____
підпис керівника

Гарант освітньої програми кандидат технічних наук,
доцент

Цензура Микола Олександрович

_____
підпис керівника

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет *інформаційних технологій* _____
Кафедра *інженерії програмного забезпечення та кібербезпеки*_____
Освітній ступінь *бакалавр*_____
Спеціальність *121 «Інженерія програмного забезпечення»*_____
Спеціалізація / освітня програма *121 «Інженерія програмного забезпечення»*

Затверджую
Зав. кафедри інженерії
програмного забезпечення
та кібербезпеки
Криворучко О.В.
«7» листопада 2019 р.

Завдання
на випускну кваліфікаційну роботу студентові

Освальд Владислав Дмитрович

1. Тема випускної кваліфікаційної роботи: *«Створення додатку з підтримкою бази даних на Unity Sengine».*
2. Затверджена наказом ректора від *«02» грудня 2019 р. № 4112*
3. Строк здачі студентом закінченої роботи _____
4. Цільова установка та вихідні дані до роботи

***Мета роботи*:** *розробка добре документованого, багаторівневого веб-додатка, в стек технологій якого входить Spring Framework.*

Об'єкт дослідження: *ігрові освітні технології і способи їх створення та реалізації на прикладі обраної предметної області.*
Предмет дослідження: *методи та 3D-технології розроблення та реалізації освітніх ігор.*

5. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*		
2.	*Вступ та перелік літературних джерел*		
3.	*Розділ 1. «Розвиток освітніх комп'ютерних ігор»*		
4.	*Розділ 2. «Розробка комп'ютерних ігор та база даних в них»*		
5.	*Розділ 3. «Розробка відеогри на Unity Engine»*		
6.	*Висновки*		
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*		
8.	*Підготовка автореферату та презентації доповіді*		
9.	*Попередній захист випускної кваліфікаційної роботи*		
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*		
11.	*Здача прошитої випускної кваліфікаційної роботи на кафедру*		
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «___»_____ 20___ р.

8. Науковий керівник
випускної кваліфікаційної роботи О.В. Криворучко

9. Гарант освітньої програми М.О. Цензура

10. Завдання прийняв до виконання студент В.Д. Освальд

11. Відгук керівника випускної кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускної кваліфікаційної роботи _____
(підпис, дата)

Відмітка про попередній захист Криворучко О.В. _____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студента В.Д. Освальда може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми М.О. Цензура

Завідувач кафедри О.В. Криворучко

«_____»_____20 ___ р.

Анотація

Освальд В.Д. «Створення додатку з підтримкою бази даних на Unity Sengine»

В випускній кваліфікаційній роботі розглянуто типи освітніх ігор, проведено аналіз їх використання у освіті та надано їх класифікація, також розглянуті вже існуючі освітні ігрові платформи. Також в випускній роботі показано послідовність розробки гри для провадження в освітній процес; яким чином відбудеться зв'язок з базою даних (навчальні матеріали) та етапи розробки; розглянуто характеристику ігрового движку та технології для розробки всієї гри. У розділі про розробку гри описуються концепт розробки гри, використані елементи інтерфейсу та можливості Unity Engine та за допомогою яких елементів відбувається взаємодія з базою даних.

Випускна кваліфікаційна робота на тему «Створення додатку з підтримкою бази даних на Unity Sengine» містить 44 сторінки, 1 таблицю, 14 рисунків та 5 додатків. Список посилань включає 13 пунктів

Ключові слова: освітня гра, ігровий движок, Unity Engine, база даних.

ANNOTATION

«Develop the application with database of Unity Sengine»

In graduation qualification work, we’ve examined types of games, analyzing use of this games in education, also get the view on existing game platforms.

Also in graduation qualification work we show sequence of game development for accepting in studying process, connection with database(learning materials) and technology of game development. In the section about game develop described concept of game, used the elements of interface and possibility of Unity engine and connection with database.

Graduation qualification work at topic «Develop the application with database of Unity Sengine» including 44 pages, 1 table, 14 figures and 5 additions. List of sources including 13 points.

Key words: education game, game engine, Unity engine, database

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

GUI – graphical user interface – графічний користувальницький інтерфейс;

IDE – – integrated development environment – інтегроване середовище розробки;

JDK – java development kit – комплект засобів розробки Java;

NDK – Android native development kit – пакет інструментаріїв і бібліотек, що дозволяє реалізувати частину програми на мові C/C++;

SDK – software development kit – комплект засобів розробки (Android)

SSAO – screen space ambient occlusion – програмна техніка (методика) в тривимірній комп'ютерній графіці, яка є наближеною імітацією глобального освітлення

ВСТУП

Ігрові технології займають важливе місце в навчальному процесі. Ігрові технології пов'язані з ігровою формою взаємодії викладача і студента через реалізацію певного сюжету (ігри, казки, спектаклі, справа-ше спілкування). При цьому освітні завдання включаються в зміст игри. В освітньому процесі використовують цікаві, театралізовані, ділові, рольові, комп'ютерні ігри. Розробкою теорії гри, її методологічних основ, з'ясуванням її соціальної природи, значення для розвитку учня займалося чимало учених. Реалізація ігрових прийомів у навчанні відбувається за такими основними напрямками:

- дидактична мета ставиться перед здобувача ми освіти у формі ігрової задачі;
- освітня діяльність підпорядковується правилам гри;
- навчальний матеріал використовується в якості її засобу, в навчальну діяльність вводиться елемент змагання, який переводить дидактичну задачу в ігрову;
- успішне виконання дидактичного завдання зв'язується з ігровим результатом.

Візуалізація є одним з найефективніших методів навчання, що допомагає набагато простіше і глибше розібратися в сутності різних явищ. Особливо корисними є візуалізація та моделювання при вивченні динамічних, що змінюються в часі об'єктів і явищ, які буває складно зрозуміти, дивлячись на просту статичну картинку в звичайному підручнику. Далеко не всі експерименти можна чи потрібно виконувати в аудиторії. Тож не дивно, що технології цифрового моделювання досить швидко прийшли в цю галузь. Зараз

у мережі інтернет представлено ціла лінійка програмних пакетів, призначених для виконання віртуальних навчальних експериментів.

Ігрові технології не тільки сприяють вихованню пізнавальних інтересів та активізації діяльності здобувачів освіти, а й виконують низку інших функцій:

- правильно організована, з урахуванням специфіки матеріалу, гра тренує пам'ять, допомагає здобувачам освіти виробити мовні вміння і навички;
- гра стимулює розумову діяльність здобувачів освіти, розвиває увагу і пізнавальний інтерес до дисципліни;
- гра - один з прийомів подолання пасивності здобувачів освіти. Функція гри - її різноманітна корисність. У кожного виду гри своя корисність

Ігри пройшли довгий шлях становлення від найпростіших двокольорних 2D ігор до сучасних 3D ігор з високим рівнем графіки. Центральним програмним компонентом комп'ютерних ігор є ігровий рушій. Він забезпечує основні технології, спрощує розробку і часто дає грі можлівість запускатіся на декількох платформах, таких як ігрові консолі та настільні операційні системи, наприклад, GNU / Linux, Mac OS X і Microsoft Windows.

Unity Engine це мультиплатформенний ігровий движок для розробки двох-і тривимірних ігор, що працюють під операційними системами Windows OS X, Windows Phone, Android, Apple IOS, Linux, а також на ігрових приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One. Є можливість створювати додатки для запуску в браузерах за допомогою спеціального модуля Unity Engine (Unity Web Player), а також за допомогою реалізації технології WebGL (програмна бібліотека, що дозволяє створювати на JavaScript інтерактивну 3D-графіку). Unity Engine підтримує дві сценарних мови JavaScript, C #.

Актуальність роботи. Протягом останнього часу дуже стрімко розвиваються індустрія навчальних комп'ютерних ігор. і у розробників ігор з'явилося більше можливостей, актуальною стає неформальна освіта, яку студенти можуть отримати використовуючи ігрові платформи. Це призводить до нового розуміння ролі університетської освіти і свободи для творчості. Тому у

цій дипломній роботі розглянуто сучасні технології для розробки навчальних ігор, визначено критерії вибору вірного інструменту для розробки ігор задля спрощення програмування і скорочення термінів розробки програмної частини.

Обґрунтування необхідності проведення дослідження. Дослідження є доцільним оскільки все більш актуальним стає неформальна освіта, яку люди отримують поза інституціональних структур, використовуючи відкриті освітні ресурси (ВОР) та платформи ігрового навчання. Це призводить до нового розуміння ролі університетської освіти.

Метою роботи є обґрунтування вибору програмного забезпечення щодо створення та реалізації баз даних освітніх ігор. *Для досягнення поставленої мети потрібно вирішити наступні завдання:*

− дослідити сучасні освітні ігрові середовища;

− проаналізувати технології для їх створення;

− ознайомитись з інструментарієм Unity GEngine;

− зпроектувати освітню ігру типу Pokemon Go для оцінювання знань з предметної області на движку Unity GEngine та показати можливість зв’язку з базою даних.

Об’єктом дослідження є ігрові освітні технології і способи їх створення та реалізації на прикладі обраної предметної області.

Предметом дослідження є методи та 3D-технології розроблення та реалізації освітніх ігор.

Методом дослідження є порівняльний аналіз способів реалізації різних провідних компаній у сфері освітніх комп’ютерних ігор.

Отримані результати можна використати для оцінювання знань здобувачів освіти та в освітньому процесі

РОЗДІЛ 1
РОЗВИТОК ОСВІТНІХ КОМП'ЮТЕРНИХ ІГОР

1.1 Аналіз використання ігор в освіті

Найбільшу цікавість, в освітньому процесі, представляють ігрові технології, так як вони не тільки сприяють вихованню пізнавальних інтересів та активізації діяльності здобувачів освіти, а й виконують ряд інших функцій.

Соціокультурний призначення гри. Гра - найсильніший засіб социалізації.

Функція міжнаціональної комунікації. Ігри національні й у той же час інтернаціональні, міжнаціональні, загальнолюдські. Ігри дають можливість моделювати різні ситуації життя, шукати вихід з конфліктів, не вдаючись до агресивності, учать розмаїтості емоцій у сприйнятті всього існуючого в житті.

Функція самореалізації людини в грі. Для людини гра важлива як сфера реалізації себе як особистості. Саме в цьому плані їй важливий сам процес игри, а не її результат, конкурентність або досягнення будь-якої мети.

Комунікативна гра. Гра - діяльність комунікативна, хоча за чисто ігровим правилам і конкретна.

Діагностична функція гри. Гра має прогностичність; вона діагностичніша, ніж будь-яка інша діяльність людини, по-перше, тому, що індивід поводиться в грі на максимумі проявів (інтелект, творчість); по-друге, гра сама по собі - це особливе «поле самовираження».

Ігротерапевична функція гри. Гра може і повинна бути використана для подолання різних труднощів, що виникають у людини в поведінці, в спілкуванні з оточуючими, у вченні

Функція корекції у грі. Корекційні ігри здатні надати допомогу учням з поведінкою, що відхиляється, допомогти їм впоратися з переживаннями, що

перешкоджають їх нормальному самопочуттю і спілкуванню зі свсрсниками в групі.

Розважальна функція гри. Розвага - це потяг до різного, різноманітного. Розважальна функція гри пов'язана зі створенням певного комфорту, сприятливої атмосфери, душевної радості як захисних механізмів, тобто стабілізації особистості, реалізації рівнів її притязань. Розвага в іграх - пошук. Гра має магію, здатну давати їжу фантазії, виводити на розважальність.

В останнє десятиліття в педагогічній літературі з'явилося багато визначень гри. Наприклад, цікаво визначення гри у Г.К.Селевко «Гра - це вид діяльності в умовах ситуацій, спрямованих на відтворення і засвоєння суспільного досвіду, в якому складається й удосконалюється самоврядування поведінкою» [3].

Авторитетом в узагальненні зв'язку між людством і грою є відомий філософ XX століття Йохан Хейзінга, який написав книгу під назвою "HOMO LUDENS" (людина граюча) [4]. Він розглянув процес життєдіяльності через гру. Гра розглядається як найважливіша характеристика людини, як культурної істоти. Проводячи докладне дослідження позначення гри на різних мовах (японській, латинській, семітському, німецькій та ін.), Він відзначає, що в деяких мовах (грецькій, китайській та ін.) такого слова немає, а є лише різні слова для визначення ігор, наприклад, змагальні ігри, ігри-вистави, азартні ігри і т.п.

Слід зазначити, що наукового єдиного загального для всіх визначення гри в літературі немає досі.

Найбільш відомі підходи до гри в історичному аспекті. К.Гросс. Він розвиває свою теорію як теорію вправ, вважаючи, що у вищих живих істот, особливо у людини, природжені реакції є недостатньо розвиненими, тому людині дано особливо довге дитинство для підготовки до життя за допомогою наслідування звичок, здібностей старшого покоління.

Сутність гри полягає в тому, що в ній більше важливий не результат, а сам процес переживань, пов'язаних з ігровими діями. При цьому, по-перше, важливий сам досвід переживання позитивних почуттів для людини, по-друге,

через переживання тільки й можна виховати позитивне ставлення до діяльності. Гра має багаті можливості сформувати позитивне ставлення і до неігрової діяльності.

Більшості ігор властиві чотири головні риси:

- це вільна розвиваюча діяльність, що здійснюється лише за бажанням, заради задоволення від самого процесу діяльності, а не тільки від результату (процедурне задоволення);
- це творчий, значною мірою імпровізований, дуже активний характер цієї діяльності («поле творчості»);
- це емоційна піднесеність діяльності, суперництво, змагальність, конкуренція, атракція (чуттєва природа гри, «емоційне напруження»);
- це наявність прямих або непрямих правил, що відображають зміст гри, логічну та часову послідовність її розвитку.

В структуру гри як діяльності, органічно входить цілепокладання, планування, реалізація мети, а також аналіз результатів, в яких особистість повністю реалізує себе як суб'єкт.

1.2 Характеристики комп’ютерних ігор та їх классифікація

Фахівці з ігрових технологій в освітній практиці пропонують таку класифікацію ігор (таблиця 1.1).

Таблиця 1.1.

Класифікація комп’ютерних ігор

Тип класу	**Прелік**
по області діяльності:	фізичні, інтелектуальні, трудовы, соціальні, психологічні;
за характером педагогічного процесу:	навчальні, тренінгові, контролюючі, узагальнюючі; пізнавальні, виховні, розвивальні; репродуктивні, продуктивні, творчі; комунікативні, діагностичні, профорієнтаційні, психотехнічні;
по ігровій методиці:	предметні, сюжетні, рольові, ділові, імітаційні, драматизації;
по предметної області:	математичні, хімічні, біологічні, фізичні, екологічні; музичні, театральні, літературні; трудові, технічні, виробничі; фізкультурні, спортивні, військово-прикладні, туристичні, народні; суспільнознавчі, управлінські, економічні, комерційні;
по ігровому середовищу:	без предметів, з предметами; настільні, кімнатні, вуличні, на місцевості; комп'ютерні, телевізійні, технічні засоби навчання (ТСН); технічні, із засобами пересування.

Комп'ютерна гра - форма розвивально-розважальної взаємодії користувача і комп'ютера, що імітує в віртуальному просторі життєві і уявні ситуації, що має значний jcdsnysq потенціал, який полягає в стимулюванні пізнавального інтересу.

Основним способом поділу відеоігор на категорії є поділ по платформах, який вказує, на якому пристрої можна запустити ту чи іншу гру. Якщо у користувача немає платформи, для якої призначена гра, то і пограти в неї він не зможе.

Персональний комп'ютер (ПК, PC, ноутбук, нетбук)

Перелік найбільш популярних серій комп'ютерних ОС: Windows (від фірми Microsoft), Mac OS (від фірми Apple), Linux (безкоштовна ОС, розроблювана світовою інтернет-спільнотою).

Ігрова консоль або приставка (PS, Xbox, Nintendo)

До найбільш популярних поколінь консолей можна віднести: Sony PlayStation (PSP, PSOne, PS2, PS3, PS4), Microsoft Xbox (Xbox, Xbox 360, Xbox One), Nintendo 3DS, Wii.

Мобільний пристрій: телефон, планшет, кишеньковий комп'ютер (КПК, PDA)

На застарілих телефонах гри є java-додатки, на сучасних телефонах ігри запускаються під мобільними ОС: Windows Mobil і Android. Для поширення мобільних ігор створені цілі глобальні сервіси типу App Store, Google Play.

Планшет, сенсорний мобільний телефон. Окремою категорією йдуть планшети і телефони з можливістю сенсорного введення (натисканням пальцями по екрану). Особливий спосіб введення даних дозволяє створювати гри з унікальних-ним геймплеєм, що використовують ці особливості, наприклад, малювання на екрані, нахил пристроїв для зміни гравітації в грі і т. д.

Ігровий автомат

Браузерна або флеш-гра (віртуальна інтернет платформа)

Браузерні ігри - ігри, здатні запускатися у вікні браузера (програми для перегляду інтернет-сторінок). Особливий пристрій браузерних ігор дозволяє грати в них з будь-якого пристрою, який може підключатися до інтернету. Всі популярні браузери: Google Chrome, Opera, FireFox, Edge, Safari - підтримують запуск невеликих програм прямо внутр. інтернет-сторінок.

Існує також класифікація комп’ютерних ігор за графікою, за змістом (жанром – рис.1.2), *по сеттінгу,* за видавничими критеріями, за типом їх поширення, за кількостю гравців

Рис. 1.2 — Схема жанрів

1.3 Програмні сервіси та платформи, що мають потенціал для геймифікації навчання

В даний час методи Гейміфікації в навчанні впроваджують різні освітні установи по всьому світу (Стокгольмська школа імені Віктора Рюдберга, проект LinguaLeo, «Роботландия» , «Професія-сор Хіггінс», The Legend of Zelda та ін.). В ході аналізу досвіду російських та зарубіжних педагогів виділені кілька комп'ютерних сервісів і платформ, які з більшим чи меншим ступенем відповідають принципам і ідеї Гейміфікації.

До групи ігрових платформ відносяться: Classcraft, MinecraftEdu, LinguaLeo, DuoLingo та ін. Найбільш поширені в освітньому середовищі такі платформи, як:

1. *DuoLingo* (https://www.duolingo.com) - чудовий приклад гейм-ції навчального процесу. Засіб об'єднує в собі безкоштовний веб-сайт для вивчення мов з платною платформою для перекладу текстів з використанням краудсорсингу. Вимагає постійного підключення до мережі Інтернет. Гейміфікація процесу навчання організована у вигляді дерева досягнень, як у справжній рольовій грі (англ. Role-Playing Game, RPG).

2. *Classcraft* (https://www.classcraft.com/ru/) - ігрова платформа, що відноситься до сфери проектування навчання. По суті, цей сервіс є ігровим

варіантом бальнорейтингової системи. Для отримання доступу необхідно зареєструватися на сайті (тобто Застосування ресурсу можливо тільки при наявності Інтернету). Після реєстрації - етап настройки персонажа (зовнішній вигляд, клас і здібності). На початковому рівні можна отримати тільки одну здатність. Щоб розвивати здібності, потрібно отримувати нові рівні. Управляє грою вчитель (майстер), він же роздає бали за різні досягнення (виконання завдань, відповіді на питання).

Уявляється система ігрових заохочень і покарань.

Друга група програмних засобів для Гейміфікації навчального процесу - це освітні квести. У цій групі виділимо: Scratch, Quandary, RibbonHero.

1. *Quandary* (http://www.halfbakedsoftware.com/quandary.php) - сервіс для проектування завдань з вибором ходу. Робота в середовищі організується у вигляді інтерактивного дослідження по якійсь певній тематиці (наприклад, для уроку біології це може бути боротьба за врожай, вирощування садових культур; на уроках хімії Гейміфікація з цим інструментом дозволить в ігровій формі відкрити нову хімічну речовину і т . п.). З 1 вересня 2009 року програмний засіб є безкоштовним, при цьому файл-інсталятор знаходиться на офіційному сайті. Значний для багатьох мінус конструктора - відсутність україномовного інтерфейсу.

2. *RibbonHero* (http://www.ribbonhero.com) - це безкоштовна надбудова до офісного пакету Microsoft Office для навчання користувачів застосуванню інструментів, доступних в новому стрічковому інтерфейсі. Після установки гри можна легко почати з будь-якої з базових програм Office, таких як Word, Excel і PowerPoint. Ігрові елементи укладаються в отриманні очок при виконанні завдань, які згруповані в чотири розділи: робота з текстом, дизайн сторінки і макет, художнє оформлення і швидкі окуляри.

3. *Scratch* (https://scratch.mit.edu) - проста, інтуїтивно зрозуміла і наочна мова програмування для знайомства молодших школярів з основами алгорітмізаціі і програмування. Розробник Scratch Мітчел Резник вважає, що найбільш ефективним способом навчання є активне пізнання - пізнання через моделювання навколишнього світу. Scratch - безкоштовний продукт, який дозволяє створювати інтерактивні додатки. Гейміфікація навчання із застосуванням Scratch дозволяє розвивати творче мислення, проектувати ефективну предметну взаємодію; формувати навички системного аналізу.

Серед сервісів, які використовуються для Гейміфікації управління навчанням, напоширінішими є:

1. *ClassDojo (https://www.classdojo.com)* - безкоштовна система управління поведінкою в класі, яка допомагає вчителям покращувати поведінку в своїх класах.
2. *Goalbook (https://goalbookapp.com)* - це онлайн-платформа, яка допомагає вчителям, батькам і самим учням спільно відстежувати їх прогрес в навчанні. Це один із прикладів реалізації методів гейміфікації контролю за успішністю. Платформа платна - від 6000 $ в рік. Україно-мовний інтерфейс відсутній.
3. *Coursera* (https://www.coursera.org) - міжнародний проект в сфері масової онлайн-освіти. Проект надає абсолютно безкоштовний доступ до великої кількості освітніх курсів кращих навчальних закллладів світу. Гейміфікація навчального процесу на платформі Coursera реалізована в рамках контролю за успішністю. Прогрес в навчанні вимірюється шляхом виконання завдань і онлайн-тестів, які оцінюються особисто викладачем або самою системою
4. *Brainscape* (https://www.brainscape.com) є мобільною платформою навчання, яка допомагає студентам вивчати і запам'ятовувати інформацію на основі спеціальних карток. Платформа застосовує адаптивні алгоритми

при створенні цих карток для запам'ятовування інформації в різних її формах (текстова, числова, графічна), змінюючи послідовність показу карток в залежності від того, що учень знає, а що викликає у нього труднощі. Метод, застосовуваний в цій платформі, відомий як метод повторіння на основі впевненості (англ. ConfidenceBased Repetition, CBR). Система вимагає постійного доступу до мережі Інтернет, україномовний інтерфейс відсутній.

1.4. Висновки до розділу 1.

Доцільність застосуванні ігрових форм навчання в освітньому процесі безумовна. В педагогічній діяльності необхідно приділяти особливу увагу механізмам впливу на мотивацію навчання, свідомість і поведінку студентів, Ці завдання можуть бути вирішені при застосуванні ігрових форм навчання, так як ігри дозволяють ефективно організувати творчу взаємодію викладача і студентів, створюють умови для формування особистісних якостей.

Характеристики, структура освітніх ігор, принципи ігрової діяльності мають свою клафікацію - за ігровою платформою, графікою, змістом, видавничими критеріям, типом їх розповсюдження, кількістю ігроків. Є безліч програмних сервісів та платформ, які мають потенціал для геймифікації навчання.

Розділ 2.
РОЗРОБКА КОМП'ЮТЕРНИХ ІГОР ТА БАЗА ДАНИХ В НИХ

2.1. Основні етапи розробоки комп'ютерної гри

Розробка комп'ютерних ігор - це досить чітко налагоджений процес, який має певні етапи, що так чи інакше проходять ігри при їх створенні. Однак життя зазвичай вносить свої корективи навіть у самі чіткі плани.

Дуже часто розробники ігор не можуть встигнути доробити гру в скількинебудь прийнятний термін - яскравий приклад - Duke Nukem Forever, випуску якого весь ігровий світ чекав багато років. Практично завжди після виходу комп'ютерної гри за нею йдуть виправлення - вся справа в тому, що розробники, знову ж таки, не вкладаються у відведені їм терміни.

Треба врахувати, що в ігровому бізнесі існує два типи компаній - розробник (developer) і видавець (publisher). Якщо розробник і видавець збігаються - процес розробки гри лише виграє - розробнику немає потреби переконувати стороннього видавця в доцільності капіталовкладень в розробку.

Типові етапи розробки комп'ютерної гри.

1. *Концептування* (Concept) На цьому першому кроці команда пропонує концепцію гри, і проводить початкоеу опрацювання ігрового дизайну. Головна мета даного етапу - це геймдізайнерська документація, що включає в себе Vision (розгорнутий до-кумент, що описує гру, як кінцевий бізнес-продукт) і Concept Document (початкове опрацювання всіх аспектів гри).

1. Прототипування (Prototyping). Важливий етап проектування будь-якої гри - це створення прототипу. Те, що добре виглядає «на папері», зовсім не обов'язково буде цікаво в реальності. Прототип реалізується для оцінки основного

ігрового процесу, перевірки різних гіпотез, проведення тестів ігрових механік, для перевірки ключових технічних моментів.

2. Серйозна помилка початківців розробників - нести тимчасову інфраструктуру і «милиці» реалізації коду в основний проект.

3. Вертикальний зріз (Vertical Slice) - отримати мінімально можливу повноцінну версію гри, що включає в себе повністю реалізований основний ігровий процес. При цьому високу якість опрацювання обов'язково потрібно втілити тільки для тих ігрових елементів, які суттєво впливають на сприйняття продукту. При цьому всі базові фичи гри присутні як мінімум в чорновій якості. Реалізовано мінімальний, але достатній для втілення повноцінного ігрового процесу набір контенту (один рівень або одна локація).

4. Виробництво контенту (Content production). розробляється достатня кількість контенту для першого запуску на зовнішню аудиторію. Реалізуються всі фічі, заплановані до закритого бета-тестування. Це найбільш тривалий етап, який може займати, для великих клієнтських проектів рік і більше.

5. Friends & Family / CBT (закрите бета-тестування) - продукт вперше демонструється досить широкому загалу, хоча і лояльному до продукту або компанії. Серед найбільш важливих задач на цьому етапі виступають: пошук і виправлення гейм-дизайнерських помилок, проблем ігрової логіки і усунення критичних багів. На цьому етапі в грі присутні вже всі ключові фичи, створено досить контенту для повноціної гри тривалий час, налаштовані збір і аналіз статистики. Тестування йде по тест-плану, проводяться стрес-тести вже із залученням реальних гравців.

6. Soft Launch / OBT (відкритий бета-тест) - триває тестування гри, але вже на широкій аудіторії. Йде оптимізація під великі навантаження. Гра повинна бути готова для прийому великого трафіку. У грі реалізований білінг і приймаються платежі.

7. Release (Видання). Ключова мета - це комерціалізаія - отримання прибутку. Базовий застосовуваний для оцінки прибутковості критерій: кількість

грошей, принесених в середньому одним игроком за весь час (LTV aka lifetime value), має перевершувати витрати на залучення цього гравця (CPI aka cost per install).

2.2 Поняття ігрового движка

Ігровий движок (англ. Game engine) - це програмна платформа для створення і розробки комп'ютерних ігор або будь-яких інших додатків з графікою, оброблюваної в реальному часі [8].

Ігровий движок, як правило, складається з наступних компонентів:

1. *Графічний движок* (англ. Graphics engine) - програмний компонент, основним завданням якого є візуалізація (рендеринг) двомірної або тривимірної комп'ютерної графіки.

2. *Фізичний движок* (англ. Physics engine) - програмний компонент, який виробляє моделювання фізичних законів у віртуальному оточенні.

3. *Звуковий движок* (англ. Sound / audio engine) - програмний компонент, що

відповідає за відтворення звуку (шумове та музичне оформлення, голосів персонажів) в комп'ютерній грі або іншому додатку.

4. *Ігровий штучний інтелект* (англ. Game artificial intelligence) - набір програмних методик, які використовуються в комп'ютерних іграх для створення ілюзії інтелекту в поведінці персонажів, керованих комп'ютером.

Рис. 2.1 — Структура ігрового движка/рушія

До появи ігрових движків нові ігри створювалися з нуля, будучи оптимізовані під цільову платформу. Необхідність використання подібного підходу обумовлювалася обмеженими можливостями апаратного забезпечення того часу, а також тим фактором, що архітектура апаратного забезпечення часто змінювалася.

2.3 Характеристики ігрових движків

Характеристики ігрового движка як засобу розробки обумовлюють область його застосування. Значна їх частина визначається компонентами, з яких він складається. Сучасний ігровий движок є комплексною програмною платформою, що складається з декількох основних компонентів, кожен з яких може бути як безпосередньою частиною движка, так і поставлятися окремо. Далі буде розглянуто кожен з використовуваних компонентів.

Графічний движок (англ. Graphics engine) - проміжне програмне забезпечення (англ. Middleware), основним завданням якого є візуалізація комп'ютерної графіки. Графічний движок визначає можливості по відображенню комп'ютерної графіки, які включають в себе:

1. Доступні режими рендеринга (2D / 3D).

2. Підтримувані програмні інтерфейси (OpenGL / DirectX).

3. Різні графічні характеристики: доступну роздільну здатність (кількість точок на одиницю площі), роздільну здатність і фільтрацію текстур, параметри освітлення і тіней.

Графічний движок може існувати як окремий продукт, але часто є частиною ігрового движка. Головною відмінністю ігрового графичного движка є обробка графіки в режимі реального часу.

На початковому етапі розвитку комп'ютерних ігор саме графічний движок становив більшу частину ігрового движка (90 - 95%). І зараз, як правило, графічні движки не поширюються окремо від ігрових, так як, по-перше, створення більшості ігор неможливо без графічного движка (виняток - деякі логічні і текстові ігри). Подруге, графічного движка самого по собі недостатньо для створення гри.

Фізичний движок (англ. Physics engine) - програмний компонент, який виробляє комп'ютерне моделювання фізичних законів реального світу у віртуальному світі, з тим або іншим ступенем наближення до реальності. Як і графічний движок, ігровий фізичний движок повинен працювати в режимі реального часу (на відміну, наприклад, від наукового фізичного движка):

NvidiaPhysX (26,8% ринку); Havok (22,7% ринку);

Bullet Physics Library (10,3% ринку);

Open Dynamics Engine (4,1% ринку).

Звуковий движок (англ. Sound / audio engine) - програмний компонент ігрового движка, який відповідає за відтворення звуку (шумове та музичне оформлення, голоси персонажів) в комп'ютерній грі або іншому додатку. Найчастіше є частиною ігрового движка, може використовувати такі відомі програмні інтерфейси: OpenAL; DirectSound3d; Environmental Audio Extensions (EAX); FMOD.

Ігровий штучний інтелект (ІШІ) (англ. Game artificial intelligence) - програмний компонент і / або набір програмних методик, що використовуються в комп'ютерних іграх для створення ілюзії інтелекту в поведінці віртуальних персонажів. На відміну від традиційного в ігровому ІІ широко застосовуються різного роду спрощення, обмани і емуляції, так як це дозволяє: знизити бюджет гри; зменшити споживання ресурсів системи.

Цільові платформи. Ігровий движок визначає доступні цільові платформи, для яких з його допомогою можна створити ігри, наприклад, пропріетарний CryEngine 2, за допомогою якого був створений бестселер Crysis, міг бути використаний тільки для створення ігор під платформу Micorsoft Windows. Unity і Unreal Engine підтримують майже всі існуючі платформи.

2.4 Засоби розробки і їх інтеграція

Для розробки комп'ютерних ігор використовуються різні технології: Adobe Flash un ActionScript, HTML, CSS, AJAX, Java, Unity3D, Adobe (Macromedia) Director, C ++, PHP, ASP .NET, Microsoft Silverlight і інші. Історично найбільш популярними були Flash і Java, так як використовуючи ці технології, можна було легко створювати інтерактивні ігри та анімацію. Однак, слід зазначити, що Adobe Flash була більш популярна при розробці мережевих ігор, ніж Java.

Однак багато хто з них (наприклад Adobe Flash) застаріли і на зміну їм прийшли більш потужні засоби.

Unreal Engine

Редактор Unity Engine - це єдине середовище розробки, що підтримує повний процес створення віртуального світу. Побудова тривимірного простору здійснюється безпосередньо всередині редактора, програмування логіки - в інтегрованому середовищі MonoDevelop на мовах JavaScript або C #. Виняток становить моделювання складної геометрії: в Unity Engine можливо лише створення графічних примітивів (куб, циліндр, сфера) і об'єднання їх в групи;

інше необхідно імпортувати з тривимірних редакторів. Для цих цілей обраний відкритий і безкоштовний 3D-редактор Blender.

До безперечної переваги як Unity, так і Blender, слід віднести велике і активне співтовариство розробників, завдяки якому можна швидко знайти рішення практично будь-яких виникли при розробці проблем.

Технології зберігання і маніпулювання даними

Одна з основних функцій системи - це обробка і зберігання даних, а також коректне їх відображення при генерації віртуального світу. Для цього використовується технологія доступу до даних Entity Framework, яка дозволяє автоматично генерувати базу даних і всі таблиці на підставі створених розробником сутнісних класів і заповнювати їх початковими даними, якщо такі були їм визначено. Дана технологія контролює всі зміни, виконані в ході розробки системи, на рівні коду і при необхідності змінює структуру бази даних. Вибір Entity Framework визначив і вибір СУБД: Microsoft SQL Server також входить в сімейство технологій від Microsoft і краще за інших гарантує плавну і безпомилкову роботу вищеописаних функцій.

Дані, необхідні для функціонування майбутньої гри «Іспит граючи», зберігаються в базі даних, логічна модель якої наведена на рис. 2.2. Крім того, частина даних зберігається на сервері у вигляді файлів.

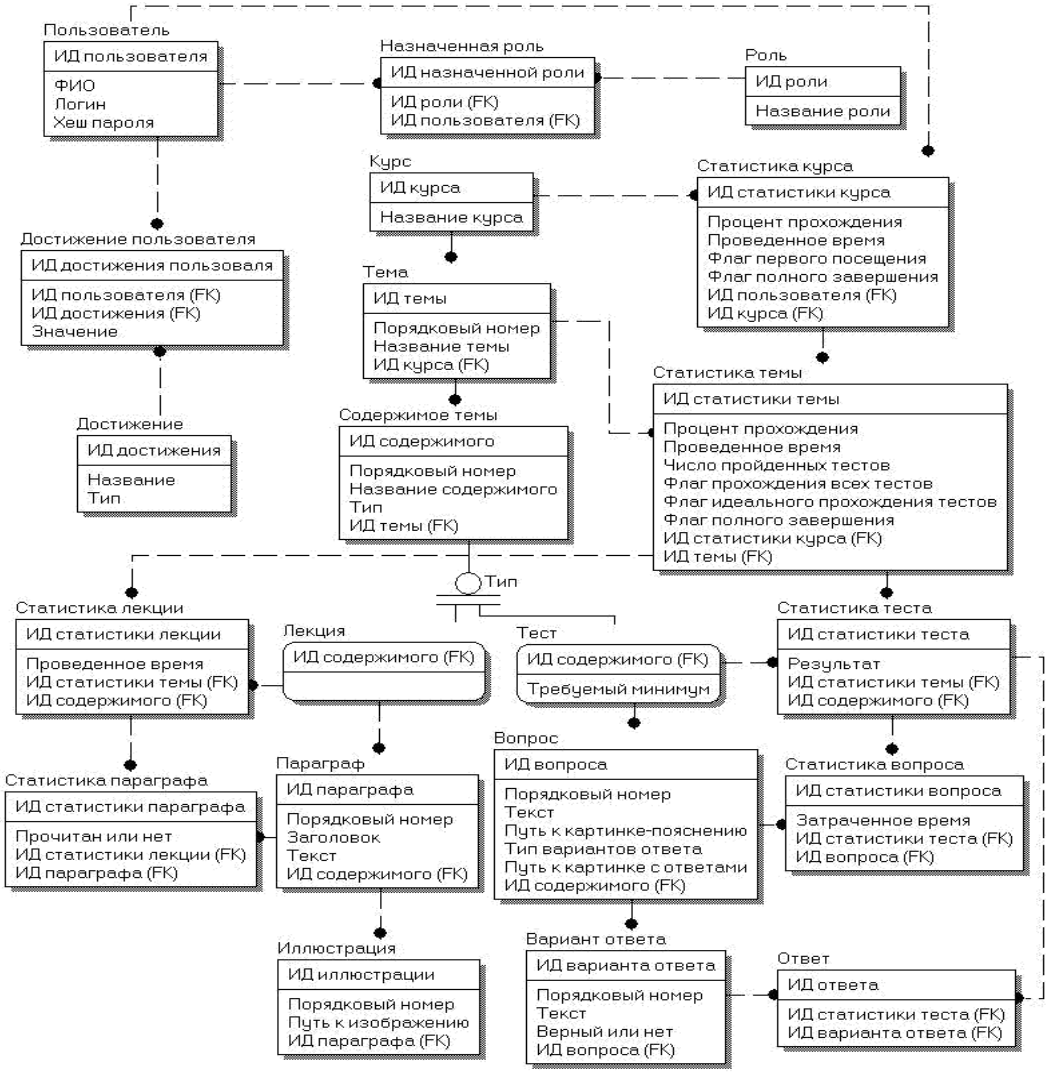

Рис.2.2. Логічна модель бази даних сиситеми

Структура навчального контенту. Основною частиною контенту системи є навчальний курс - сукупність текстових і ілюстраційних матеріалів, згрупованих за тематичними ознаками. Структура курсу має деревоподібну форму і представлена на рис.2.2 - корінь дерева розпадається на теми, що складаються з лекцій і тестів; лекції розбиваються на параграфи, кожний з яких може мати ілюстраціями; тести містять ряд питань з різною кількістю варіантів відповіді, кожен з яких може бути вірним або невірним. Варіанти відповіді можуть бути

представлені або в текстовому форматі, або у вигляді зображення; крім того, ілюстрацією може бути забезпечений саме питання.

За розробку структури навчального курсу відповідає викладач, для цього в передбачається системі є вбудований редактор курсів, що надає інтерфейс для редагування, додавання або видалення будь-якого елементу дерева. Крім того, викладач має можливість завантажити навчальний матеріал з файлу MS Word, який повинен мати

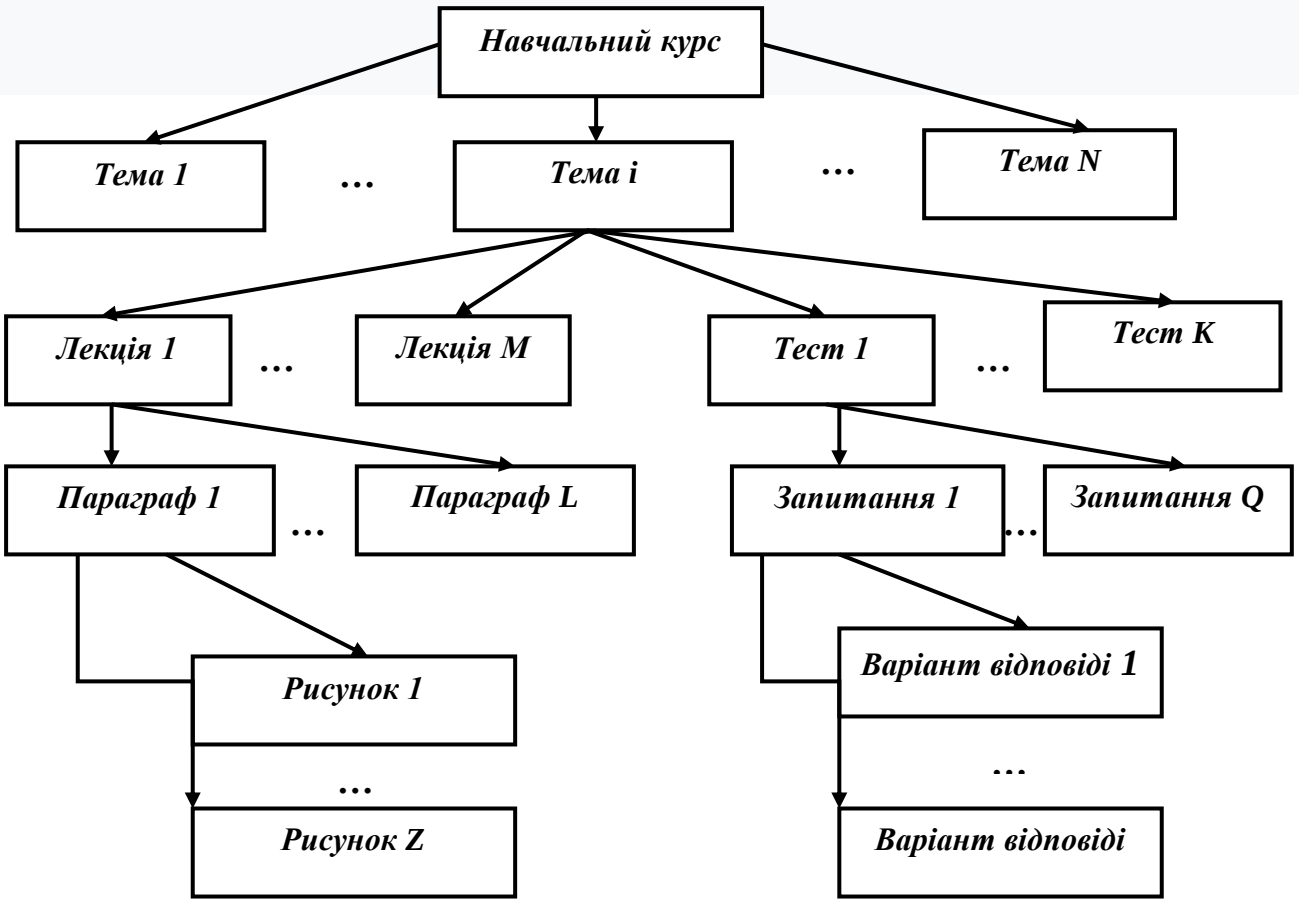

Рис. 2.3. Структура навчального курсу

Структура віртуального простору. Тривимірний навчальний простір складається з двох частин: постійної і динамічної. Постійна частина оформлена у вигляді холу, містить вступну інформацію про університет, факультети, кафедри і розробників, а також включає термінал для вибору курсу зі списку всіх доступних курсів. Динамічна частина являє собою набір пов'язаних кімнат / коридорів та

персонажів і генерується автоматично на основі структури обраного курсу і кімнат-шаблонів, всередину яких завантажується конкретний зміст. Приклад схеми тривимірного простору представлений на рис. 2.4..

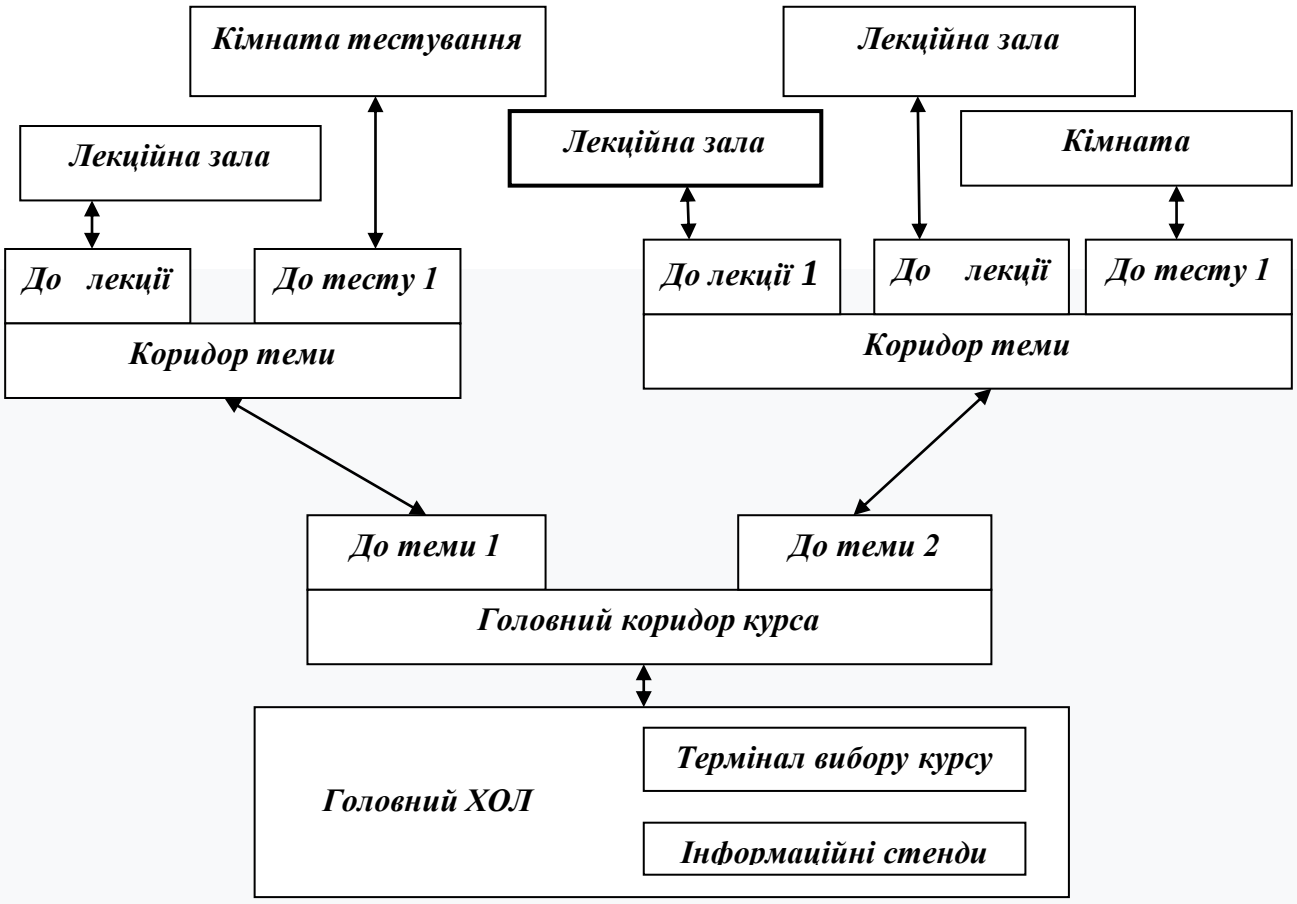

Рис. 2.4.Приклад схеми трьохмірного навчального простору

Взаємодія користувача з об'єктами віртуального світу здійснюється через персонажа-аватара.

Система відстежує прогрес студента всередині тривимірного простору і зберігає його в базі даних, а також відображає статистику його дій, включаючи проведене в світі час, число пройдених тестів і вірних відповідей на питання, а також відсоток завершення курсу / теми.

Якщо процес навчання включає елементи ігрового підходу, в тому числі постійне заохочення, засноване на системі балів досвіду з рольових відеоігор; балами нагороджуються будь-які дії користувача, починаючи від незначних

другорядних (читання вступної інформації, дослідження світу) і закінчуючи безпосередньо навчальними (читання лекцій, проходження тестів).

Архітектура системи. Майже всі віртуальні навчальні середовища базуються на технології клієнт-сервер, тому гра «Іспит граючи» не є винятком. Архітектура системи представлена на рис. 2.5.

Клієнтська частина передбачає просто веб-браузер, який використовується для перегляду сторінок на сервері (користувачеві тільки необхідно встановити на своєму комп'ютері невеликий плагін Unity Web Player). 3D-сцени віртуального світу інтегруються в HTML-сторінки, тому навчають, може переміщатися по віртуальному простору точно так же, як за звичайними Інтернет-сайтам.

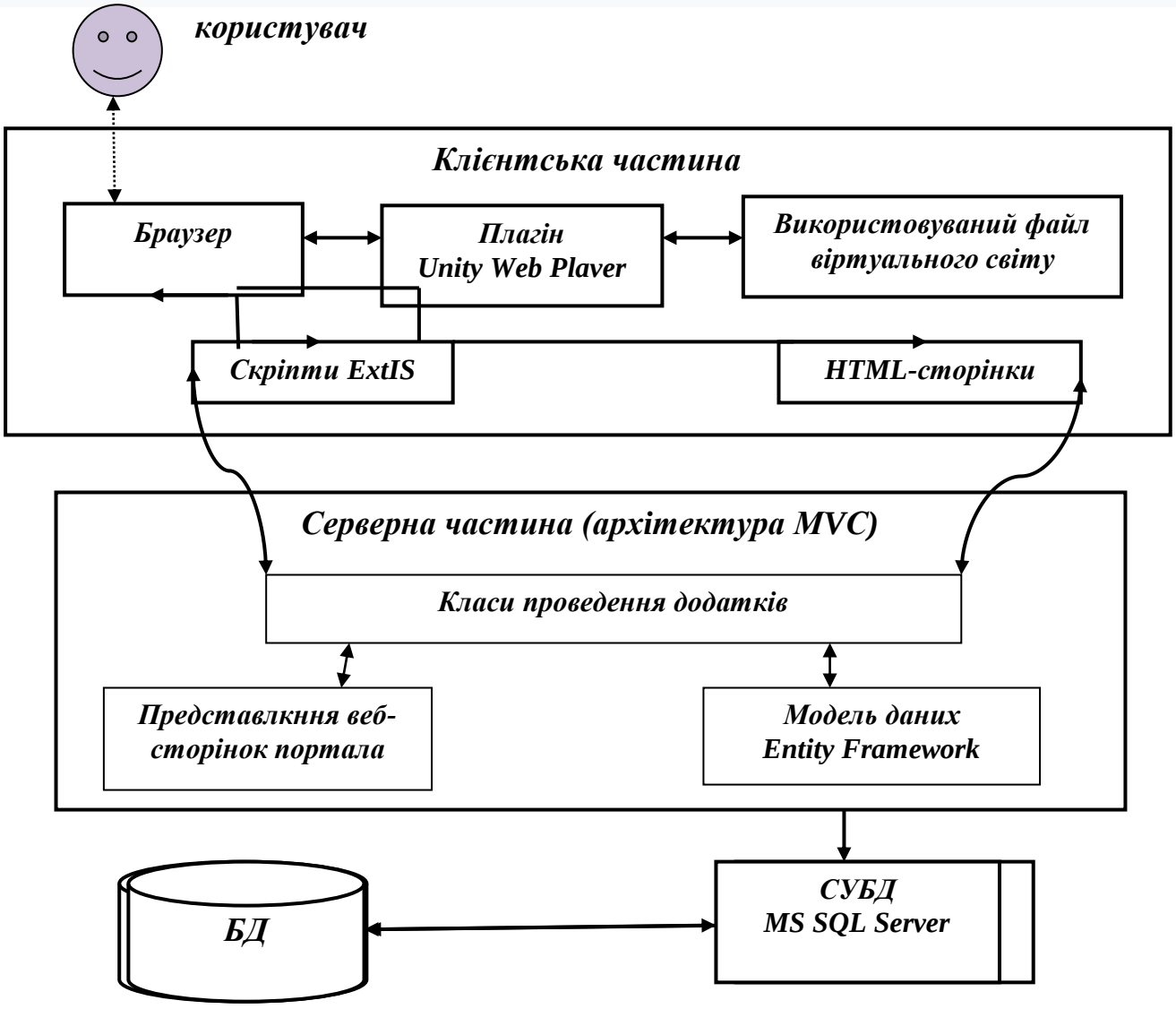

Рис.2.5.Архітектура розроблюваної системи- гри «Іспит, граючи»

Серверна частина системи реалізує архітектуру MVC, яка визначає три рівні:

- рівень представлення веб-сторінок порталу;
- рівень бізнес-логіки програми та доступу до даних;
- рівень даних.

Як протоколу обміну даними по мережі використовується протокол TCP / IP. Чотири контролера з групи поведінки обслуговують чотири сторінки сайту з групи уявлення. Головним компонентом моделі (рівень даних) є контекст бази даних; в ньому дається перерахування всіх сутнісних класів, включених в модель, і через нього працюють з базою даних все контролери.

2.5.Інструментарій роброблки системи

Інтерфейс Unity складається з декількох частин: вкладка Scene, вкладка Game, панель інструментів, вкладка Hierarchy, панель Inspector, вкладки Project і Console. У кожній частині є власне призначення, при цьому всі вони грають важливу роль в процесі створення гри. На рисунку 2.6. зображений інтерфейс програми UnityEnige .

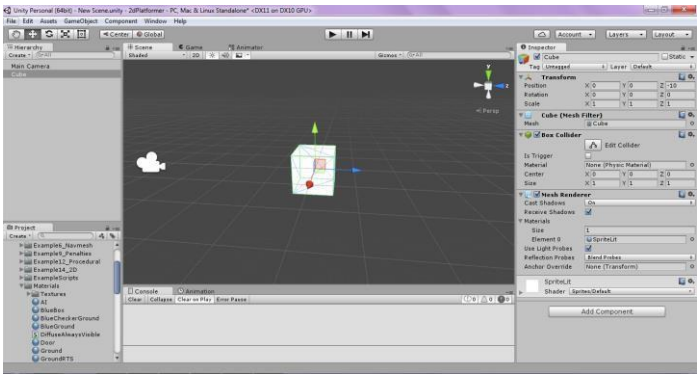

Рис. 2.6 – Частини інтерфейсу Unity Enige

Хоча компоновка ігрових ресурсів відбувається в візуальному редакторі, потрібний керуючий ними код, що забезпечує інтерактивність гри. Unity підтримує всього 2 мови програмування JavaScript і C #.

Програмування здійснюється не всередині Unity Enige, код існує у вигляді окремих файлів, місце розташування яких ви повідомляєте Unity Enige. Файли сценаріїв можуть створюватися в додатку Unity, але в будь-якому випадку вам буде потрібно текстовий редактор або IDE, де буде писатися код для цих спочатку порожніх файлів. У комплекті з Unity поставляється додаток MonoDevelop як міжплатформене інтегроване середовище розробки (IDE) для мови C # з відкритим вихідним кодом. Більш детальну інформацію про цю програму можна отримати на сторінці www.monodevelop.com, але при цьому необхідно використовувати версію, що йде в комплекті з Unity, а не скачаним з цього сайту додатком, так як в базову програму було внесено ряд змін для кращої інтеграції з Unity Engine.

2.6. Висновки до розділу 2

В даний час віртуальна навчальна система – гра «Іспит граючи» знаходиться в стані проектування., але архітектура програмного забезпечення має чітку визначену структуру

Основні аспекти розробки ігор в цілому і на Unity Engine дає підставу зробити висновок, що він показав себе як дуже гнучкий, зручний і простий інструмент для створення ігор, що не вимагає установки додаткового програмного забезпечення (для розробки гри використовувався тільки Unity), при цьому значних недоліків при роботі з ним помічено не було. Unity Engine постійно розвивається і досить часто виходять нові версії (в яких з'являються нові можливості, а також поліпшуються старі), тому деякі аспекти даної роботи будуть неактуальні через деякий час. Встановлено можливість впровадження систем

віртуальної і доповненої реальності для створення освітніх ігор. Визначено особливості розробки ігор з прив'язкою до об'єктів реального світу і карти.

Розділ 3
РОЗРОБКА ВІДЕОГРИ НА UNITY Engine

Інструмент Unity Engine надає початківцям розробникам можливість негайно приступити до роботи, але перед цим необхідно все ретельно продумати і обговорити всі нюанси. Наявність такого гнучкого інструменту як Unity Enige не вирішує всі проблеми і тому необхідно чітко уявляти, що саме повинно вийти в результаті.

3.1 Розробока гри «Іспит, граючи»

Концепція гри. Для вивчення принципів роботи з інструментом Unity Engine розроблено відеогру з доповненою реальністю, що використовує служби геолокації для прив'язки інформації (модель вченого-покемона) до реальної точки на місцевості.

Завданням групи студентів буде створити свою віртуальну команду відомих вчених-екзаменаторів (аналогів-покемонів). Протягом цієї пригоди ви зможете дізнатися про галузь науки або певну дисципліну, виконати місії, допомогти тим, хто потребує, зібрати предмети та артефакти та зв’язатися зі своїми друзями!

Ви зможете зібрати команду відомих науковців з певної царини науки, але для цього вам доведеться точно відповісти на деякі питання, виконати завдання та тести, що стосуються цієї дисципліни.

Ігровий процес і механіка. Здебільшого графіка не має бути ключовим компонентом цієї гри, спрощена естетика складає жанр сам по собі, що, безумовно, подобається деяким людям. Подібні ігри просто не можна порівняти з іншими іграми на основі геолокації.

Механіка гри може бути різною – або ви, як зазвичай просто показуєте пальцями по екрану, щоб оглянути карту, або буквально переміщаєте телефон, щоб обертатися навколо карти, що робить взаємодії дуже цікавими!

Якщо говорити про взаємодії, то всі взаємодії в грі стосуватимуться різних галузей Науки та Техніки. Чи то через перегляд того, що ми будемо називати "освітніми" відеороликами чи вивченням матеріалу підручників, щоб збирати "Аури" або очки та бали через відповіді на окремі питання або виконання тестів, кожна взаємодія стосуватиметься певонї галузі науки і техніки.

Зміст питань і тестів залежить від того, здає студент іспит з певної дисципліни чи проводиться перевірка залишкових знань студнта з певної спеціальності (тобто комплексні питанння та тести). Створення контенту тестових завдань є окремим завданням. Можна використовувати відомі он-лайн сервіси, подібні до наведених в Додатку Д. Наприклад, Ньютон може спитати щодо поліному Ньютона (якщо йдеться про обчислювальну маьематику) або про закони механіки (якщо про Прикладну механіку). У випадку швидкої відповіді або розв’язку теста вчений надасть вам підказку щодо місця знаходження наступнорго вченого (покемона). Для відпочинку і обмірковування відповідей в грі треба передбачити у людних місцях «покезупинки», де можна знайти цінні предмети

У вашого персонажа (якого ви можете обрати з поміж велиикої кількості аватарів) є духовність, здоров'я, голод та спрага. Протягом усіх ваших пригод ви повинні гарантувати, що тримаєте ці показники на гідному рівні. Для цього можна слухати різні розповіді про історію або філософію науки, допомогати друзям, тощо.

Тоді як гравець пересувається реальним світом, його аватар подорожує картою завдяки використанню GPS. Якщо поряд знаходиться вчений-покемон, гра повідомляє про це і пропонує відшукати його, користуючись камерою смартфона чи планшета. Зображення покемона накладається на картинку навколишнього світу, отриману з камери. Іноді очки аури та бали відшукуються

так само, як і покемони, за них відбуваються внутрішньоігровий обмін їх на корисні речі, які прискорюють прогрес гравця. Всі такі обміни і знахідки складаються до інвентаря, який вміщує певну кількість (500 у грі Покемон го) предметів. Покемони-вчені мають різну складність тестів, позначену кольором кільця навколо них.

Знаходячи або вирощуючи нових покемонів, розвиваючи їх і беручи участь у змаганнях, гравець віртуально збільшує свій рівень розвитку, отримуючи нові можливості. Йому стає до снаги знаходити все сильніших вчених-покемонів (з більш рейтинговими або простішими питаннями -тестами), винагорода зростає.

Досягши певного рівня, він може долучитися до однієї з команд «тренерів», які змагатимуться між собою.

Інші особливості. Гра дозволятиме спілкування між гравцями в грі, що досить цікаво. Ви можете з'єднуватися із співгрупниками, спілкуватися з ними тощо.

Гра не є суто розважальною, її метою є досягти того, щоб перевірка знань проходила у цікавій і активній формі. При її плануванні не малось на увазі, що гра охопить людей поза громадою кампуса університету, але для тих, хто цікавиться наукою і певною цариною техніки, це здається хорошим способом перевірити свої знання та повеселитися з однолітками!

Додаток передбачає прогулянку кампусом університету для зустрічі та запрошення якомога більшої кількості ігрових персонажів на eTeam; додаток має використовувати соціальну взаємодію, текстові повідомлення та відображати фізичні зустрічі користувачів у різних місцях, щоб здійснювати допомогу членам своєї команди або інші завдання, змагаючись з ними і залучаючи інші eTeams.

Здоров'я вимірюється набутими рівнями води, хліба та духовності.

У додатку показано різні рейтинги гравців, і гравці також можуть запросити інших реальних людей, а також персонажів, яких вони зустріли, бути в їхніх командах знавців певних розділів науки і техніки.

3.2. Процес разробоки гри на движку Unity Engine. Створення карти

Встановити Unity Engine на робочий комп'ютер доволі просто (Див. Додаток А) Для даного проекту досить безкоштовної версії. Після натискання на кнопку «безкоштовне завантаження» відбувається скачування установника Unity Engine. Цей інсталятор важить менше 1 Мб і всі необхідні файли скачує після початку установки.

Перебираємо всіх персонажів-покемонів, зберігаємо дані про покемонів в змінних, (атрибути елементів приходять нам в файлі, ми їх перебираємо і зберігаємо в змінних).

Рис. 3.1 – Папки завантажені в Unity Engine

Створюємо класи для зручного форматування даних, створюємо скрипт, моделі не повинні успадковуватися ні від чого.

```
public class PokemonModel
{
    public int Type { get; set; }

}
```

Беремо з нього type *int* . Створюємо новий клас (енуматор). одаємо туди імена персонажів-покемонів

```
public enum ETypes
{
    Bulbasaur,
    Charmander,

}

public class PokemonModel
{
    public ETypes PokemonType { get; set; }
    public int MyProperty { get; set; }

}
```

Задаємо позицію і характеристики покемонів

Lat - позиція x

Lon - позиція y

Exp - рівень

Damage - шкоди

Health - життя

Заносимо вручну ці параметри покемонів в файл xml

```
<?xml version="1.0" encoding="UTF-8"?>
<pokemons>
<pokemon type="0" lat="42.3637" lon="-71.05686" id="0" exp="0" damage="10"
    health="100"></pokemon>
<pokemon type="2" lat="42.3657" lon="-71.05686" id="1" exp="0" damage="10"
    health="100"></pokemon>
<pokemon type="2" lat="42.3687" lon="-71.05686" id="2" exp="0" damage="10"
    health="100"></pokemon>
</pokemons>
```

Створюємо List покемонів, яких отримано з сервера.

```
1 reference
public List<PokemonModel> PokemonModels { get; set; }
```

Створюємо module персонажі –покемони. Передаємо в нього атрибути. Конвертуємо srting в тип int, після цього додаємо *id, Lat* . *Lon* типу Single (float) Damage Exp Health конвертуємо в тип *int*.

```
    PokemonModel pokemonModel = new PokemonModel();
    ///<pokemon type="0" lat="42.3637" lon="-71.05686"></pokemon>
    int pokemonTypeInt = System.Convert.ToInt32(item.Attribute("type").Value);
    pokemonModel.PokemonType = (ETypes)pokemonTypeInt;

    pokemonModel.Id = System.Convert.ToInt32(item.Attribute("id").Value);

    pokemonModel.Lat = System.Convert.ToSingle(item.Attribute("lat").Value);
    pokemonModel.Lon = System.Convert.ToSingle(item.Attribute("lon").Value);

    pokemonModel.Exp = System.Convert.ToInt32(item.Attribute("exp").Value);
    pokemonModel.Damage = System.Convert.ToInt32(item.Attribute("damage").Value);
    pokemonModel.Health = System.Convert.ToInt32(item.Attribute("health").Value);

    PokemonModels.Add(pokemonModel);
}
```

створюємо Prefabs

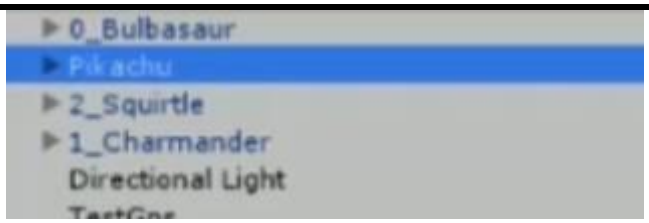

Рис. 3.2 – Меню з вибором префаба

І додаємо їм компоненти

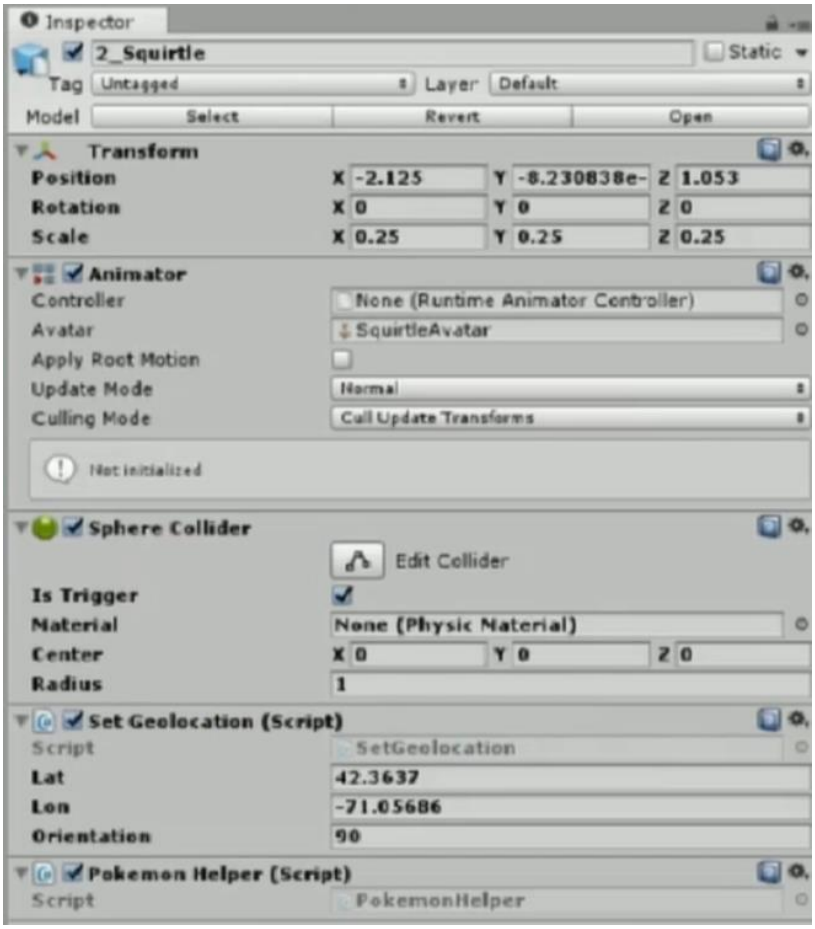

Рис. 3.3 – Компоненти префабів

Так виглядає папка Prefabs (база даних покемонів)

Рис. 3.4. – Моделі покемонів

Перебираємо PokemonModels (моделі покемонів)

```
        for (int i = 0; i < PokemonModels.Count; i++)
        {
            var item = PokemonModels[i];

            GameObject pokemon = Instantiate(PokemonPrefabs[item.PokemonType]);
        }
    }
```

Методом Setlocation отримуємо (item.Lan, item.Lon, item.Origint) після цього кожен покемон стане на потрібну позицію.

```
GameObject pokemon = Instantiate(PokemonPrefabs[(int)item.PokemonType]);
SetGeolocation setGeolocation = pokemon.GetComponent<SetGeolocation>();
setGeolocation.SetLoacation(item.Lat, item.Lon, item.Orint);

PokemonHelper pokemonHelper = pokemon.GetComponent<PokemonHelper>();
pokemonHelper.LoadPok
```

Зберігаємо модель на тому об'єкті, який інстантіейтім

```
public void LoadPokemon(PokemonModel item)
{
    MyPokemonModel = item;

}
```

Ініціюємо масив створених на мапі покемонів, які вже розміщені на мапі

```
1 reference
public List<PokemonHelper> PokemonHelpers { get; set; }
```

Отримуємо доступ до game_Helper для коррдінат GPS

```
GameHelper _gameHelper;
```

Створюємо перевірку для виконання

```
while (!_gameHelper.GpsFix)
{
    yield return null;
}
```

Перевіряємо роботу нашого додатку

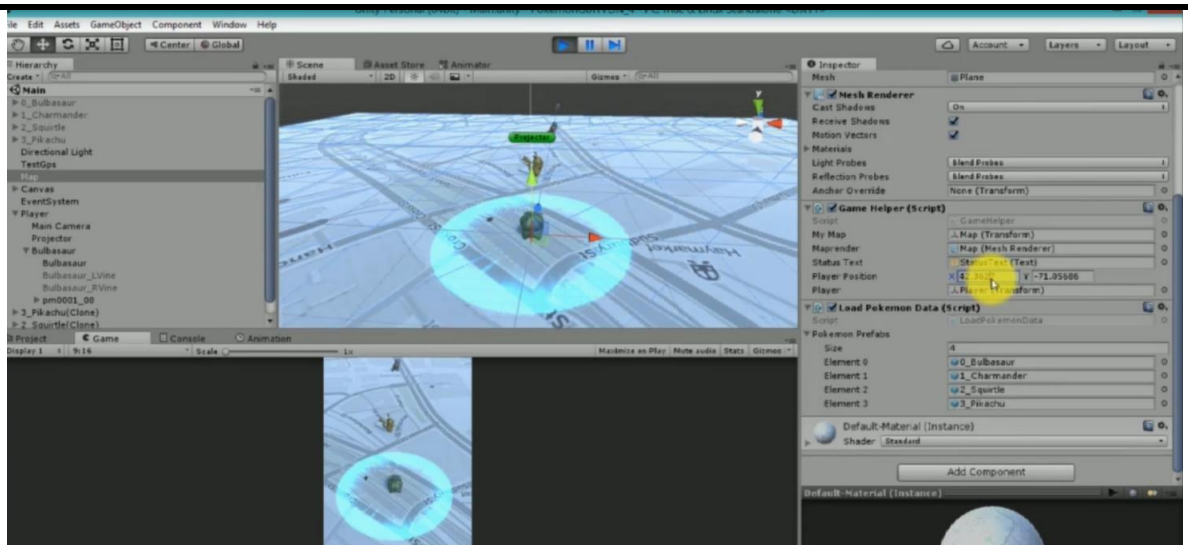

Рис. 3.5. Відображення додатку з покемонами у вікні Unity Enige

Решта атрибутів кожного персонажа-покемона наведена у Додатках Б, В

3.3. Збереження бази даних в Unity Engine

Для збереження бази даних в Unity використовувалаися вбудовані PlayerPrefs і sqlite. Sqlite тепер можна використовувати з коробки в безкоштовній версії. Були зручні sql таблиці для зберігання даних про курси, викладачів та студентів. Найбільш зручним інструментом для управління таблицями є sqlitebrowser.

Запис і читання в PlayerPrefsмає наступний вигляд:

```
public class DB : MonoBehaviour {

 public int currentShipBodyState
 {
get { return PlayerPrefs.GetInt("currentShipBodyState"); }
set { PlayerPrefs.SetInt("currentShipBodyState", value); }
 }
 }
```

з наступними елементами та записом:

```
db.currentShipBodyState = 100;

int shbs = db.currentShipBodyState;
```

Для використання Sqlite в редакторі досить створити каталог StreamingAssets і помістити туди файл бази даних. Для того щоб працювати з базою даних на android пристрої, файл бази потрібно помістити в Application.persistentDataPath. В офіційній документації характеризується як загальнодоступний каталог який також не буде перезаписано при оновленні програми. Також необхідно створити каталог Plugins і Plugins \ Android. У перший помістіть sqlite3.dll і System.Data.dll в Plugins \ Android помістіть libsqlite3.so.

Універсальний код для з'єднання з базою даних:

```
public IDbConnection connector()
{
// если игра запускается на ПК
 if (Application.platform != RuntimePlatform.Android)
{
//database.Open(Application.dataPath + "/StreamingAssets/db.sqlite");
dbconn = (IDbConnection)newSqliteConnection("URI=file:" + Application.dataPath +
"/StreamingAssets/db.sqlite");
dbconn.Open(); //Open connection to the database.

}
// android
Else
{

// расположение базы даннях
string filepath = Application.persistentDataPath + "/" + "db.sqlite";

// если базы данных по заданному пути нет, размещаем ее там if
(!File.Exists(filepath))
{

WWW loadDB = new WWW("jar:file://" + Application.dataPath + "!/assets/" +
"db.sqlite");
while (!loadDB.isDone) { }
File.WriteAllBytes(filepath, loadDB.bytes);
 }

dbconn = (IDbConnection)new SqliteConnection("URI=file:" +
Application.persistentDataPath + "/db.sqlite");
 dbconn.Open(); //Open connection to the database.
```

```
}

return dbconn;

}
```

Запити до бази даних, це звичайні sql запити:

```
public List<string> getItemInfoFromIslandByItemName(string item)
 {

 dbconn = connector();
 dbcmd = dbconn.CreateCommand();

string sqlQuery = "SELECT island_resources.amount, island_resources.weight, island_resources.title, island_resources.price, island_resources.sell_price, ship_resources.amount FROM island_resources INNER JOIN ship_resources ON island_resources.title=ship_resources.title WHERE island_resources.title='" + item + "' AND island_resources.island_id=" + zones_filter(current_zone_name) + " ";


dbcmd.CommandText = sqlQuery;
reader = dbcmd.ExecuteReader();

System.Object[] values = new System.Object[reader.FieldCount]; int fieldCount = reader.GetValues(values);

List<string> list = new List<string>();

for (int i = 0; i < fieldCount; i++)
{
list.Add( values[i].ToString() );
}

clean();
return list;
}
```

3.4. Висновки до розділу 3

Розроблена освітня відеогра на основі технології доповненної реальності, що використовує прив'язку до об'єктів реального світу і карти, на кшталт відомої гри Pokemon Go, при доробці якої можна з отримати гіднуосвітню гру.

В Unity з'ясовано роль компаса, акселерометра і гіроскопа для постановки об'єкта на рівень горизонту, Вивчена можлівість портування даної гри під будь-яку платформу (яка підтримує Unity Engine), з невеликими витратами і незначними змінами в коді.

Підтримка бази даних в Unity Engine відбувається доволі просто, за допомогою вбудованої PlayerPrefs і sqlite.

ВИСНОВКИ

Застосування ігрових форм навчання в освітньому процесі є доцільним. Ігри, або ігрові додатки дозволяють ефективно організувати творчу взаємодію викладача і студентів, створюють умови для формування особистісних якостей.

Існує безліч інструментальних засобів для розробки освітніх ігор. Основною концепцією Unreal Engine є використання в сцені легко керованих об'єктів, які, в свою чергу, складаються з великої кількості компонентів. Він показав себе як дуже гнучкий, зручний і простий інструмент для створення ігор, що не вимагає установки додаткового програмного забезпечення (для розробки гри використовувався тільки Unity Engine), при цьому значних недоліків при роботі з ним помічено не було. Unity Engine постійно розвивається і досить часто виходять нові версії (в яких з'являються нові можливості, а також поліпшуються старі), тому деякі аспекти даної роботи будуть неактуальні через деякий час.

В ігрових додатках є можливість застосовувати різні зразки ігор різних жанрів, і визначати критерії, яким мають відповідати освітні ігри. Встановлено можливість впровадження систем віртуальної і доповненої реальності для створення освітніх ігор. Особливістю розробки ігор з прив'язка до об'єктів реального світу і карти.

На перспективу подальшого розвитку системи (ігоровго додатку) слід згадати реалізацію режиму, який дасть системі соціалізуючу складову, а також додавання експертних елементів, за допомогою яких система зможе адаптуватися під успіхи або невдачі конкретного користувача і видавати йому актуальні рекомендації, тим самим роблячи процес навчання більш індивідуальним та інтелектуальним. В додатку з'ясовано роль компаса, акселерометра і гіроскопа для визначення положення пристрою в просторі і швидкості руху персонажа. Вивчена можлівість портування даної гри під будь-яку платформу (яка підтримує

Unity Engine), з невеликими витратами і незначними змінами в коді. Можливе, в продовження розробки, додавання декількох типів покемонів-вчених (або викладачів), використання вже ісеуючих он-лайн сервісів для тетових завдань, доопрацювання меню (додати можливість налаштувань гри), реалізацію системи поліпшень, зробити додаткові режими, додати деякі неігрові персонажі і описати взаємодію з ними, а також доопрацювати сюжет гри.

Стосовно освітнього процессу підготовки студентів на рівні «бакалавр» галузі знань 12 «Інформаційні технології» було б непогано передбачити опрацювання таких тем в окремих дисциплінах «Основи створення ігрових додатків (на базі GameMaker)» (Basics of Game Applications Development with GameMaker Studio), «Особливості тестування ігрових додатків» (Features of Game Applications Testing).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Козловський, Є.О. & Кравцов, Г.М. (2011). Віртуальна лабораторія в структурі системи дистанційного навчання. *Інформаційні технології в освіті*, 10, 102 – 109.
2. Козловский, Е.О. & Кравцов, Г.М. (2012). Объектная модель структуры программного обеспечения виртуальной лаборатории в системе Херсонский виртуальный университет. *Інформаційні технології в освіті*, 12, 55 – 60.
3. Буч, Г. (2008). *Объектно-ориентированный анализ и проектирование*.
4. Соловов, А. В. (2002). Виртуальные учебные лаборатории в инженерном образовании. *Индустрия образования*, 2, 386 – 392.
5. Меньшиков, Д. В., Эйхман, Е. А. & Юн, С. Г. (2011). Основные подходы к разработке системы построения виртуальных моделей и демонстраций. *Новые образовательные технологии в вузе (НОТВ – 2011): сб. материалов восьмой междунар. науч.-метод. конф., 2–4 февр. 2011 г.*, 373– 378.
6. Козловський, Є. О. & Кравцов, Г. М. (2014). Мультимедійна віртуальна лабораторія з фізики в системі дистанційного навчання. *Інформаційні технології в освіті*, 18, 80 – 89.
7. *Unity (game engine).* [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Unity_%28game_engine%29.
8. *Наглядная физика.* [Електронний ресурс]. – Режим доступу: http://www.virtulab.net/index.php?option=com_content&view=section&layout=blog&id=5&Itemid=94 .
9. Лабораторна робота №4. *Школа сучасних знань.* [Електронний ресурс]. – Режим доступу: http://www.zhu.edu.ua/mk_school/pluginfile.php/15939/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B 1%D0%BE%D1%82%D0%B0%20%E2%84%964.pdf.
10. Олифер, В. Г. & Олифер, Н. А. (2006). *Компьютерные сети. Принципы, технологии, протоколы*.
11. Pirker J. Understanding Physical Concepts using an Immersive Virtual Learning Environment / J. Pirker, S. Berger, C. Gutl, J. Belcher, P.H. Bailey // Proceedings of the 2nd European Immersive Summit (Paris, 26-27 November 2012). – P. 183-191.

12. CSE 3541:Computer Game and Animation Techniques [Електронний ресурс]. – Режим доступу: http://web.cse.ohio-state.edu/~wang.3602/courses/cse3541-2015-spring/

13. Kent B.R. 3D Scientific Visualizationwith Blender / B.R. Kent. – San Rafael: Morgan&Claypool Publishers, 2015. – 91 p.

ДОДАТКИ

Додаток А

Інструкція по встановненню Unity Engine на персональний КП

Для початку необхідно встановити Unity Engine на робочий комп'ютер. Для даного проекту досить безкоштовної версії. Після натискання на кнопку «безкоштовне завантаження» відбувається скачування установника Unity Engine. Цей інсталятор важить менше 1 Мб і всі необхідні файли скачує після початку установки.

Крім самої Unity Engine на комп'ютер встановлюється Mono Develop (IDE для написання сценаріїв Unity Engine), він використовується в Unity за замовчуванням, однак в налаштуваннях можна змінити параметр (Рис. А.1) і використовувати Visual Studio (попередньо необхідно встановити плагін для інтеграції з Unity Engine). Там же, як видно з малюнка, необхідно вказати шлях до Android SDK, JDK, NDK, для компіляції гри під ОС Android.

При запуску, Unity Engine пропонує провести авторизацію (це необхідно для роботи з Asset Store з Unity Enige), для цього необхідно спочатку створити обліковий запис (якщо натиснути на словосполучення «create one» в браузері відкриється сайт https://accounts.unity3d.com/sign-up , де і відбувається реєстрація нових користувачів Unity), але, якщо немає необхідності використовувати Asset Store, можна вибрати пункт «Work offline» (Рис. А.3). Далі можна вибрати один з існуючих проектів (поряд з ім'ям проекту написана версія Unity Enige в якій він був створений) або створити новий (Рис. А.4). Для створення нового проекту потрібно натиснути на кнопку «New», після чого потрібно буде ввести ім'я проекту, шлях до нього, вибрати режим 3D або 2D, а також є можливість вибрати стандартні набори ресурсів, для імпорту в проект (Рис. А.5).

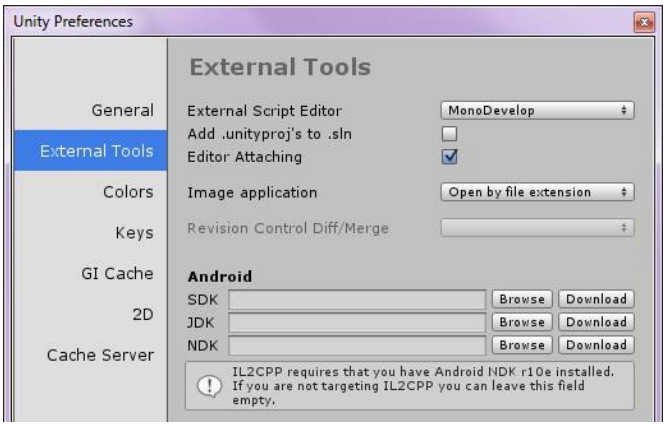

Рис. А.1. Вікно налаштувань Unity Engine

Після створення проекту відкриється порожня сцена, в якій буде тільки один об'єкт - камера. Для початку потрібно зберегти сцену, для цього в меню «File» є пункт «Save Scene». Для того щоб в майбутньому звертатися до сцен на ім'я, краще дати їм зрозумілі імена, тому ця сцена називається «MainLevel». У вікні Project (Рис. А.6) можна працювати з файлами проекту: імпортувати (шляхом перенесення або через пункт "import package» у відповідному меню), створювати файли, папки (для зручності зазвичай створюють окрему папку під кожен тип об'єктів: для скриптів, матеріалів, спрайтів і т.д.), переміщати файли (між папками, в сцену з папки проекту і навпаки).

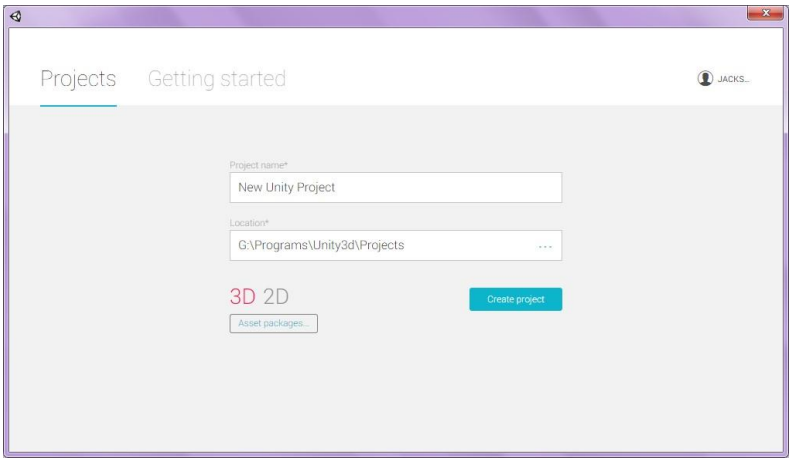

Рис. А.2. Створення нового проекта

Після цього створюється нова папка з ім'ям «Scenes», і в неї поміщається сцена MainLevel. Решта папок створювалися у міру необхідності. Папка Standard Assets містить імпортований з Asset Store однойменний набір.

Завантаження карти . Завантажимо карту, потім поставимо на неї точки появи "покемонів-вчених". Точки будемо вантажити з xml-документа, який покладемо на google-drive. Також зробимо просту взаємодію між покемонами і нейтралами. У грі реалізуємо можливості використання Gps-координати реального телефону.

Ресурс *Map-квест,*

Рис. А.3. Ресурс Map Quest

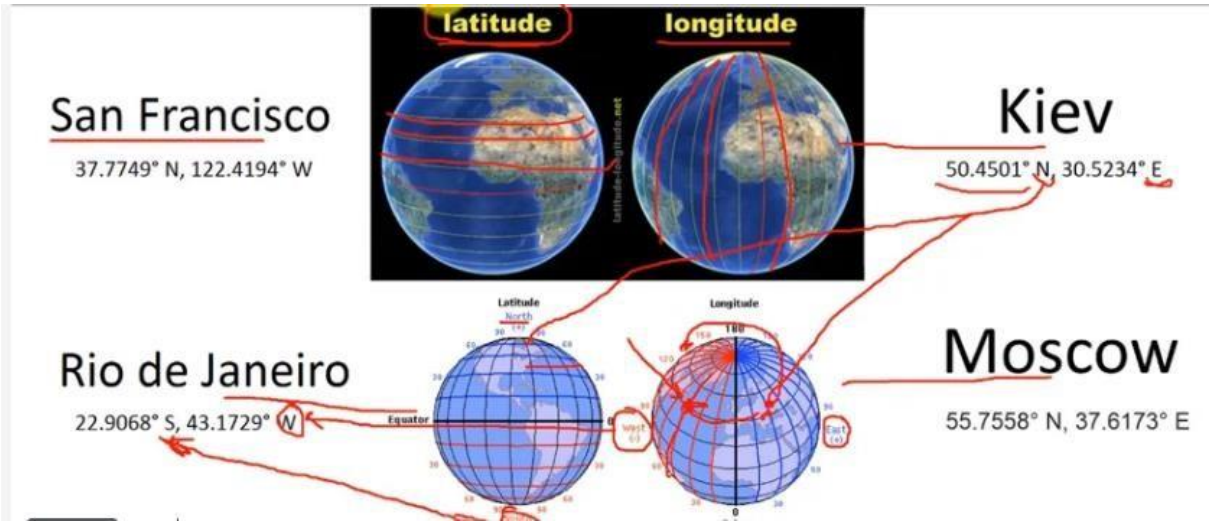

Рис. А.5. Вибір координат міста

Реєстрація на мап квест

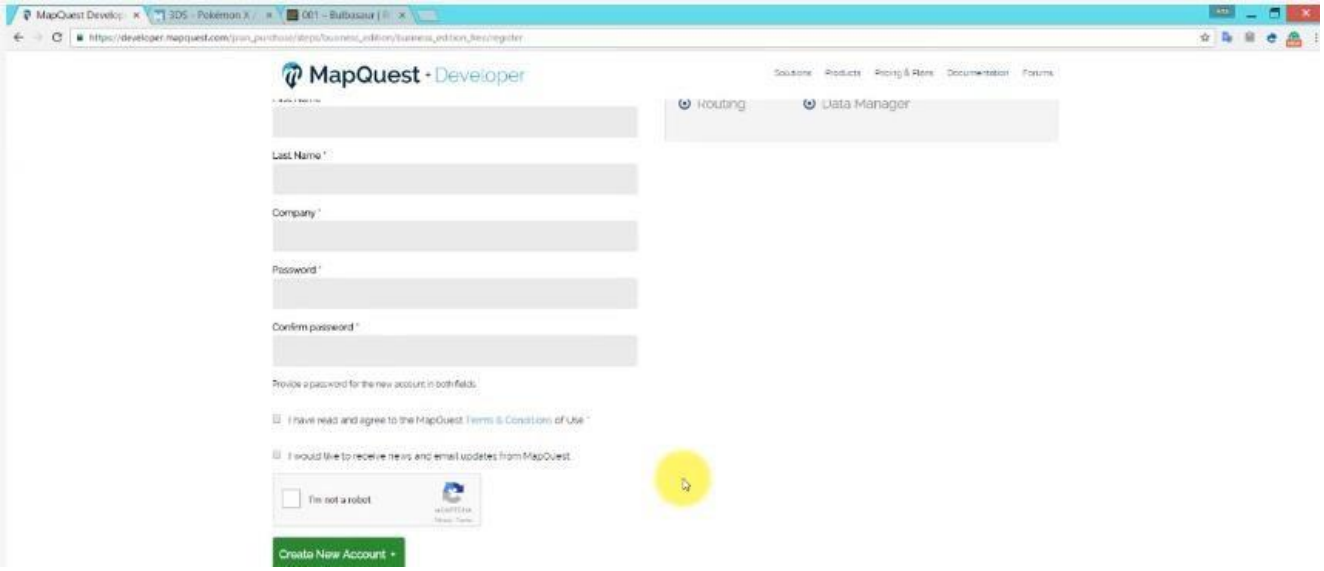

Рис. А.6 – Реєстрація на Map Quest

Вибір *Створення My application*

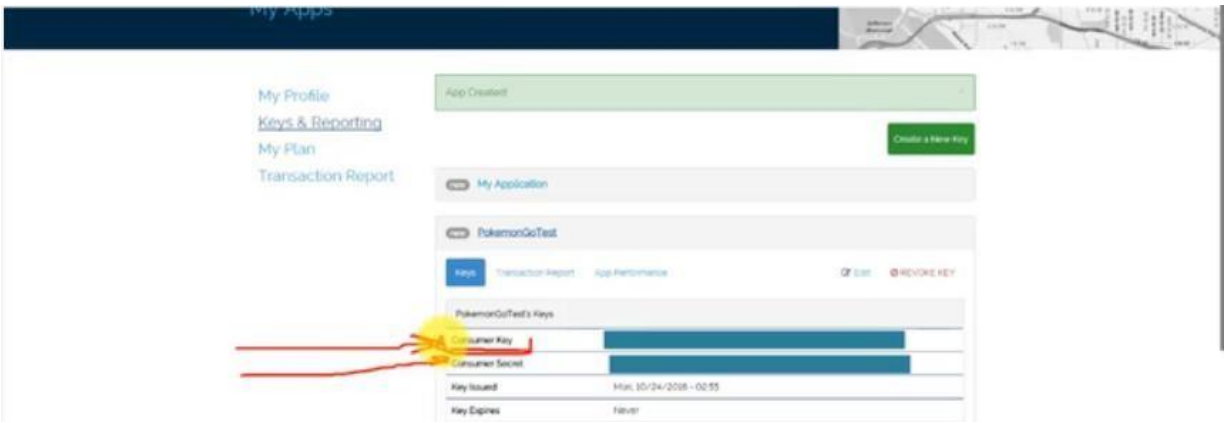

Рис. А.7 – Додаток в Application Creator

Відкриваємо Unity Engine

Рис. А.8 – Проект в Unity Engine

Loadmap.cs

```
public class LoadMap : MonoBehaviour
{
    const string Key = "[redacted]";

    public Renderer maprender;
    public Vector2 =
        new Vector2(42.3627, -71.05686f);  //Latitude, Longitude

    int _zoom = 17;
    string _maptype = "map";
    string _url;

    void Start()
    {
        StartLoadMap(PlayerPosition);
    }
```

Рис. А.9 – Завантаження карти в проект Unity Engine

url –адреса

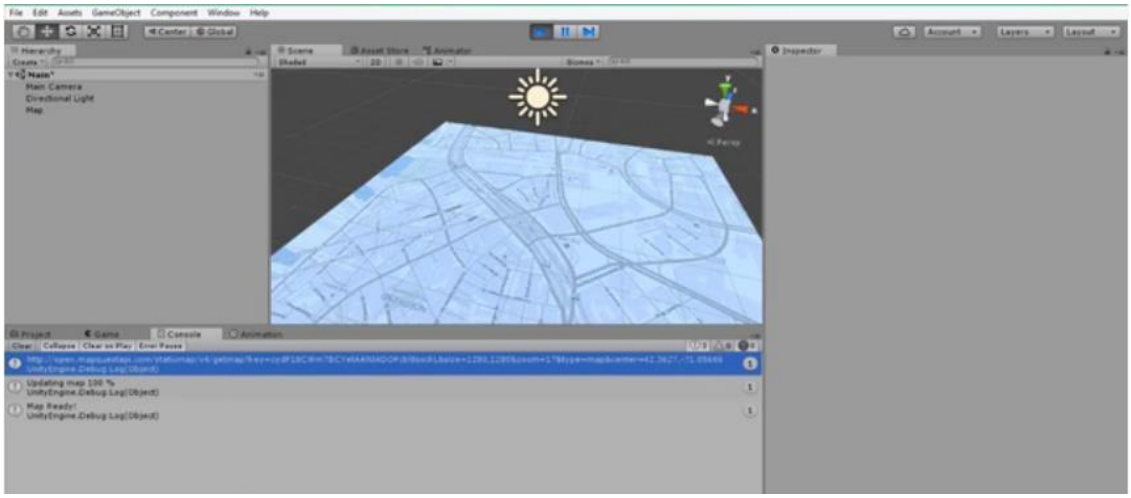

Рис. А.10 – Заиис url адреси в проекті Unity Engine

Здійсюємо load image

```
    {
        _url = "http://open.mapquestapi.com/staticmap/v4/getmap?key="
            + "&size=1280,1280&zoom=" + _zoom
            + "&type=" + _maptype
            + "&center=" + PlayerPosition.x + "," + PlayerPosition.y
        // Debug.Log(_url);

        StartCoroutine(LoadImage());
    }

    private IEnumerator LoadImage()
    {
        WWW www = new WWW(_url);
        while (!www.isDone)
        {
            yield return null;
        }
```

Рис. А.11 – Завантаження зображння карти в Скрипт

Завантаження карти

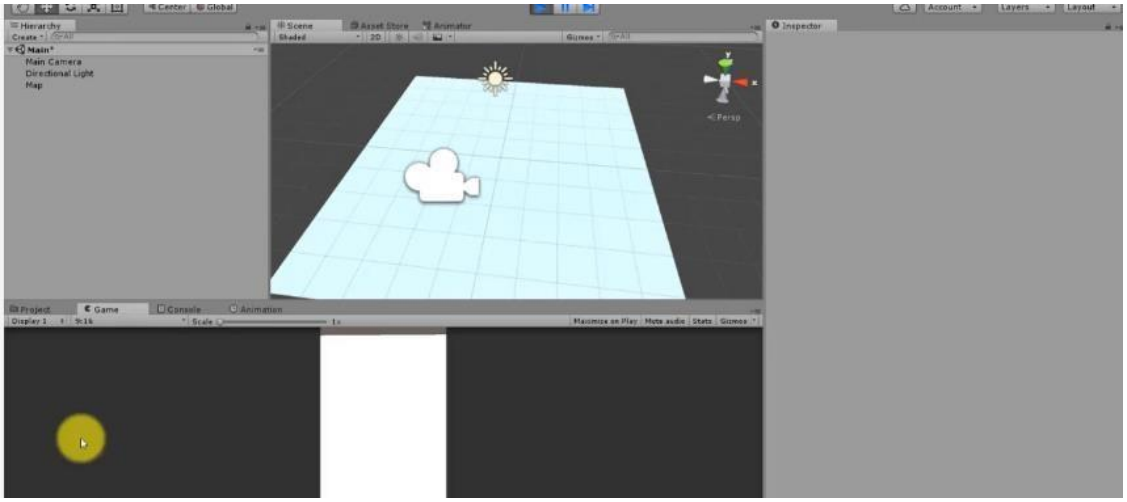

Рис. А.12 – Завантаження карти в проект Unity Engine

Точка на карті google

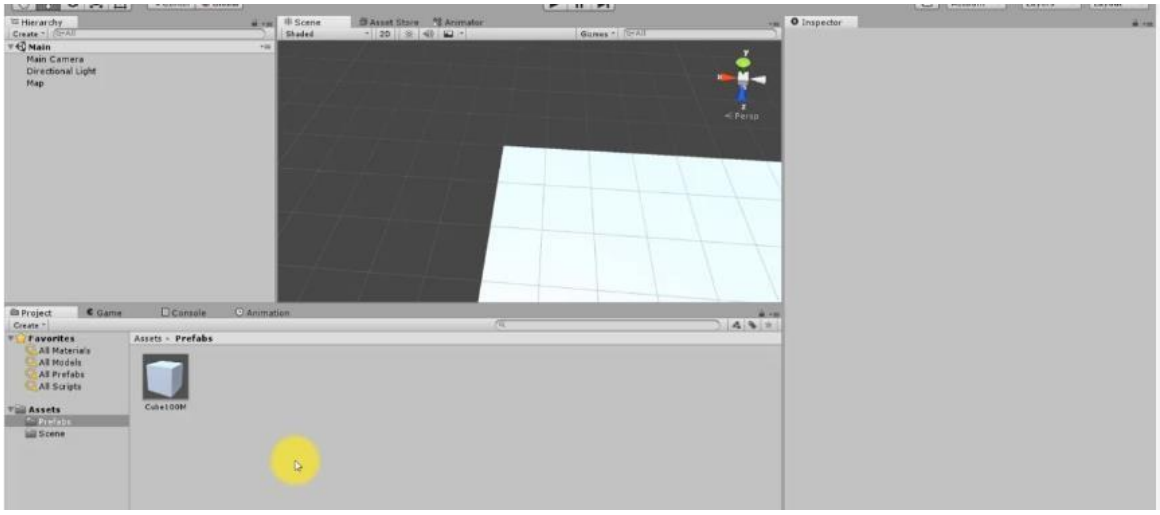

Рис. А.13 – Створення точки на Google Map

Отримання координат ігрока (location service)

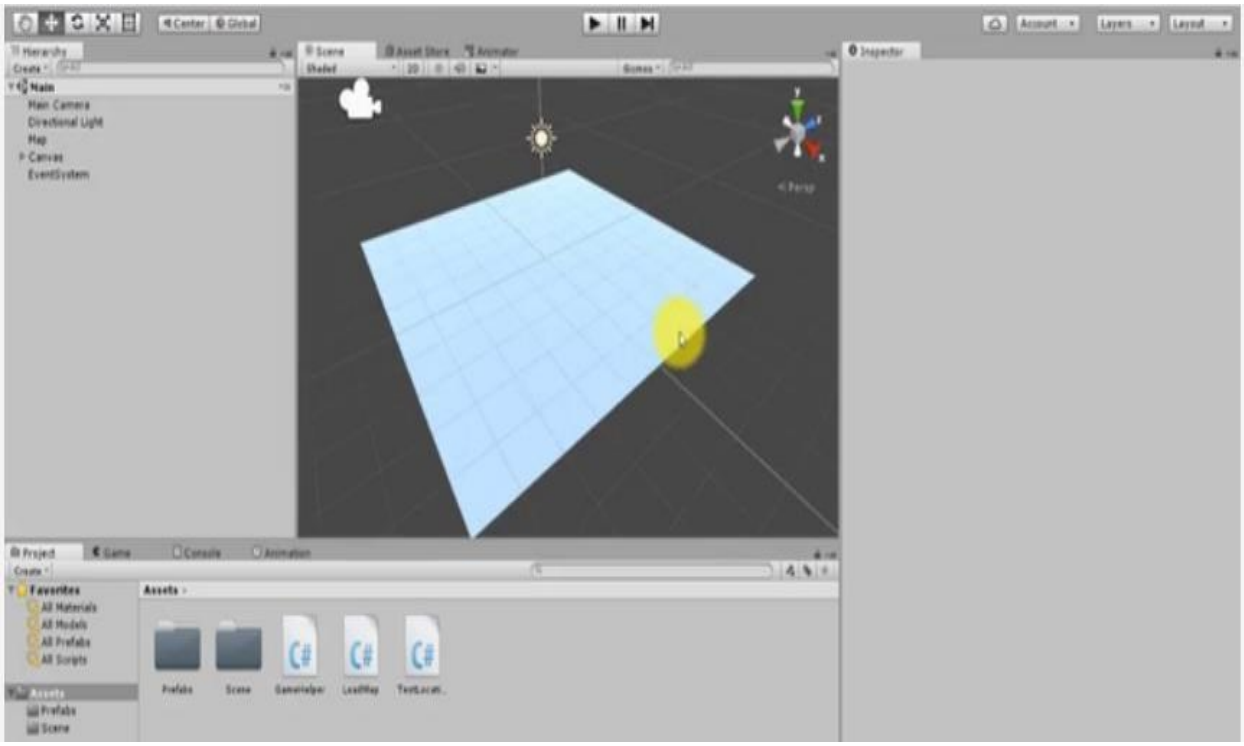

Рис. Б.1 – Занесення скриптів і даних в проект Unity Engine

Скрипт Text.location

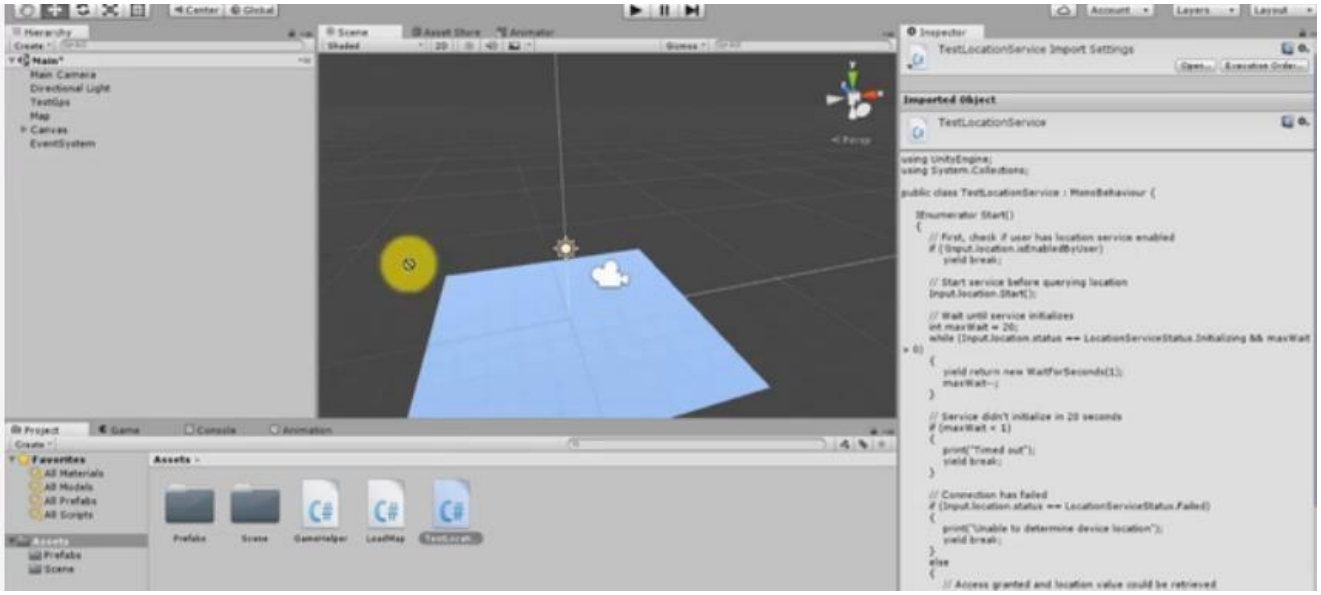

Рис. Б.2. Скрипт Text.location в проекті Unity Engine

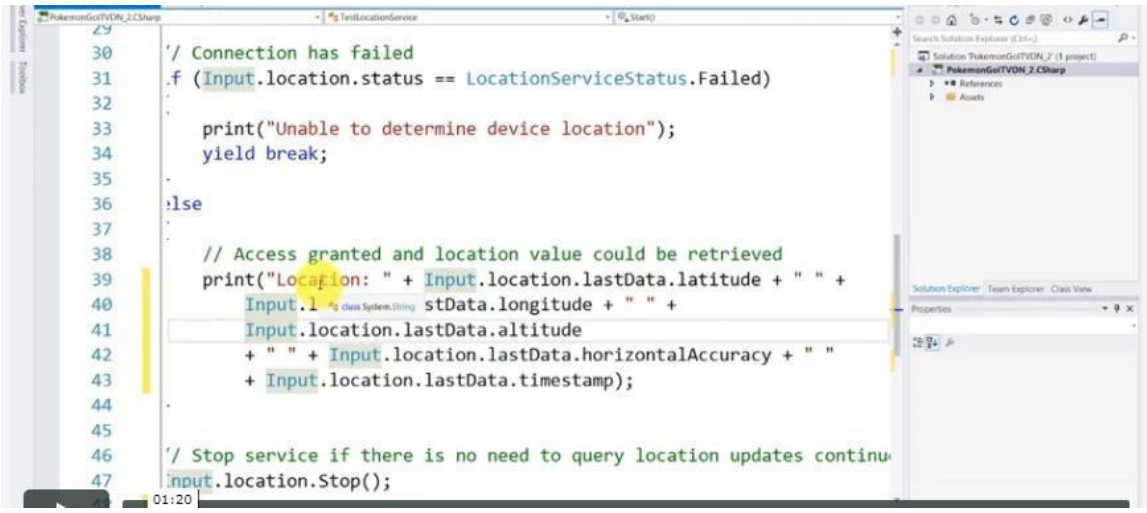

Рис. Б.3 – Скрипт Text.location

Game helper

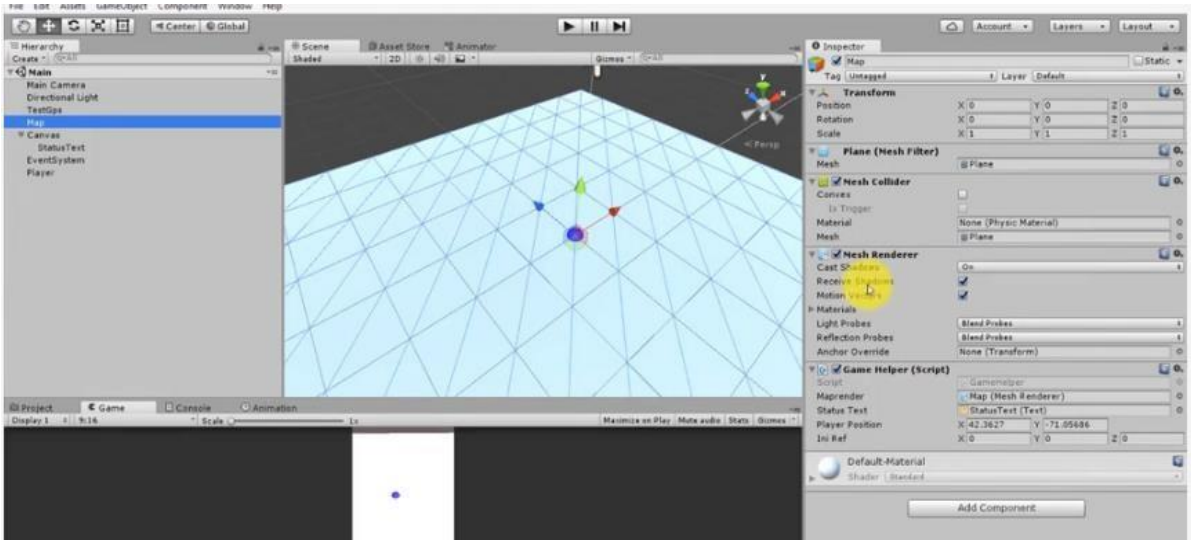

Рис.Б.4 Відображення точки гравця у вікні Unity Engine

Рис. Б.5 – Позиція гравця в скрипті

Location service

LocationService.Start

public void **Start**(float **desiredAccuracyInMeters** = 10f, float **updateDistanceInMeters** = 10f);

Parameters

Description

Starts location service updates. Last location coordinates could be.

Retrieved via `Input.location.lastData`. Service does not start to send location data immediately. Code should check `Input.location.stat` status. desiredAccuracyInMeters - desired service accuracy in meters. Using higher value like 500 usually does not require to turn GPS chip or power. Values like 5-10 could be used for getting best accuracy. Default value is 10 meters. updateDistanceInMeters - the minimum distance (device must move laterally before `Input.location` property is updated. Higher values like 500 imply less overhead. Default is 10 meters.

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

Рис. Б.6 – Вікно Location service

Статус game helper

```
30
31      int _counter;
        0 references
32      IEnumerator Start()
33      {
34
35          Input.location.Start(5, 5);
36          Input.compass.enabled = true;
37          Status = "Initializing Location Services..";
38
39          // Wait until service initializes
40          while (Input.location.status == LocationServiceStatus.Initia
41          {
42              yield return new WaitForSeconds(1);
43              Debug.Log("Wait " + _counter);
44              Status = "Wait " + _counter;
45              _counter++;
46          }
47
48          if ( counter >= WaitTime)
```

Рис. Б.7 Ініціалізація статусу Location service

координати

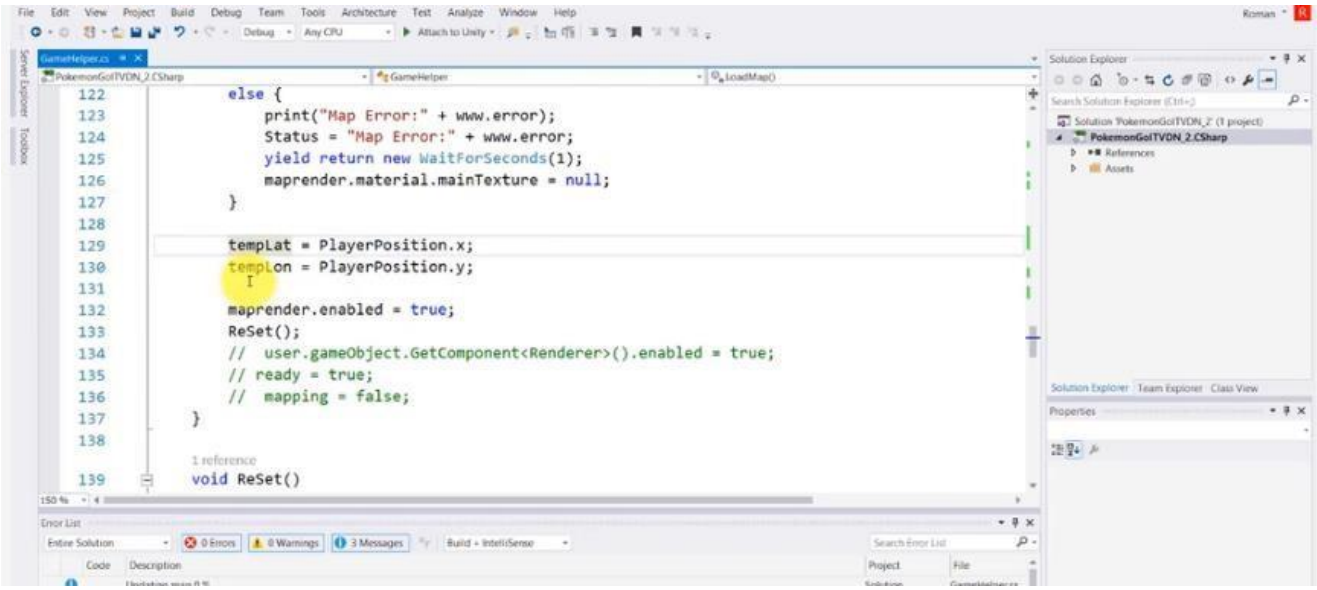

Рис. Б.8 Визначення координат гравця

Отримуємо координати ігрока

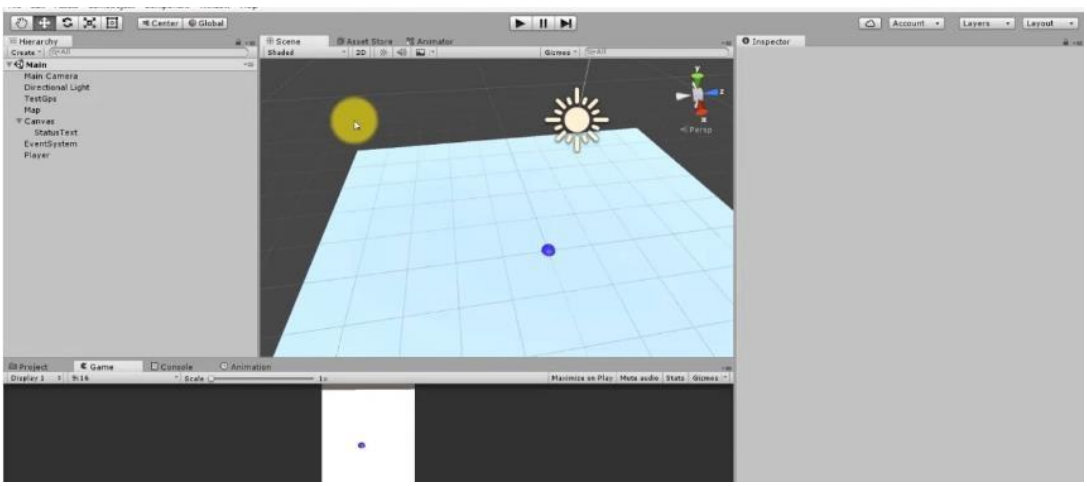

Рис. Б.9 – Гравець на карті у вікні Unity Engine

Сайти з моделями покемонів

Оскільки створювана версія гри є тестовою, то в якості моделей вчених-покемонів будемо використовувати моделі покемонів, які є в достатній кількості і є безкоштовними.

Рис. Б.10 – Модель Ньютона

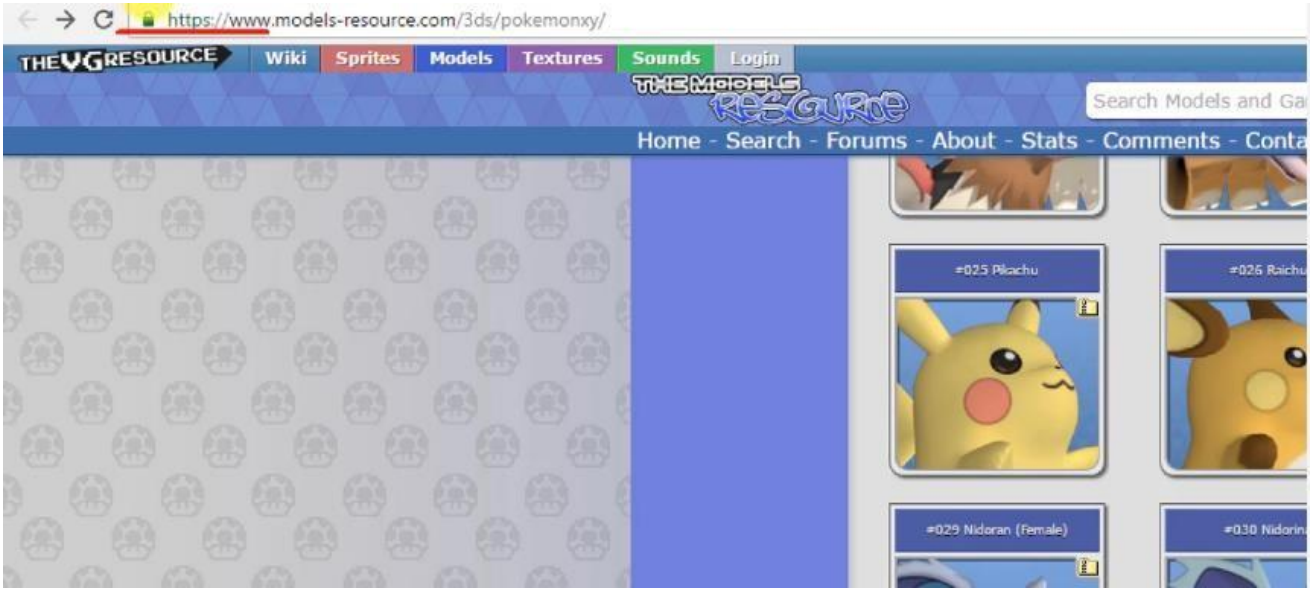

Рис. Б.11 – Сайт Пикачу

Завантажуємо моделі в Unity Engine

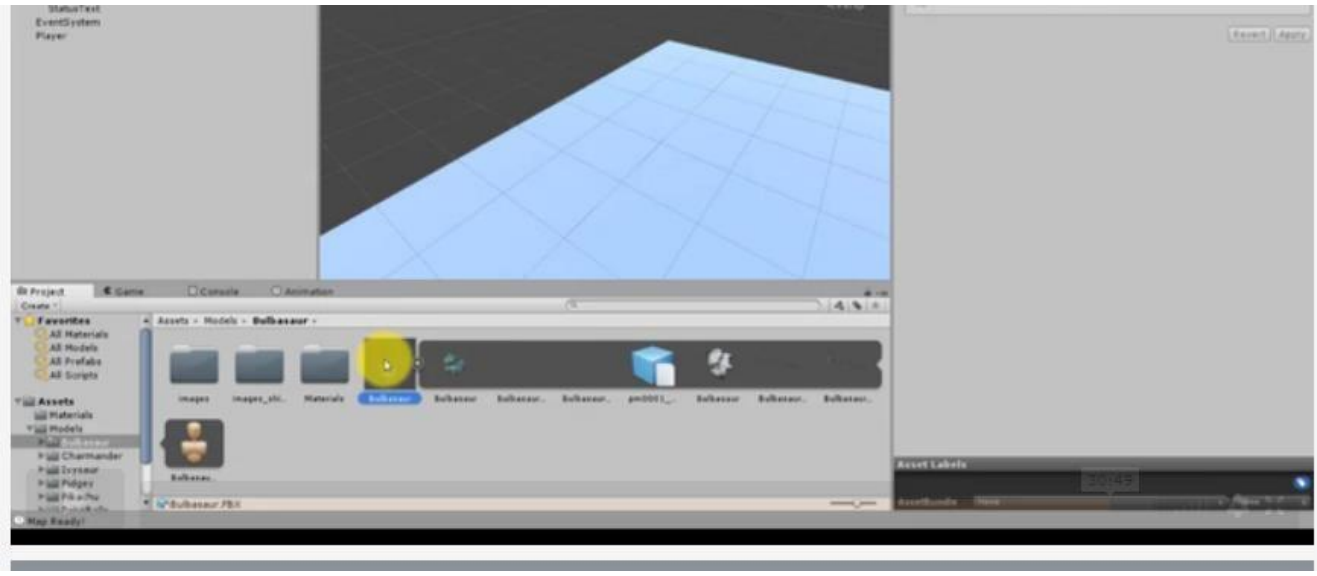

Рис. 3.12 Завантаження моделей в Unity Engine

Моделі на карті

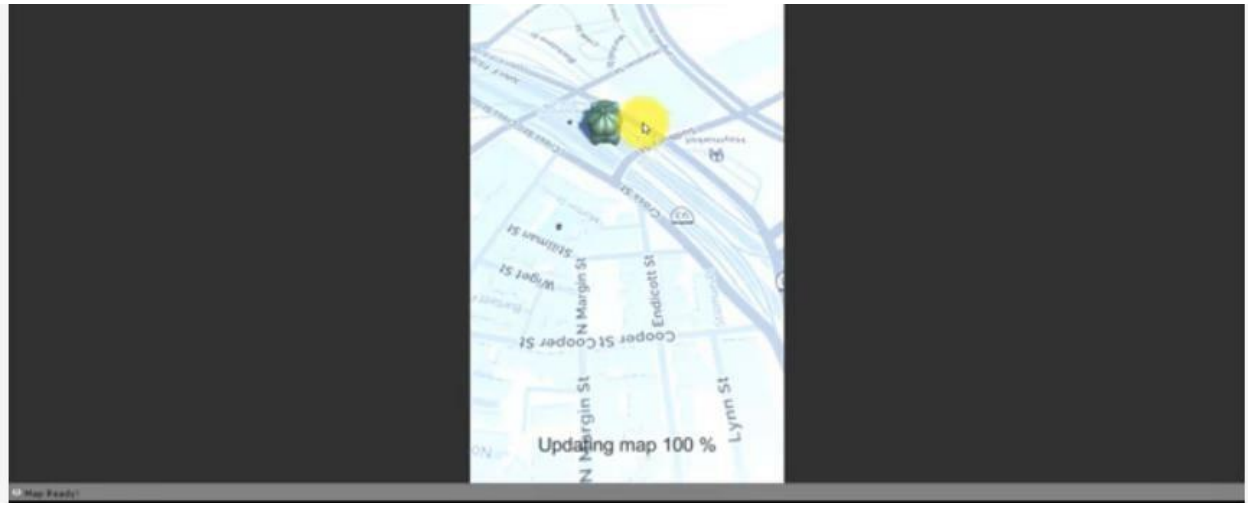

Рис. 3.13 –Розміщення модлей на карті в Unity Engine

Анімація персонажа

Unity Engine надає розробникам відмінний інструмент для роботи з анімацією. У персонажа вже є компонент Animator (рис. В.1). Тепер треба зробити анімації і налаштувати контролер анімації.

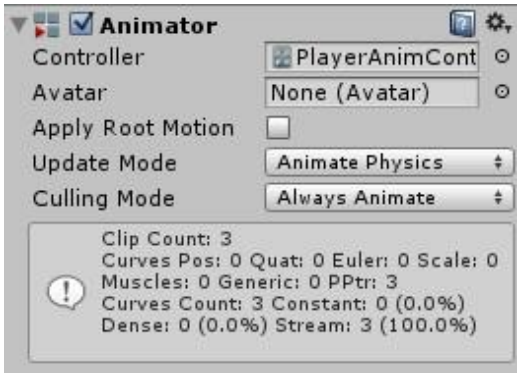

Рис. В.1 Налаштування компонента Animator

Для створення анімації в Unity використовується вікно Animation. Можна анімувати будь-який параметр будь-якого компонента об'єкта, наприклад, його положення (параметр Position компонента Transform), або замінити спрайт в компоненті Sprite Renderer. Вікно Animation складається з часової шкали, де будуть розміщуватися ключові кадри, кнопки: пуск, стоп, запис, додати ключовий кадр, додати подію, до наступного ключового кадру, до попереднього ключового кадру, а також зі списку аніміруємих параметрів (рис. В.2).

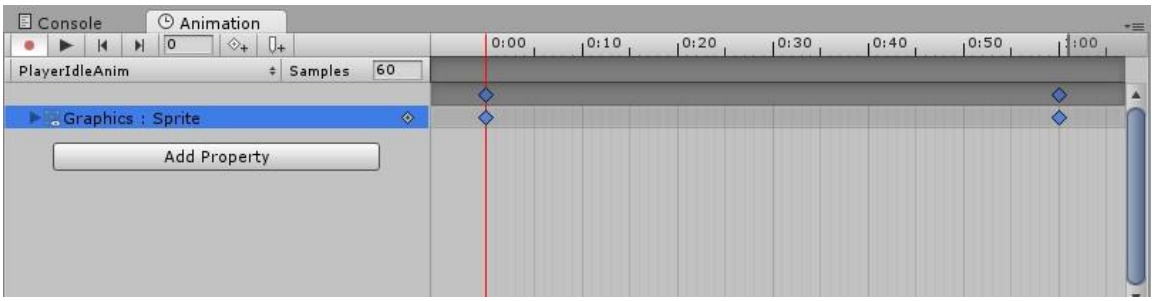

Рис. В.2 Створення анімації персонажа

Отже, для створення анімації потрібно натиснути кнопку запису, вибрати параметри значення, яких необхідно змінити в процесі анімації, додається

ключовий кадр, змінюється значення параметра, додається ще один ключовий кадр і т.д. В даному випадку змінюється спрайт персонажа. Для анімації ходьби використовується 7 кадрів, 4 кадри з яких різні, а 3 повторюються для створення повного циклу анімації, а для анімації стрибка і бездіяльності всього 2 кадри. Параметр Samples визначає швидкість анімації (кількість кадрів в секунду).

Тепер необхідно налаштувати контролер анімації (рис.В.3). Для цього в Unity використовується вікно Animator. Контролер анімації описує порядок і умови відтворення анімацій. Стан (анімація) PlayerIdleAnim (анімація бездіяльності) встановлюється за умовчанням (тобто ця анімація буде відтворюватися першою). Стан Any State має перехід в PlayerJumpAnim (анімація стрибка), це означає що з будь-якого стану можна перейти в стан стрибка. Для опису переходів використовуються додаткові параметри, такі як Speed (швидкість), Ground (булева змінна визначає чи стоїть персонаж на землі), vSpeed (вертикальна швидкість).

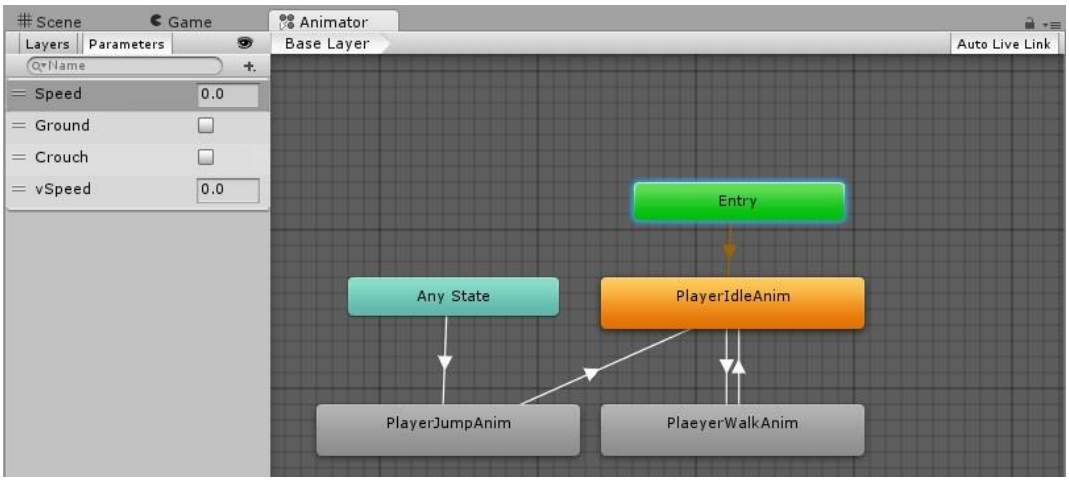

Рис. В.3 Налаштування контролера анімації

Далі необхідно написати сценарій персонажа. У ньому будуть міститися його властивості (здоров'я, швидкість пересування), а також методи впливу на нього

(отримання шкоди, знищення персонажа при значенні здоров'я менше 0 або 0).

Лістинг програмного коду

class TestLocationService
:

```
using UnityEngine;
using
System.Collections;

public class TestLocationService : MonoBehaviour {

   IEnumerator Start()
   {
      // First, check if user has location service
enabled        if
(!Input.location.isEnabledByUser)
         yield break;

      // Start service before querying location
      Input.location.Start();

      // Wait until service
initializes        int maxWait =
20;
      while (Input.location.status == LocationServiceStatus.Initializing && maxWait >
0)
      {
         yield return new WaitForSeconds(1);
         maxWait--;
      }

      // Service didn't initialize in 20
seconds        if (maxWait < 1)
      {
         print("Timed out");
yield break;
      }

      // Connection has failed
      if (Input.location.status == LocationServiceStatus.Failed)
```

```
        {
            print("Unable to determine device
location");            yield break;
        }
else
{
            // Access granted and location value could be retrieved
print("Location: " + Input.location.lastData.latitude + " " +
                Input.location.lastData.longitude + " " +
                Input.location.lastData.altitude
                + " " + Input.location.lastData.horizontalAccuracy + " "
                + Input.location.lastData.timestamp);
        }

        // Stop service if there is no need to query location updates continuously
        Input.location.Stop();
    }
}
```

class SetGeolocation :
MonoBehaviour public class
SetGeolocation : MonoBehaviour
{

```
    public float lat;
public float lon;
    public float orientation;

    private float initX;
private float initZ;

    private        bool        gpsFix;
private  float  fixLat;       private
float    fixLon;               private
GameHelper _gameHelper;
    void Awake()
    {
        _gameHelper =
GameObject.FindObjectOfType<GameHelper>();         gpsFix =
_gameHelper.GpsFix;
    }
```

```
    IEnumerator Start()
    {
        while (!gpsFix)
        {
            gpsFix = _gameHelper.GpsFix;
            yield return null;
        }
        initX =
_gameHelper.IniRef.x;        initZ
= _gameHelper.IniRef.z;

        yield return new WaitForSeconds(1);
        GeoLocation();
    }

    void GeoLocation()
    {
        Vector3 pos = transform.position;
        pos.x = (float)(((lon * 20037508.34) / 18000) - initX);
        pos.z = (float)(((Mathf.Log(Mathf.Tan((90 + lat) * Mathf.PI / 360))
            / (Mathf.PI / 180)) * 1113.19490777778) -
initZ);        pos.y = 0;
        transform.position = pos;
        Vector3 eAngles = transform.eulerAngles;
        eAngles.y = orientation;
transform.eulerAngles = eAngles;
    }

    public void SetLoacation(float latitude, float longitude, float newOrientation)      {
lat = latitude;        lon = longitude;
        orientation = newOrientation;
        GeoLocation();
    }
}
```

class LoadMap

```
public class LoadMap : MonoBehaviour
{
    const string Key = "cydP18CWm7BCYetAANI4DOPcbl8ssckL";
```

```
    public Renderer maprender;
Vector2 PlayerPosition =
       new Vector2(50.437990f, 30.521626f);  //Latitude, Longitude
                                    //   new Vector2(42.3627f, -71.05686f);  //Latitude,
Longitude

    int _zoom = 17;
string _maptype =
"map";
    string _url;

    void Start()
    {
        StartLoadMap(PlayerPosition);
    }


    private void StartLoadMap(Vector2 playerPosition)
    {
        _url = "http://open.mapquestapi.com/staticmap/v4/getmap?key=" + Key
            + "&size=1280,1280&zoom=" + _zoom
            + "&type=" + _maptype
            + "&center=" + PlayerPosition.x + "," + PlayerPosition.y;
       // Debug.Log(_url);

        StartCoroutine(LoadImage());
    }

    private IEnumerator LoadImage()
    {
        WWW www = new
WWW(_url);        while
(!www.isDone)
        {
            Debug.Log("progress = " +
www.progress);            yield return null;
        }

        if (www.error == null)
        {
            Debug.Log("Updating map 100
%");            Debug.Log("Map
Ready!");            yield return new
```

```
WaitForSeconds(0.5f);
maprender.material.mainTexture = null;
Texture2D tmp;
            tmp = new Texture2D(1280, 1280, TextureFormat.RGB24, false);
maprender.material.mainTexture = tmp;
            www.LoadImageIntoTexture(tmp);
        }
        else {
            Debug.Log("Map     Error:"     +
www.error);                yield return new
WaitForSeconds(1);
maprender.material.mainTexture = null;
        }

        maprender.enabled = true;
    }
}
```

class PokemonHelper

```
public class PokemonHelper : MonoBehaviour
{

    public PokemonModel MyPokemonModel { get; set; }
    BattleHelper _battleHelper;
    void Start()
    {
        _battleHelper = GameObject.FindObjectOfType<BattleHelper>();
    }

    // Update is called once per
frame     void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if (!_battleHelper.IsBattle)
        {
            _battleHelper.StartBattele(MyPokemonModel);
        }
       // Destroy(gameObject);
    }
```

```
    public void LoadPokemon(PokemonModel item)
    {
        MyPokemonModel = item;

    }
}
```

class PlayerHelper

```
public class PlayerHelper : MonoBehaviour
{
    public PokemonModel MyPokemonModel { get; set; }

    // Use this for
initialization     void Start()
    {
        MyPokemonModel = new PokemonModel()
        {
            PokemonType = ETypes.Pikachu,
            Health = 200,
            Damage = 10,
            Exp = 21
        };
    }

    // Update is called once per
frame     void Update()
    {

    }
}
```

class BattlePokemonHelper

```
public class BattlePokemonHelper : MonoBehaviour
{
    public PokemonModel MyPokemonModel { get; set; }

    public int MaxHealth { get;
set; }     public int Health { get;
set; }     public int Level { get;
set; }
    public string Name { get; set; }
```

```
    public bool IsDead { get;
set; }          // Use this for
initialization     void Start()
    {

    }

    // Update is called once per
frame     void Update()
    {

    }

    internal void Load(PokemonModel myPokemonModel)
    {
        MyPokemonModel = myPokemonModel;

        MaxHealth = MyPokemonModel.Health;
        Health = MyPokemonModel.Health;
        Level = MyPokemonModel.Exp;
        Name = myPokemonModel.PokemonType.ToString();
    }

    internal void TakeDamage(int damage)
    {
        int health = Health - damage;

        if (health <= 0)
        {
            Health = 0;
            IsDead = true;
        }

        Health = health;
    }
}
```

class LoadPokemonData

```
using UnityEngine; using
System.Collections; using
System.Xml.Linq; using
```

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

public class LoadPokemonData : MonoBehaviour
{
   public GameObject[] PokemonPrefabs;

   string _xml = "";

   public List<PokemonModel> PokemonModels { get; set; }
   public List<PokemonHelper> PokemonHelpers { get; set; }

   GameHelper _gameHelper;
   IEnumerator Start()
   {
      PokemonModels = new List<PokemonModel>();
      PokemonHelpers = new List<PokemonHelper>();
      _gameHelper = GetComponent<GameHelper>();

      while (!_gameHelper.GpsFix)
      {
         Debug.Log("Wait!");
yield return null;
      }

      WWW www = new
WWW("https://docs.google.com/uc?id=0B8_KhHuqPhc6YU05aHpFdnA2
Wms");        while (!www.isDone)
      {
         yield return null;
      }

      Debug.Log(www.text);
      _xml = www.text;


      XDocument doc = XDocument.Parse(_xml);
      XElement element = doc.Element("pokemons");
      IEnumerable<XElement> elements = element.Elements();

      foreach (XElement item in elements)
      {
         PokemonModel pokemonModel = new PokemonModel();
```

```
            ///<pokemon type="0" lat="42.3637" lon="-
71.05686"></pokemon>            int pokemonTypeInt =
System.Convert.ToInt32(item.Attribute("type").Value);
pokemonModel.PokemonType = (ETypes)pokemonTypeInt;

            pokemonModel.Id = System.Convert.ToInt32(item.Attribute("id").Value);

            pokemonModel.Lat = System.Convert.ToSingle(item.Attribute("lat").Value);
pokemonModel.Lon = System.Convert.ToSingle(item.Attribute("lon").Value);
pokemonModel.Orint = System.Convert.ToSingle(item.Attribute("orint").Value);

            pokemonModel.Exp = System.Convert.ToInt32(item.Attribute("exp").Value);
pokemonModel.Damage = System.Convert.ToInt32(item.Attribute("damage").Value);
pokemonModel.Health = System.Convert.ToInt32(item.Attribute("health").Value);

            PokemonModels.Add(pokemonModel);
        }

        Debug.Log("PokemonModels.Count = " + PokemonModels.Count);


        for (int i = 0; i < PokemonModels.Count; i++)
        {
            var item = PokemonModels[i];

            GameObject pokemon = Instantiate(PokemonPrefabs[(int)item.PokemonType]);
SetGeolocation setGeolocation = pokemon.GetComponent<SetGeolocation>();
setGeolocation.SetLoacation(item.Lat, item.Lon, item.Orint);

            PokemonHelper pokemonHelper =
pokemon.GetComponent<PokemonHelper>();
pokemonHelper.LoadPokemon(item);

            PokemonHelpers.Add(pokemonHelper);
        }


    }

    // Update is called once per
frame     void Update()
    {
```

```
    }

    internal void DestroyPokemon(PokemonModel myPokemonModel)
    {
        PokemonHelper remove = null;
foreach (var item in
PokemonHelpers)
        {
            if (item.MyPokemonModel.Id == myPokemonModel.Id)
            {
                remove = item;
            }
        }

        Destroy(remove.gameObject);
        PokemonHelpers.Remove(remove);
    }
}
```

class GameHelper

```
public class GameHelper : MonoBehaviour
{
    const string Key =
"cydP18CWm7BCYetAANI4DOPcbl8ssckL";     const int
WaitTime = 10;

    public bool GpsFix { get; set; }

    /// <summary>
    /// Url
propertyes     ///
</summary>
    string Url = "";        public Transform myMap;        int
_multiplier = 2; //1 для size=640x640 tile, 2 для
size=1280*1280

    public Renderer maprender;
public Text StatusText;

    public Vector2 PlayerPosition =
        new Vector2(42.3627f, -71.05686f);  //Latitude, Longitude
```

```
    private double tempLat;
private double tempLon;
private Vector3 _iniRef;
public Vector3 IniRef
    {
        get { return _iniRef; }
        set { _iniRef = value; }
    }

    int _zoom = 17;
    string _maptype = "map";

    private LocationInfo _loc;
float _download = 0;
    public string Status { set { StatusText.text = value; } }

    int _counter;
IEnumerator Start()
    {

        Input.location.Start(5, 5);
        Input.compass.enabled = true;
        Status = "Initializing Location Services..";

        // Wait until service initializes
while (Input.location.status ==
LocationServiceStatus.Initializing &&
            _counter < WaitTime)
        {
            yield return new WaitForSeconds(1);
            Debug.Log("Wait " + _counter);
            Status = "Wait " + _counter;
            _counter++;
        }

        if (_counter >= WaitTime)
        {
            Status = "_counter >=
WaitTime";            yield return new
WaitForSeconds(4);
            Application.Quit();
yield return null;
        }
```

```
        if (Input.location.status == LocationServiceStatus.Failed)
        {
            Status = "Input.location.status ==
LocationServiceStatus.Failed";            yield return new
WaitForSeconds(4);
            Application.Quit();
yield return null;
        }
else
        {
          LocationInfo loc = Input.location.lastData; Debug.Log("Input.location.lastData");
          yield return new WaitForSeconds(2);
            /// Только для телефонов -----------------------------------------------------
            // PlayerPosition.x = loc.latitude;
            // PlayerPosition.y = loc.longitude;

            //Set Position
            _iniRef.x = (float)((PlayerPosition.y * 20037508.34 / 180) / 100);
            _iniRef.z = (float)(System.Math.Log(System.Math.Tan((90 + PlayerPosition.x)
                * System.Math.PI / 360)) / (System.Math.PI / 180));
            _iniRef.z = (float)((_iniRef.z * 20037508.34 / 180) /
100);            _iniRef.y = 0;

            GpsFix = true;
            ///Все гуд
            LoadMap(PlayerPosition);
        }

        InvokeRepeating("UpdateMyPosition", 1, 0.5f);
        InvokeRepeating("UpdateMap", 1, 3f);
        //  InvokeRepeating("Orientate", 1, 0.05f);
    }

    public Transform Player;
private bool _mapLoaded;
    public bool UpdatedPossition { get; set; }

    private Vector3
_newUserPos;     void
UpdateMap()
    {
        if (GpsFix && _mapLoaded && UpdatedPossition)
```

```
            LoadMap(PlayerPosition);
    }

    const float DistanceMapUpdate = 2;
Vector2 _lastPlayerPosition;
    void UpdateMyPosition()
    {
        if (GpsFix)
        {
            LocationInfo loc = Input.location.lastData;
            /// Только для телефонов -----------------------------------------------------
            // PlayerPosition.x = loc.latitude;
            // PlayerPosition.y = loc.longitude;

            if (Vector3.Distance(_lastPlayerPosition, Player.position) >
DistanceMapUpdate)                UpdatedPossition = true;

           // Debug.Log(" - " + Vector3.Distance(_lastPlayerPosition, Player.position));

        _newUserPos.x = (float)(((PlayerPosition.y * 20037508.34 / 180) / 100) -
  _iniRef.x); _newUserPos.z = (float)(System.Math.Log(System.Math.Tan((90
                                                                        + PlayerPosition.x)
                * System.Math.PI / 360)) / (System.Math.PI / 180));
            _newUserPos.z = (float)(((_newUserPos.z * 20037508.34 / 180) / 100) -
_iniRef.z);
            Player.position = _newUserPos;
        }
    }

    void Orientate()
    {
        //if (!simGPS && gpsFix)
        //{
        //    heading = Input.compass.trueHeading;
        //}
        //else {
        //    heading = user.eulerAngles.y;
        //}
    }
```

```
    private void LoadMap(Vector2 playerPosition)
    {
        _mapLoaded = false;
        Url = "http://open.mapquestapi.com/staticmap/v4/getmap?key=" +
            Key + "&size=1280,1280&zoom=" +
            _zoom + "&type=" + _maptype +
            "&center=" + PlayerPosition.x + "," + PlayerPosition.y;
        _lastPlayerPosition = Player.position;
        UpdatedPossition = false;

        StartCoroutine(LoadMap());
    }

    private IEnumerator LoadMap()
    {
        WWW www = new WWW(Url);

        while (!www.isDone)
        {
            _download = (www.progress);
            Debug.Log("Updating map " + System.Math.Round(_download *
100) + " %");                                Status = "Updating map " +
System.Math.Round(_download * 100) + " %";            yield return null;
        }

        if (www.error == null)
        {
            Debug.Log("Updating map 100 %");
            Debug.Log("Map Ready!");
           Status = "Updating map 100 %\nMap Ready!";
           yield return new WaitForSeconds(0.5f);
            maprender.material.mainTexture = null;
Texture2D tmp;
            tmp = new Texture2D(1280, 1280, TextureFormat.RGB24,
false);            maprender.material.mainTexture = tmp;
www.LoadImageIntoTexture(tmp);
        }
else {
            print("Map Error:" +
www.error);            Status = "Map
Error:" + www.error;            yield
return new WaitForSeconds(1);
maprender.material.mainTexture = null;
```

```
        }

        tempLat = PlayerPosition.x;
tempLon = PlayerPosition.y;

        maprender.enabled = true;
        ReSet();
        ReScale();

        _mapLoaded = true;
    }

    void ReSet()
    {
        Vector3 newPosition = new Vector3();        newPosition.x =
(float)(((tempLon * 20037508.34 / 180) / 100) - _iniRef.x);
newPosition.z = (float)(System.Math.Log(System.Math.Tan((90 +
tempLat)
            * System.Math.PI / 360)) / (System.Math.PI / 180));
        newPosition.z = (float)(((newPosition.z * 20037508.34 / 180) / 100) - _iniRef.z);
transform.position = newPosition;
    }

    void ReScale()
    {
        Vector3 newScale = myMap.localScale;
        newScale.x = (float)(_multiplier * 100532.244f /
(Mathf.Pow(2, _zoom)));        newScale.z = newScale.x;
myMap.localScale = newScale;
    }


    // Update is called once per
frame     void Update()
    {

    }
}
```

класс PokemonModel

```
using UnityEngine;
using System.Collections;

public class PokemonModel
{
   public int Id { get; set; }     public
ETypes PokemonType { get; set; }

   public float Lat { get; set; }
public float Lon { get; set; }
public float Orint { get; set; }

   public int Exp { get; set; }
public int Damage { get; set;
}     public int Health { get;
set; }


public enum ETypes
{
   Bulbasaur,
   Charmander,
   Squirtle,
   Pikachu,
   Pidgey,
Ivysaur
}
```

class loopScript

```
using UnityEngine;
using
System.Collections;

public class loopScript : MonoBehaviour {

      public GameObject chosenEffect;
   public float loopTimeLimit = 2.0f;

        void Start ()
         {
                PlayEffect();
         }
```

```
        public void PlayEffect()
          {
                StartCoroutine("EffectLoop");
          }


        IEnumerator EffectLoop()
          {
  GameObject effectPlayer = (GameObject) Instantiate(chosenEffect,
transform.position, transform.rotation);

              yield return new WaitForSeconds(loopTimeLimit);

                Destroy (effectPlayer);
                 PlayEffect();
          }
}
```

class lightScript

```
using
UnityEngine;
using System.Collections;

public class lightScript : MonoBehaviour
{
          bool Impact = false;
public float Sqr;
        // Use this for initialization
         void Start ()
          {
                 Impact = true;
              gameObject.GetComponent<Light>().intensity = 7;
           Sqr = gameObject.GetComponent<Light>().intensity * gameObject.GetCompo-
nent<Light>().intensity * (( gameObject.GetComponent<Light>().intensity < 0.0f ) ? -
1.0f : 1.0f);
          }

       // Update is called once per frame
         void Update ()
          {
```

```
                if (Impact)
                 {
                  gameObject.GetComponent<Light>().intensity -= (1.0f / Time.deltaTime) * Sqr
* .0001f;
                    if (gameObject.GetComponent<Light>().intensity <= 0)
                        {
                              Destroy (gameObject);
                        }
                 }
        }
}
```

class LoadSceneOnClick

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using
System.Collections;
using
System.Collections.Gen
eric;

public class LoadSceneOnClick : MonoBehaviour
{
   public void LoadScene1()  {
            SceneManager.LoadScene ("etfx_explosions");
        }
   public void LoadScene2()  {
      SceneManager.LoadScene ("etfx_explosions2");
        }
   public void LoadScene3()  {
      SceneManager.LoadScene ("etfx_portals");
        }
   public void LoadScene4()  {
            SceneManager.LoadScene ("etfx_magic");
        }
   public void LoadScene5()  {
      SceneManager.LoadScene ("etfx_emojis");
        }
   public void LoadScene6()  {
      SceneManager.LoadScene ("etfx_sparkles");
        }
   public void LoadScene7()  {
```

```
        SceneManager.LoadScene ("etfx_fireworks");
          }
    public void LoadScene8()  {
        SceneManager.LoadScene ("etfx_powerups");
    }
    public void LoadScene9()  {
        SceneManager.LoadScene ("etfx_swordcombat");
    }
    public void LoadScene10() {
        SceneManager.LoadScene("etfx_maindemo");
    }
    public void LoadScene11() {
        SceneManager.LoadScene("etfx_combat");
    }
    public void LoadScene12() {
SceneManager.LoadScene("etfx_");
    }
}
```

class DragMouseOrbit

```
using UnityEngine;
using UnityEngine;
using
System.Collections;
[AddComponentMenu("Camera-Control/Mouse drag Orbit with
zoom")] public class DragMouseOrbit : MonoBehaviour
{     public Transform
target;     public float
distance = 5.0f;     public
float xSpeed = 120.0f;
public float ySpeed =
120.0f;     public float
yMinLimit = -20f;
public float yMaxLimit
= 80f;     public float
distanceMin = .5f;
public float distanceMax
= 15f;     public float
smoothTime = 2f;
float rotationYAxis =
0.0f;     float
rotationXAxis = 0.0f;
float velocityX = 0.0f;
```

```
float velocityY = 0.0f;
// Use this for
initialization     void
Start()
   {
       Vector3 angles =
transform.eulerAngles;
rotationYAxis = angles.y;
rotationXAxis = angles.x;
       // Make the rigid body not change
rotation        if
(GetComponent<Rigidbody>())
       {
           GetComponent<Rigidbody>().freezeRotation = true;
       }
   }
   void LateUpdate()
   {
       if (target)
       {
           if (Input.GetMouseButton(1))
           {
               velocityX += xSpeed * Input.GetAxis("Mouse X") * distance *
0.02f;                velocityY += ySpeed * Input.GetAxis("Mouse Y") * 0.02f;
           }
           rotationYAxis += velocityX;
rotationXAxis -= velocityY;
           rotationXAxis = ClampAngle(rotationXAxis, yMinLimit, yMaxLimit);
           Quaternion fromRotation = Quaternion.Euler(transform.rotation.eulerAngles.x,
transform.rotation.eulerAngles.y, 0);
           Quaternion toRotation = Quaternion.Euler(rotationXAxis, rotationYAxis, 0);
           Quaternion rotation = toRotation;

           distance = Mathf.Clamp(distance - Input.GetAxis("Mouse ScrollWheel") * 5,
distanceMin, distanceMax);
           RaycastHit hit;            if
(Physics.Linecast(target.position, transform.position, out
hit))
           {
               distance -= hit.distance;
           }
           Vector3 negDistance = new Vector3(0.0f, 0.0f, -distance);
           Vector3 position = rotation * negDistance + target.position;
```

```
            transform.rotation = rotation;
            transform.position = position;
            velocityX = Mathf.Lerp(velocityX, 0, Time.deltaTime *
smoothTime);            velocityY = Mathf.Lerp(velocityY, 0,
Time.deltaTime * smoothTime);
        }
    }
    public static float ClampAngle(float angle, float min, float max)
    {
        if (angle < -
360F)            angle
+= 360F;         if
(angle > 360F)
angle -= 360F;
        return Mathf.Clamp(angle, min, max);
    }
}
```

class UICanvasManager

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

public class UICanvasManager :
MonoBehaviour {            public static
UICanvasManager GlobalAccess;       void
Awake () {            GlobalAccess = this;
        }

        public bool MouseOverButton =
false;            public Text PENameText;
      public Text ToolTipText;

 // Use this for initialization  void
Start () {
                if (PENameText != null)
                    PENameText.text = ParticleEffectsLibrary.GlobalAccess.GetCurrentPE-
NameString();
        }
```

```
    // Update is called once per frame
      void Update () {

  // Mouse Click - Check if mouse over button to prevent spawning particle effects while
hovering or using UI buttons.
                if
(!MouseOverButton) {
// Left Button Click
                    if (Input.GetMouseButtonUp (0)) {
                            // Spawn Currently Selected Particle System
                              SpawnCurrentParticleEffect();
                        }
                }

            if (Input.GetKeyUp (KeyCode.A)) {
                    SelectPreviousPE ();
                }
            if (Input.GetKeyUp (KeyCode.D)) {
                    SelectNextPE ();
                }
        }

   public void UpdateToolTip(ButtonTypes toolTipType) {
            if (ToolTipText != null) {
                  if (toolTipType == ButtonTypes.Previous) {
                          ToolTipText.text = "Select Previous Particle Effect";
                        }
                        else if (toolTipType == ButtonTypes.Next) {
                ToolTipText.text = "Select Next Particle Effect";
                        }
                }
        }
     public void ClearToolTip() {
            if (ToolTipText != null) {
                    ToolTipText.text = "";
                }
        }

     private void SelectPreviousPE() {
            // Previous

ParticleEffectsLibrary.GlobalAccess.PreviousParticleEffect();
if (PENameText != null)
```

```
                PENameText.text = ParticleEffectsLibrary.GlobalAccess.GetCurrentPE-
NameString();
        }
      private void SelectNextPE() {
                // Next
            ParticleEffectsLibrary.GlobalAccess.NextParticleEffect();
              if (PENameText != null)
                PENameText.text = ParticleEffectsLibrary.GlobalAccess.GetCurrentPE-
NameString();
        }

       private RaycastHit rayHit;
     private void SpawnCurrentParticleEffect() {
              // Spawn Particle Effect
                Ray mouseRay =
Camera.main.ScreenPointToRay(Input.mousePosition);            if
(Physics.Raycast (mouseRay, out rayHit)) {
                ParticleEffectsLibrary.GlobalAccess.SpawnParticleEffect (rayHit.point);
                }
        }

        /// <summary>
        /// User interfaces the button click.
        /// </summary>
        /// <param name="buttonTypeClicked">Button type clicked.</param>
        public void UIButtonClick(ButtonTypes buttonTypeClicked) {
                switch
        (buttonTypeClicked) {     case
        ButtonTypes.Previous:
        // Select Previous Prefab
        SelectPreviousPE();
                        break;
case ButtonTypes.Next:
// Select Next Prefab
SelectNextPE();
                        break;
        default:
// Nothing
break;
                }
        }
}
```

```
using UnityEngine;
using
System.Collections;

public class PEDestoryTimed : MonoBehaviour {

 // Use this for initialization  void
Start () {

        }

      // Update is called once per frame
        void Update () {

        }
}

using UnityEngine;
using
System.Collections;
using
UnityEngine.UI;
using UnityEngine.EventSystems;

public enum ButtonTypes {
        NotDefined,
        Previous,
        Next
}

public class PEButtonScript : MonoBehaviour, IEventSystemHandler,
IPointerEnterHandler, IPointerExitHandler {

        private Button myButton;
        public ButtonTypes ButtonType = ButtonTypes.NotDefined;
        // Use this for initialization
        void Start () {
            myButton = gameObject.GetComponent<Button> ();
        }

        public void OnPointerEnter(PointerEventData eventData) {
                // Used for Tooltip
            UICanvasManager.GlobalAccess.MouseOverButton = true;
```

```
            UICanvasManager.GlobalAccess.UpdateToolTip (ButtonType);
        }

    public void OnPointerExit(PointerEventData eventData) {
                // Used for Tooltip
            UICanvasManager.GlobalAccess.MouseOverButton = false;
             UICanvasManager.GlobalAccess.ClearToolTip ();
        }

      public void OnButtonClicked () {
                // Button Click Actions
            UICanvasManager.GlobalAccess.UIButtonClick(ButtonType);
        }
}

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

public class ParticleEffectsLibrary : MonoBehaviour {
        public static ParticleEffectsLibrary
GlobalAccess;       void Awake () {
GlobalAccess = this;

            currentActivePEList = new List<Transform> ();

            TotalEffects = ParticleEffectPrefabs.Length;

             CurrentParticleEffectNum = 1;

                // Warn About Lengths of Arrays not matching
        if (ParticleEffectSpawnOffsets.Length !=
TotalEffects) {
                        Debug.LogError ("ParticleEffectsLibrary-
ParticleEffectSpawnOffset: Not all arrays match length, double check counts.");
                }
            if (ParticleEffectPrefabs.Length != TotalEffects) {
   Debug.LogError ("ParticleEffectsLibrary-ParticleEffectPrefabs: Not all arrays match
length, double check counts.");
                }

             // Setup Starting PE Name String
```

```
            effectNameString = ParticleEffectPrefabs [CurrentParticleEffectIndex].name + "
(" +
CurrentParticleEffectNum.ToString() + " of " + TotalEffects.ToString() + ")";
        }

        // Stores total number of effects in arrays - NOTE: All Arrays must match
        length. public int TotalEffects = 0;
        public int CurrentParticleEffectIndex
= 0; public int CurrentParticleEffectNum =
0; //    public string[]
ParticleEffectDisplayNames;     public
Vector3[] ParticleEffectSpawnOffsets;
 // How long until Particle Effect is Destroyed - 0 = never
public float[] ParticleEffectLifetimes;
    public GameObject[] ParticleEffectPrefabs;

        // Storing for deleting if looping
particle effect        private string
effectNameString = "";    private
List<Transform> currentActivePEList;

       void Start () {
        }

    public string GetCurrentPENameString() {
                return ParticleEffectPrefabs [CurrentParticleEffectIndex].name + " (" +
CurrentParticleEffectNum.ToString() + " of " + TotalEffects.ToString() + ")";
        }

     public void PreviousParticleEffect() {
            // Destroy Looping Particle Effects
  if (ParticleEffectLifetimes [CurrentParticleEffectIndex] == 0) {
if (currentActivePEList.Count > 0) {
   for (int i = 0; i < currentActivePEList.Count; i++) {      if
(currentActivePEList [i] != null) {
                                    Destroy (currentActivePEList [i].gameObject);
                                }
                            }
                        currentActivePEList.Clear ();
                    }
                }
```

```
                // Select Previous Particle
Effect                    if
(CurrentParticleEffectIndex > 0) {
                        CurrentParticleEffectIndex -= 1;
                } else {
                       CurrentParticleEffectIndex = TotalEffects - 1;
                }
            CurrentParticleEffectNum = CurrentParticleEffectIndex + 1;

                // Update PE Name String
                  effectNameString = ParticleEffectPrefabs
[CurrentParticleEffectIndex].name + " (" + CurrentParticleEffectNum.ToString() + " of
" + TotalEffects.ToString() + ")";
        }
      public void NextParticleEffect() {
                // Destroy Looping Particle Effects
          if (ParticleEffectLifetimes [CurrentParticleEffectIndex] ==
         0) {   if (currentActivePEList.Count > 0) {
            for (int i = 0; i < currentActivePEList.Count; i++) {     if
         (currentActivePEList [i] != null) {
                                        Destroy (currentActivePEList [i].gameObject);
                                    }
                                }
                            currentActivePEList.Clear ();
                        }
                }

                // Select Next Particle Effect
             if (CurrentParticleEffectIndex < TotalEffects - 1) {
                        CurrentParticleEffectIndex += 1;
                } else {
                        CurrentParticleEffectIndex = 0;
                }
            CurrentParticleEffectNum = CurrentParticleEffectIndex + 1;

                // Update PE Name String
                  effectNameString = ParticleEffectPrefabs
[CurrentParticleEffectIndex].name + " (" + CurrentParticleEffectNum.ToString() + " of
" + TotalEffects.ToString() + ")";
        }

    private Vector3 spawnPosition = Vector3.zero;
  public void SpawnParticleEffect(Vector3 positionInWorldToSpawn) {
```

```
            // Spawn Currently Selected Particle Effect
  spawnPosition = positionInWorldToSpawn +
ParticleEffectSpawnOffsets[CurrentParticleEffectIndex];
  GameObject                                newParticleEffect                                =
GameObject.Instantiate(ParticleEffectPrefabs[CurrentParticleEffectIndex],
spawnPosition,
ParticleEffectPrefabs[CurrentParticleEffectIndex].transform.rotation)                    as
GameObject;
                                        newParticleEffect.name = "PE_" +
ParticleEffectPrefabs[CurrentParticleEffectIndex];
            // Store Looping Particle Effects Systems
                if (ParticleEffectLifetimes [CurrentParticleEffectIndex] == 0) {
        currentActivePEList.Add (newParticleEffect.transform);
                }

currentActivePEList.Add(newParticleEffect.transform);
// Destroy Particle Effect After Lifetime expired
            if (ParticleEffectLifetimes [CurrentParticleEffectIndex] != 0) {
                        Destroy(newParticleEffect,
ParticleEffectLifetimes[CurrentParticleEffectIndex]);
                }
        }
}
```

Перелік вільних і комерційних движків ігор

Вільні движки:

- Agar (англ.) - високорівнева каркас графічного додатку, використовуемое для розробки як 2D, так і 3D комп'ютерних ігор.
- Axiom Engine - відгалуження, що містить в основі графічний движок OGRE.
- Boom - порт вихідного коду гри Doom від TeamTNT.
- CheapHack (англ.) - паче не розробляється движок на основі TomazQuake.
- Crystal Entity Layer - розширення движка Crystal Space 3D Engine.
- Crystal Space - повний каркас для розробки 3D додатків.
- DarkPlaces - один з багатьох удосконалених вільно поширеноюняемое Quake engines.
- Daybreak motor - об'єктно-орієнтована графічний движок для XNA (XNA 3d engine - Xbox, Windows).
- Delta3D (англ.) - з'єднує інші добре відомі безкоштовні проекти в просте API, спочатку розроблявся U.S. Navy.
- DGD - об'єктно-орієнтована MUD движок.
- Eternity Engine - вихідний порт движка гри Doom.
- Exult (англ.) - вільна реінкарнація ігрового движка Ultima VII.
- FIFE (англ.) - вільний 2D ISO движок, що підтримує ресурси Fallout 1 і 2.
- Game Blender - під-додаток Blender для створення ігор.
- Game Maker - об'єктно-орієнтований програмний продукт для разництва ігор з drag-anddrop інтерфейсом і своїм скриптовою мовою.
- GQ - движок гри Quake, який включає в себе фичи з TomazQuake і DarkPlaces.
- GZDoom - вихідні ігрового движка Doom, засновані на ZDoom.
- Genesis3D - 3D движок реального часу для Windows.
- GemRB - вільна реалізація Infinity Engine.

- HGE - Haaf's Game Engine, движок для створення 2D ігор.
- Jogre - клієнт / серверний ігровий движок, написаний на мові Java, з-тримає API для онлайнових ігор реального часу, будь-то шашки, шахи, та інше.
- MGF - Mortem's Game Framework - активно розвивається ігровий фреймворк з відкритим вихідним кодом для ігрової консолі PSP на мові C ++.
- MRPGe - ігровий движок для Visual Basic, для двомірних RPG-ігор, з вбудованим скриптмовою. Вельми змінюємо на ранніх стадіях розробки.
- M.U.G.E.N - двомірний движок для ігор жанру файтинг (наприклад, Mortal Kombat).
- Multiverse Network - MMOG платформа, що включає сервер, клієнт, і ін-струментарій.
- Nebula Device - тривимірний ігровий движок реального часу, розробити конструкцію танний німецької студією ігор Radon Labs.
- OctLight - ігровий движок на Java, що використовує scene-graph, і отрісови-вающий на базі Lightweight Java Game Library (LWJGL), використовуючи OpenGL.
- OMEGA Engine - простий у використанні 2D движок. Використовує в якості рендера Direct3D8 або OpenGL, для звуку - DirectSound.
- OSlib - Old School Library - бібліотека для портування старих ігор на PSP або написання 2D ігор в стилі ігор минулих років на мові C ++.
- Pentagram - проект створює ігровий движок, який використовується в випущеної грі Ultima VIII: Pagan.
- PLIB - ігровий движок, що включає в себе: трехмерку, звук, музику, гра-фические інтерфейс користувача, менеджера вікон, і портіруемость на Linux / Windows / MacOSX.
- pH Engine - потужний і гнучкий ООП движок, що використовує Direct3D9 (Рендер), DirectSound (Звук) і TCP / IP (Мережа). Можливість «нарощування» воз-можностей за допомогою плагінів.

- ProQuake - вдосконалений движок гри Quake.
- QSP - популярний багатоплатформовий російський движок для створення ігор в жанрі

Interactive Fiction.

- RealmForge - ігровий движок з відкритим кодом для Microsoft .NET

Framework, попередник Visual3D.NET.

- rRenderer - ігровий движок сучасного рівня написаний на VB6, для DX8.
- Sauerbraten - вдосконалений движок, відгалуження від движка Cube.
- SmartX - ігровий движок використовує платформу .NET Framework.

На-писаний на C #, але є можливість інтеграції в мови підтримують

технологію .NET.

- Storm3D - движок використовує Direct3D 9, написаний на C ++.
- Stratagus - багатоплатформовий ігровий движок для стратегічних ігор реального часу.
- Telejano - змінений движок гри Quake.
- URQ - популярний російський движок для створення ігор в жанрі Interactive Fiction.
- vbGORE - движок з відкритим вихідним кодом, на основі якого можна

створити двомірну онлайнову гру з деякими 3D ефектами.

- Yake - компонент-заснований, об'єктно-орієнтована, частково ро-

довой движок, написаний на C ++.

- ZDoom - один з багатьох початкових кодів Doom.

Комерційні движки:

- Arcane Engine - розроблений Wolfpack Studios для Shadowbane.
- Auran Jet - движок, розроблений австралійською компанією AURAN і

використовуваний в грі Trainz.

- Baja Engine - движок професійної якості, який використовується для гри The Lost Mansion.
- Blitz3D - графічний движок зі своєю мовою программіровнаія BlitzBasic.

- C4 Engine - з'являється ігровий движок наступного покоління від Еріка Ленгел.
- Dark engine - застарілий движок, використаний для ігор Looking Glass Studios.
- Earth-4 Engine - графічний движок, який використовується в Earth 2160.
- Explorations - творець двомірних MMO.
- IMUSE - спеціально розроблений движок для синхронізації музики з візуальними діями.
- KjAPI - технологія C ++ для створення ігор і тривимірних додатків для ПК.
- Medusa (ігровий движок) - ігровий тривимірний движок на C ++ розробити конструкцію танний Palestar і використаний в грі DarkSpace MMO. У ньому є симуляція ігрового світу, єдиний інструмент контролю версій, реалізація активів, кроссплатформенная підтримка і вбудована система клієнт / сервер.
- Odyssey Engine - використаний для створення тривимірних рольових ігор, а також ігри Star Wars: Knights of the Old Republic.
- ORE - онлайновий движок для рольових ігор.
- Quasar - ігровий об'єктно-орієнтований наступного поколе-ня, що розробляється компанією Syide Technologies.
- Power Render - основна мета цього пакета - розробка ігор і тривимірних уявлень.
- Reality Engine - тривимірний ігровий движок компанії Artificial Studios.
- Retribution Engine - ігровий движок для створення ігор в стилі Екшн.
- Revolution3D - тривимірний графічний движок, розроблений X-Dream Project.
- Shark 3D - щось середнє між Spinor для ПК, відео іграми і тривимірними додатками реального часу.
- Silent Storm engine - тривимірний ігровий движок для тактичних стратегій (напр. Silent Storm).

- Torque Game Engine - змінена версія ігрового тривимірного движка, спочатку розроблена Dynamix для гри 2001 FPS Tribes 2.
- TOSHI - багатоплатформовий ігровий движок четвертого покоління, раз-працюємо Blue Tongue Entertainment.
- Truevision3d - тривимірний ігровий движок, що використовує DirectX API.
- Unigine - багатоплатформовий middleware.
- Unity (game engine) - легко використовуваний ігровий тривимірний движок.
- Vicious Engine - ігровий движок, портіруемость під Microsoft Windows, Sony PlayStation 2, Microsoft Xbox, і Sony PlayStation Portable.
- 3DGame Studio - повноцінний ігровий конструктор, ціла система система для створення тривимірних ігор.
- Visual3D.NET - тривимірна платформа для візуальної розробки окруже-ня, побудована на Microsoft .NET 2.0 і XNA Framework для розробки під PC, Xbox 360, і пристроїв Windows Mobile підтримує C #, Visual Basic, J # (Java), C ++. NET, JScript.NET, IronPython, і візуальне скріптованія.
- Virtools - движок для створення ігор, тренажерів і симуляторів, 3D Internet і систем віртуальної реальності.
- WGAF - ігровий движок, розроблений Guild Software, який вико-ється в їх MMORPG Vendetta Online.
- White Engine - ігровий движок сьомого покоління, що належить Square-Enix, буде використаний в їх проектах для PS3. Продумана можливість використання заздалегідь намальовані якісної CGI графіки в реальному часі.
- Xors3d Engine - движок для створення тривимірних ігор. Почав свою сущест-вованіе як динамічно підключається для Blitz3D, яка по-зволяет використовувати GAPI DirectX9.
- Zero - тривимірний ігровий движок, який використовується в Star Wars: The Clone Wars, Star Wars: Battlefront, і Star Wars: Battlefront II.

- LyN engine - двигун компанії Ubisoft, який використовується у власних розроб-Ботках.
- Vicarious Visions Alchemy - движок компанії Intrinsic Graphics.
- Cry Engine 1-3 - ігровий движок, створений німецькою приватною компанією Crytek в 2002 році і спочатку використовуваний в шутере від першої особи Far Cry.
- Id Tech - сімейство ігрових движків, розроблених американської когось паніей id Software.
- Source - ігровий движок, розроблений корпорацією Valve. Його особливо-стями вважаються модульна основа і гнучкість, синхронізація руху губ з промовою, технологія вираження емоцій і система фізики, яка працює по мережі.
- Serious Engine - ігровий движок, розроблений хорватською компанією Croteam і вперше використаний в грі Serious Sam 2001 року випуску. Тех-монолог перебувала в розробці протягом трьох років. Згодом компа-нией Croteam були розроблені більш досконалі ігрові движки - Serious Engine 2 і Serious Engine 3.