МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
т.в.о. завідувача кафедри
_____ Сафонова С.О.
«____»_____ 2020 р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Дослідження та розробка IDE для компільованих ігрових додатків

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 123 - "Комп'ютерна інженерія"

Науковий керівник роботи: _____ М. Є. Щербакова
(підпис) (ініціали, прізвище)

Консультант з охорони праці: _____ Я. О. Критська
(підпис) (ініціали, прізвище)

Студент: _____ Н. А. Мовшук
(підпис) (ініціали, прізвище)

Група: КІ-18дм

Сєвєродонецьк 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки
Кафедра комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки _____
(шифр і назва)

Спеціальність 123 - "Комп'ютерна інженерія"
(шифр і назва)

ЗАТВЕРДЖУЮ:
Т.в.о. завідувача кафедри
_____С.О. Сафонова
«_____» _____2020р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Мовшуку Нестору Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка IDE для компільованих ігрових додатків
керівник проекту Щербакова Марина Євгенівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "11" жовтня 2019 р. № 135/15.15
2. Строк подання студентом роботи 10.01.2020 р.
3. Вихідні дані до роботи Матеріали науково-дослідної практики; програмне забезпечення ігрового середовища розробки.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз методів і засобів розробки ігрових додатків та постановка задачі досліджень. Аналіз і опис використовуваних математичних, структурних та функціональних рішень. Програмна реалізація ігрового середовища розробки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Електронні плакати.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Критська Я. О., ст. викл.		

7. Дата видачі завдання 11.10.2019 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Отримання завдання, збір матеріалів	18.10.19- 20.10.19	Виконано
2	Огляд літератури й обґрунтування необхідності дослідження	21.10.19 –27.10.19	Виконано
3	Дослідження методів розробки ігрових додатків	28.10.19 – 12.11.19	Виконано
4	Аналіз і опис використовуваних математичних, структурних та функціональних рішень	11.11.19 –20.11.19	Виконано
5	Реалізація ігрового середовища розробки	21.11.19 – 27.12.19	Виконано
6	Оформлення пояснювальної записки	28.12.19 – 08.01.20	Виконано
7	Підготовка та подання магістерської роботи до захисту	09.01.20 – 10.01.20	Виконано

Студент _____ Н. А. Мовшук
(підпис) (ініціали, прізвище)

Науковий керівник _____ М. Є. Щербакова
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Мовшук Н.А. Дослідження та розробка IDE для компільованих ігрових додатків.

Досліджено особливості та проведено аналіз ігрових середовищ розробки. Розроблено нові програмно-архітектурні концепції, спеціалізовані високопродуктивні структури та механізми. Для підвищення якості вихідного програмного забезпечення використаний ітеративний підхід розробки. На підставі результатів дослідження реалізований програмний прототип для створення ігрових додатків.

Ключові слова: IDE, MGE, програмне забезпечення, ігрове середовище розробки, архітектура, ігровий шаблон, C++.

АННОТАЦИЯ

Мовшук Н.А. Исследование и разработка IDE для компилируемых игровых приложений.

Исследованы особенности и проведён анализ игровых сред разработки. Разработаны новые программно-архитектурные концепции, специализированные высокопроизводительные структуры и механизмы. Для повышения качества выходного программного обеспечения использован итеративный подход разработки. На основании результатов исследования реализован программный прототип для создания игровых приложений.

Ключевые слова: IDE, MGE, программное обеспечение, игровая среда разработки, архитектура, игровой шаблон, C++.

ABSTRACT

Movshuk N.A. Research and developers of IDE for compiled game applications.

A features and the analysis of game development environments are investigated. New software-architectural concepts, specialized high-performance structures and mechanisms have been developed. An iterative development approach was used to improve the quality of the source software. Based on the results of the study, a software prototype for creating gaming applications was implemented.

Key words: IDE, MGE, software, environment development games, architecture, game template, C++.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Ігрова індустрія на даний момент є однією з актуальних, затребуваних і таких, що швидко розвиваються галузей виробництва, особливо це стосується ігрових консольних додатків. Зародившись в середині 1970-х років, вона зросла у величезну індустрію розваг з оборотами в мільярди доларів. Розробкою ігор зараз займаються як великі компанії (Konami, Square Enix, Ubisoft, Electronic Arts, Blizzard Entertainment, Activision, Rockstar Games), так і невеличкі фірми, спільноти та окремі індивідуальні розробники.

Використання ігрового середовища значно спрощує процес розробки ігор. Зазвичай для реалізації досить тривіальних задач, наприклад: відображення внесених змін при зміні геометрії ландшафту в режимі реального часу, наглядного розташування ігрових об'єктів на сцені, використовуються сотні бібліотечних функцій, тісно переплетених між собою. За допомогою ігрового середовища розробки подібні конструкції спрощуються до виклику двох функцій. Це дозволяє розробникам працювати над грою більш абстрактно, не замислюючись про низькорівневе представлення компонентів двигуна і те, як вони працюють і як пов'язані між собою. До того ж виконуваний код, написаний з використанням ігрового середовища, виглядає більш організованим і більш керованим, що знижує загальну кількість виникаючих програмних помилок.

На сьогоднішній день не існує конкретних стандартних інструментів для розробки ігрових додатків. Практично кожна компанія покладається на свої власні розробки, паралельно при цьому надаючи свої готові рішення кінцевим споживачам на певних умовах використання. Також варто відзначити значну частину індивідуальних розробників, які створюють ігрові додатки без використання готових середовищ розробки в угоду більш вільній, незалежній і розширеній розробці власних додатків.

Тому обґрунтованим є вибір теми магістерської роботи, в якій вирішується науково-прикладна задача розробки IDE для компіляції ігрових додатків, тобто створення кінцевого програмного додатку відразу у вигляді машинного коду за допомогою вбудованого компілятора з використанням розробленого ігрового двигуна.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана у Східноукраїнському національному університеті ім. В. Даля у відповідності з державними програмами і планами в межах НДР “Дослідження механізмів захисту від кібератак”, реєстраційний номер 0116U007997.

Мета і завдання дослідження. Метою магістерської роботи є дослідження архітектури, методів і алгоритмів для створення прикладного програмного середовища розробки ігрових додатків будь-якої складності та розробка програмного прототипу ігрового середовища розробки орієнтованого на ефективне використання системних ресурсів за рахунок використання спеціальних внутрішніх механізмів.

Метою дослідження є підвищення якості та розширення прийнятих концепцій ігрової розробки, шляхом застосування нових програмних методів та архітектурних рішень.

Для досягнення мети дослідження необхідно вирішити такі завдання:

– проведення огляду наукових досліджень, спрямованих на вирішення завдань ігрової розробки;

– дослідження архітектури, методів і алгоритмів для створення прикладного програмного середовища розробки ігрових додатків;

– аналіз придатності використання публікацій в якості базису деяких підсистем ігрового середовища розробки;

– визначення вимог до розроблюваної IDE;

– проектування програмної архітектури IDE;

– створення програмного прототипу, метою якого є перевірка придатності узгоджених для застосування концепцій, архітектурних і технологічних рішень;

– створення демонстраційного ігрового додатку.

Об'єкт дослідження - середовище розробки компільованих ігрових додатків.

Предмет дослідження - програмні та архітектурні методи розробки.

Методи дослідження. Проведені в роботі дослідження основані на методах аналізу і моделювання. При розробці прикладної програми для досягнення поставлених завдань використовується метод “ітеративної розробки” [1], завдяки чому забезпечується паралельна розробка з аналізом отриманих результатів та подальшим коректуванням програмного забезпечення.

Наукова новизна отриманих результатів:

– подальший розвиток методу розробки ігрових додатків за допомогою IDE;

– розробка альтернативних методів програмування ігрових проектів з використанням IDE.

Практичне значення отриманих результатів полягає в тому, що основні наукові положення роботи реалізовані у виді розрахункових моделей та програмних засобів, які утворюють прикладну інформаційну технологію - середовище розробки для компіляції ігрових додатків.

Особистий внесок здобувача полягає у розробленні нових моделей, методів та інструментальних засобів, що дозволяють вирішити поставлені задачі. Усі основні результати отримані автором особисто. Автору належать наступні публікації: “Деталізація сферично-динамічного рельєфу в комп’ютерній графіці” [2], “Розробка та аналіз програмного рішення файлового кешування даних в мові C++ на основі технології memory mapped files” [3], “Динамічна деталізація сферичного рельєфу в комп’ютерній графіці” [4].

Апробація результатів роботи. Основні положення, ідеї, висновки магістерської роботи доповідалися та обговорювалися на IV регіональний форум «IT-Ідея 2018» і IX Всеукраїнській науково-практичній конференції «Майбутній науковець – 2018».

Публікації. За темою магістерської роботи з викладом її основних результатів опубліковано три наукових праці, серед яких одна стаття у науковому фаховому виданні України, та дві тези доповідей на конференціях.

Структура та обсяг магістерської роботи. Магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел з 38 найменувань та додатку на 5 сторінках. Загальний обсяг роботи складає 98 сторінок. Магістерська робота містить 51 рисунок та 8 таблиць.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ І ЗАСОБІВ РОЗРОБКИ ІГРОВИХ ДОДАТКІВ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕНЬ

1.1 Аналіз та опис об’єкта дослідження

Об’єктом дослідження є середовище розробки компільованих ігрових додатків, яке складається з трьох основних частин:

– середовище розробки ігрових додатків;
– ігрового двигуна;
– компілятора.

Основні завдання об'єкта досліджень:

– визначення вимог до розроблюваної IDE;
– проектування програмної архітектури IDE;
– створення програмного прототипу, метою якого є перевірка придатності узгоджених для застосування концепцій, архітектурних і технологічних рішень;
– створення демонстраційного ігрового додатку.

Основні проблеми проектування та реалізації об'єкта дослідження:

– алгоритмічна складність;
– трудомісткість розробки;
– інформаційна складність;
– неоднозначність поставлених рішень.

1.1.1 Середовище розробки ігрових додатків

Середовище розробки [5] - комплекс програмних методів і засобів, які використовуються розробниками для подальшого спрощення та ефективної розробки програмного забезпечення.

Мета середовища полягає в об'єднанні різних утиліт в одному модулі, дозволяючи програмісту абстрагуватися від виконання допоміжних завдань, зосередитися на вирішенні власне алгоритмічному завданні та уникнути втрати часу при виконанні типових технічних дій. Таким чином, підвищується продуктивність

праці розробника. Тісна інтеграція задач розробки може підвищити продуктивність за рахунок можливості введення додаткових функцій на проміжних етапах роботи.

Основні складові середовища розробки включають в себе:

– текстові та графічні редактори;
– компілятор та/або інтерпретатор;
– засоби автоматизації збірки;
– відладчик.

На відміну від стандартного уявлення середовища розробки, ігрове середовище орієнтовано на ігрову архітектуру з використанням вбудованого ігрового двигуна, що значно розширює спеціалізовані функціональні можливості для максимального продуктивності роботи розробника при виконання поставлених завдань.

Основною відмінною рисою ігрового середовища розробки є наявність вбудованого графічного редактора, за допомогою якого можна безпосередньо взаємодіяти з ігровим простором, а саме: візуального редагування полігонального ландшафту, ігрових об'єктів, логічних зв'язків і інших подібних речей. Завдяки чому значно зростає швидкість та якість розробки, так як результати подібних змін відображаються в режимі реального часу - вони будуть миттєво представлені у кінцевого користувача, що значно спрощує налагодження внесених змін.

Також варто відзначити значні вимоги до ігрового середовища розробки, вони полягають у підвищеній відмовостійкості та надійності, так як поява критичної помилки з подальшим завершення роботи програми розробки вкрай негативно позначається на розробці. Окрім втрачених напрацювань і часу розробника, також є суттєвий ризик в отриманні повністю непрацездатного проекту, що тягне за собою додаткові витрати часу і сил на пошук та виправлення можливих дефектів в проекті, викликаних непередбаченим збоєм в роботі програми.

Іншою значною вимогою до ігрового середовища розробки, від якої в свою чергу залежить швидкість, масштабність і складність розробки проекту та подальшу затребуваність у розробників - є продуктивність роботи. По мірі ускладнення ігрової сцени пропорційно підвищується кількість необхідних системних ресурсів, що в кінцевому підсумку веде до можливого перевищення допустимої кількості виділених системних ресурсів комп'ютера, які негативно позначається на подальшій продуктивності роботи програми і може призвести до збою її виконання.

1.1.2 Ігровий двигун

Ігровий двигун - центральне програмне ядро будь-якої відеоігри, що представляє собою своєрідну вузькоспеціалізовану операційну систему, оскільки включає всі модулі останньої, дозволяє в першу чергу полегшити та прискорити розробку гри шляхом уніфікації і систематизації її внутрішньої структури. За рахунок чого досягається можливість повторного використання і розширення для розробки різних ігрових додатків без істотних змін.

Ігровий двигун є одним з найскладніших в написанні програмних додатків, який в свою чергу складається з десятки різних компонентів, кожен з яких автоматично налаштовується в залежності від вимог до гри.

У функціональну складову якого входить:

– візуалізатор;
– фізична підсистема;
– власна файл-ресурсна система;
– засоби роботи з багатопотоковістью;
– система вводу-виводу;
– аудіо підсистема;
– штучний інтелект;
– фізична підсистема;
– мережева підсистема.

1.1.3 Компілятор

Компілятор [6] - збірка програми, що включає трансляцію всіх модулів програми, написаних на одному або декількох вихідних мовах програмування високого рівня і/або мовою асемблера, в еквівалентні програмні модулі на низькорівневій мові близькому до машинного коду виконує фізичної програмованої машини.

Результат компіляції - виконавчий програмний модуль, який має максимально можливу продуктивність, однак прив'язаний до конкретної операційної системи (сімейства або підродини ОС) і процесору (сімейства процесорів) і не буде працювати на інших.

1.2. Аналіз існуючих програмних рішень

У середині 90-х після появи відеопроцесорів, здатних обробляти тривимірну графіку стали з'являтися програмні інтерфейси, що спрощує її розробку. Слідом за цим з'явилися візуалізатори OpenGL і Direct3D, які на багато років вперед визначили способи графічного виведення в іграх. Після чого в 1996-му році вийшла гра Quake на Quake Engine. Цей движок надав колосальний вплив на ігрову індустрію ставши одним з перших двигунів, здатним обробляти в реальному часі повністю тривимірну графіку.

Через деякий час двигун став доступний для безкоштовного використання за умовами ліцензії General Public License [7], яка передбачає собою право копіювати, модифікувати і поширювати його вміст. Це справило великий вплив на розвиток світу відкритого програмного забезпечення, а також породило велику кількість сторонніх ігрових двигунів, заснованих на Quake engine. Дерево розвитку ігрових двигунів, заснованих на Quake Engine представлено на рисунку 1.1.

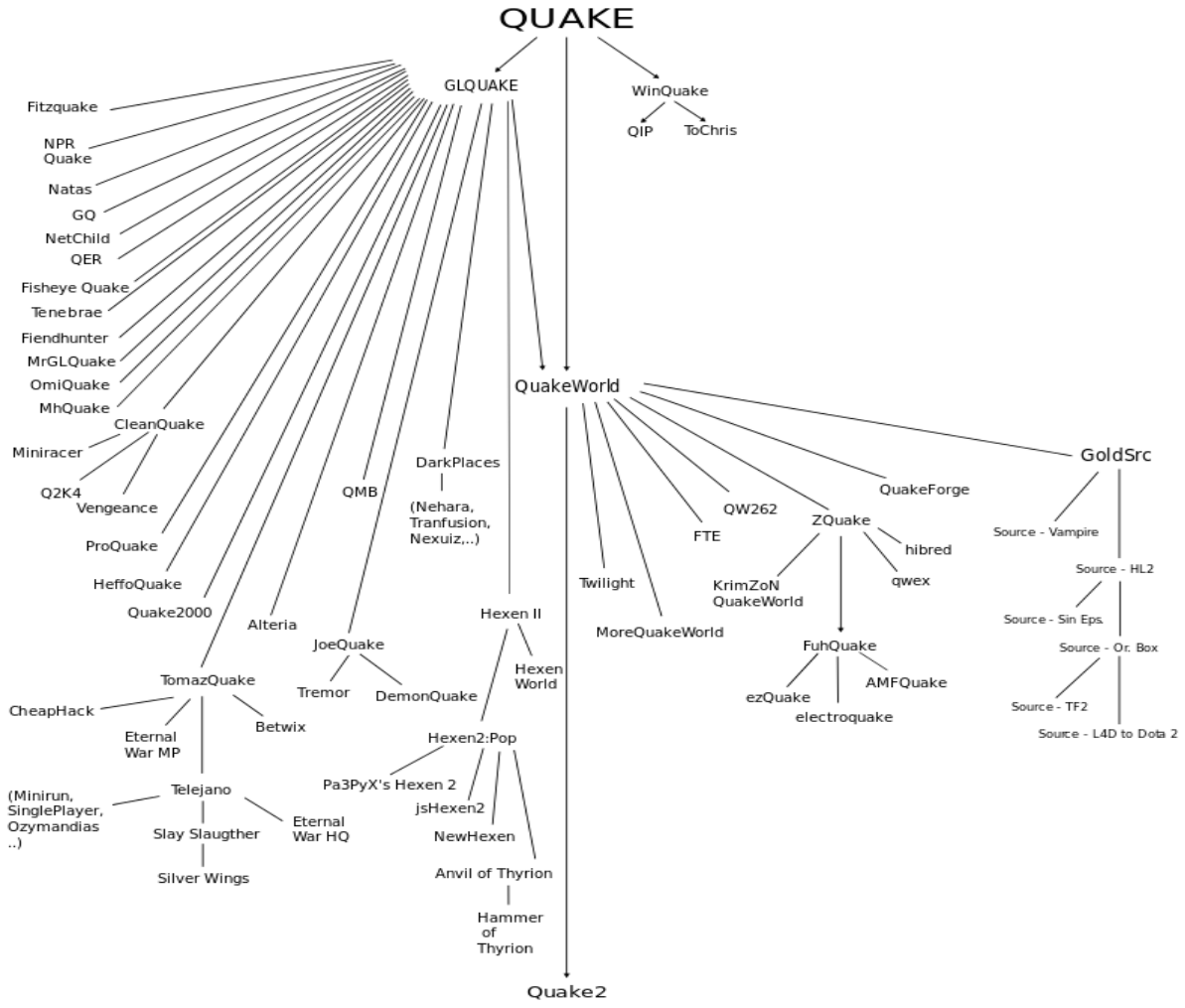

Рисунок 1.1, аркуш 1 - Дерево розвитку ігрових двигунів, заснованих на Quake Engine

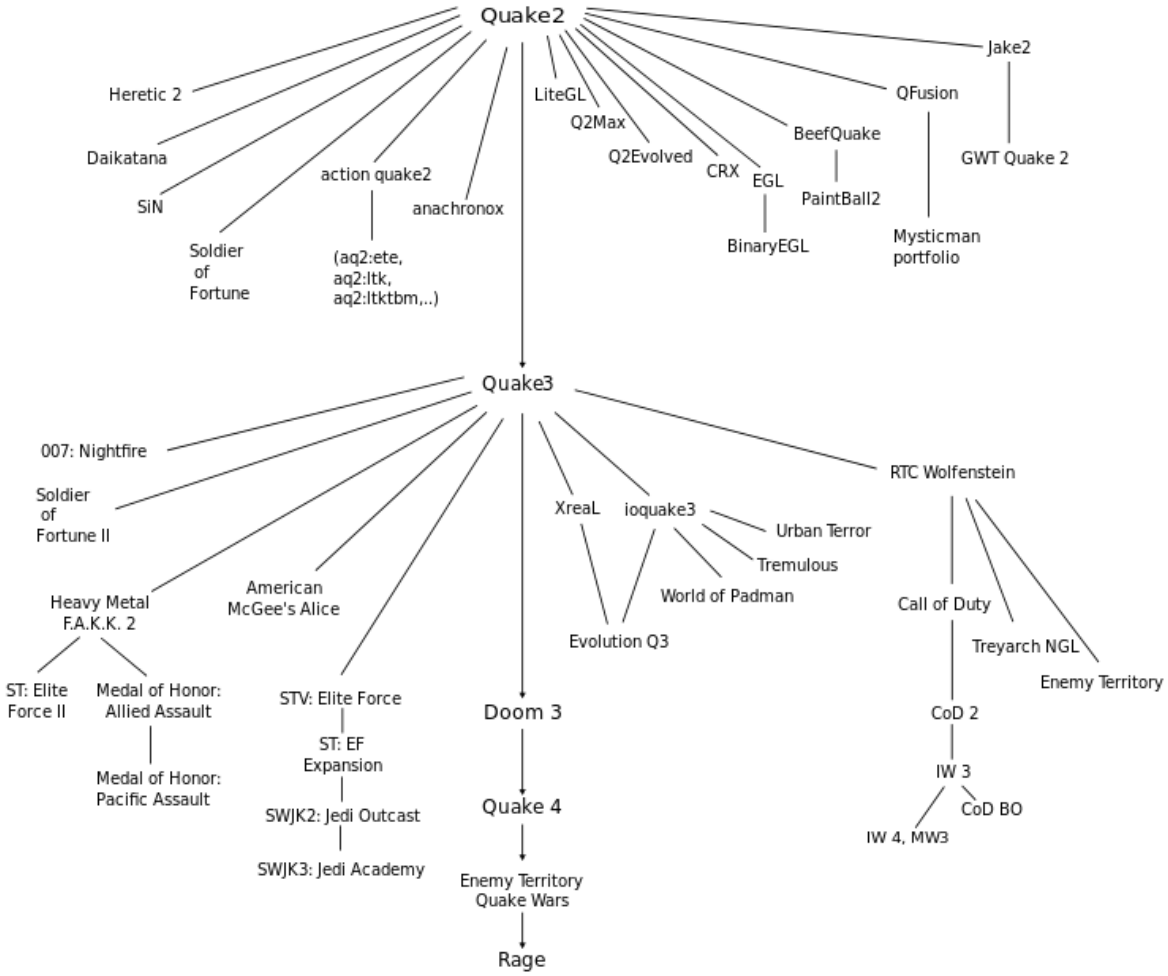

Рисунок 1.1, аркуш 2

У сучасних моделях ігрових середовищ розробки [8, 9], двигуни використовуються в якості механічної основи ігор. Складаються з безлічі компонентів-модулів, які реалізують ігровий функціонал у вигляді відображення, обробки графіки, звуку, штучного інтелекту, залишаючи тільки накласти контент, що буде відповідати конкретно до розробленої гри.

Модульний дизайн ігрових двигунів дозволяє розробникам легко змінювати його частини та модифікувати їх з для створення нових ігор з новими моделями, графікою, звуками, сценарієм, змінювати існуючий матеріал і додавати нові функції. Завдяки цьому на базі існуючих двигунів було створено і створюється безліч нових ігор.

Основним недоліком модульних ігрових двигунів є їх універсальність систем та механізмів, що супроводжує собою труднощі і неможливість оптимізації та використання механізмів специфічних для певної платформи або завдання ігрового проекту.

У багатьох випадках для вирішення подібного недоліку програмістам доводиться допрацьовувати ігровий движок.

Основними лідируючими середовищами розробки ігрових додатків є:

– Unreal Engine;
– CryEngine;
– HeroEngine;
– Source;
– Project Anarchy;
– Unity.

Unreal Engine [10] один з найбільш популярних двигунів для розробки AAA-ігор (рис.1.2), написаний на мові C++, що розробляється Epic Games. Після створення першої гри на Unreal 1 в 1998 році. З тих пір різні версії Unreal Engine були використані в десятках ігор, у тому числі Deus Ex, Lineage II, Thief: Deadly Shadows, серіях ігор Brothers in Arms, Tom clancy's Splinter Cell, Tom clancy's Rainbow Six і, звичайно, в серії ігор Unreal. Будучи пристосованим в першу чергу для шутерів від першої особи, двигун використовувався і при створенні ігор інших жанрів.

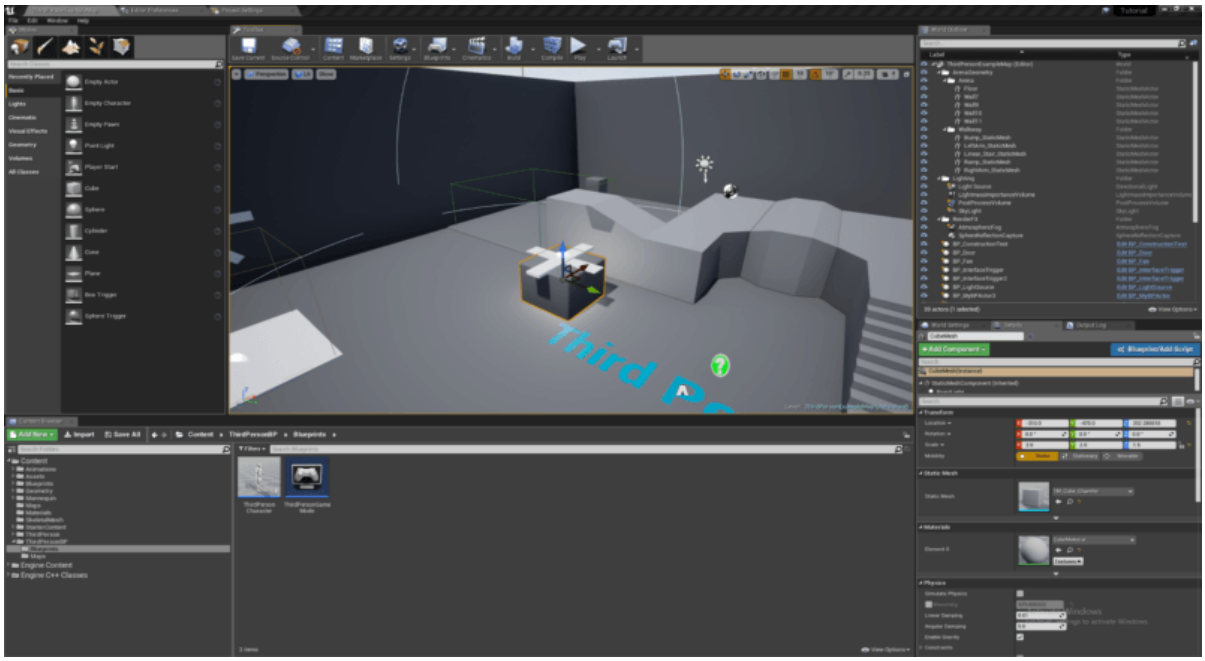

Рисунок 1.2 - Вікно програми в режимі редагування розташування об'єктів

Основні переваги:

– повністю безкоштовний, лише при 5% роялті згідно ліцензійної винагороди;

– можливість використовувати в проекті файли з вихідним кодом на C++;
– відмінна технічна підтримка і механізм апгрейту;
– широкий асортимент інструментів для різних цілей;
– зручний механізм оновлень;
– мультиплатформеність.

CryEngine [11] орієнтований на створення крос-платформних ігор AAA-класу, призначених для PC і консолей (рис.1.3). В даний час підтримуються платформи Xbox 360, Xbox One, PlayStation 3-4, WiiU, з технології візуалізації DirectX 9-12, але без підтримки мобільних платформ. CryEngine має повну підтримку механізму мультиплеєрних ігор та приголомшливий список технологій візуалізації. Фізичний компонент двигуна CryPhysics також працює незалежно від фізичних API, таких як PhysX. Вбудована система анімації пропонує кілька відмінних підсистем: індивідуалізація персонажів, параметрична скелетна анімація, процедурне деформування руху. Також заслуговує на окрему увагу вбудована система штучного інтелекту, що дозволяє обробляти поведінку не тільки персонажів, але і транспортних засобів. Вона складається з трьох модулів: розумні об'єкти, алгоритми динамічного виявлення шляху, а також система, керована сценаріями.

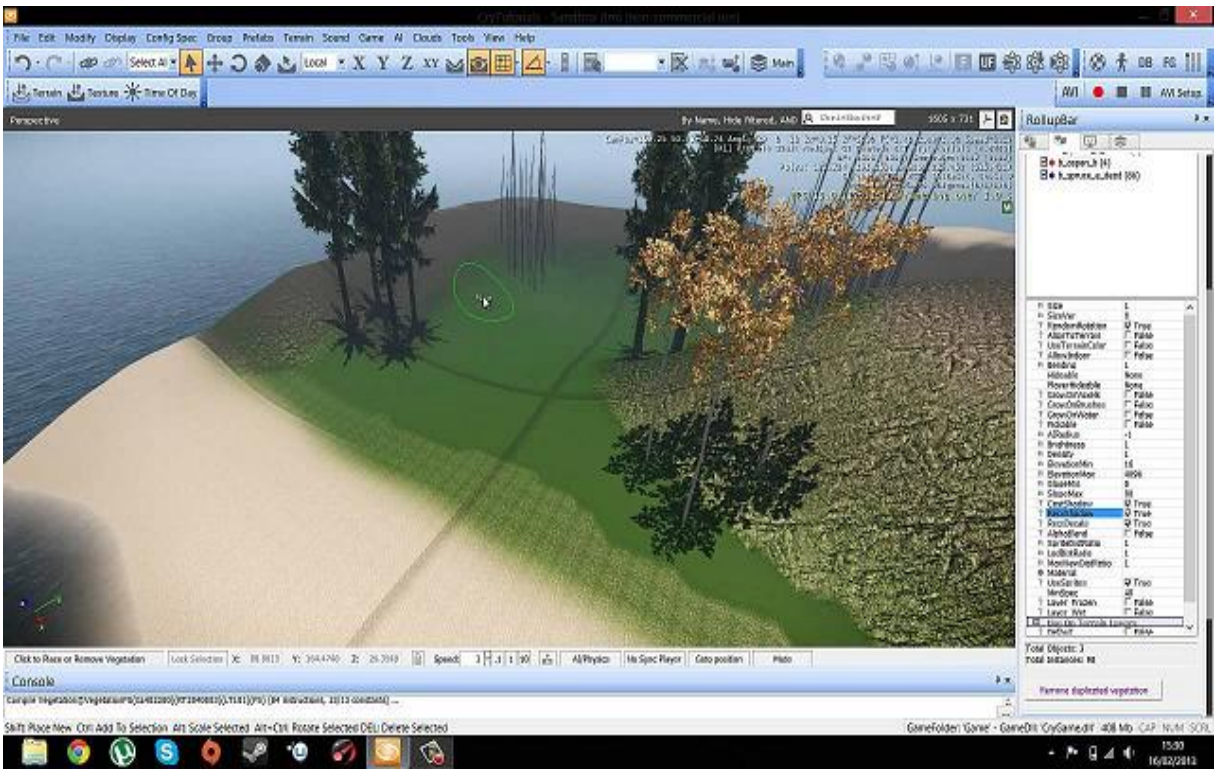

Рисунок 1.3 - Вікно програми в режимі редагування місцевості

Основні переваги:

– не складне створення реалістичною графіки;
– створення потужного звукового супроводу;
– найпростіший процес створення AI;
– розробки ігор для PC, Xbox 360, PlayStation 3 з підтримкою DirectX9-12.

Основні недоліки:

– недбала технічна підтримка безкоштовної версії;
– підтримувана ОС редактору розробки тільки Microsoft Windows;
– відносно високий поріг входження.

HeroEngine [12] популярний як двигун для розробки MMO ігор (рис.1.4), включає розробку клієнта і сервера. Для розробки використовується власна мова програмування (HeroScript), але можливо використовувати C++. Підтримується тільки операційна система Windows. Основним відштовхуючим фактором є висока ціна ліцензії вартістю близько 100$ в рік.

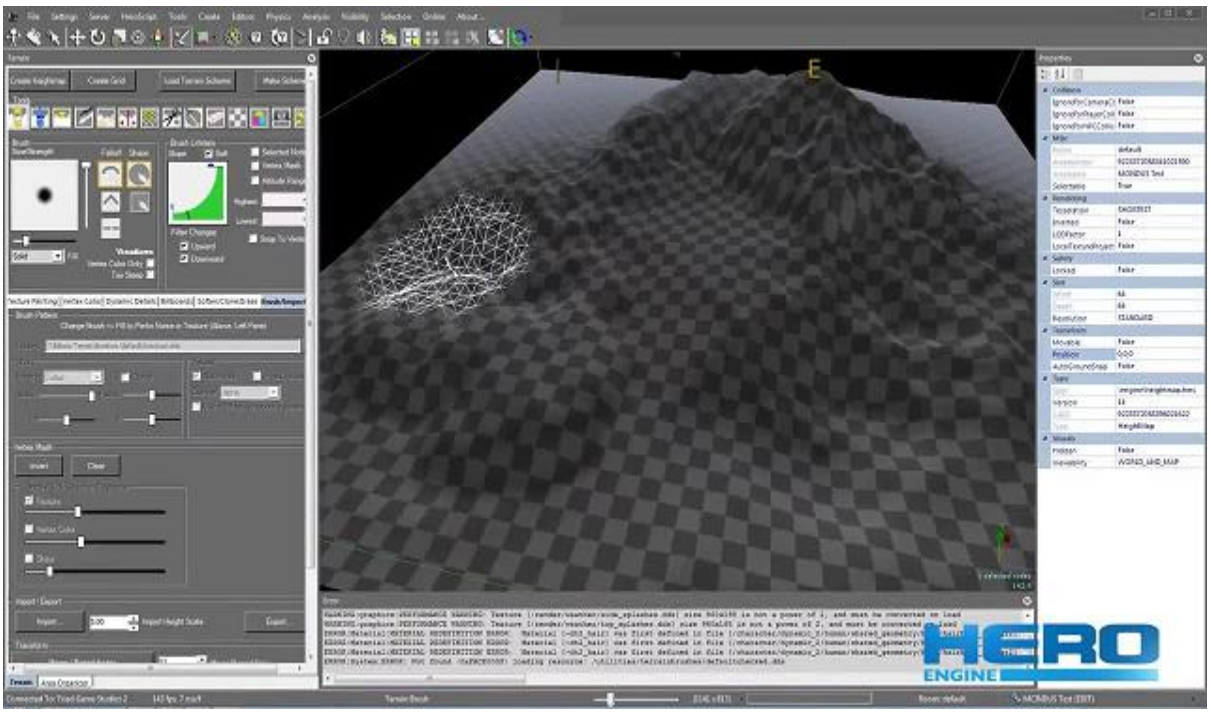

Рисунок 1.4 - Вікно програми в режимі побудови рельєфу

Основні переваги:

– дуже потужний інструмент для створення AI;
– окремий зручний сервіс HeroCloud, через який здійснюється техпідтримка.

Основні недоліки:

– 100 доларів на рік за одне робоче місце;
– HeroEngine з HeroCloud дуже дорого коштують разом;
– потужний скриптовий двигун, але дуже незручний в управлінні;
– відносно високий поріг входження.

Source [13] - розроблений компанією Valve Corporation для створення власних комп'ютерних ігор. Версія двигуна була вперше використана для ігор Half-Life: Source і Counter-Strike: Source. Завдяки модульній структурі двигун постійно доопрацьовувався, підтримуючись на актуальному рівні (рис.1.5). Source містить в собі складний мережевий код, що забезпечує ефективну підтримку для 32 гравців по LAN і Internet, також містить у собі повний набір інструментів для дизайну, анімації персонажів, створення демонстраційних версій і багато чого іншого.

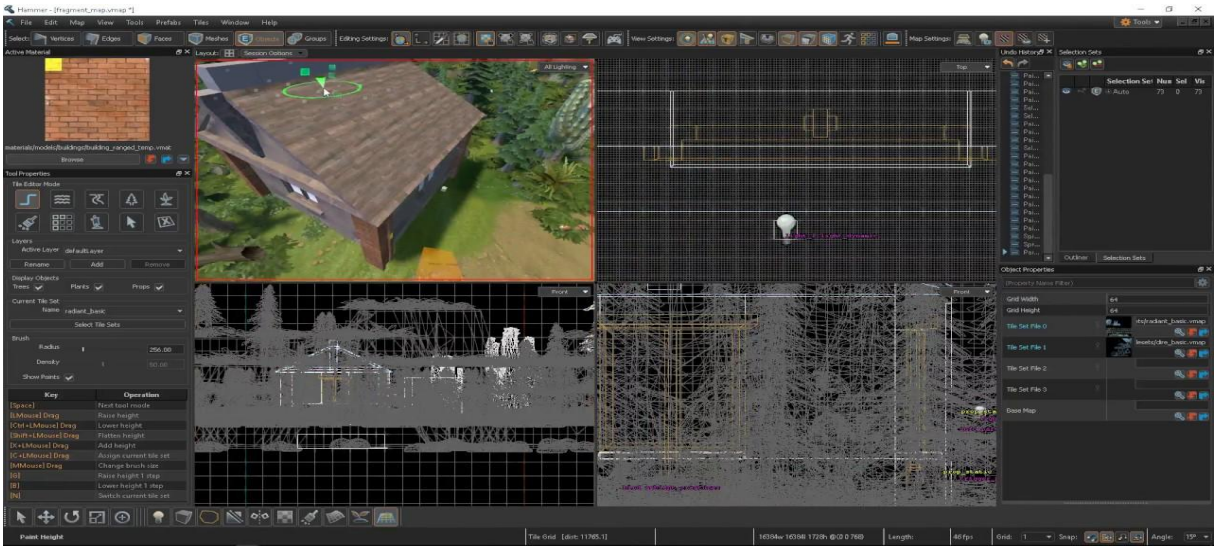

Рисунок 1.5 - Вікно програми в режимі перегляду моделі

Основні переваги:

– оптимізація для слабких ПК;
– постійне розширення двигуна;
– швидкий багатокористувацький режим;
– низький поріг входження.

Основні недоліки:

– підходить тільки для ігор з малим простором;
– постійна некоректна геометрія освітлення.

Project Anarchy [14] - ігровий двигун який подобається багатьом розробникам. Безкоштовна ліцензія - під час розробки гри платити доведеться тільки в тому випадку, якщо ви захочете купити модуль для експорту у другу платну платформу, що не входить в основний набір.

Цей двигун орієнтований в першу чергу на розробку мобільних ігор, але підтримується можливість використовувати один проект для компіляції під всі заявлені платформи, в тому числі і на стаціонарні (рис.1.6). При цьому розробка і випуск ігор на цьому двигуну вільні для більшості популярних платформ і для будь-яких компаній, незалежно від їх розміру та фінансового обороту.

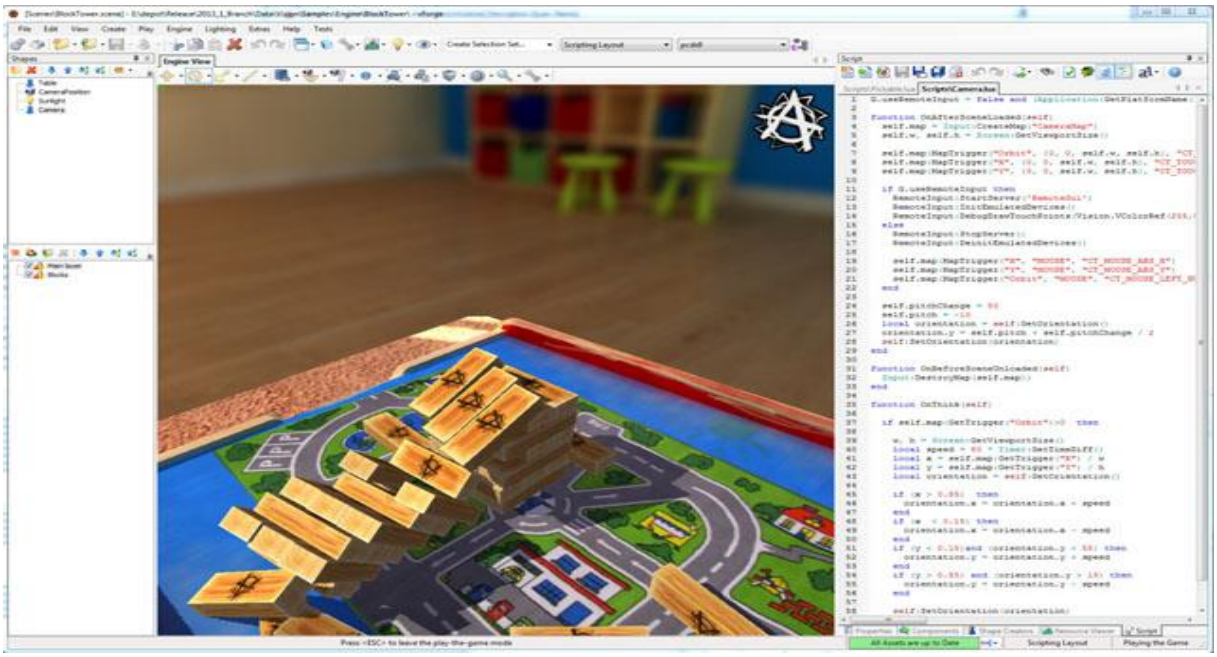

Рисунок 1.6 - Вікно програми в режимі коригування скриптів

Основні переваги:

– безкоштовна ліцензія для платформ: iOS, Android і Tizen;
– відмінне співтовариство;
– якісна документація;
– функції Fmod для аудіо-супроводу;
– відмінний AI.

Основні недоліки:

– немає можливості розробки гри для таких платформ, як Linux і Mac;
– дорога ліцензія для платформи Windows.

Unity [15] - найпопулярніше міжплатформне середовище розробки для створення 2D і 3D-ігор. Для досягнення лідируючих позицій в ігровій індустрії розробники негайно реалізують її в Unity всі нові ігрові/графічні технології. Крім розробки одиночних ігор для PC, також існує підтримка портування гри під інші ОС, консолі і мобільні технології за додаткову доплату близько 1500 доларів за кожну платформу: iOS, Android, BlackBerry.

Також є ціла індустрія, що працює над створенням доповнень і розширень двигуна, серед них є як спеціалізовані серверні рішення для Unity, так і засоби для розробки користувальницького інтерфейсу та конструкторів, призначених для створення ігор певних жанрів (рис.1.7).

Особливість в Unity полягає в низькому порігу входження для початківців користувачів, завдяки цьому, а також тому, що інді-версія безкоштовна, навколо двигуна організувалося величезне співтовариство. Низький поріг входження є результатом грамотного дизайну додатку - багато речей можна виконати за допомогою різних редакторів, не написавши при цьому ні строчки виконавчого коду на JavaScript або C#. Вихідний код на C/C++ закритий.

Значною перевагою є наявність безкоштовної версії за умови, що дохід з гри не складе вище 100 000 доларів в рік. Що є вдалим рішенням для стартапів і початкових розробників.

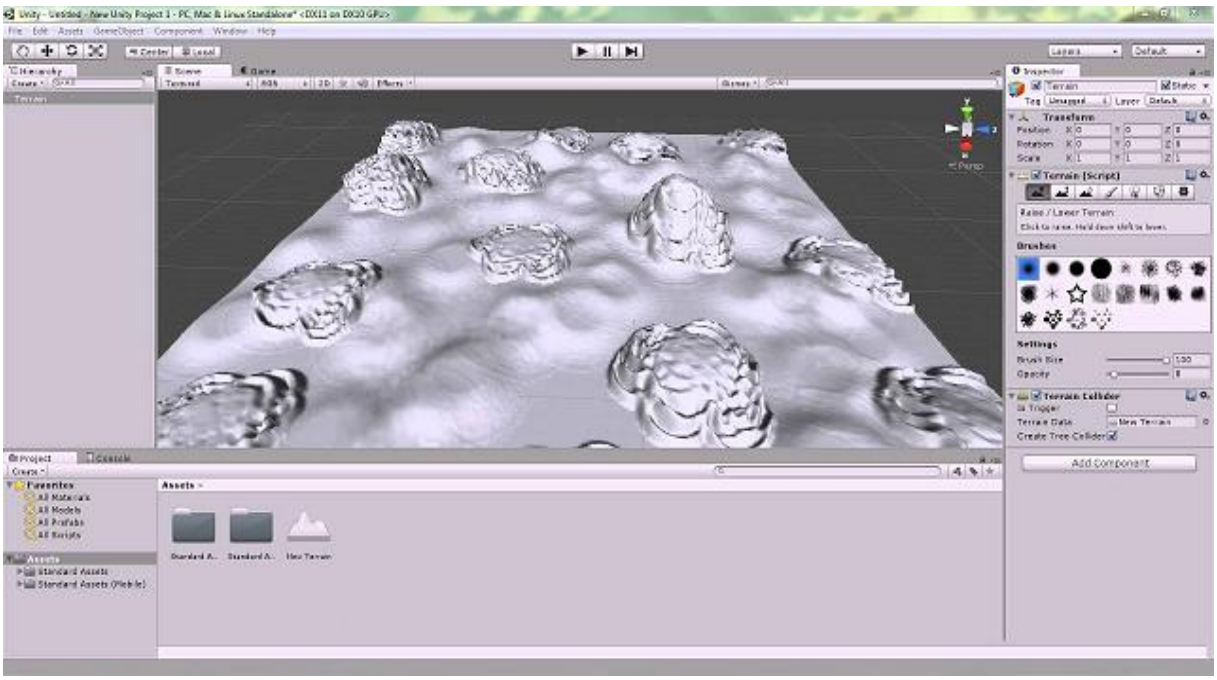

Рисунок 1.7 - Вікно програми в режимі редагування полігональної сітки рельєфу

Основні переваги:

– вигідна ліцензійна політика;
– сумісність з будь-якою платформою;
– доступний і зрозумілий інтерфейс;
– можливість доповнення функціоналу;
– можливість використання систем контролю версій;
– підтримка імпорту великої кількості форматів;
– відмінна технічна підтримка;
– низький поріг входження;
– сценарії на C#, JavaScript.

Основні недоліки:

– обмежений набір безкоштовних інструментів;
– обмежений випуск під платформи у безкоштовній версії;
– процес налагодження гри забирає багато часу;
– проблеми з налаштуванням мережевого режиму для гри;
– постійні проблеми з малою продуктивністю і великим споживанням пам'яті для кінцевого ігрового додатку.

1.3 Постановка мети і завдань магістерської роботи

Метою магістерської роботи є дослідження архітектури, методів і алгоритмів для створення прикладного програмного середовища розробки ігрових додатків будь-якої складності та розробка програмного прототипу ігрового середовища розробки орієнтованого на ефективне використання системних ресурсів за рахунок використання спеціальних внутрішніх механізмів.

Метою дослідження є підвищення якості та розширення прийнятих концепцій ігрової розробки, шляхом застосування нових програмних методів та архітектурних рішень.

Для досягнення мети дослідження необхідно вирішити такі завдання:

– проведення огляду наукових досліджень, спрямованих на вирішення завдань ігрової розробки;
– дослідження архітектури, методів і алгоритмів для створення прикладного програмного середовища розробки ігрових додатків;
– аналіз придатності використання публікацій в якості базису деяких

підсистем ігрового середовища розробки;

– визначення вимог до розроблюваної IDE;

– проектування програмної архітектури IDE;

– створення програмного прототипу, метою якого є перевірка придатності узгоджених для застосування концепцій, архітектурних і технологічних рішень;

– створення демонстраційного ігрового додатку.

Для проведення досліджень доцільно застосовувати методи аналізу і моделювання. Також при розробці прикладної програми використовується метод ітеративної розробки [1], завдяки чому забезпечується паралельна розробка з аналізом отриманих результатів та подальшим коректуванням розробки програмного забезпечення.

Висновки до розділу 1

За результатами дослідження сформовано наступні висновки:

1) використання ігрового середовища значно спрощує процес розробки ігор. Дозволяючи розробникам працювати над грою більш абстрактно, не замислюючись про низькорівневе представлення компонентів двигуна і те, як вони працюють і як пов'язані між собою. До того ж виконуваний код, написаний з використанням ігрового середовища, виглядає більш організованим і більш керованим, що знижує загальну кількість виникаючих програмних помилок;

2) ґрунтуючись на аналізі існуючих середовищ розробки ігрових додатків із вбудованими ігровими двигунами, як комерційних так і безкоштовних, підтверджується доцільність розробки IDE для створення ігрових додатків, націлившись спочатку на конкретні потреби, а в подальшому реалізувати його архітектуру таким чином, що б його можна було легко підтримувати актуальним і адаптувати під проекти будь-якого роду. Основним підтверджуючим фактором є висока вартість комерційних рішень, та накладеними обмеженнями при використанні безкоштовних рішень;

3) обрані програмні та інструментальні засоби розробки протягом тривалого часу неодноразово підтверджували свою перевагу у використанні, відрізняючись стабільністю, економністю і продуктивністю роботи, при використанні мінімуму системних ресурсів без зменшення якості;

4) робота над проектом по створенню ігрового середовища розробки представляє із себе досить складний і великий проект, що дозволяє істотно підняти рівень навичок в програмуванні та розробки складних автоматизованих програм, актуальність якого буде забезпечена протягом тривалого часу.

РОЗДІЛ 2
АНАЛІЗ І ОПИС ВИКОРИСТОВУВАНИХ МАТЕМАТИЧНИХ, СТРУКТУРНИХ ТА АРХІТЕКТУРНИХ РІШЕНЬ

В даному розділі більш детально описуються причини прийняття тих чи інших рішень пов'язаних з розробкою ігрового середовища розробки, а саме: архітектурних, структурних і функціональних рішень. Також наводяться теоретичні розрахунки, орієнтовані на підтвердження доцільності прийнятих рішень націлених на виконання поставлених завдань описаних раніше в першому розділі роботи.

2.1 Опис архітектури програмного забезпечення

Архітектура програмного забезпечення (англ. software architecture) - це структура програми або обчислювальної системи, яка складається з програмних компонентів, декількох рівнів абстракції та відносинами між ними. В свою чергу архітектура може мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

Процес проектування архітектури програмного забезпечення включає в себе збір вимог, їх аналіз і створення проекту для компонента програмного забезпечення у відповідність з вимогами. Успішна розробка ПЗ повинна забезпечувати баланс неминучих компромісів внаслідок суперечливих вимог; відповідати принципам проектування і рекомендованим методам, виробленим з часом; і доповнювати сучасне обладнання, мережі та системи управління. Надійна архітектура програмного забезпечення вимагає значного досвіду в теоретичних і практичних питаннях, а також уяви, необхідного для перетворення сценаріїв і вимог.

Архітектура програмного забезпечення включає в себе структурованого рішення, що відповідає всім технічним і робочим вимогам, одночасно оптимізуючи загальні атрибути якості, такі як продуктивність, безпека і керованість. Сюди входить серія рішень, заснованих на широкому діапазоні факторів. Кожне з яких, може значно впливати на якість та продуктивність розробки.

Програмне забезпечення рідко буває автономним. Зазвичай завжди має взаємодіяти з зовнішніми іншими службами та мережевими функція для виконання перевірки автентичності, отримання інформації та надання інтегрованих середовищ роботи користувачів. Без відповідної архітектури може бути складно, якщо взагалі можливо, здійснити розгортання, експлуатацію, обслуговування та успішну інтеграцію

з іншими системами.

Архітектуру програмного забезпечення можна розглядати як зіставлення між метою компонента і відомостями про реалізацію в коді. Правильне розуміння архітектури забезпечить оптимальний баланс вимог і результатів. Програмне забезпечення з добре продуманою архітектурою буде виконувати зазначені завдання з параметрами вихідних вимог, одночасно забезпечуючи максимально високу продуктивність, безпеку та надійність.

2.1.1 Вимоги до архітектури

Вимоги до програмного забезпечення з кожним разом стають все більш складними, оскільки користувачі очікують все більше від додатків. Можливостей простих автономних настільних додатків більше не достатньо для більшості комерційних і ділових ситуацій. У світі розвиненого зв'язку додатки повинні взаємодіяти з іншими додатками і службами, а також працювати в різних середовищах та портативних пристроях.

Ці труднощі впливають не тільки на архітектуру, але також і на розгортання, обслуговування та адміністрування програмного забезпечення. Сукупна вартість володіння програмного забезпечення тепер в основному складається з витрат, що виникають після розгортання. Додаток з добре продуманою архітектурою забезпечить мінімальну сукупну вартість володіння завдяки зниженню витрат і часу, необхідних для розгортання програми, забезпечення його роботи, оновлення для задоволення мінливих вимог і усунення проблем.

Успішне програмне забезпечення також має відповідати декільком важливим критеріям:

– забезпечувати безпеку, бути стійким і надійним для мінімізації збоїв і відповідних витрат;

– працювати з необхідними параметрами у відповідність до вимог користувачів, такими як максимальний час відгуку або певне робоче навантаження;

– бути простим у підтримці для зниження витрат на адміністрування та підтримку і досить розширюваним для включення неминучих оновлень, які з часом будуть потрібні.

З усіма цими факторами пов'язані деякі компроміси. Наприклад, реалізація найбільш безпечних механізмів з використанням складного шифрування вплине на

продуктивність. Реалізація безлічі параметрів конфігурації та оновлення може ускладнити розгортання та адміністрування. Крім того, чим складніше архітектура, тим дорожче її реалізація. Правильна архітектура повинна забезпечувати баланс цих факторів з метою отримання оптимального результату для певного сценарію.

2.1.2 Проектування архітектури

Проектування архітектури програмного забезпечення, яка мала достатню масштабованість, відповідність сучасним технологіям, бути досить гнучкою і підтримувати різні типи ігрових додатків, забезпечила б надійний фундамент для подальшого розвитку і стала б початком стандартизації взаємодії між компонентами використовуваними в грі. Така архітектура розвивалася і розширювалася б разом зі збільшенням складності сучасних ігор і при цьому не обмежувала б творчий підхід розробників ігор. Для того щоб досягти цієї мети, підсумкова архітектура повинна відповідати наступним вимогам:

– підтримка концепції компонентної розробки;
– приховування деталей реалізації та доступність застосування без знання предметної області;
– гнучкість і кодифікованість;
– масштабованість і експлуатаційна надійність.

Підтримка концепції компонентної розробки

Архітектура повинна мати сувору логічну структуру, а також необхідно розділити систему на логічно незалежні частини, щоб кожна з підсистем могла бути розроблена і видалена незалежно від інших. Таким чином ці компоненти повинні легко інтегруватися в підсумковому ігровому додатку і не вимагати для цьому великих змін ігрового коду.

Приховування деталей реалізації та доступність застосування без знання предметної області

Архітектурна вимога про приховування деталей реалізації та доступності застосування існує через різні можливості, необхідні в різних іграх. У сучасних іграх потрібно висококласна графіка, реалістична фізика, людиноподібний штучний інтелект, звук кінематографічної якості. Навіть якщо розробник використовує різні компоненти для створення всього цього в грі, він повинен добре знати предметну область - для того, щоб використовувати конкретний компонент коректно.

Гнучкість і модифікованість

Гнучкість є ключ до майбутнього ігрової розробки. Пропонована архітектура не залежить ні від жанру гри, ні від технологій, що втілюються в ній і дозволяє розробникам створювати різні ігри, використовуючи різні технології. Для того, щоб ця архітектура прижилася в ігровій індустрії, вона повинна бути досить гнучка, щоб будь-яка гра змогла використовувати її.

Масштабованість і експлуатаційна надійність

Іншими критичними вимогами до архітектури, що також випливають з проблеми швидкого зростання вартості розробки, є масштабованість і експлуатаційна надійність. Успішні ігри часто з'являються заново з різними доповненнями і поліпшеною якістю.

2.2 Загальна архітектура розроблюваної ігрової IDE

Як описано в першому розділі, ігрове середовище розробки представляє з себе комплекс взаємопов'язаних програмних методів і засобів спрямованих на спрощення і підвищення продуктивності розробки ігрового програмного забезпечення.

Як і будь-яке інше програмне забезпечення подібного рівня, середовище розробки має певну архітектуру. Оскільки не існує єдиної архітектури для подібного типу проекту, а існуючі рішення не виконають поставлені вимоги, які полягають в: ефективності, гнучкості, розширюваності і високій сумісності з мовою програмування C++, була розроблена спеціалізована архітектура “об'єктно-орієнтований моноліт”, яка задовольняє поставленим вимогам та завданню проекту розробки.

Архітектура об'єктно-орієнтованого моноліту представляє собою взаємопов'язані і взаємозалежні компоненти, що знаходяться в єдиному адресному просторі виконуваної програми. Завдяки даному рішенню мінімізуються затримки на спілкування між компонентами, так як для отримання специфічних даних досить безпосередньо звернутися до відповідного компоненту, що значно скорочує менеджер повідомлень, залишаючи останньому тільки обробку необхідних подій.

Дана архітектура орієнтована на використання спеціалізованих підсистем і функціональних частин, пов'язано це з тим, що при використанні універсального підходу при зверненні до компонентів необхідно буде виконати додаткові перевірки на приналежність даних, крім загального зниження продуктивності, також підвищується складність і розмір виконуваного коду.

Використання спеціалізованого підходу вирішує дані недоліки, проте залишається необхідність у висококваліфікованому програмісті, який знає всі нюанси і

особливості даної розробки.

Більш наочний зразок даної архітектури представлено на рисунку 2.1.

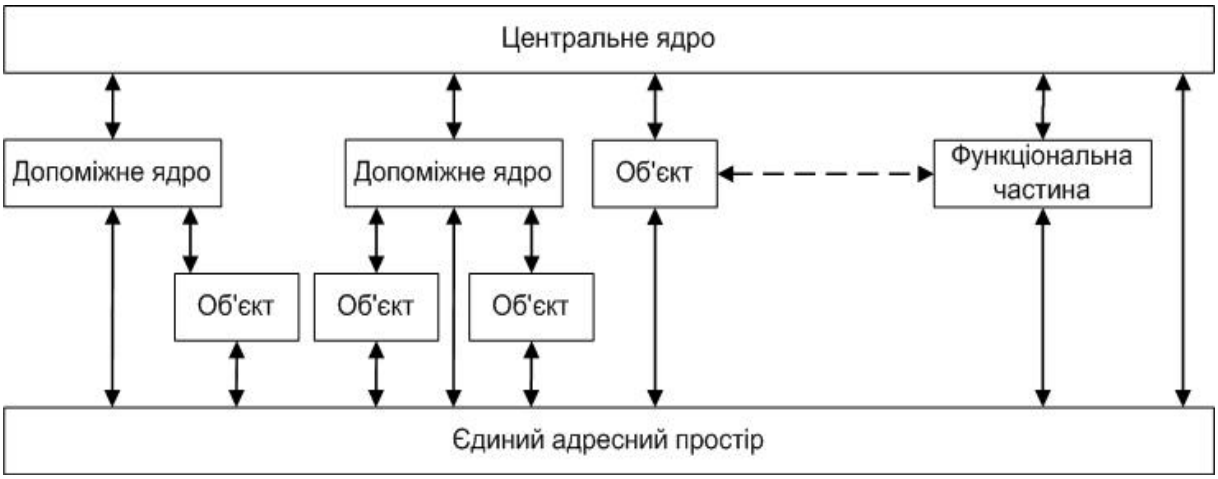

Рисунок 2.1 – Зразок об'єктно-орієнтованої монолітної архітектури

Компонентами даної архітектури є:

– центральне ядро, має найвищий пріоритет над усіма іншими компонентами, що відповідає за контроль над роботою всієї системи;

– допоміжні ядра, що мають високий пріоритет з можливістю виконання у власному потоці виконання, до складу якої входять підсистеми згрупованих компонентів за певною спільною ознакою;

– об'єкти, що мають середній пріоритет виконання, є як кінцевим видом, так і розширюваною частиною іншого об'єкта;

– функціональні частини, призначені на виконання часто незалежних частин виконуваного коду.

Дана архітектура має строгі типи зв'язків:

– глобальні зв'язки, доступні всім системам, підсистемам, компонентам;

– виняткові зв'язки, що мають прямий доступ до необхідних частин системи, підсистеми, компонентів, функціональних частин. Без будь-яких проміжних переходів;

– локальні зв'язки, доступні в межах групи систем, підсистем, компонентів, функціональних частин;

– закриті, зв'язку в межах іменованого простору, без успадкування або можливої передачі іншим виконуваним частинам відмінною від поточної.

Дана архітектура має ряд переваг, а саме:

– забезпечення високої продуктивності виконання, яка досягається за рахунок безпосереднього використання рідного API виконуваної OS, та використання прямого доступу до даних загального призначення без використання важких синхронізуючих механізмів;

– бути єдиною неподільною програмною одиницею - полягає в об'єднанні різних об'єктів програми в межі однієї програми однієї платформи одного адресного простору;

– спрощена розробка і розгортання - так як об'єкти монолітного програмного забезпечення взаємопов'язані і взаємозалежні, це дозволяє програмному забезпеченню бути самодостатнім. Крім того, всі дії виконуються з одним каталогом, що спрощує розгортання;

– гнучкість і розширюваність - так як основні компоненти архітектури крім ядер представлені у вигляді об'єктів то при зміні, оновленні, розширенні, доводиться модифікувати лише відповідальні об'єкти за свої частини.

Також в даній архітектурі є негативні моменти, а саме:

– при зміні, оновленні, розширенні кодової бази необхідно враховувати всі взаємозв'язки і особливості відповідальних за свої ділянки об'єктів, що накладає певні вимоги до кваліфікації та знання архітектури проекту на програміста;

– складність перенесення на нові архітектури процесора або операційних систем через значну прив'язку до останніх;

– висока складність модифікацій частин ядер системи, так як навіть незначна модифікація може спричинити за собою необхідність в повному перегляді і модифікації всіх дочірніх об'єктів ядра.

Архітектура ігрового середовища розробки являє собою складну і дуже масштабовану систему, розширювану в залежності від типу шаблону ігрового проекту.

Загальна архітектура для всіх типів шаблону ігрового проекту представлена на рисунку 2.2.

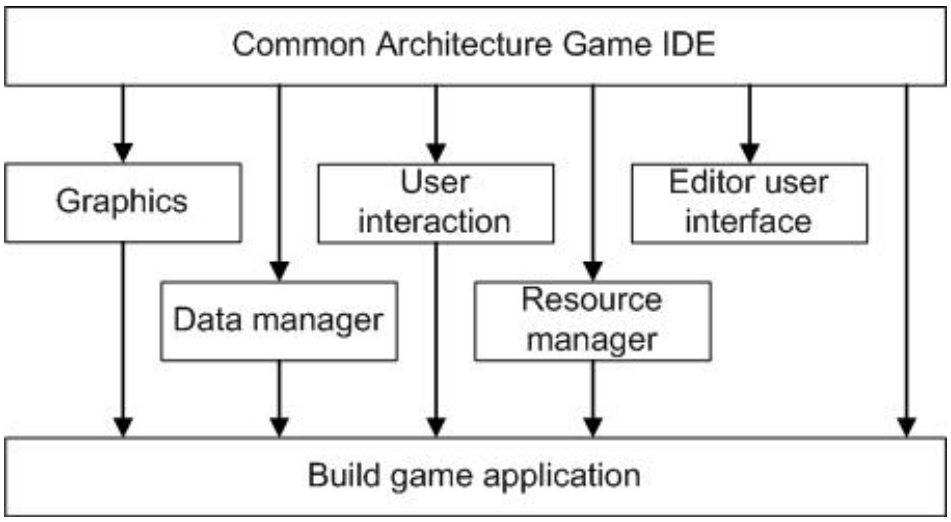

Рисунок 2.2 – Загальна архітектура ігрової IDE

В якості перевірки придатності застосування архітектурних і технологічних рішень, в даній магістерській роботі було поставлено завдання - розробити демонстраційний програмний прототип, до складу якого входять базові підсистеми для створення демонстраційного ігрового додатка.

Ключовою особливістю розроблюваної архітектури і підсистем є - *використання спеціалізованих підсистем і механізмів згідно типу шаблону ігрового проекту.*

На відміну від існуючих ігрових середовищ розробки, базова архітектура яких орієнтована на використання певної спеціалізації ігрового жанру, а також універсальності механізмів і підсистем, що призводить до певних обмежень і складнощів в розробці ігрових жанрів і ігрових концепцій відмінних від базової, які в свою чергу відбиваються на якості і продуктивності кінцевого програмного забезпечення.

Дана розробка орієнтована на відмову від універсальності підсистем і механізмів, залежних від обраного ігрового жанру, системи виконання, в угоду шаблонної спеціалізації. Шаблонна спеціалізація - являє собою наявність визначених статичних механізмів і набору підсистем які строго відповідають обраному типу ігрового шаблону.

Розроблювані ігрові програми мають наступні переваги:

– найвища продуктивність ігрових додатків - так як підсистеми і механізми розроблюваного ігрового проекту спочатку орієнтовані на особливості виконуваної операційної системи;

– більш ефективна і спрощена розробка - так як підсистеми, механізми і графічний користувальницький інтерфейс редактора спочатку орієнтований відповідно

до типу шаблону проекту;

– відсутність заглушених і/або баластних підсистем і механізмів - так як використовуються підсистеми і механізми безпосередньо необхідні відповідно до обраного типу шаблону проекту;

– мінімалізм IDE - до складу програмного забезпечення ігрового IDE входять тільки використовувані ресурси згідно обраного шаблону ігрового проекту.

Також є і негативні моменти:

– висока складність і відповідальність при виборі стартового типу шаблону ігрового проекту;

– складність перенесення ігрового проекту на інший тип шаблону, відмінним від поточного без використання вбудованого майстра перенесення проектів;

– підтримка платформ, ігрових жанрів, систем виконання тільки в разі наявності необхідного шаблону.

2.3 Дослідження придатності використання публікацій в якості базису деяких підсистем ігрового середовища розробки

Завдяки розроблюваної спеціалізованої архітектури IDE, а також підсистем в цілому, кожен тип шаблону ігрового проекту має можливість в роздільній розробці. Це в свою чергу дозволяє використовувати найбільш ефективні технології, методи та способи відповідно до області типу ігрового шаблону.

В даному підрозділі описується використання двох публікацій до підсистем карти і менеджера даних для досягнення поставленого завдання дипломного проекту.

Використовувані публікації:

– деталізація сферично-динамічного рельєфу в комп’ютерній графіці [2];

– розробка та аналіз програмного рішення файлового кешування даних в мові C++ на основі технології memory mapped files [3].

2.3.1 Підсистема карти типу динамічна обмежена карта висот

Загальна підсистема карти представляє собою набір механізмів для: створення, управління, складання та рендерингу всієї ігрової місцевості згідно шаблону ігрового проекту.

В даний момент буде розглядатися тип карти “динамічна обмежена карта висот”,

зі скороченою назвою в межах проекту розробки - DLMH (Dynamic Limited Height Map). Визначення "динамічна" в даному контексті означає, що будь-яка ділянка карти окрім зміни в редакторі, схильна до змін безпосередньо в процесі гри.

Даний тип карти являє собою обмежений масив MxM кратний ступені двійки, розмірність якого задається на етапі створення ігрового проекту і не підлягає зміні в подальшому. Більш наочний приклад карти висот представлений на рисунку 2.3.

	0	1	2	3	4
0	1	2	2	2	3
1	1	1	2	2	2
2	0	1	2	2	1
3	0	1	1	1	1
4	0	0	0	0	0

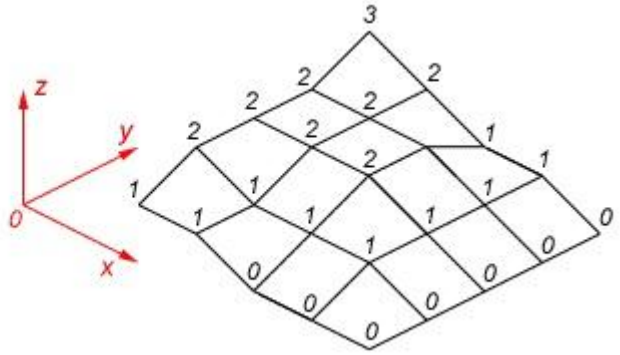

Рисунок 2.3 – Масив і графічне представлення масиву карти висот

Використання карти висот має значну перевагу, так як для отримання і/або зміни даних значень вершин досить виконати два зміщення по стовпцю та рядку масиву. Використання тільки 2 операцій зміщення, позитивно позначається на швидкості виконання всього додатка в цілому, так як на рівні процесора дані операції обходяться значно дешевше, ніж при використанні інших арифметичних операції.

Негативним моментом є обмеження на розмірність карти, так як для зберігання "сирих" даних необхідна значна кількість оперативної пам'яті.

Наприклад, при розмірності карти в 8192м^2, з урахуванням використання деталізованої карти, час проходження гравця в один напрямок складе близько години, в свою чергу необхідна кількість оперативної пам'яті згідно з формулою розрахунків 2.1, становитиме 268 435 456 байт оперативної пам'яті комп'ютера, для використання тільки даних вершини.

$$SizeByte = SizeMap^2 * BytePer \quad (2.1)$$

Де sizeBite - кінцевий необхідний розмір в байтах;

SizeMap - розмірність карти в м^2;

BytePer - точність висоти в байтах на 1 вершину.

Для сучасних комп'ютерів, що володіють в середньому по 8GB оперативної пам'яті, споживана кількість пам'яті може бути в межах допустимої норми. Основним же обмеженням є вихідні дані полігональної сітки місцевості збережені в відеопам'яті, в базовий склад якої входять координати вершин по 3 осях XYZ, так як використовується 3 мірний простір, і 2 текстурні координати на вершину для визначення положення текстури по вертикалі і горизонталі, загальна мінімальна вимога до відеопам'яті обчислено згідно з формулою розрахунків 2.2.

$$SizeByte = SizeMap^2 * (XYZ + UI) * Type \quad (2.2)$$

Де SizeBite - кінцевий необхідний розмір в байтах;

SizeMap - розмірність карти в $м^2$;

XYZ - використовувані осі для представлення тривимірного простору;

UI - текстурні координати по вертикалі і горизонталі;

Type - розмір стандартного типу даних геометрії=4.

$$1\ 342\ 177\ 280 = 8192^2 * (3 + 2) * 4$$

Використання 1,34 гігабайта для рендерингу тільки каркасної геометрії є неприпустимим незалежно від потужності сучасних відеоприскорювачів. Так як окрім геометрії місцевості існують і інші об'єкти, що вимагають підвищеної деталізації, в тому числі і необхідність зберігання в відеопам’яті високо деталізованих текстур, ведучи в кінцевому підсумку до переповнення відеопам'яті, що в кінцевому підсумку призведе до суттєвого зниження продуктивності всієї ігрової сцени. У кращому випадку частини даних будуть пересилатися з більш повільної пам'яті кожен кадр, призводячи до суттєвої зниження частоти виведення кадру із-за системних затримок під час запиту і пересилання даних по шині даних, в іншому випадку відбудеться помилка завантаження даних, призводячи до збою виконавчої програми.

Таким чином підсистема карти повинна мати керуючий механізм для динамічної зміни рівнів деталізації. Принцип дії якого буде в використанні менш деталізованої геометрії для найбільш віддалених ділянок місцевості і використанні підвищеної деталізації поруч зі спостерігачем, для забезпечення високої і плавної частоти кадрів протягом всієї роботи.

Для вирішення даного завдання, в якості базису підсистеми карти, буде використана авторська опублікована робота “Деталізація сферично-динамічного

рельєфу в комп’ютерній графіці” [2].

Дана робота представляє з себе рішення для візуалізації великомасштабного сферично-динамічного рельєфу з використанням механіки зміни рівнів деталізації розроблений на основі "дерева квадрантів" [16]. Також в даній роботі є розроблені механізми для читання та збереження даних, коригування, та обробку рельєфу, які згідно з результатами здатні обробляти рельєф загальною площею в 1 610 612 736 м^2.

Характерною особливістю в даній роботі є її висока швидкість виконання, простота методу і мінімізація споживаних ресурсів, не дивлячись на призначення для сферично-динамічного рельєфу. Ця робота використовує спосіб подання куба в якості сфери, створеної з 6 окремих незалежних площин збудованих на всі сторони куба, після виконання деформації якої перетворюється в повноцінну сферу (рис. 2.4).

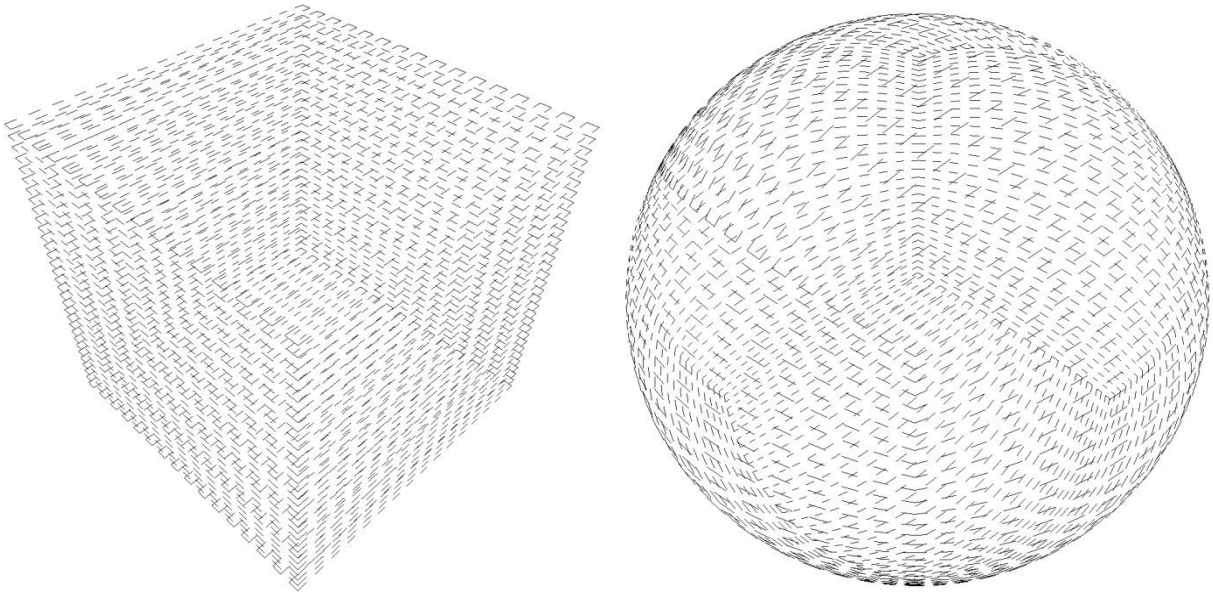

Рисунок 2.4 – Використання 6 площин куба в якості сфери

Кожна площина, яка управляється за допомогою відокремленого "дерева квадрантів" [16], розбиває всю місцевість на певні квадрати відповідно до рівня деталізації в залежності від позиції спостерігача, завдяки чому досягається ефективна зміна рівнів деталізації, висока швидкість роботи програми та мінімізація споживаних ресурсів.

2.3.2 Підсистема менеджера даних

Менеджер даних являє собою механізм який використовується всіма підсистемами, призначений для управління даними необхідними протягом всієї роботи

ігровий IDE, головним завданням якої полягає в розвантаженні займаної оперативної пам'яті незалежно від заповнення.

Необхідністю додатково механізму управління пам'яттю, відмінним від операційної системи, пояснюється в більш ефективному використанні оперативної пам'яті і файлу підкачки.

Так як у випадку використання пам'яті вище встановлений операційною системою буде виконаний механізм, за яким не використовувана оперативна пам'ять буде скинута в файл підкачки до моменту її повторної потреби. Даний метод є відмінним для вирішення з переповненням пам'яті, однак так як виділена пам'ять всередині програми має невизначене розташування у фізичній пам'яті, то при використанні подібної гнучкості доводиться йти на компроміс - у вигляді підвищених системних викликів в момент звернення до блока даний, який знаходяться у файлі підкачки.

Таким чином, для підвищення продуктивності і мінімізації системних затримок введена підсистема менеджера даних, яка розділяє та управляє пам'яттю відповідно до її пріоритету.

Всього існує 3 рівня даних:

1) дані малого розміру (до 64 байт), які знаходяться завжди в оперативній пам'яті, типом даних які є:

1.1) параметри проекту;

1.2) системні параметри;

1.3) системні дані IDE;

1.4) прапори та режими роботи IDE.

2) мало використовувані дані невизначеного розміру розташовані у файлі кешування, побудованого на технології MMF [23], типом даних які є:

2.1) буфери кінцевої геометрії графічних об'єктів;

2.2) інші буферизовані дані використовуваними різними підсистемами.

3) рідко використовувані дані, які знаходяться у файловому сховищі, типом даних які є:

3.1) текстурні та інші ресурси;

3.2) геометричні дані об'єктів.

На відміну від першого рівня даних, для якого використання спеціальних механізмів не має потреби в увазі простоти і ефективності роботи при використанні тільки API ОС, другий та третій рівень даних при використанні технології MMF [17] представляють із себе складний механізм, так як необхідно мати механізми

синхронізації при зміні даних різними підсистемами, а також підтримувати актуальність виділеного простору плоть до закінчення останнього сполучного об'єкта, а при відсутності таких зв'язків видалити займаний простір.

Технологія MMF (Memory Mapped Files), являє собою спосіб роботи з файлами на зовнішніх накопичувачах, при якому весь файл або до певної його частини представляється відповідна ділянка віртуальної пам'яті. Причому читання та запис за цією адресою призводить до читання або запису даних, які знаходяться безпосередньо на диску. Також за допомогою потужних внутрішніх механізмів файл відображення можна зробити загальнодоступним для спільного звернення процесами до однієї ж ділянки пам’яті.

Крім спрощеної роботи з файлом, основною перевагою є менше навантаження на операційну систему, так як необхідний доступ до відображеного файлу в пам'яті здійснюється через диспетчер пам'яті операційної системи, який автоматично налаштовує процесор так, що сторінки RAM зберігають сусідні фрагменти файлу (як правило, 4 КБ), утворюють безперервний діапазон адрес, надаючи їх програмі по мірі необхідності.

Завдяки чому при наявності невеликої кількості оперативної пам'яті (наприклад 128 мегабайт), можна легко переглянути файл розміром 1 гігабайт або більше, не приходячи до великих накладних витрат для системи. Також виграш в продуктивності відбувається при запису з пам'яті на диск, та при необхідності оновлення великої кількості даних, що знаходяться в пам'яті, вони можуть бути одночасно (за один прохід головки HDD) записані на зовнішній накопичувач.

Для вирішення поставлених завдань, в якості базису для механізму середньо пріоритетних даних, буде інтегрована і використана публікація “Розробка та аналіз програмного рішення файлового кешування даних в мові C++ на основі технології memory mapped files” [3].

Дане програмне рішення є відмінним як в якості базису, так і для всього механізму в цілому, так як включає в себе безпосередньо поставлені завдання, а саме:

- управління дескриптором файлу;
- управління дескриптором файлу відображення;
- управління початковою адресою зіставлення даних файлу в пам'яті;
- динамічне управління простором пам'яті для занесення даних в різні ділянки пам'яті;
- контроль зайнятості ділянок пам'яті іншими даними;
- реєстрації звільнених ділянок пам'яті;

– зберігання довжини даних для подальшого читання та запису в ділянки пам'яті;

– контролювання доступу до вмісту при використанні в багатопотоковому режимі;

– синхронізація займаних/звільнених ділянок пам'яті в багатопотоковому режимі;

– коректна поведінка протягом всієї роботи програми.

Крім цього дана робота забезпечує наявність високої продуктивності і фіксований малий розмір власної структури, що становить 8 байт при використанні 32/64 розрядної архітектури процесора.

Структура заголовка даних (рис 2.5) являє собою наявність 4 біт для лічильника блокувань призначений для ведення обліку зв'язків, 28 біт для розміру збережених даних та 32 біта в якості точки початку даних в просторі кешованої пам'яті.

SHeaderData																															
Байт	Бит																														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
4	Лічильник блокування				Розмір даних																										
8	Позиція початку даних																														

Рисунок 2.5 – Структура заголовка даних кешованої пам'яті

Так як дане програмне рішення використовує безперервну кешовану пам'ять, в якості обліку вільного простору використовується певна структура (рис 2.6), яка складається з 32 біт для обліку кількості вільного простору та 32 біт для вказівки точки початку даних у просторі кешованої пам'яті.

SFreeSpace																															
Байт	Бит																														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
4	Кількість вільного простору																														
8	Позиція початку вільного простору																														

Рисунок 2.6 – Структура вільного простору кешованої пам'яті

Крім усього дане програмне рішення ефективно використовує виділену кешовану пам'ять завдяки вбудованому внутрішньому механізму. При запиті на виділення необхідного розміру пам'яті в першу чергу буде виділено ділянку, чий вільний простір дорівнюватиме запитуваному, в іншому випадки при наявності

безперервного ділянки вільного простору він буде обрізаний на розмір запитуваного. У випадку повернення займаної пам'яті, простір, що звільняється, в першу чергу буде віддано найближчій на кордоні вільній ділянці, в іншому випадку буде створено новий запис вільної ділянки пам'яті.

Більш докладні результати ефективності виконання представлені безпосередньо в публікації.

2.4 Використання кватерніонів в якості основи для матриці повороту

Кватерніони надають зручне математичне позначення положення і обертання об'єктів в просторі. Завдяки поданню обертання в чотирьох мірному вимірі кватерніони дозволяють реалізувати гладке і безперервне обертання, а також уникнути проблеми пов'язаної з неможливістю повороту навколо осі, як у випадку з кутами Ейлера [18], незалежно від зробленого обертання по інших осях.

Підсумковий кватерніон повороту є результатом множення декількох кватерніонів обертання. Важливою особливістю є порядок множення кватерніонів, так як множення кватерніонів не є комутативним.

В якості кінцевої матриці використовуються 3 кватерніона обертання по осях XYZ визначені як крен, рискання і тангаж, властивості яких представлені на рисунку 2.7.

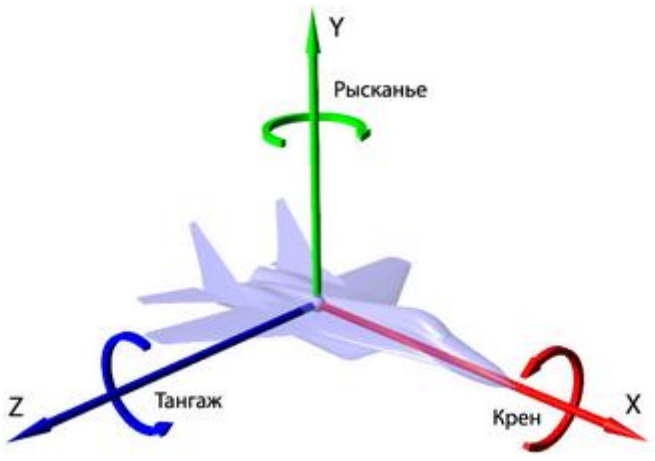

Рисунок 2.7 – Визначення кутів повороту

Для отримання матриці повороту, спочатку необхідно скласти 3 кватерніона згідно осям XYZ, по формулі 2.3.

$$Q_{xyzw}\,X = \left[1 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad \cos\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right)\right]$$

$$Q_{xyzw}\,Y = \left[0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 1 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad \cos\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right)\right] \qquad (2.3)$$

$$Q_{xyzw}\,Z = \left[0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 0 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad 1 * \sin\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right) \quad \cos\left(\frac{\left(A*\frac{PI}{180}\right)}{2}\right)\right]$$

Де Q_{xyzw} XYZ - кватерніон обертання по осі XYZ;

A - кут повороту в градусах відповідної осі.

Після чого отримані кватерніони необхідно помножити в суворій послідовності X * Y * Z, згідно з формулою 2.4.

$$Q1 * Q2 = \begin{bmatrix} Q1.w * Q2.w - Q1.x * Q2.x - Q1.y * Q2.z + Q1.z * Q2.z \\ Q1.w * Q2.x + Q1.x * Q2.w + Q1.y * Q2.z + Q1.z * Q2.y \\ Q1.w * Q2.y - Q1.x * Q2.z + Q1.y * Q2.w + Q1.z * Q2.x \\ Q1.w * Q2.z + Q1.x * Q2.y - Q1.y * Q2.x + Q1.z * Q2.w \end{bmatrix} \qquad (2.4)$$

Для отримання кінцевої матриці повороту необхідно перевести кватерніон в матрицю обертання, для цього слід виконати формулу приведення 2.5.

$$M = \begin{bmatrix} 1 - 2 * y^2 - 2 * z^2 & 2 * x * y - 2 * z * w & 2 * x * z + 2 * y * w \\ 2 * x * y + 2 * z * w & 1 - 2 * x^2 - 2 * z^2 & 2 * y * z - 2 * x * w \\ 2 * x * z - 2 * y * w & 2 * y * z + 2 * x * w & 1 - 2 * x^2 - 2 * y^2 \end{bmatrix} \qquad (2.5)$$

Матриця повороту в програмі використовується для повороту точок навколо заданої системи координат. У той час, як окремі точки прив'язуються до нових координат, відстані між ними не змінюються.

Використання кватерніонів значно розширює область використання, а також забезпечує очікуване виконання. Крім усього кватерніони дозволяють оптимізувати роботу програми, так як використовується менше математичних операції.

2.5 Використання трикутників в якості основи для побудови регулярної полігональної сітки

Полігональна сітка являє собою набір топологічно - пов’язаних простих геометричних двовимірних примітивів, які описують поверхню об'єкта. Цими

примітивами є полігони, які представляють собою фігури з прямими сторонами (3 і більше сторін), визначеними за допомогою точок тривимірного простору (вершин) і з'єднують ці точки лініями (ребра) (рис. 2.8). Внутрішня область полігону називається гранню.

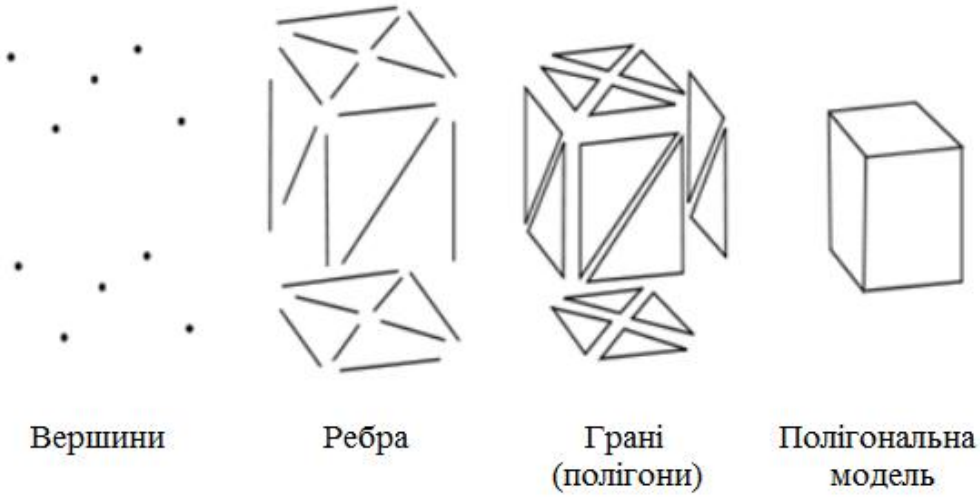

Рисунок 2.8 – Складові полігональної моделі куба

Визначення складові полігональної моделі**:**

– вершина - це позиція разом з іншою інформацією, такою як колір, нормальний вектор і координати текстури;

– ребро - це з'єднання між двома вершинами;

– грань - це замкнута множина ребер, в якій трикутна грань має три ребра;

– полігон - це набір, який лежить в одній площині граней.

Регулярна сітка - це модель, яка описує координати окремих точок поверхні способом представленим на рисунку 2.9.

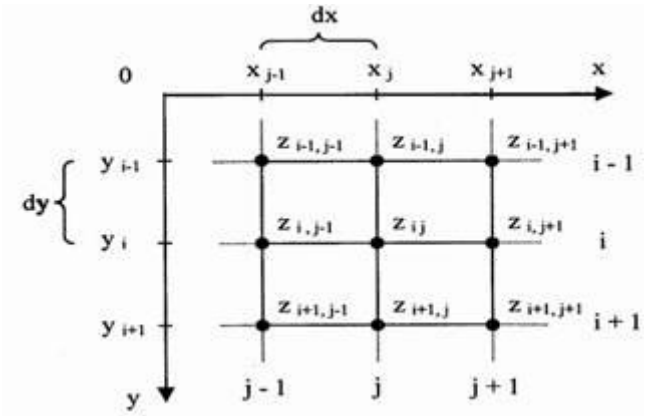

Рисунок 2.9. Загальний вигляд рівномірної сітки висот

Кожній точці сітки з індексами i, j приписується значення висоти z_{ij}. Індексам i, j відповідають певні значення координат x, y. У свою чергу відстань між вузлами є однакова - dx по осі x дорівнює dy по осі y. В спрощеному вигляді дана модель являє собою двомірний масив, кожен елемент якого зберігає значення власної висоти.

Основна область призначення рівномірної сітки призначена для опису рельєфу земної поверхні.

Позитивні риси рівномірної сітки:

– простота опису поверхонь;

– можливість швидко дізнатися висоту будь-якої точки за допомоги інтерполяції.

Недоліки рівномірної сітки:

– поверхні, які відповідають неоднозначній функції висоти у вузлах сітки, не можуть бути змодельовані;

– для опису складних поверхонь необхідна велика кількість вузлів, яка може бути обмежена об'ємом пам'яті комп'ютера.

Після чого для рівномірної сітки відразу проводиться тріангуляція, оскільки реалізація необхідних в процесі роботи алгоритмів машинної графіки для цього примітиву найбільш проста. Розбиття на трикутники проводиться шляхом проведення діагоналі з точки (i, j) в точку (i - 1, j + 1).

Висновки до розділу 2

1) використання розробленої архітектури “об'єктно-орієнтованого моноліту”, є відмінним рішенням для програми з високими вимогами до високої швидкості виконання і мінімізації споживаної пам'яті. Надає високу свободу дій для програміста, але в свою чергу накладає високі вимоги до знання всієї розроблюваної системи і прийнятих концепцій;

2) використання шаблонної спеціалізації проекту має суттєві переваги, оскільки розроблювані підсистеми і програми мають високу продуктивність та спрощену розробку. Також тому що використовуються спеціалізовані механізми для вирішення завдань, специфічних для використовуваного типу ігрового проекту;

3) використання публікацій в якості базису деяких підсистем розроблюваної програми, крім прискорення розробки, також підвищує і якість, оскільки для використовуваних методів і способів неодноразово проводилися аналітичні тести;

4) використання кватерніонів дозволяє не тільки уникнути багатьох складнощів і проблем, пов'язаних зі стандартними методами, але і дозволяє підвищити продуктивність виконання, а також дозволяє розробляти більш складні і досконалі рішення.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО СЕРЕДОВИЩА РОЗРОБКИ

Ігрове середовище розробки створене згідно з описаною раніше архітектурою п.2.2, даною особливістю якої є взаємопов'язані і взаємозалежні підсистеми, об'єкти та інші функціональні частини. У зв'язку з чим дана розробка являє собою роздільні виконавчі модулі (програми) кожен з яких виконує строго своє призначення, що входять в єдине визначення програмного забезпечення MGE.

До базового складу програмного забезпечення MGE входить:

- бібліотека системних ресурсів “MGE_LibSR.dll”;
- бібліотека мовної локалізації загального призначення “MGE_LibLL.dll”;
- програмне забезпечення “MGELauncher.exe”.

Так як в базовий склад не входить програмне забезпечення редактора, в якості демонстрації розроблених підсистем і прийнятих раніше концепцій, а також виконання поставлених раніше завдань проекту, опис буде вестися для розроблюваного ігрового шаблону з кодовим ім'ям "MGEW732DX9CSDLMH" опис спеціалізації якого надано в підпунктах далі.

3.1 Аналіз та опис використовуваних програмних і інструментальних засобів

Для досягнення поставленої мети програмне забезпечення буде розроблятися в IDE Visual Studio 2012 який є лідером в області середовищ розробок, виконуватиметься на мові програмування високого рівня C++, з використанням наступних функціональних бібліотек:

- Windows API;
- DirectX 9.0L(Ex);
- Fast Light Tool kit;
- GekConfigFile;
- GekBaseData.

Microsoft Visual Studio 2012 [19, 20] - це лінійка продуктів компанії Microsoft, що включає в себе інтегроване середовище розробки програмного забезпечення, і ряд

інструментальних засобів, які дозволяють розробити консольні додатки, додатки з графічним інтерфейсом за технологію Windows Forms, а також сайти, додатки, служби, як в рідному так і в переносному коді для платформ підтримуючих Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone і Silverlight. Включає в себе редактор вихідного коду з технологію IntelliSense, а також можливість рефакторінгу коду. Вбудований відладчик може працювати на рівні вихідного і машинного коду.

Переваги вибору Visual Studio:

– найпотужніше передове середовище розробки серед подібних аналогів;
– висока швидкість і стабільність роботи;
– невеличкі системні вимоги;
– потужний налагоджувальний механізм;
– багато функціональних можливостей.

Переваги вибору Visual Studio для проекту:

– дозволяє виконати повний цикл розробки ПО без необхідності використання зовнішніх програмних та інструментальних засобів;
– простота, зручність і багатофункціональність інтерфейсу користувача;
– невеликий базовий розмір кінцевого виконавчого файлу програми;
– власний досвід і навички роботи в Visual Studio 2012;
– дуже добре зарекомендувала себе в роботі протягом тривалого часу;
– ефективність і швидкість виконання кінцевої розробленої програми.

Мова програмування C++ - це компільоване, високорівнева статично типізована мова програмування загального призначення, для використовування при системному програмуванні, розробки програмного забезпечення, написання драйверів, серверно-клієнтських програм, розробки розважальних програм, наприклад відеоігор.

Розроблена Б'ярном Страуструпом в 1979 року з початковою назвою “Сі з класами”. Згодом Страуструп перейменував мову на C++ у 1983 р. Базується на мові С, так як при її створенні прагнули зберегти сумісність з мовою С.

Вибір С в якості бази для створення нової мови програмування полягає в тому, що мова С:

– є багатоцільовим, лаконічним і відносно низькорівневою мовою;
– підходить для вирішення більшості системних завдань;
– виповнюється скрізь і на всьому;

– стикується з середовищем програмування UNIX.

По мірі розробки C++ в нього були включені інші засоби, які в свою чергу перекривали можливості конструкцій C, у зв'язку з чим неодноразово піднімалося питання про відмову від сумісності мов шляхом видалення застарілих конструкцій. Тим не менш, сумісність була збережена з наступних міркувань:

– збереження чинного коду, написаного на C і прямо перенесеного в C++;

– виключення необхідності перенавчання програмістів, які раніше вивчали C (потрібність тільки вивчити нові засоби C++);

– виключення плутанини між мовами, при їх спільному використанні (при спільному використанні мов, їх відмінності повинні бути або мінімальними, або настільки великими, щоб мови було неможливо переплутати).

Переваги вибору C++ для проекту:

– гнучкість і необмежена варіантність виконання поставленої задачі;

– строга типізація даних;

– підтримка різних стилів та технологій програмування, включаючи традиційне директивне програмування, об’єктно-орієнтоване програмування, узагальнене програмування, метапрограмування (шаблони, макроси);

– не прив’язаність до конкретної платформи або системи;

– велика швидкість і ефективність виконання програмного коду;

– повний контроль та свобода дій, без обов’язкових механізмів безпеки при роботі з пам'яттю, приведення типів, тощо;

– власні знання та досвід робити, які дозволяють програмувати на рівні “власної мови” для вирішення поставлених завдань.

Основним недоліком при виборі інших C подібних мов, таких як Java та C#, є відмінний стиль програмування, строгі обмеження, такі як відсутність вказівник в Java і не пряме виконання програмного коду на процесорі. А також малі спеціалізовані знання та досвід для ефективного програмування на мові відмінною від C++.

Windows API (windows application programming interfaces) [21] - це набір чітко визначених підпрограм, протоколів взаємодії і базових функцій для програмування додатків до операційних систем сімейств Microsoft Windows корпорації Майкрософт.

Windows API спроектований для використання в мові C, або C++ для написання

прикладних програм призначених для роботи під керуванням операційної системи MS Windows. Найближчий спосіб взаємодії прикладної програми з ОС Windows для швидкої і ефективної розробки програмного забезпечення включає в себе безліч функцій, структурних даних і числових констант, для прямої та повноцінної розробки додатків.

Переваги вибору для проекту:

– Windows API розроблено на мові С що відмінно поєднується з мовою проекту розробки (C++);
– найближчий спосіб взаємодії прикладної програми з ОС Windows;
– має чітку структуру і стандартизацію;
– має чітку та повну загальнодоступну технічну документацію;
– надання повного контролю над об'єктами виконання.

DirectX [22] - це ряд технологій, завдяки яким комп'ютери на основі ОС Windows стають ідеальним середовищем для запуску та відображення додатків і багатих елементами мультимедіа, таки як: кольорова графіка, відео, тривимірна анімація і стереозвук. DirectX включає оновлення, що підвищують безпеку і продуктивність, а також нові функції, що відносяться до різних технологій, до яких додаток може звертатися за допомогою DirectX API. DirectX API - це набір низькорівневих інтерфейсів прикладного програмування (API) для розробки ігрових і мультимедійних додатків під платформу Microsoft Windows.

В проекті буде використовуватися компонент DirectX Graphics версії 9.0L. Основна частина якого полягає в графічному конвеєрі [23], призначеного для виведення тривимірної (Direct3D) і двомірної (Direct2D) графіки, дія яких розгортається в реальному часі.

Fast Light Tool kit [24] – багатоплатформна бібліотека інструментів для побудови графічного інтерфейсу користувача (GUI), розроблена Біллом Спіцтаком (Bill Spiztak).

FLTK створювалася для підтримки 3D графіки і тому має вбудований інтерфейс до OpenGL, який дуже добре підходить і для програмування звичайних інтерфейсів користувача.

Дана бібліотека використовує свої власні незалежні від системи віджети, графіки і події, які дозволяють писати програми однаково працюючих і виглядаючи на різних операційних системах. На відміну від інших подібних бібліотек таких як Qt, GTK,

wxWidgets, FLTK обмежується тільки графічною функціональністю. Тому вона має малий розмір та дозволяє компонуватися статично. Також бібліотека не використовує складних макросів, препроцесорів і подібних можливостей мови C++ (шаблони, виключення, простору імен).

За допомогою бібліотеки FLTK виконуються усі поставлені задачі, а саме:

– єдиний стиль відображення на різних операційних системах;

– велика швидкість виконання, а також простий графічний інтерфейс;

– написаний безпосередньо поверх Window API для максимальної швидкості і оптимізації для розміру і продуктивності коду;

– малий розмір виконавчого файлу;

– мала потрібність в системних ресурсах операційної системи;

– великий набір елементів для створення графічного інтерфейсу;

– розроблено на мові C++.

GekConfigFile - самописна приватна бібліотека яка є інтелектуальною власністю автора даної випускної кваліфікаційної роботи, призначеної для роботи із зовнішніми файлами конфігураційних налаштувань. Спроектована на C++ для використання в прикладних програмах призначених для роботи під керуванням операційної системи MS Windows.

Відмінними рисами від аналогів для використання в проекті розробки є:

– відсутність ліцензій та інших сторонніх авторських вимог по відношенню до авторського права;

– відсутність проблем безпеки пов'язане з виконанням стороннього коду крім роботи ключь-значень при завантаженні файлу конфігурації;

– підтримка кодування файлу в UTF-8, для простого зовнішнього редагування;

– підтримка типів даних аналогічних в Ci;

– підтримка текстового типу кодування ASCII і UTF-8 без обмежувальних символів і обов’язкових спеціалізованих вставок;

– підтримка багатопоточності з використанням API ОС Windows, для економії ресурсів системи і підвищення швидкості виконання;

– кешування даних для швидкої роботи з конфігураційними даними протягом всього відкритого сеансу роботи;

– підтримка механізмів відмовостійкості в разі непередбаченого збою програми виконання в момент збереження конфігураційних даних;

– оптимізовано для використання на x32 або x64 розрядної платформі додатка;

– захист програми за рахунок відсутності доступу до вихідного коду бібліотеки, що ускладнює модифікацію відмінну від авторських.

GekBaseData - самописна приватна бібліотека яка є інтелектуальною власністю автора даної випускної кваліфікаційної роботи, призначеної для роботи із базою даних для використання в прикладних програмах, написаних на C++.

Складається з 3 частин, які при потребі підлягають об'єднанню завдяки єдності інтерфейсів:

– кешованої бази даних;

– кешованої бази даних з підтримкою зберігання в зовнішньому файлі;

– кешованої бази даних з підтримкою багатопоточності.

Внутрішні механізми та структура даних які лягли в основу публікаційної роботи “Розробка та аналіз програмного рішення файлового кешування даних в мові C++ на основі технології memory mapped files ” [3].

Завдяки неодноразовому тестуванню протягом часу, а також виконаному в публікації більш детальному аналізу ефективності роботи, підтверджується придатність використання даної бібліотеки в якості основи файлів даних для розроблюваної IDE.

Основними перевагами використання в проекті є:

– відсутність ліцензій та інших сторонніх авторських вимог по відношенню до авторського права;

– відсутність проблем безпеки пов'язані з виконанням стороннього коду, окрім роботи ключь-значень при завантаженні файлу бази даних;

– приховування інформації від перегляду за допомогою стандартних засобів;

– підтримка багатопоточності з використанням API ОС Windows для економії ресурсів системи і підвищення швидкості виконання;

– кешування даних для швидкої роботи з даними протягом всього відкритого сеансу роботи;

– підтримка механізмів відмовостійкості в разі непередбаченого збою програми виконання в момент збереження даних;

– оптимізовано для використання на x32 або x64 розрядної платформі додатка;

– захист програми за рахунок відсутності доступу до вихідного коду бібліотеки, що ускладнює модифікацію відмінну від авторських.

3.2 Стандартизація розробок MGE

В залежності від використовуваного шаблону ігрового проекту використовуються різні версії програм редактора, які в свою чергу мають фундаментальні відмінності різних використовуваних технологій, підсистем, функціональних можливостей.

Для досягнення синхронізації і забезпечення єдності розроблюваних програм MGE використовується загальний файл стандартизації і угод “SyncDefineHeader.h”, опис розділів яких представлено в таблиці 3.1 та 3.2.

Таблиця 3.1 – Підрозділ найменування exe розроблюваних програм

Найменування	Значення
DefParam_Exe_MGE_Launcher	"Mad Game Editor.exe"
DefParam_Exe_MGE_W732DX9CSDLMH	"MGEW732DX9CSDLMH.exe"

Таблиця 3.2 – Підрозділ параметрів за замовчуванням загального призначення

Найменування	Значення
DefParam_FLTK_ColorBackground	0x2E2E2E
DefParam_FLTK_ColorBackgroundLabelListText	0x363636
DefParam_FLTK_ColorLabelText	0xE6E6E6
DefParam_FLTK_ColorISelection	0xE4
DefParam_FLTK_FLTK_FontSize	16
DefParam_FLTK_ScrollBarSize	20
DefParam_MaxLen_NameProject	64
DefParam_MaxLen_NameApp	64

З метою підвищення продуктивності і мінімізації споживаної оперативної пам'яті використовується тип даних unsigned int, який має розмір 32 біт для зберігання режимів і прапорів в якості параметрів. Мінімізація і підвищення продуктивності полягає в стислому зберіганні прапорів і параметрів в одній 4 байтовой змінній, розташованій в одному регістрі процесора, замість використання порядку 32 байт на кожен параметр або прапор.

Необхідні витрати на виконання бітових операцій І, АБО, ЗРУШЕННЯ, компенсуються використанням 32 бітової змінної рівної регістрам процесора. У випадку використання типу даних bool в якості прапорів при зверненні потрібно виконати додаткове приведення типу до 32 бітового регістру процесора, для виконання

арифметичних і логічних операцій.

В якості стандарту та уникання можливої плутанини в процесі розробки кожен тип параметра має власний псевдонім типу змінної. Завдяки чому, при будь-яких змінах, надалі досить змінити тип даних в 1 місці і це буде мати ефект на всі розробки. Також використання псевдоніма допомагає при налагодженні додатків, так як можна дізнатися безпосередньо приналежність змінної в випадки труднощів при розшифровці імені змінної.

Основними параметрами є:

– параметри типу шаблону проекту;
– параметри проекту;
– параметри системи (зарезервовано для розширення);
– параметри графіки;
– параметри DirectX3D 9.0 L (тільки для MGEW732DX9CSDLMH);
– параметри камери.

Структури параметрів з урахуванням зміщення та кількості займаних біт представлено в рисунку 3.1, опис полів структур представлено в таблиці 3.3.

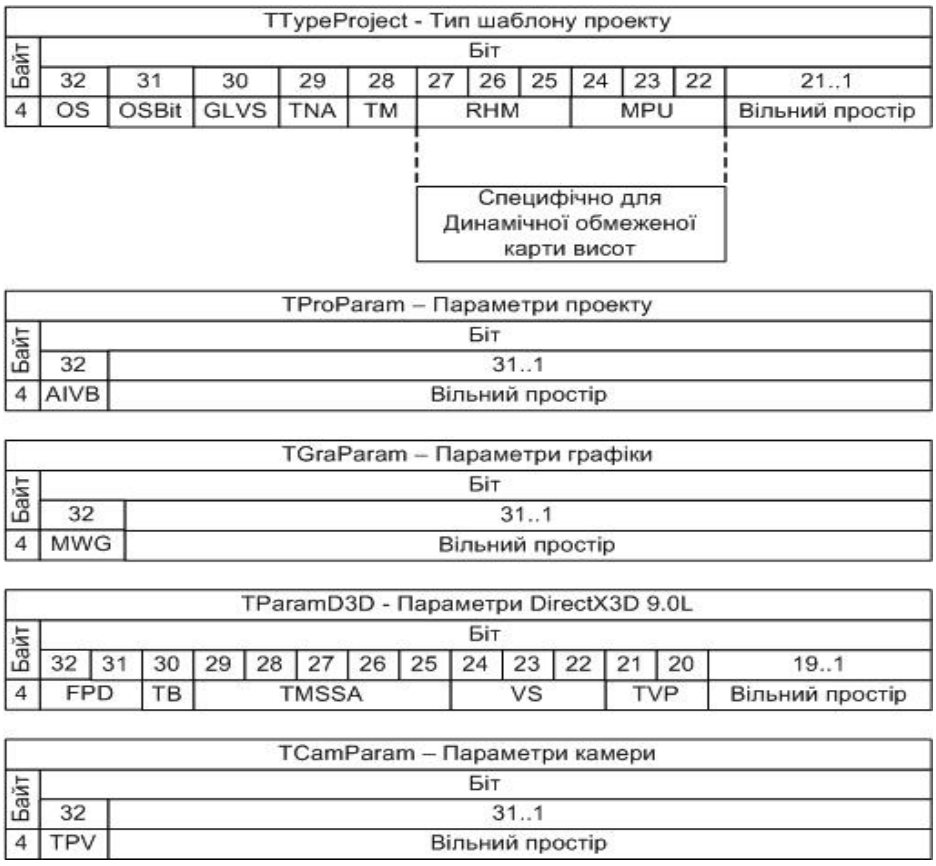

Рисунок 3.1 – Структура параметрів

Таблиця 3.3 – Опис полів структури параметрів

Параметр	Опис
Тип шаблону проекту:	
OS	Тип операційної системи
OSBit	Розрядність операційної системи
GLVS	Графічна бібліотека / Версія шейдера
TNA	Тип мережевої архітектури
TM	Тип карти
RHM	Розмір карти висот
MPU	Метрів на одиницю
Параметри проекту:	
AIVB	Автоінкрементування версії збірки
Параметри графіки:	
MWG	Режим каркасної геометрії
Параметри DirectX3D 9.0L:	
FPD	Формат пікселя для заднього буфера і дисплея
TB	Потрійна буферизація
TMSSA	Тип мультисемплінгу
VS	Режими синхронізації частоти зміни кадрів з частотою монітора
TVP	Тип обробки вершин
Параметри камери:	
TPV	Тип проекції виду

Крім загальних параметрів і визначень в файл стандарту входять також стандартизація структури файлу основного проекту MGE (.mge) і файлу користувацьких параметрів середовища розробки (.usp). Завдяки чому забезпечується єдиний інтерфейс між усіма розробками MGE.

Самі файли розширення mge і usp є файлами бази даних бібліотеки GekBaseData. Відмінною особливістю для використання в проекті є швидкість роботи, так як дана бібліотека розроблена для C++ в якості типу даних підтримуються деякі типи даних (char, int, long long) або будь-який тип даних (void*).

В якості ключа полів виступає 64 бітний хеш ключ, який генерується з рядка вбудованими функціями бібліотеки або задається безпосередньо. 64 бітний ключ, введений крім підвищення значень для унікальності, є основною причиною в ефективному використанні процесора, розвантаження шини даних при передачі між пам'яттю і процесором та мінімізації споживаної пам'яті, так як при зверненні використовується 1-2 регістра процесора на 64/32 розрядної архітектурі, інші регістри можуть бути віддані на інші операції необхідні для виконання дій над ключем.

Опис структури і належних ідентифікаторів для файлів розширення mge і usp представлено в таблиці 3.4.

Таблиця 3.4 – Опис полів структури mge і usp

Найменування	HashKey	Розмір даних	Опис
Основний файл проекту MGE (.mge):			
ProParam	17534960018052767614	4 байт	Параметри проекту
NameProject	10927193470207497325	<=64 байт	Ім'я проекту
NameApp	10928428214687834412	<=64 байт	Ім'я програми
VersionProject	7726473537859609569	8 байт	Версія проекту
TypeProject	192954315090787009	4 байт	Тип шаблону проекту
Файл користувацьких параметрів середовища розробки MGE (.usp):			
SysParam	13554032394392926943	4 байт	Параметри системи
GraParam	17465211552403588414	4 байт	Параметри графіки
CamParam	11742710939896140382	4 байт	Параметри камери
SecInterface:: GroupParam	7958406239347290601	20 байт	Група параметрів інтерфейсу (позиція і розмір вікна та інших віджетів)
SecGraphics:: ParamD3D	12352050392226477323	4 байт	Група параметрів DirectX3D 9.0 L
SecGraphics:: QualityLevelMSAA	1564280967232201254	4 байт	Рівень якості мультисемплінгу
SecCamera:: GroupParam	15735183646551254240	32 байт	Група параметрів камери (позиція, кут напрямку, швидкість переміщення і кут видимості камери)

В якості наглядного представлення реалізації даних структур в проекті, в додатку А представлений програмний вид структури “Тип шаблону проекту”.

3.3 Базові зовнішні бібліотеки даних MGE

Динамічна бібліотека (DLL) системних ресурсів являє собою єдиний набір ресурсів використовуваних безпосередньо в додатках. Відмінною особливістю зберігання ресурсів у зовнішній бібліотеці є механізм DLL, що дозволяє ефективно використовувати пам'ять і дисковий простір, використовуючи для цього тільки один екземпляр бібліотечного модуля для декількох додатків. Крім загального скорочення споживання оперативної пам'яті при використанні декількох додатків MGE, також скорочується розмір виконуваних файлів.

Динамічна бібліотека (DLL) мовної локалізації загального призначення являє собою єдиний набір визначень всієї текстової інформації всіх додатків MGE встановленої мови (додаток Б). Допоміжним підтримуваним кодуванням тексту є UTF-

8, що є стандартом у багатьох додатках і веб-просторі для кодування символів unicode. Також в свою чергу UTF-8 має повну сумісність з 7-бітної кодуванням ASCII, що є основним кодуванням у внутрішніх механізмах додатків MGE.

3.4 Програмне забезпечення MGELauncher

Програмне забезпечення MGELauncher призначене для створення та/або відкриття відповідного редактора MGE згідно типу шаблону ігрового проекту.

З метою ефективного використання системних і мінімізації споживаних ресурсів комп'ютера, даний модуль винесений в окрему виконавчу програму, так як є загальним для всіх редакторів, завдяки чому знижується системні вимоги до основних редакторів ігрової розробки.

Даний модуль являє собою просту структуру (рис 3.2), складається з створення проекту і/або відкриття редактора відповідного типу.

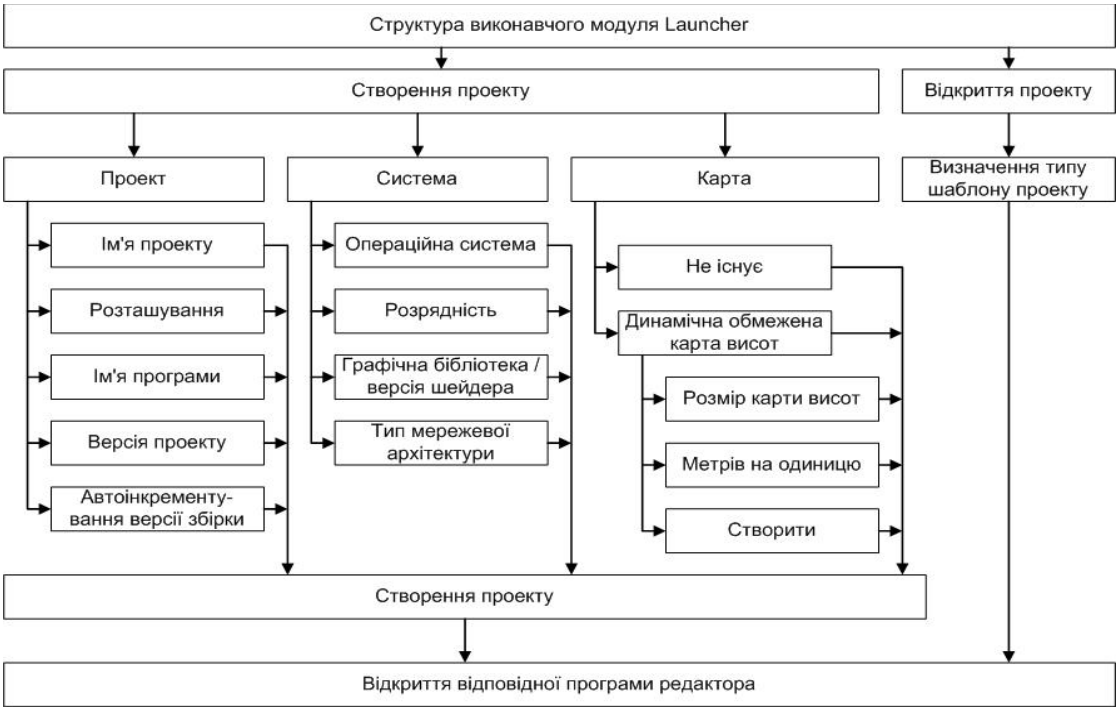

Рисунок 3.2 – Структура виконавчого модуля MGELauncher

Створення проекту являє собою вказівку параметрів нового проекту, на основі яких надалі буде визначено тип шаблону проекту, що є незмінним протягом всієї розробки. Після чого виконується перевірка вводу параметрів, по успішності якої виконується генерація базового рішення проекту, до складу якої входить базові директорії проекту, файл проекту, після чого виконується запуск проекту згідно

приналежності типу редактора (додаток В).

Також підтримується альтернативні варіанти запуску програми в режимі відкриття проекту, які полягають у запуску через файл проекту MGE (.mge) або через консоль з вказівкою шляху до файлу проекту в якості параметра запуску.

Інтерфейс розробленої програми

Інтерфейс програми виконаний з упором на швидкість виконання, простоту управління і мінімізації споживаних ресурсів комп'ютера. У зв'язку з чим інтерфейс представляє з себе класичний стиль з використанням переважно темної теми, призначеної для роботи в нічний і денний час доби, а також з метою зниження напруги на очі досягається при використанні високоякісної матриці дисплея.

Графічні елементи управління виконані з використанням бібліотеки FLTK, що має в своєму складі всі необхідні віджети для складання інтерфейсу будь-якої складності. Дана бібліотека відрізняється високою швидкістю роботи і мінімальними системними вимогами, завдяки чому інтерфейс має миттєвий відгук на дії користувача.

При запуску програми користувач відразу потрапляє до головному меню (рис 3.3), в якому надається вибір подальших дій, а саме створення проекту або відкриття існуючого проекту.

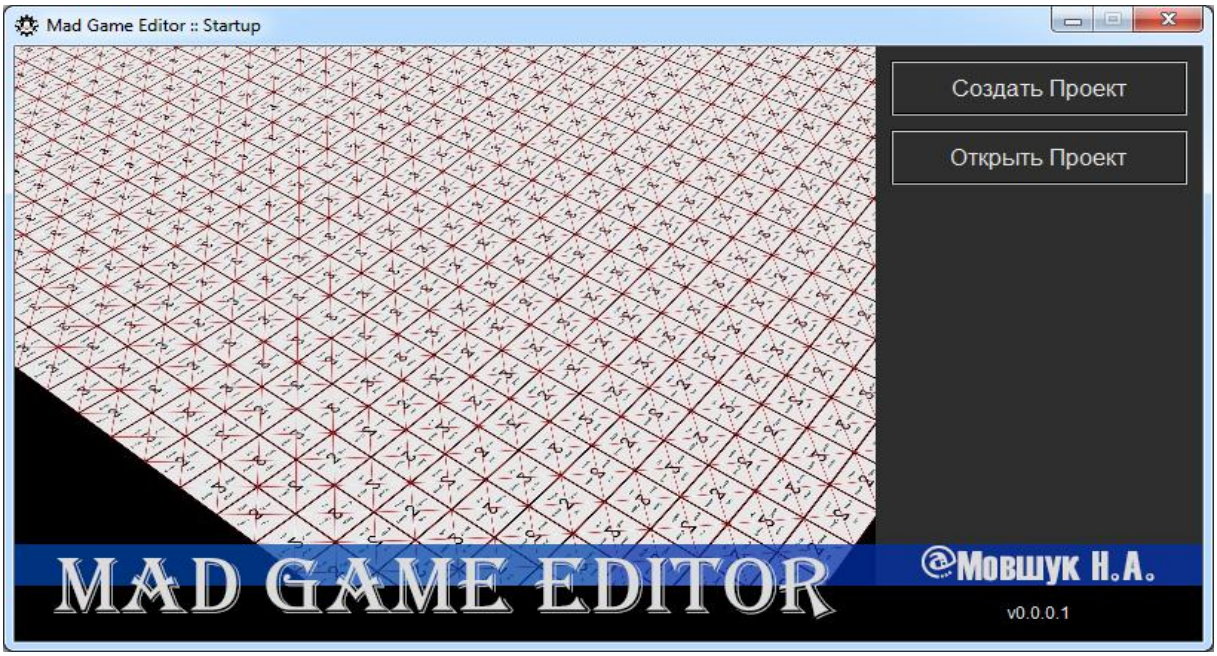

Рисунок 3.3 – Головне меню програми

У випадку вибору “створення проекту” виконується перехід в розділ створення проекту, в якому налаштовуються параметри майбутнього ігрового проекту і визначається тип шаблону, по успішному закінченню якого буде відкрита відповідна

шаблоном програма редактора.

Даний розділ має 3 вкладки:

– вкладку “Проект” (рис. 3.4), призначений для налаштування основних параметрів проекту;

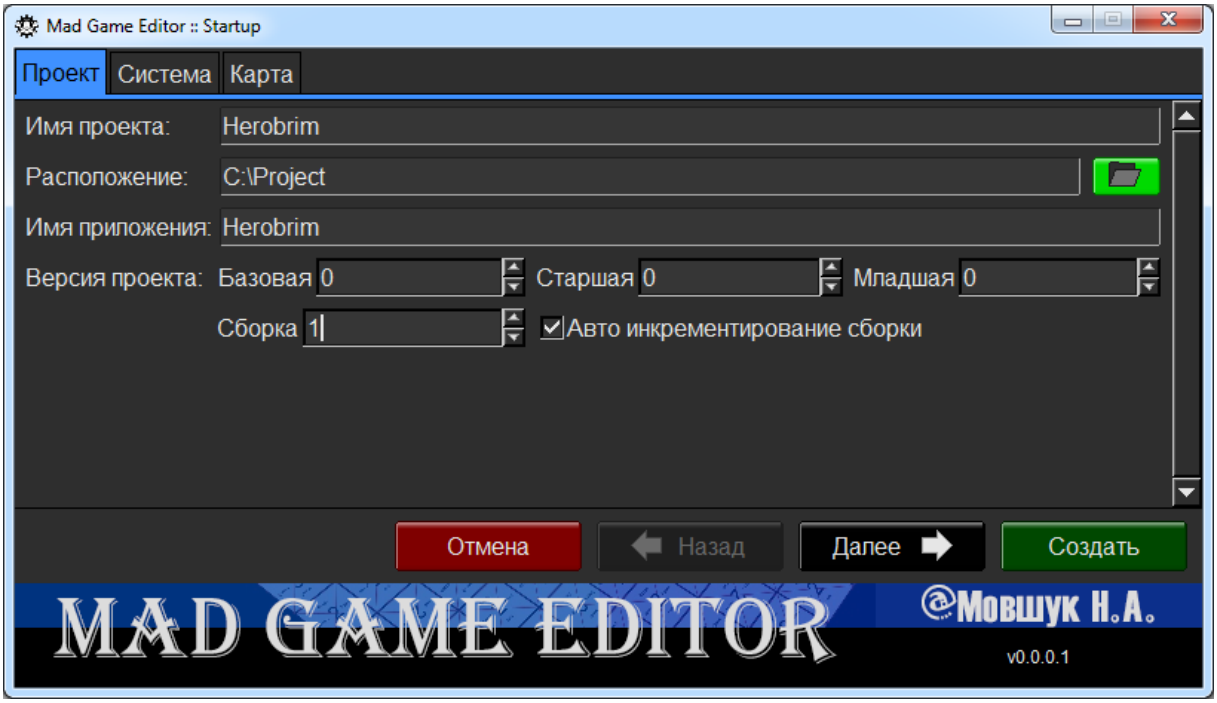

Рисунок 3.4 – Розділ створення проекту, вкладка “Проект”

– вкладку “Система” (рис. 3.5), призначену для налаштування параметрів платформи, використовуваних технологій і рішень проекту;

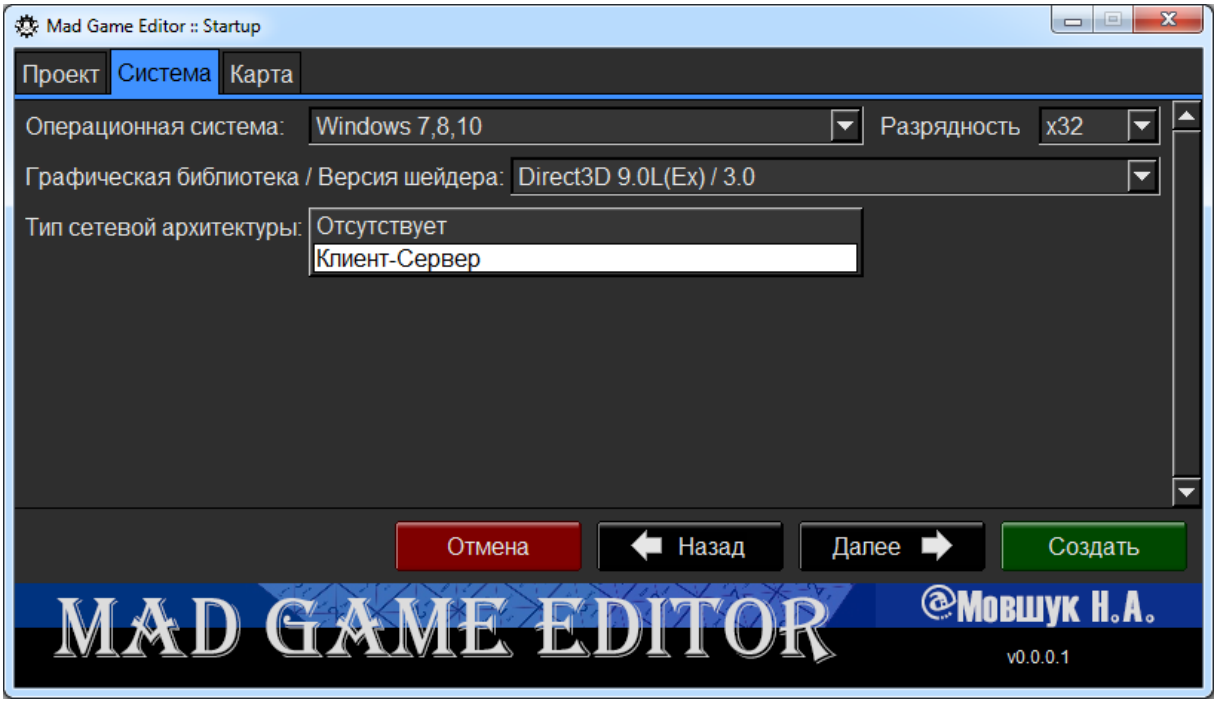

Рисунок 3.5 – Розділ створення проекту, вкладка “Система”

– вкладку “Карта” (рис. 3.6), призначену для налаштування параметрів ігрової карти, параметри якої залежать від вибору типу карти.

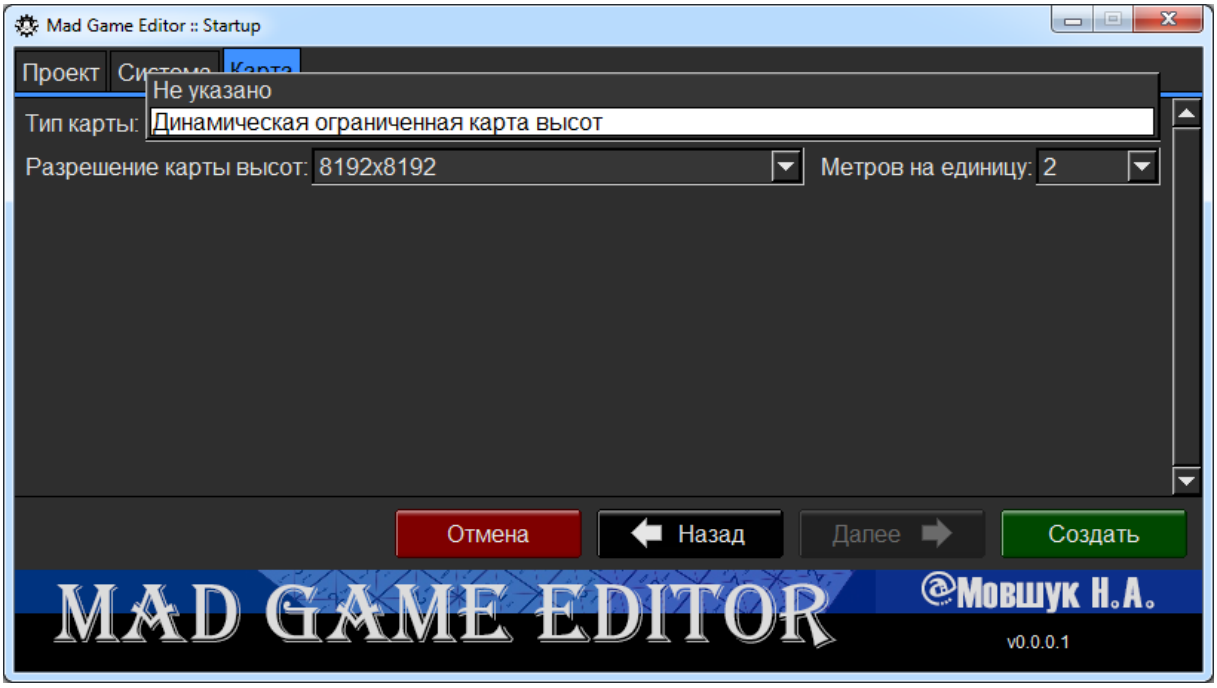

Рисунок 3.6 – Розділ створення проекту, вкладка “Карта”

У разі некоректності введення, буде видана помилка згідно типу виключення з опис некоректностей, а також буде переміщено і позначено відповідне поле некоректного введення (рис. 3.7).

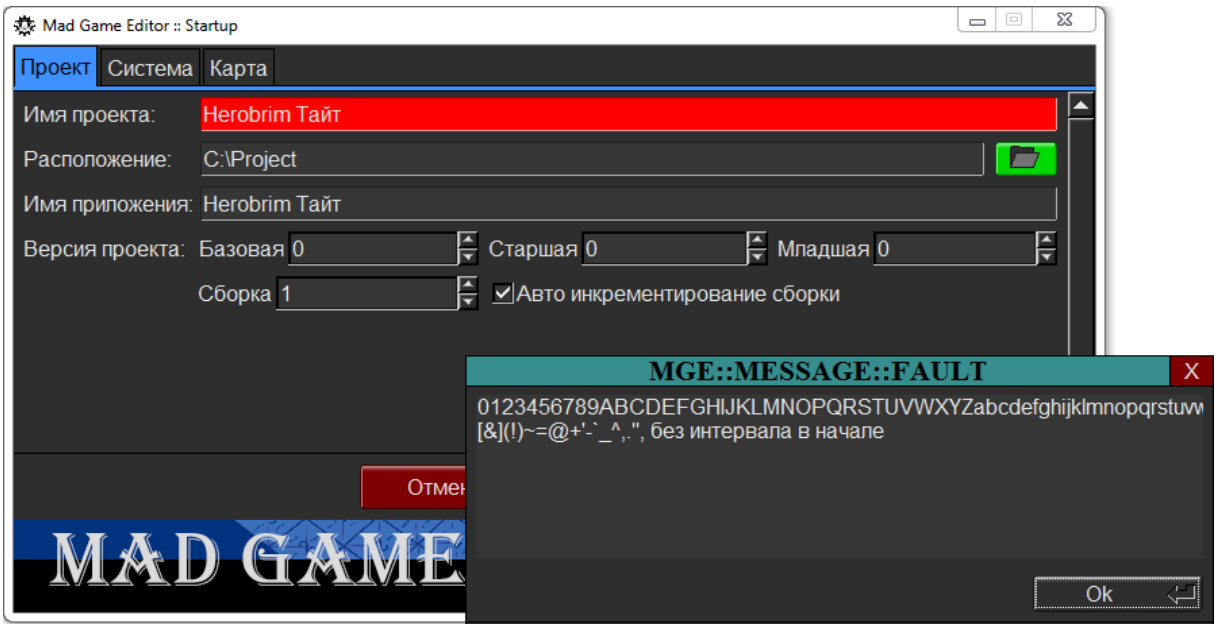

Рисунок 3.7 – Подія в разі некоректного введення

У разі вибору користувачем “открытия проекта”, з'явиться стандартне діалогове вікно вибору файлів згідно запускається OS (рис 3.8).

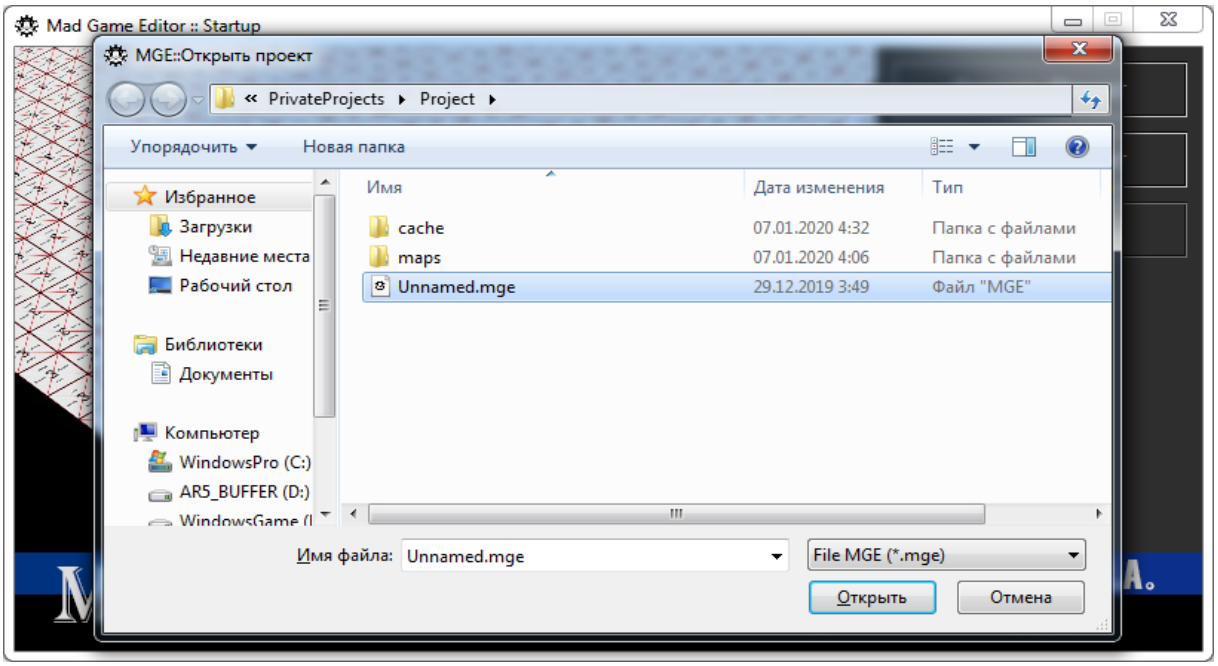

Рисунок 3.8 – Діалогове вікно відкриття проекту

3.5 Виконавчий модуль редактору типу MGEW732DX9CSDLMH

Даний виконуваний модуль є основним редактором всього ігрового проекту для типу ігрового шаблону MGEW732DX9CSDLMH, повна розшифровка якого надана на рисунку 3.9.

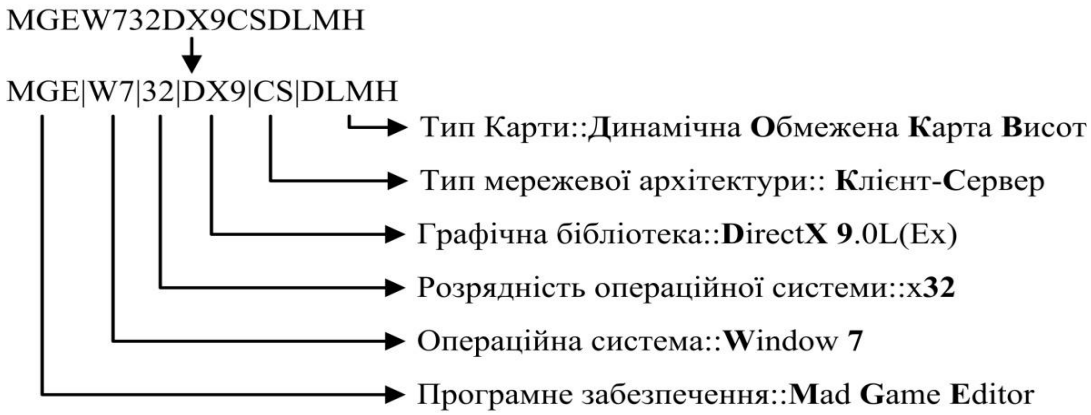

Рисунок 3.9 – Розшифровка ігрового шаблону

3.5.1 Архітектура прототипу редактора

В якості базової архітектури використовується раніше описана архітектура IDE яка заснована на «Об'єктно-орієнтованому моноліті», згідно специфіці якої всі підсистеми та інше функціональні частини орієнтовані виключно для роботи з шаблоном ігрового проекту MGEW732DX9CSDLMH.

Для відповідності ігровому шаблону, базова архітектура була розширена для включення таких підсистем як: камера, журнал, карта. Так як одним з поставлених завданням досліджень є "*створення програмного прототипу, метою якого є перевірка придатності узгоджених для застосування концепцій, архітектурних і технологічних рішень*", деякі з підсистем будуть заглушені і використовуватися не будуть. Підсумкова архітектура представлена на рисунку 3.10.

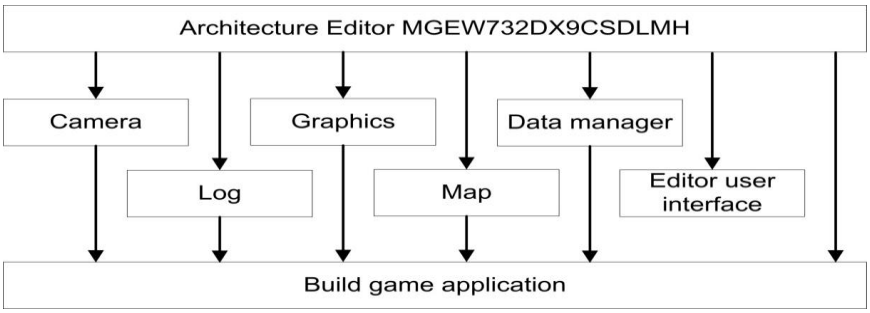

Рисунок 3.10 – Архітектура прототипу редактора MGEW732DX9CSDLMH

3.5.1.1 Підсистема менеджера даних

Підсистема менеджера даних складається з методу і структур описаними раніше в п.2.3.2. В якості реалізації дана підсистема являє собою клас "CCacheMemory" (додаток Г), UML діаграма якого представлена на рисунку 3.11.

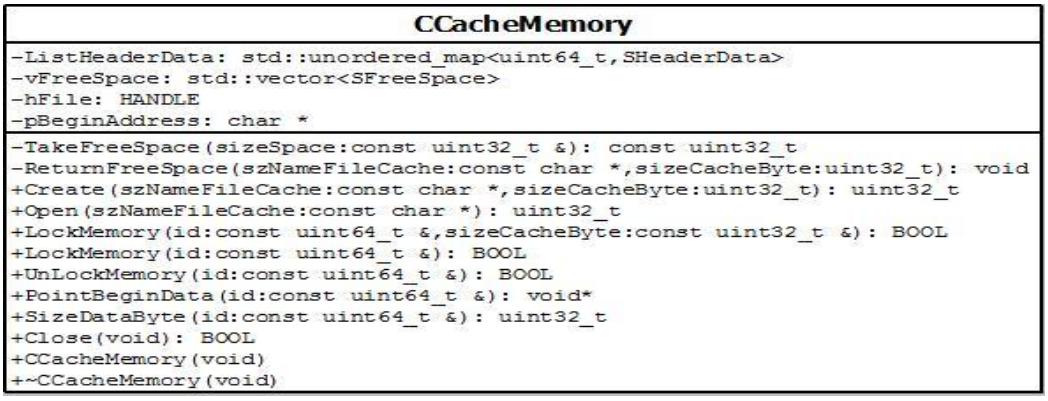

Рисунок 3.11 – UML діаграма CCacheMemory

Для роботи з даною підсистемою і класом в цілому досить створити клас за допомогою new або оголосити в якості глобальної змінної.

Так як основний напрямок всіх підсистем є швидкість виконання, був проведений підтверджуючий аналіз виконання, метою якого є порівняння необхідної кількості команд для створення класу. В якості тесту в проекті MGE було створено 2 частини однакового коду окрім створення і руйнування класу. В результаті якого підтверджується переваги створення об'єктів в глобальній області видимості.

Результати тестів, а також використаний код представлений на рисунку 3.12.

```
CCacheMemory Test1;
000000013FC99B9E  mov         edx,1
000000013FC99BA3  lea         rcx,[rsp+60h]
000000013FC99BA8  call        CCacheMemory::__autoclassinit (013FC81901h)
000000013FC99BAD  lea         rcx,[rsp+60h]
000000013FC99BB2  call        CCacheMemory::CCacheMemory (013FC81753h)
000000013FC99BB7  nop

Test1.Create("1.test1",65536);
000000013FC99BB8  mov         r8d,10000h
000000013FC99BBE  lea         rdx,[PathFile+20h (013FD3F1F8h)]
000000013FC99BC5  lea         rcx,[rsp+60h]
000000013FC99BCA  call        CCacheMemory::Create (013FC8170Dh)

void *AdresData=Test1.PointBeginData(67);
000000013FC99BCF  mov         qword ptr [rsp+348h],43h
000000013FC99BDB  lea         rdx,[rsp+348h]
000000013FC99BE3  lea         rcx,[rsp+60h]
000000013FC99BE8  call        CCacheMemory::PointBeginData (013FC8195Bh)
000000013FC99BED  mov         qword ptr [rsp+0F8h],rax

Test1.Close();
000000013FC99BF5  lea         rcx,[rsp+60h]
000000013FC99BFA  call        CCacheMemory::Close (013FC81A3Ch)
000000013FC99BFF  nop
000000013FC99C00  lea         rcx,[rsp+60h]
000000013FC99C05  call        CCacheMemory::~CCacheMemory (013FC8196Ah)

4 mov, 8 lea, 6 call, 2 nop
--------------------------
20 commands
```

```
CCacheMemory *Test2=new CCacheMemory;
000000013FC99C0A  mov         r9d,33h
000000013FC99C10  lea         r8,[PathFile+28h (013FD3F200h)]
000000013FC99C17  mov         edx,1
000000013FC99C1C  mov         ecx,88h
000000013FC99C21  call        operator new (013FD10B7Eh)
000000013FC99C26  mov         qword ptr [rsp+358h],rax
000000013FC99C2E  cmp         qword ptr [rsp+358h],0
000000013FC99C37  je          main+120h (013FC99C50h)
000000013FC99C39  mov         rcx,qword ptr [rsp+358h]
000000013FC99C41  call        CCacheMemory::CCacheMemory (013FC81753h)
000000013FC99C46  mov         qword ptr [rsp+460h],rax
000000013FC99C4E  jmp         main+12Ch (013FC99C5Ch)
000000013FC99C50  mov         qword ptr [rsp+460h],0
000000013FC99C5C  mov         rax,qword ptr [rsp+460h]
000000013FC99C64  mov         qword ptr [rsp+350h],rax
000000013FC99C6C  mov         rax,qword ptr [rsp+350h]
000000013FC99C74  mov         qword ptr [rsp+100h],rax

Test2->Create("1.test2",65536);
000000013FC99C7C  mov         r8d,10000h
000000013FC99C82  lea         rdx,[PathFile+48h (013FD3F220h)]
000000013FC99C89  mov         rcx,qword ptr [rsp+100h]
000000013FC99C91  call        CCacheMemory::Create (013FC8170Dh)

void *AdresData=Test2->PointBeginData(67);
000000013FC99C96  mov         qword ptr [rsp+360h],43h
000000013FC99CA2  lea         rdx,[rsp+360h]
000000013FC99CAA  mov         rcx,qword ptr [rsp+100h]
000000013FC99CB2  call        CCacheMemory::PointBeginData (013FC8195Bh)
000000013FC99CB7  mov         qword ptr [rsp+108h],rax

Test2->Close();
000000013FC99CBF  mov         rcx,qword ptr [rsp+100h]
000000013FC99CC7  call        CCacheMemory::Close (013FC81A3Ch)

delete Test2;
000000013FC99CCC  mov         rax,qword ptr [rsp+100h]
000000013FC99CD4  mov         qword ptr [rsp+370h],rax
000000013FC99CDC  mov         rax,qword ptr [rsp+370h]
000000013FC99CE4  mov         qword ptr [rsp+368h],rax
000000013FC99CEC  cmp         qword ptr [rsp+368h],0
000000013FC99CF5  je          main+1E3h (013FC99D13h)
000000013FC99CF7  mov         edx,1
000000013FC99CFC  mov         rcx,qword ptr [rsp+368h]
000000013FC99D04  call        CCacheMemory::`scalar deleting destructor' (013FC81A5Ah)
000000013FC99D09  mov         qword ptr [rsp+468h],rax
000000013FC99D11  jmp         main+1EFh (013FC99D1Fh)
000000013FC99D13  mov         qword ptr [rsp+468h],0

25 mov, 3 lea, 6 call, 2 cmp, 2 je, 2 jmp
------------------------------------------
40 commands
```

Рисунок 3.12 – Порівняння ефективності виконання підходів створення CCacheMemory, в режимі дизассемблера

Після створення об'єкта CCacheMemory необхідно або відкрити існуючий кешований файл за допомогою методу Open із зазначенням імені файлу, або створити новий за допомогою методу Create із зазначенням імені файлу і константним максимальний розміром кешованої пам'яті. Зворотнім значенням є розмір, який обчислюється за формулою 3.1, або 0 в разі помилки.

$$Size = \frac{\text{SizeFile } +(\text{GranMem } -1)}{\text{GranMem}} * \text{GranMem} \quad (3.1)$$

де SizeFile - необхідний розмір файлу кешування;

GranMem - гранулярність розподілу пам'яті операційної системи;

Size - результат обчислювання розміру файлу з урахуванням гранулярності пам'яті.

CCacheMemory містить вбудований механізм управління пам'яттю, для взаємодії з яким використовуються 5 методів:

1) LockMemory версія 1 - збільшує лічильник блокування пам'яті з резервуванням зазначеного розміру пам'яті;
2) LockMemory версія 2 - збільшує лічильник блокування пам'яті виділеної ділянки;
3) UnLockMemory - зменшує лічильник блокування пам'яті;
4) PointBeginData - отримання початкової адреси даних;
5) SizeDataByte - отримання розміру зарезервованої ділянки пам'яті в байтах.

Найважливішим моментом використання методу PointBeginData - ділянка даних що повертається є адресою на загальну кешовану пам'ять, завдяки чому надається прямий доступ до ділянки кешованої пам'яті без використання проміжних можливих перевірок. Що в свою чергу може породити псування всієї кешованої пам'яті в випадки виходу за межі виділеної ділянки пам'яті.

3.5.1.2 Підсистема графіки

Підсистема графіки є глобальним безкласовим об'єктом, що знаходиться в просторі імені nGraphics. В наявності якого тільки функції ініціалізації і руйнування. Основний об'єкт входить до складу і є керованим в підпросторі імені nD3D.

Підрозділ nD3D є центральною керуючою системою над графікою, в основі якої використовується DirectX 9.0L(Ex). У свою чергу надаючи можливість прямого управліннями над багатьма станами "сцени" за допомогою зовнішніх функцій. Окрім функцій, також надається прямий доступ до багатьох необхідних дескрипторів і керуючих змінних. Більш детальна UML структура представлена в рисунку 3.13.

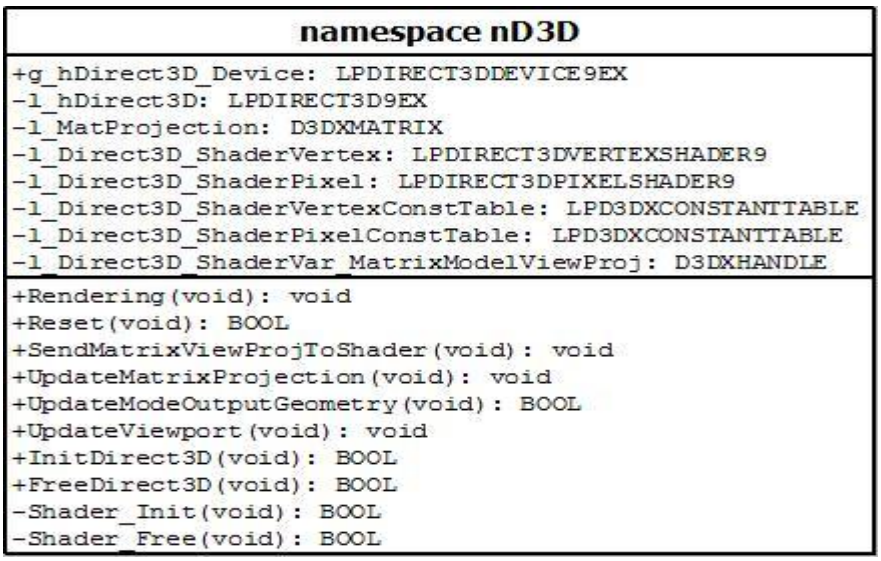

Рисунок 3.13 – UML діаграма nD3D

3.5.1.3 Підсистема карти

Підсистема карти є глобальним безкласовий об'єктом, що знаходиться в просторі імені nSecMap, з набором функцій для управління картою в ігровому редакторі. Основою підсистемою карти є метод і тип карти DLMH описаний раніше в п.2.3.1.

Реалізація даної підсистеми являє собою використання “дерева квадрантів” [22] як механізму зміни деталізації. Дерево квадрантів являє собою розбиту площину на 4 частини, кожна з якої може рекурсивно ділиться на наступні 4 частини, більш наочний приклад представлений на рисунку 3.14.

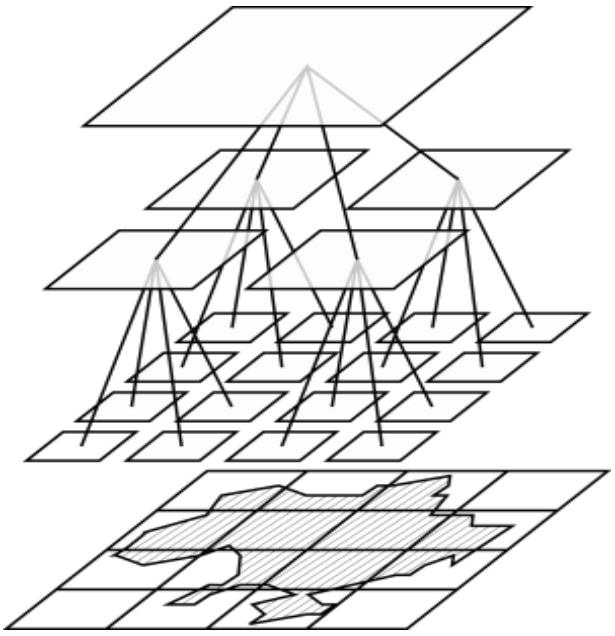

Рисунок 3.14 – Рівні деталізації з використанням дерево квадрантів

Завдяки чому вся місцевість розбивається на певні квадрати відповідно до рівня деталізації. Залежно від положення спостерігача, відбувається деталізація певних ділянок місцевості, в той же час віддалені ділянки мають найменшу можливу деталізацію.

В якості структури управління деревом квадрантів розроблений клас CQuard (рис. 3.15). Основне завдання даного класу полягає в управлінні ступені деталізації в залежності від позиції спостерігача, генерації геометричних даних карти, рендеринг видимої ділянки місцевості.

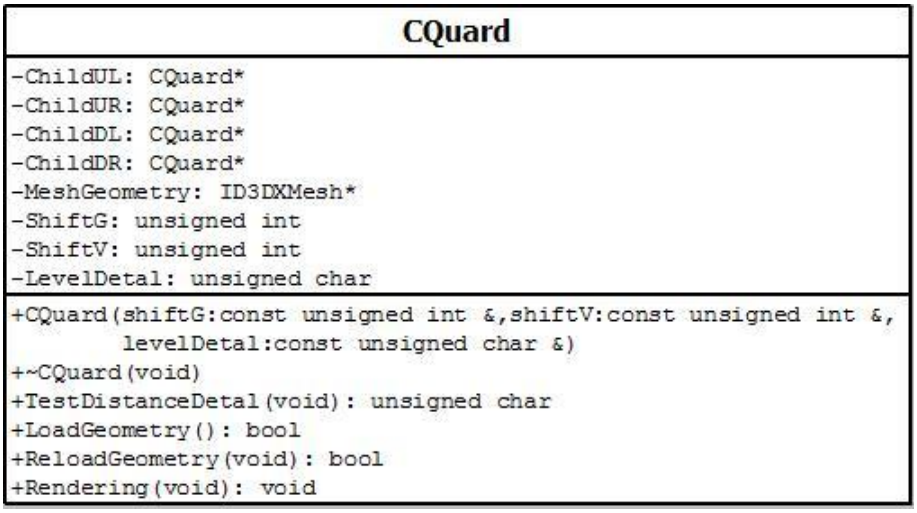

Рисунок 3.15– UML діаграма CQuard

3.5.1.4 Підсистема камери

Підсистема камери є глобальним безкласовим об'єктом, що знаходиться в просторі імені nSecCamera, з набором функцій для управління виключно камерою в ігровому редакторі. Основне призначення підсистеми полягає в створенні матриці виду (Matrix view), що складається з матриці повороту, вектора розташування і кутів нахилу.

В якості основи для отримання матриці повороту використовується метод представлений в п.2. 4. так як вихідними даними є положення миші в межах вікна по x і y осях, підсумкові кути обчислюються за формулами 3.2, 3.3.

$$AnglePitch = PreAnglePitch + (PreY - Y) * CofMouseY \qquad (3.2)$$

Де AnglePitch - отриманий кут тангажа;

PreAnglePitch - минулий кут тангажа;

PreY - минулі координати миші по осі Y.

Y - координати миші по осі Y;

CofMouseY - чутливість миші по осі Y.

$$AngleYaw = PreAngleYaw + (PreX - X) * CofMouseX \quad (3.3)$$

Де AngleYaw - отриманий кут рискання;

PreAngleYaw - минулий кут рискання;

PreX - минулі координати миші по осі X.

X - координати миші по осі X;

CofMouseX - чутливість миші по осі X.

Отримана матриця повороту множиться на вектор напрямку, після чого результат заноситься в глобальну змінну g_MatView для подальшого використання необхідними підсистемами.

3.5.1.5 Підсистема журналу

Підсистема журналу є глобальним безкласовим об'єктом, що знаходиться в просторі імені nSecLog з набором функцій для відображення діалогового вікна про повідомлення, скидання повідомлення в файл журналу і базові функції по ініціалізації і руйнуванні об'єкта журналу.

Основним механізмом є використання коду помилки в якості ідентифікатора повідомлень. Структура коду представляє собою ціле 4 байтове значення під псевдонімом TLog, до складу якого входить 5 секторів фіксованого розташування і бітової розмірності.

Більш детальна структура представлена на рисунку 3.16.

TLog – Код лога																													
Байт	Біт																												
	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
4	Тип			Джерело									Надлежит							Тип додаткового значення						Код						

Рисунок 3.16 – Структура TLog

Тип - в якості типу вказується 1 з 8 можливих значень в залежності від повідомлення. Основними типами коду є:

– не зареєстрований (Not registered);

– інформація (Information);
– похибка (Fault);
– втручання (Intervention);
– попередження (Warning);
– помилка (Error);
– фатальний (Fatal);
– критичний (Critical).

Джерело - вказує на знаходження даного повідомлення в програмному коді. Як орієнтир можуть виступати: функції, класи, простору імен або будь-яких загальних ознак.

Належність - вказує на приналежність коду журналу до бібліотеки, системи або дії користувача.

Тип додаткового значення - вказує на тип даних що повертаються, у випадку його відсутності встановлюється Void.

Код - будь-яке унікальне число, звіт якого починається з джерела.

В якості автоматизації процесу ведення журналу, а також перевіркою на контролю введення, коди журналу зберігаються в Microsoft Office Excel в окремій вкладці “коди журналу” (рис. 3.17).

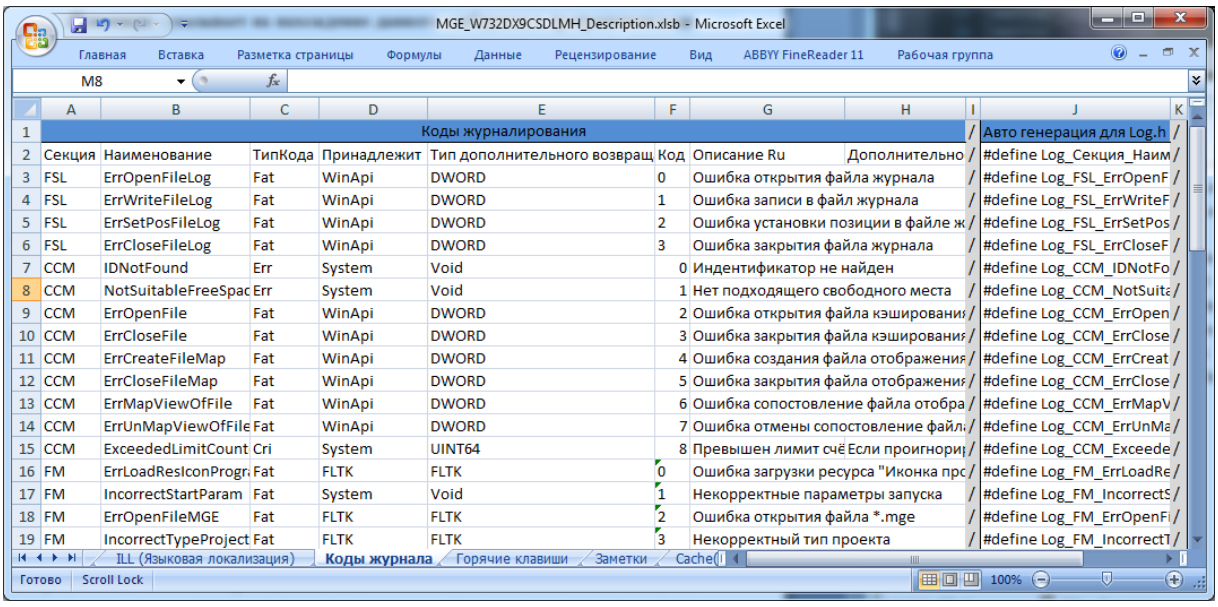

Рисунок 3.17 – Файл визначень проекту, вкладка “Коды журнала”

В якості автоматизації створена спеціальна утиліта, яка на вимогу оновлює коди журналу у файлі Log.h (рис.3.18).

```
49  //Коды
50    //Начало раздела автоматической генерации кодов журнала, изменять код внутри секции ЗАПРЕЩЕНО!, он просто будет перетёрт при автоге
51    //BEGIN_SECTION_SALL
52  #define Log_FSL_ErrOpenFileLog        _Log_Typ_Fat|_Log_Srs_FSL|_Log_Bln_WinApi|_Log_Rtr_DWORD|0 //Ошибка открытия файла журнала
53  #define Log_FSL_ErrWriteFileLog       _Log_Typ_Fat|_Log_Srs_FSL|_Log_Bln_WinApi|_Log_Rtr_DWORD|1 //Ошибка записи в файл журнала
54  #define Log_FSL_ErrSetPosFileLog      _Log_Typ_Fat|_Log_Srs_FSL|_Log_Bln_WinApi|_Log_Rtr_DWORD|2 //Ошибка установки позиции в файле
55  #define Log_FSL_ErrCloseFileLog       _Log_Typ_Fat|_Log_Srs_FSL|_Log_Bln_WinApi|_Log_Rtr_DWORD|3 //Ошибка закрытия файла журнала
56  #define Log_CCM_IDNotFound        _Log_Typ_Err|_Log_Srs_CCM|_Log_Bln_System|_Log_Rtr_Void|0 //Индентификатор не найден
57  #define Log_CCM_NotSuitableFreeSpace       _Log_Typ_Err|_Log_Srs_CCM|_Log_Bln_System|_Log_Rtr_Void|1 //Нет подходящего свободного ме
58  #define Log_CCM_ErrOpenFile        _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|2 //Ошибка открытия файла кэширования
59  #define Log_CCM_ErrCloseFile       _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|3 //Ошибка закрытия файла кэширования
60  #define Log_CCM_ErrCreateFileMap       _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|4 //Ошибка создания файла отображени
61  #define Log_CCM_ErrCloseFileMap        _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|5 //Ошибка закрытия файла отображения
62  #define Log_CCM_ErrMapViewOfFile       _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|6 //Ошибка сопоставление файла отобр
63  #define Log_CCM_ErrUnMapViewOfFile       _Log_Typ_Fat|_Log_Srs_CCM|_Log_Bln_WinApi|_Log_Rtr_DWORD|7 //Ошибка отмены сопоставление фа
64  #define Log_CCM_ExceededLimitCounterLock         _Log_Typ_Cri|_Log_Srs_CCM|_Log_Bln_System|_Log_Rtr_UINT64|8 //Превышен лимит счётчик
65  #define Log_FM_ErrLoadResIconProgram       _Log_Typ_Fat|_Log_Srs_FM|_Log_Bln_FLTK|_Log_Rtr_FLTK|0 //Ошибка загрузки ресурса "Иконка
66  #define Log_FM_IncorrectStartParam       _Log_Typ_Fat|_Log_Srs_FM|_Log_Bln_System|_Log_Rtr_Void|1 //Некорректные параметры запуска
67  #define Log_FM_ErrOpenFileMGE       _Log_Typ_Fat|_Log_Srs_FM|_Log_Bln_FLTK|_Log_Rtr_FLTK|2 //Ошибка открытия файла *.mge
68  #define Log_FM_IncorrectTypeProject        _Log_Typ_Fat|_Log_Srs_FM|_Log_Bln_FLTK|_Log_Rtr_FLTK|3 //Некорректный тип проекта
69  #define Log_FM_ErrSetCurrentDirectory       _Log_Typ_Fat|_Log_Srs_FM|_Log_Bln_WinApi|_Log_Rtr_DWORD|4 //Ошибка установки рабочего ка
70  #define Log_FM_InterventionFileMGE_NameProject          Log_Typ_Inf|_Log_Srs_FM|_Log_Bln_GDBC|_Log_Rtr_EErrWar|5 //Вмешательство в фай
```

Рисунок 3.18 – Зразок структури файлу Log.h

Залежно від типу коду журналу, подається відповідне типу коду діалогове вікно (рис 3.19) з описом помилки, попередження, тощо.

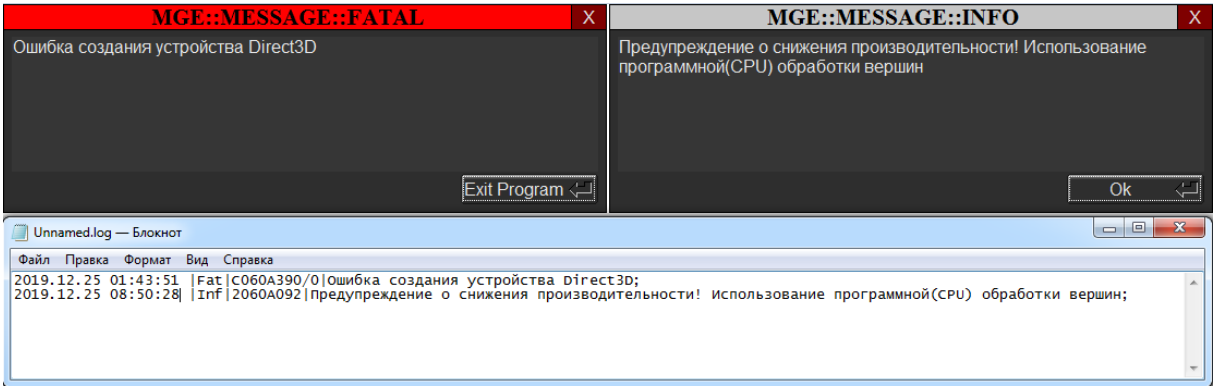

Рисунок 3.19 – Приклад діалогових вікон з файлом журналу ігрового проекту

На момент завершення розробки MGE файл кодових визначень містив: 8 джерел, 8 приналежностей, 10 типів додаткового значення, 179 кодів журналу.

Дана підсистема є результатом удосконалень протягом багатьох авторських проектів, включаючи дипломний проект бакалавра на тему “Програмне забезпечення для віддаленого контролю комп'ютерів спільного використання” захищений в 2018 році.

3.5.1.6 Підсистема збірки ігрового додатка

Дана підсистема являє собою лише прототип для повноцінного механізму складання додатків, так як багато необхідних підсистем не включені до складу розроблюваної MGE. Таким чином єдиною використовуваною підсистемою є карта,

дані якої будуть використані в кінцевому ігровому додатку.

Використовувана структура представлена на рисунку 3.20.

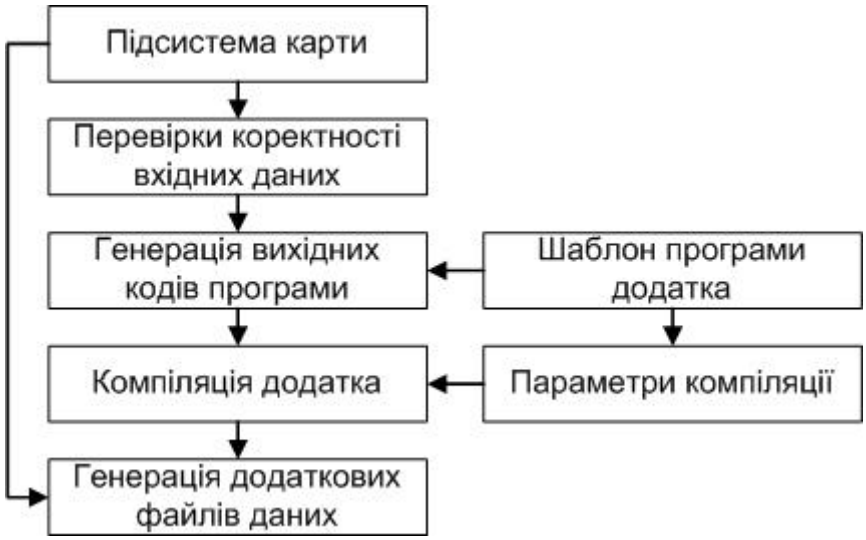

Рисунок 3.20 – Використовуваний прототип структури збірки програми

Основний процес компіляції, полягає у наданню компілятору з генерованих вихідних кодів, що є набором необхідних даних зібраних з пов'язаних підсистем.

Використовуваний компілятором прототипу програми є “оптимізуючий компілятор Microsoft (R) C/C++ версії 17.00.50727.1 для x86”.

Процес компіляції полягає у виклику сценарію CompilerProject.bat, знаходиться в директорії Compiler в папці місцезнаходження Mad Game Editor.exe. В процесі якого виконується компіляція вихідного файлу проекту SourseDemo.cpp, згенерованого на етапі “Генерація вихідних кодів програми”. При успішності виконання запускається файл і файли ресурсів будуть знаходиться в директорії “Ім'я проекту\debug”.

3.5.2 Інтерфейс розробленої програми

Інтерфейс програми виконаний з упором на швидкість виконання, простоту управління і мінімізації споживаних ресурсів комп'ютера. У зв'язку з чим інтерфейс представляє з себе класичний стиль з використанням переважно темної теми, призначеної для роботи в нічний і денний час доби, а також з метою зниження напруги на очі досягається при використанні високоякісної матриці дисплея.

Графічні елементи управління виконані з використанням бібліотеки FLTK, що має в своєму складі всі необхідні віджети для складання інтерфейсу будь-якої складності. Дана бібліотека відрізняється високою швидкістю роботи і мінімальними

системними вимогами, завдяки чому інтерфейс має миттєвий відгук на дії користувача.

Запуск програми здійснюється трьома способами:

1) за допомогою програми MGELauncher;

2) за допомогою файлу проекту MGE призначеного виключно для типу ігрового шаблону MGEW732DX9CSDLMH;

3) за допомогою командного рядка із зазначенням в якості параметра файлу проекту MGE призначеного виключно для типу ігрового шаблону MGEW732DX9CSDLMH.

Після виконання перевірки на приналежність проекту до типу редактора, з'явиться головне вікно редактора (рис. 3.21).

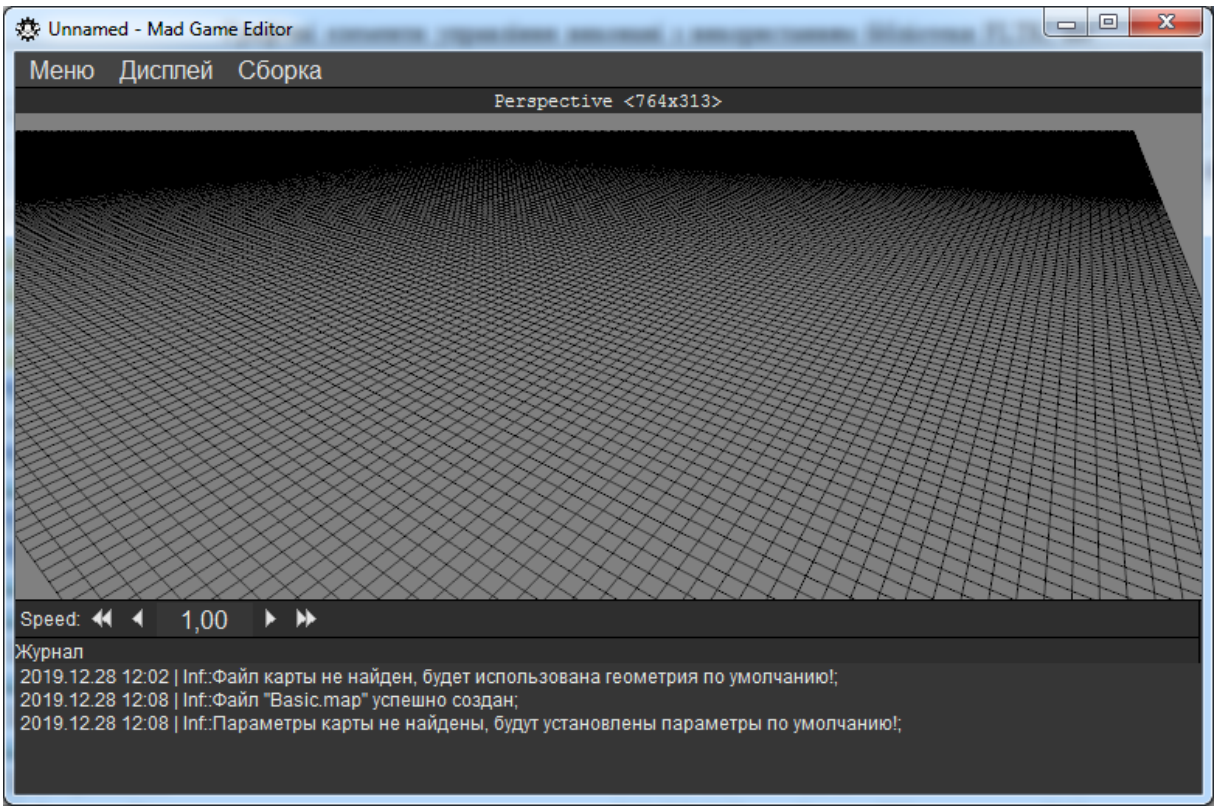

Рисунок 3.21 – Головне меню програми в момент першого запуску

Так як дана програма є всього лише прототипом, доступними функціями є: редагування геометрії за допомогою миші, збереження змін, переміщення камери в просторі, збирання демонстративного додатку.

Висновки до розділу 3

1) Використання роздільних виконуваних модулів має істотні плюси для програм, чий склад підсистем змінюється в залежності від призначення або типу ігрового проекту. Полегшує розробку і дозволяє більш ефективно реалізувати поставлені завдання;

2) введення стандартизації для розробок, що мають більше 2 виконуваних модулів, є не тільки доброю ознакою розробки, але і забезпечує пасивний механізм захисту від розбіжностей версій структур і / або прийнятих рішень між усіма розробками;

3) використання хеш ключів фіксованого розміру забезпечує захист від можливого стороннього втручання, а використовуючи механізми мають підвищену швидкість виконання на відміну від використання стандартних рядкових ключів;

4) використання DLL дозволяє більш ефективно використовувати сумісну пам'ять у випадку з запуском декількох додатків однієї розробки, а також дозволяє скоротити і усунути використання повторюваних даних;

5) використання утиліт автоматизації для занесення безлічі однотипних даних, таких як коди журналу та визначення мовної локалізації, істотно полегшує розробку, а також значно економить робочий час.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної магістерського проекту було дослідження та розробка IDE для компільованих ігрових додатків, і як результат був створений програмний додаток. Так як в процесі проектування та розробки буде використовуватися ПК, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп’ютера.

4.1 Загальні питання з охорони праці

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. В законі України «Про охорону праці» [25] визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

При роботі з обчислювальною технікою змінюються фізичні і хімічні фактори навколишнього середовища: виникає статична електрика, електромагнітне випромінювання, змінюється температура і вологість, рівень вміст кисню і озону в повітрі. Повітря забруднюється шкідливими хімічними речовинами антропогенного походження за рахунок деструкції полімерних матеріалів, які використовуються для обробки приміщень та обладнання. Неправильна організація робочого місця сприяє загальному і локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, викривлення хребта і розвитку остеохондрозу.

4.1.1 Правові та організаційні основи охорони праці

Основним організаційним напрямом у здійсненні управління в сфері охорони праці є усвідомлення пріоритету безпеки праці і підвищення соціальної відповідальності держави, і особистої відповідальності працівників.

Державна політика в галузі охорони праці визначається відповідно до Конституції України Верховною Радою України і спрямована на створення належних, безпечних і здорових умов праці, запобігання нещасним випадкам та професійним захворюванням.

Трудові відносини, що діють між працівниками і роботодавцями в Україні регулюються Кодексом законів про працю України [26], відповідно до якого права працюючої людини на охорону праці охороняються всебічно.

До нормативно-правових актів з охорони праці відносяться правила, норми, регламенти, положення, стандарти, інструкції та інші документи, обов'язкові до виконання.

4.1.2 Організаційно-технічні заходи з безпеки праці

В організації/підприємстві проводиться навчання і перевірка знань з питань охорони праці відповідно до вимог Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці України від 26.01.2005 4 15, зареєстрованого в Міністерстві юстиції України 15.02.2005 за 4 231/10511 [27].

Обов'язковими вимогами враховане наступне:

– не слід допускати до роботи осіб, що в установленому порядку не пройшли навчання, інструктаж та перевірку знань з охорони праці, пожежної безпеки та цих Правил;

– на підприємстві/організації, де експлуатуються ЕОМ з відео дисплейними терміналами (ВДТ) і периферійними пристроями (ПП), розробляється інструкція з охорони праці відповідно до Положення про розробку інструкцій з охорони праці, затвердженого наказом Держнаглядохоронпраці від 29.01.98 4 9, зареєстрованого в Міністерстві юстиції України 07.04.98 за 4 226/2666 [28];

– обов'язкові організаційні заходи перед початком, під час і після завершення роботи повинні включати перевірку (візуально) наявності і справності електрообладнання та його заземлення, а під час виконання роботи вимогу «не

залишати без нагляду обладнання, яке працює». Після закінчення роботи - вимагається прибирання робочого місця, відключення всіх електроприладів від електромережі.

Не допускається:

– виконувати обслуговування, ремонт та налагодження ЕОМ з ВДТ і ПП безпосередньо на робочому місці оператора;

– зберігати біля ЕОМ з ВДТ і ПП папір, дискети, інші носії інформації, запасні блоки, деталі тощо, якщо вони не використовуються для поточної роботи;

– відключати захисні пристрої, самочинно проводити зміни у конструкції та складі ЕОМ з ВДТ і ПП або їх технічне налагодження;

– працювати з ВДТ, у яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на екрані тощо;

– працювати з матричним принтером за відсутності вібраційного килимка та зі знятою (піднятою) верхньою кришкою.

4.2 Аналіз стану умов праці

Робота над створенням IDE для компільованих ігрових додатків системи проходитиме в приміщенні. Для даної роботи достатньо однієї людини, для якої надано робоче місце зі стаціонарним комп’ютером.

4.2.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в таблиці 4.1.

Таблиця 4.1 – Розміри приміщення

Найменування	**Значення**
Довжина, м	5
Ширина, м	3
Висота, м	2,5
Площа, м2	15
Об’єм, м3	37,5

Згідно з [29] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації.

4.2.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за [30] і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність (таб.4.2).

Таблиця 4.2 - Характеристики робочого місця

Найменування параметра	Фактичне значення	Нормативне значення
Висота робочої поверхні, мм	750	680 ÷800
Висота простору для ніг, мм	730	не менше 600
Ширина простору для ніг, мм	660	не менше 500
Глибина простору для ніг, мм	700	не менше 650
Висота поверхні сидіння, мм	470	400 ÷500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	500	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷800

Приміщення кабінету має об’єм 37,5 м^3, площу – 15 м^2.

Температура в приміщенні протягом року коливається у межах 18–24°С, відносна вологість — близько 50%. Система вентилювання приміщення — природна неорганізована, а опалення — централізоване.

Розміщення вікон забезпечує природне освітлення з коефіцієнтом природного освітлення не менше 1,5%, а загальне штучне освітлення, яке здійснюється за допомогою восьми люмінесцентних ламп, забезпечує рівень освітленості не менше 200 Лк.

За ступенем пожежної безпеки приміщення належить до категорії В.

4.2.3 Навантаження та напруженість процесу праці

За фізичним навантаженням робота відноситься до категорії легкі роботи (Іа), її виконують сидячи з періодичним ходінням. Щодо характеру організування виконання дипломної роботи, то він підпадає під нав'язаний режим, оскільки певні розділи роботи необхідно виконати у встановлені конкретні терміни. За ступенем нервово-психічної напруги виконання роботи можна віднести до II – III ступеня і кваліфікувати як помірно напружений – напружений за умови успішного виконання поставлених завдань.

Під час виконання робіт використовують ПК та периферійні пристрої (лазерні та струменеві), що призводить до навантаження на окремі системи організму. Такі перекоси у напруженні різних систем організму, що трапляються під час роботи з ПК, зокрема, значна напруженість зорового аналізатора і довготривале малорухоме положення перед екраном, не тільки не зменшують загального напруження, а навпаки, призводять до його посилення і появи стресових реакцій.

Наявні психофізіологічні небезпечні та шкідливі фактори:

а) фізичного перевантаження:
   - статичного;
   - динамічного;

б) нервово-психічного перевантаження:
   - розумового перенапруження;
   - монотонності праці;
   - перенапруження аналізаторів;
   - емоційних перевантажень.

Роботу за дипломним проектом визнано, таку, що займає 50% часу робочого дня та за восьмигодинної робочої зміни рекомендовано встановити додаткові регламентовані перерви тривалістю 15 хвилин через кожну годину роботи.

4.3 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки будуть надалі вирішені питання необхідності забезпечення достатньої кількісті освітлення, вентиляції повітря, організації заземлення, тощо.

4.3.1 Аналіз небезпечних та шкідливих факторів при виробництві виробу

Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00-7.15-18 [34], які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП.

Роботи за проектами виконують у кабінетах, де використовують різноманітне електрообладнання, зокрема персональні комп'ютери (ПК) та периферійні пристрої.

Основними робочими характеристиками персонального комп'ютера є наступні:

– робоча напруга U = +220В ± 5%;

– робочий струм I = 2А;

– споживана потужність P = 350 Вт.

Робоче місце має відповідати вимогам Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 4 7 [30].

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3).

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	**Джерела факторів (види робіт)**	**Кількісна оцінка**	**Нормативні документи**
1	2	3	А
Фізичні			
- підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	4	ГОСТ 12.1.030-81 [7] ГОСТ 13109-97 [8]
- недостатність природного світла	порушення умов праці (вимог до приміщень)	2	ДБН В.2.5-28:2018 [9]
- недостатнє освітлення робочої зони	порушення гігієнічних параметрів виробничого середовища	3	ДБН В.2.5-28:2018 [9]
психофізіологічні:			
- фізичні (статичне – сидіння)	порушення умов праці (організації місця праці сидіння користувача,) та організації робочого часу - безперервна робота)	2	НПАОП 0.00-7.15-18 [10] ДСанПіН 3.3.2.007-98 [6]

Продовження таблиці 4.3

Небезпечні і шкідливі виробничі фактори	Джерела факторів (види робіт)	Кількісна оцінка	Нормативні документи
- нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових)	- пошук інформації для постановки теми; - пошук та аналіз аналогів і літератури; - пошук наявних технологій, моделювання та аналіз алгоритмів; - виконання роботи за темою диплома, тестування; - оформлення роботи	4	НПАОП 0.00-7.15-18 [10] ДСанПіН 3.3.2.007-98 [6]

4.3.2 Пожежна безпека

Приміщення оснащено системою автоматичної пожежної сигналізації, має 1 вогнегасник ВП-5 із зарядом вогнегасної речовини 8-12 кг, відповідно до вимог чинного законодавства України. Проходи до засобів пожежогасіння вільні, не захаращуються та у разі потреби забезпечувати евакуацію всіх людей, які перебувають у приміщенні через один евакуаційний вихід з дверима, що відчинятися в напрямку виходу з будівлі від робочого місця. В приміщенні наявна затверджена «План-схема евакуації з кабінету (приміщення)».

Пожежна безпека при застосуванні ЕОМ забезпечується:

– системою запобігання пожежі,

– системою протипожежного захисту,

– організаційно-технічними заходами.

Згідно ДСТУ Б В.1.1-36:2016 [35] таке приміщення, площею 25 м^2, відноситься до категорії "В" (пожежонебезпечної) та для протипожежного захисту в ньому проектом передбачено устаткування автоматичною пожежною сигналізацією із застосуванням датчиків-сповіщувачів РІД-1 (сповіщувач димовий ізоляційний) в кількості 1 шт., і застосуванням первинних засобів пожежогасіння.

Горючими матеріалами в приміщенні, де розташовані ЕОМ, є:

– поліамід – матеріал корпусу мікросхем, горюча речовина, температура самозаймання 420º Сж

– полівінілхлорид – ізоляційний матеріал, горюча речовина, температура запалювання 335º С, температура самозаймання 530º Сж

– склотекстоліт ДЦ – матеріал друкарських плат, важкогорючий матеріал, показник горючості 1.7А, не схильний до температурного самозаймання,

– пластикат кабельний №.489 – матеріал ізоляції кабелів, горючий матеріал, показник горючості більше 2.1ж

– деревина – будівельний і обробний матеріал, з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, температура запалювання 255º С, температура самозаймання 399º С.

Простори усередині приміщень в межах, яких можуть утворюватися або знаходиться пожежонебезпечні речовини і матеріали відповідно до [35] відносяться до пожежонебезпечної зони класу П-IIа. Це обумовлено тим, що в приміщенні знаходяться тверді горючі та важкозаймисті речовини та матеріали. Приміщенню, у якому розташоване робоче місце, присвоюється II ступень вогнестійкості.

Продуктами згорання, що виділяються на пожежі, є: окис вуглецю; сірчистий газ; окис азоту; синильна кислота; акромін; фосген; хлор і ін. При горінні пластмас, окрім звичних продуктів згорання, виділяються різні продукти термічного розкладання: хлорангідридні кислоти, формальдегіди, хлористий водень, фосген, синильна кислота, аміак, фенол, ацетон, стирол. [36].

4.3.3 Електробезпека

На робочому місці виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три- провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК, укладено по підлозі поруч зі стінами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання.

4.4 Гігієнічні вимоги до параметрів виробничого середовища

4.4.1 Параметри мікроклімату

Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря. В даному приміщенні проводяться роботи, що виконуються сидячи і не потребують динамічного фізичного напруження, то для нього відповідає категорія робіт Іа. Отже оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають [29] і наведені в табл. 4.4:

Таблиця 4.4 – Норми мікроклімату робочої зони об’єкту

Період року	Категорія робіт	Температура С0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	легка-1 а	22 - 24	40 – 60	0,1
Тепла	легка-1 а	23 - 25	40 – 60	0,1

4.4.2 Освітлення

Збільшення освітленості сприяє поліпшенню працездатності навіть в тих випадках, коли процес праці практично не залежить від зорового сприйняття.

Освітленість приміщення має велике значення при роботі на ПЕОМ. Вона багато в чому визначається колірною і мережевий обстановкою. Для зменшеного поглинання світла стеля і стіни вище панелей (1,5-1,7м.). Якщо вони не облицьовані звукопоглинальним матеріалом, фарбуються білою водоемульсійною фарбою (з коефіцієнтом відбиття не менше 0,7). Для забарвлення стіни панелей рекомендується віддавати перевагу світлим фарбам.

Природне освітлення, коли робочі місця з ПЕОМ розташовуються в один ряд по довжині приміщення на відстані 0,8 - 1,0 м від стіни з віконними прорізами, і екрани знаходяться перпендикулярно цієї стіни. Основний потік природного світла при цій повинен бути зліва. Не допускається спрямування основного світлового потоку природного світла праворуч, ззаду і спереду працює на ПЕОМ. Оптимальна відстань очей до екрана відео монітора повинна становити 60-70 см, допустиме не менше 50 см. Розглядати інформацію ближче 50 см не рекомендується.

У приміщенні, де розташовані ЕОМ передбачається природне бічне освітлення, рівень якого відповідають [33]. Джерелом природного освітлення є сонячне світло.

Регулярно повинен проводитися контроль освітленості, який підтверджує, що рівень освітленості задовольняє ДБН [33] і для даного приміщення в світлий час доби достатньо природного освітлення.

Розрахунок освітлення.

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше – 1/8, в побутових – 1/10:

$$S_b = \left(\frac{1}{5} \div \frac{1}{10}\right) * S_n \quad (4.1)$$

де Sb – площа віконних прорізів, м2;

Sn – площа підлоги, м2;

Sn = a·b = 5·3,5 = 17,5 м2;

S = 1/8·17,5 = 2,1875 м2;

Приймаємо 2 вікна площею S=1,1 м2 кожне.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 5 м, ширина 3,5 м, світильниками ЛПО2П, оснащеними лампами типа ЛБ (дві по 80 Вт) з світловим потоком 5400 лм кожна.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E*S*Z*K}{F*U*M} \quad (4.2)$$

де E – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м2; S = 17,5 м2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу

приміщення і т.п. – 0,575;

М – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 * 17{,}5 * 1{,}1 * 1{,}5}{5400 * 0{,}575 * 2} \approx 1$$

Приймаємо освітлювальну установку, яка складається з 1-го світильника, який складається з двох люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

4.4.3 Шум та вібрація, електромагнітне випромінювання

Рівень шуму, що супроводжує роботу користувачів персональних комп'ютерів, коливається у межах 50–65 дБА [37]. Шум такої інтенсивності на тлі високого ступеня напруженості праці негативно впливає на функціональний стан користувачів.

У приміщенні з ЕОМ коректований рівень звукової потужності не перевищує 45 дБА. Оскільки рівень шуму не перевищує гранично допустимих величин встановлені санітарними нормами, заходи для зниження шуму не проводяться. Вібрація на робочому місці в приміщенні, що розглядається, відповідає нормам [37]. Допустимий рівень вібрацій на робочому місці: - для 1 ступеня шкідливості до 3 дБ; - для 2-3 - 1-6 дБ; - для 3 - більше 6 дБ.

Для захисту від електромагнітного випромінювання передбачаються наступні заходи: застосування нових плазмових моніторів, LG W2271TC, віддалення робочого місця не менше, ніж на 0,4 – 0,5 м, оскільки напруженість електричного поля зменшується при віддаленні від джерела поля, встановлення раціональних режимів роботи персоналу, раціональне розміщення в робочому приміщенні устаткування, що випромінює електромагнітну енергію.

4.4.4 Вентилювання

Здійснюється провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.5 Розрахунок захисного заземлення (забезпечення електробезпеки будівлі)

Загальний опір захисного заземлення визначається за формулою:

$$R_{ззп} = \frac{R_з \cdot R_п}{R_п \cdot n \cdot \eta_з + R_з \cdot \eta_п}, \quad (4.3)$$

де Rз - опір заземлення, якими когут бать труби, опори, кути і т.п., Ом;

Rш - опір опори, яке з'єднує заземлювачі, Ом;

п - кількість заземлювачів;

ηз - коефіцієнт екранування заземлювача; приймається в межах 0,2 ÷ 0,9; ηз = 0,7

ηш - коефіцієнт екранування сполучної стійки; приймається в межах 0,1 ÷ 0,7; ηш = 0,5;

Опір заземлення визначається за формулою:

$$R_з = \frac{\rho}{2\pi \cdot l} \cdot \left(\ln \frac{2 \cdot l}{d} + \frac{1}{2} \ln \frac{4 \cdot t + l}{4 \cdot t - l} \right), \quad (4.4)$$

де ρ - питомий опір ґрунту, залежить від типу грунту, Ом·м;

для піску - А00 ÷ 700 Ом·м; приймаємо ρ = А00 Ом·м;

l - довжина заземлювача, м; для труб - 2-3 м; l = 3 м;

d - діаметр заземлювача, м; для труб - 0,03-0,05 м; d = 0,05 м;

t - відстань від середини забитого в ґрунт заземлювача до рівня землі, м; t = 2 м.

$$R_з = \frac{400}{2 \cdot 3{,}14 \cdot 3} \left(\ln \frac{2 \cdot 3}{0{,}05} + \frac{1}{2} \ln \frac{4 \cdot 2 + 3}{4 \cdot 2 - 3} \right) = 110, \text{ Ом} \quad (4.5)$$

Опір смуги, що з'єднує заземлювачі, визначається за формулою:

$$R_ш = \frac{\rho}{2\pi \cdot L} \cdot \ln \frac{2 \cdot L^2}{b \cdot t^1}, \quad (4.6)$$

де L - довжина смуги, що з'єднує заземлювачі (м) і приблизно дорівнює периметру будівлі: $П_{буд.} = 42 \cdot 2 + 38 \cdot 2 = 160$ м; L = 160 м;

b - ширина смуги, м; b = 0,03 м;

t_1 - глибина заземлення від рівня землі, м; t_1 = 0,5 м.

$$R_n = \frac{400}{2 \cdot 3{,}14 \cdot 160} \cdot \ln \frac{2 \cdot 160^2}{0{,}03 \cdot 0{,}5} = 5{,}99 \text{, Ом}$$

Кількість заземлювачів захисного заземлення визначається за формулою 4.7:

$$n = \frac{2 \cdot R_з}{4 \cdot \eta_з}, \tag{4.7}$$

де 4 - допустимий загальний опір, Ом;

2 - коефіцієнт сезонності.

Визначаємо загальний опір захисного заземлення:

$$R_{ззп} = \frac{110 \cdot 5{,}99}{5{,}99 \cdot 79 \cdot 0{,}7 + 110 \cdot 0{,}5} = 1{,}7 \text{ Ом} \tag{4.8}$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{ззп} < 4$ Ом.

При виникненню пожеж при роботі на ПЕОМ від таких можливими джерел запалювання як:

– іскри і дуги коротких замикань;

– перегрів провідників, резисторів та інших радіодеталей ПЕОМ, від тривалої перевантаження та наявність перехідного опору;

– іскри при розмиканні і розмиканні ланцюгів;

– розряди статичної електрики;

– необережному поводженню з вогнем, а також вибухи газоповітряних і пароповітряних сумішей.

4.6 Охорона навколишнього природного середовища

Діяльність з використання комп’ютерної техніки впливає на навколишнє природне середовище.

Основним екологічним аспектом в процесі діяльності за даними спеціальностями є процеси впливу на атмосферне повітря та процеси поводження з відходами, які утворюються, збираються, розміщуються, передаються на видалення

(знешкодження), утилізацію, тощо в ІТ галузі.

Немає впливу на атмосферне повітря при нормальних умовах праці, бо в приміщенні не використовуються сканери, принтери та інші джерела викиду забруднюючих речовин в повітря робочої зони.

В процесі діяльності користувача виникають процеси поводження з відходами ІТ галузі.

Нижче надано перелік відходів, що утворюються в процесі роботи:

– відпрацьовані люмінесцентні лампи - I клас небезпеки;
– змінні носії інформації - IV клас небезпеки;
– макулатура - IV клас небезпеки.

Висновки до розділу 4

1) в результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над проектом, для того щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

2) приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

ВИСНОВКИ

1) Ігрове середовище розробки дозволяє розробникам працювати над грою більш абстрактно, не замислюючись про низькорівневе представлення компонентів двигуна, а також те, як компоненти працюють і пов'язані між собою. До того ж виконуваний код, написаний з використанням ігрового середовища, виглядає більш організованим і керованим, що знижує загальну кількість виникаючих програмних помилок.

2) Аналіз існуючих середовищ розробки з вбудованими ігровими двигунами, як комерційних, так і безкоштовних, підтверджує доцільність розробки IDE для створення ігрових додатків спочатку під конкретні вимоги, а в подальшому з реалізацією його архітектури таким чином, щоб легко підтримувати його актуальним і адаптувати під проекти будь-якого роду.

3) Робота над проектом по створенню ігрового середовища розробки представляє із себе досить складний і великий проект, який дозволяє істотно підняти рівень навичок в програмуванні складних автоматизованих програм; ці навички будуть актуальними протягом тривалого часу.

4) Використання розробленої власноруч архітектури є відмінним рішенням для програми з високими вимогами до швидкості виконання і мінімізації спожитої пам'яті. Це надає свободу дій програмісту, але, в свою чергу, накладає додаткові вимоги до знання всієї розроблюваної системи і прийнятих концепцій.

5) Використання робіт, опублікованих автором, в якості базису деяких підсистем розробленої програми дозволило прискорити розробку і підвищити її якість, оскільки для використаних методів і способів неодноразово проводилися аналітичні тести.

6) Розробка прототипу програми для перевірки придатності узгоджених для застосування концепцій, архітектурних і технологічних рішень, є вірним рішенням. В процесі реалізації ця розробка неодноразово удосконалювалася в бік збільшення продуктивності, зниження спожитої пам'яті та загальної оптимізації.

7) Розроблений програмний продукт є унікальним, оскільки використовувані концепції і розроблені механізми написані без використання готових рішень та сторонньої допомоги. Проект “Mad Game Editor” є інтелектуальною власністю автора випускної кваліфікаційної роботи.

Даний проект, у зв’язку з його актуальністю та перспективою подальшого розвитку, буде й надалі вдосконалюватися автором для розробки ігрових додатків.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Інкрементна розробка [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Ітеративна_та_інкрементна_розробка;

2. Деталізація сферично-динамічного рельєфу в комп'ютерній графіці 2018р. IX Всеукраїнській науково-практичній конференції «Майбутній науковець – 2018» / Мовшук Н.А., Щербакова М. Є./ Електронне видання. ст. 35-37;

3. Розробка та аналіз програмного рішення файлового кешування даних в мові C++ на основі технології memory mapped files 2020р. Наукові вісті Далівського університету /Мовшук Н.А. / Електронне видання. 2020 №17. С. - .(прийнято до друку);

4. Динамічна деталізація сферичного рельєфу в комп'ютерній графіці 2018р. IV регіональний форум «IT-Ідея 2018» / Мовшук Н.А., Лавриненко О.О. / Електронне видання. ст. 59-61;

5. Інтегроване середовище розробки [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Інтегроване_середовище_розробки;

6. Компілятор [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Компілятор;

7. GNU General Public Licenseки [Електронний ресурс]. Режим доступу: https://opensource.org/licenses/gpl-license;

8. Ігрові двигуни [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Game_engine ;

9. Список ігрових двигунів [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/List_of_game_engines;

10. Unreal Engine [Електронний ресурс]. Режим доступу: https://www.unrealengine.com;

11. CryEngine [Електронний ресурс]. Режим доступу: https://www.cryengine.com;

12. HeroEngine [Електронний ресурс]. Режим доступу: http://www.heroengine.com;

13. Source [Електронний ресурс]. Режим доступу: https://developer.valvesoftware.com/wiki/Source_2;

14. Project Anarchy [Електронний ресурс]. Режим доступу: https://devtribe.ru/p/other-engines/projanarch;

15. Unity 3D [Електронний ресурс]. Режим доступу: https://unity.com;

16. Дерево квадрантів [Електронний ресурс]. Режим доступу:

https://uk.wikipedia.org/wiki/Дерево_квадрантів

17. Memory mapped files [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Memory-mapped_file;

18. Ейлерові кути [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Ейлерові_кути;

19. Visual Studio IDE [Електронний ресурс]. Режим доступу: https://visualstudio.microsoft.com/ru/?rr=https%3A%2F%2Fwww.google.com.ua%2F;

20. Microsoft Visual Studio [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio;

21. Windows API [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list;

22. DirectX [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/DirectX;

23. Графічний конвеєр [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/windows/uwp/graphics-concepts/graphics-pipeline.

24. FLTK [Електронний ресурс]. Режим доступу: https://www.fltk.org;

25. Закон України «Про охорону праці». Вводиться в дію Постановою ВР № 2695-XII від 14.10.92, ВВР, 1992, № 49, ст.669. Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12;

26. Кодекс Законів про Працю України (КЗпПУ). Затверджується Законом № 322-VIII від 10.12.71 ВВР, 1971. Режим доступу: https://zakon.rada.gov.ua/laws/show/322-08;

27. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 0.00-4.12-05). Наказ від 26.01.2005 №15. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0231-05;

28. Положение о разработке инструкций по охране труда (НПАОП 0.00-4.15-98). Наказ від 29.01.1998 № 9. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0226-98;

29. Санітарні норми мікроклімату виробничих приміщень (ДСН 3.3.6.042-99). Постанова №42 від 01.12.99. Режим доступу: https://zakon.rada.gov.ua/rada/show/va042282-99;

30. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПІН 3.3.2.007-98). Затверджено Постановою Головного державного санітарного лікаря України 10 грудня 1998 р. N 7. Режим доступу: https://zakon.rada.gov.ua/rada/show/v0007282-98;

31. Правила безопасной эксплуатации електроустановок (НПАОП 40.1-1.01-97). Наказ від 06.10.1997 № 257. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0011-98;

32. Норми якості електричної енергії в системах електропостачання загального призначення (ГОСТ 13109-97). Дата введенья 01.01.1999. Режим доступу: http://odz.gov.ua/lean_pro/standardization/files/elektromagnitnaja_sovmestimost_2014_03_11_1.pdf

33. Природне і штучне освітлення (ДБН В.2.5-28:2018). Наказ Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264Режим доступу: http://www.minregion.gov.ua/wp-content/uploads/2018/12/V2528-1.pdf;

34. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями (НПАОП 0.00-7.15-18). Зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за № 508/31960. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18;

35. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою (ДСТУ Б В.1.1-36:2016). Наказ від 15.06.2016 №158. Режим доступу: http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=65419;

36. Система стандартов безопасности труда. Пожаровзрывоопасность веществ и материалов. Номенклатура показателей и методы их определения (ГОСТ 12.1.044-89). Дата введенья 12.12.1989. Режим доступу: http://online.budstandart.com/ru/catalog/doc-page.html?id_doc=51048;

37. Санітарні норми виробничого шуму, ультразвуку та інфразвуку (ДСН 3.3.6.037-99). Наказ від 01.12.1999 №37. Режим доступу: https://zakon.rada.gov.ua/rada/show/va037282-99.

38. Методичні вказівки до виконання та захисту магістерської роботи за спеціальностями 122 "Комп’ютерні науки та інформаційні технології", 123 "Комп’ютерна інженерія" (для студентів денної та заочної форм навчання) / Уклад.: СкаргаБандурова І.С., Рязанцев О.І., Барбарук В.М., Щербакова М.Є. – Сєвєродонецьк: Вид-во СНУ ім. В. Даля, 2016. – 80 с.

Додаток А

Програмне представлення структури “Тип шаблону проекту”

```
//TypeProject_Проект параметры типа (F - Флаги, M - Режимы)
typedef UINT TTypeProject;
//Занятость битов TTypeProject 4 байт/32 бита - 1111 1000  0000 0000  0000 0000  0000 0000
//+Для типа карты Динамическая ограниченная карта высот
//Занятость битов TTypeProject 4 байт/32 бита - 0000 0111  1110 0000  0000 0000  0000 0000
//Смещения
#define _TypeProjectM_Shift_OS                  31u //Смещение бит, Операционная система
#define _TypeProjectM_Shift_OSBit               30u //Смещение бит, Разрядность операционной системы
#define _TypeProjectM_Shift_GraphLibVerSha      29u //Смещение бит, Графическая библиотека / Версия шейдера
#define _TypeProjectM_Shift_TypeNetArch         28u //Смещение бит, Тип сетевой архитектуры
#define _TypeProjectM_Shift_TypeMap             27u //Смещение бит, Тип карты
//Для типа карты Динамическая ограниченная карта высот
#if _MGE_ExecutableModule_Launcher||_MGE_ExecutableModule_w732DX9CSDLMH//Специфично для MGE_Launcher | MGE_w732DX9CSDLMH
  #define _TypeProjectM_Shift_ResolutionHeightMap   24u //Смещение бит, Разрешение карты высот
  #define _TypeProjectM_Shift_MetersPerUnit         21u //Смещение бит, Разрешение карты высот::Метров на единицу
#endif
//Маска
#define TypeProjectM_Mask_OS                    0x80000000u //0x1000 0000  0000 0000  0000 0000  0000 0000 - 1 бит, Операционная система
#define TypeProjectM_Mask_OSBit                 0x40000000u //0x0100 0000  0000 0000  0000 0000  0000 0000 - 1 бит, Разрядность операционной системы
#define TypeProjectM_Mask_GraphLibVerSha        0x20000000u //0x0010 0000  0000 0000  0000 0000  0000 0000 - 1 бит, Графическая библиотека/Версия шейдера
#define TypeProjectM_Mask_TypeNetArch           0x10000000u //0x0001 0000  0000 0000  0000 0000  0000 0000 - 1 бит, Тип сетевой архитектуры
#define TypeProjectM_Mask_TypeMap               0x08000000u //0x0000 1000  0000 0000  0000 0000  0000 0000 - 1 бит, Тип карты
//Для типа карты Динамическая ограниченная карта высот
#if _MGE_ExecutableModule_Launcher||_MGE_ExecutableModule_w732DX9CSDLMH//Специфично для MGE_Launcher | MGE_w732DX9CSDLMH
  #define TypeProjectM_Mask_ResolutionHeightMap     0x70000000u  //0x0000 0111  0000 0000  0000 0000  0000 0000 - 3 бит, Разрешение карты высот
  #define TypeProjectM_Mask_MetersPerUnit           0xE0000000u  //0x0000 0000  1110 0000  0000 0000  0000 0000 - 3 бит, Разрешение карты высот::Метров на единицу
#endif
//Значения режимов
#define TypeProjectM_OS_Windows7_8_10           (0x0u<<_TypeProjectM_Shift_OS)                  //Операционная система - Windows 7,8,10
#define TypeProjectM_OSBit_x32                  (0x0u<<_TypeProjectM_Shift_OSBit)               //Разрядность операционной система - x32
#define TypeProjectM_GraphLibVerSha_D3D90L_Sha30 (0x0u<<_TypeProjectM_Shift_GraphLibVerSha)     //Графическая библиотека/Версия шейдера - Direct3D 9.0L(Ex) / 3.0
#define TypeProjectM_TypeNetArch_Not            (0x0u<<_TypeProjectM_Shift_TypeNetArch)         //Тип сетевой архитектуры - отсутствует
#define TypeProjectM_TypeNetArch_ClientServer   (0x1u<<_TypeProjectM_Shift_TypeNetArch)         //Тип сетевой архитектуры - клиент-сервер
#define TypeProjectM_TypeMap_DynamicLimitedHeightMap (0x0u<<_TypeProjectM_Shift_TypeMap)        //Тип карты - Динамическая ограниченная карта высот
//Для типа карты Динамическая ограниченная карта высот
#if _MGE_ExecutableModule_Launcher||_MGE_ExecutableModule_w732DX9CSDLMH//Специфично для MGE_Launcher | MGE_w732DX9CSDLMH
  #define TypeProjectM_ResolutionHeightMap_128  (0x0u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот -  128x128
  #define TypeProjectM_ResolutionHeightMap_256  (0x1u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот -  256x256
  #define TypeProjectM_ResolutionHeightMap_512  (0x2u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот -  512x512
  #define TypeProjectM_ResolutionHeightMap_1024 (0x3u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот - 1024x1024
  #define TypeProjectM_ResolutionHeightMap_2048 (0x4u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот - 2048x2048
  #define TypeProjectM_ResolutionHeightMap_4096 (0x5u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот - 4096x4096
  #define TypeProjectM_ResolutionHeightMap_8192 (0x6u<<_TypeProjectM_Shift_ResolutionHeightMap) //Разрешение карты высот - 8192x8192
  #define TypeProjectM_MetersPerUnit_1          (0x0u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу -  1
  #define TypeProjectM_MetersPerUnit_2          (0x1u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу -  2
  #define TypeProjectM_MetersPerUnit_4          (0x2u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу -  4
  #define TypeProjectM_MetersPerUnit_8          (0x3u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу -  8
  #define TypeProjectM_MetersPerUnit_16         (0x4u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу - 16
  #define TypeProjectM_MetersPerUnit_32         (0x5u<<_TypeProjectM_Shift_MetersPerUnit) //Метров на единицу - 32
#endif
namespace nFileMGE{
/*Структура файла *.mge
  TProParam     -4 байт - Проект параметры
  char *NameProject -n байт - Имя проекта
  char *NameApp     -n байт - Имя приложения
  TVer          -8 байт - Версия проекта
  TTypeProject  -4 байт - Тип проекта
*/
  const uint64_t CID_ProParam=17534960018052767614ull;//Проект параметры
  namespace nSecHeader{//Сектор заглавия
    const uint64_t CID_NameProject    =10927193470207497325ull;//Имя проекта
    const uint64_t CID_NameApp        =10928428214687834412ull;//Имя приложения
    const uint64_t CID_VersionProject =7726473537859609569ull;//Версия проекта
    const uint64_t CID_TypeProject    =19295431509078700911ull;//Тип проекта
  };
};
```

Додаток Б

Програмне представлення бібліотеки мовної локалізації загального призначення

```
//Lib_LL.cpp - Определяет реализацию функций и прочего, и экспортированные функции для приложения DLL
#include "Lib_LL.h"

//##########################################################
//#                       CONST                            #
//##########################################################
LibLL::
//Начало раздела автоматической генерации языковой локализации, изменять код внутри секции ЗАПРЕЩЕНО!, он просто будет перетёрт при автогенерации
//BEGIN_SECTION_SALL
MGE_Lib_LL_API const char LibLL::cALL_Button_No[]="\320\235\320\265\321\202";
MGE_Lib_LL_API const char LibLL::cALL_Button_Yes[]="\320\224\320\260";
MGE_Lib_LL_API const char LibLL::cALL_Button_Ok[]="Ok";
MGE_Lib_LL_API const char LibLL::cALL_Button_Cancel[]="\320\236\321\202\320\274\320\265\320\275\320\260";
MGE_Lib_LL_API const char LibLL::cALL_Button_Close[]="\320\227\320\260\320\272\321\200\321\213\321\202\321\214";
MGE_Lib_LL_API const char LibLL::cALL_Button_ExitProgram[]="Exit Program";
MGE_Lib_LL_API const char LibLL::cALL_Button_Back[]="@<+<-                \320\235\320\260\320\267\320\260\320\264";
MGE_Lib_LL_API const char LibLL::cALL_Button_Next[]="\320\224\320\260\320\265\320\265   @<+>-";
MGE_Lib_LL_API const char LibLL::cALL_Button_Create[]="\320\241\320\276\320\267\320\264\320\260\321\202\321\214";
MGE_Lib_LL_API const char LibLL::cALL_Button_Ignore[]="\320\230\320\263\320\275\320\276\321\200\321\270\321\200\320\276\320\262\320\260\321\202\321\214";
MGE_Lib_LL_API const char LibLL::cALL_ErrLoadIncludeResource[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\267\320\260\320\263\321\200\321\203\320\267\320\272\320\270 \320\272\320\276\320\274\320\277\320\276\320\275\320\265\320\275\320\242";
MGE_Lib_LL_API const char LibLL::cALL_AllowedSymbolsNameFile[]="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_[&](!)>=@+'-^;.\" \320\261\320\265\320\267\320\274\320\217 \
MGE_Lib_LL_API const char LibLL::cALL_ErrCreateDirectory[]="\320\236\321\210\320\270\320\261\320\272\320\260 \321\201\320\276\320\267\320\264\320\260\320\275\320\270\321\217 \
MGE_Lib_LL_API const char LibLL::cALL_ProjectWithThisNameAlreadyExists[]="\320\237\321\200\320\276\320\265\320\272\321\202 \321\201 \321\202\320\260\320\272\320\274 \
MGE_Lib_LL_API const char LibLL::cALL_NotSpecified[]="\320\235\320\265 \321\203\320\272\320\260\320\267\320\260\320\275\320\276";
MGE_Lib_LL_API const char LibLL::cALL_IncorrectStartParam[]="\320\235\320\265\321\200\320\275\320\276 \321\203\320\272\320\260\320\267\320\260\320\275\321\213
MGE_Lib_LL_API const char LibLL::cALL_ErrorOpenFileProject[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\276\321\202\320\272\321\200\321\213\321\202\320\270\321\21
MGE_Lib_LL_API const char LibLL::cALL_TypeProjectNotMatchExeModule[]="\320\242\321\270\320\277 \320\277\321\200\320\276\320\265\320\272\321\202\320\260 \320\275\320\265 \321\
MGE_Lib_LL_API const char LibLL::cALL_ErrAllocationByteMemory_1[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\262\321\213\320\264\320\265\320\273\320\275\320\
MGE_Lib_LL_API const char LibLL::cALL_ErrAllocationByteMemory_2[]="\320\261\320\260\320\271\321\202 \320\277\320\260\320\274\321\217\321\202\320\270;";
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Menu[]="\320\234\320\265\320\275\321\216";
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_MenuSaveProject[]="\320\241\320\276\321\205\321\200\320\260\320\275\320\270\321\202\320\260\320\271 \320\277\321\200\320\276\320\265\32
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_MenuCloseProject[]="\320\227\320\260\320\272\321\200\321\213\321\202\321\214 \320\277\321\200\320\276\320\265\320\272\321\202";
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Display[]="\320\224\320\270\321\201\320\277\320\273\320\265\321\271";
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Display_ModeWireframe[]="\320\240\320\265\321\266\320\270\320\274 wireframe";
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Display_ProjectionView[]="\320\242\321\200\320\265\321\202\321\214 \320\274\320\260\321\202\321\200\320\270\321\206\321\213 \320\277\321\200\320
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Display_ProjectionView_OrthogonalProjection[]="\320\236\321\200\321\202\320\276\320\263\320\276\320\275\320\260\321\213\321\214\
MGE_Lib_LL_API const char LibLL::cCWE_MenuBar_Display_ProjectionView_PerspectiveProjection[]="\320\237\321\200\320\277\321\200\320\270 \321\201\320\276\321\205\321\200\320\260\
MGE_Lib_LL_API const char LibLL::cALL_ErrorSaveParamUSP[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\277\321\200\320\260\320\267\320\272\320\270 \
MGE_Lib_LL_API const char LibLL::cALL_ErrorLoadParamUSP[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\267\320\260\320\263\321\200\321\203\320\267\320\272\320\270 \
MGE_Lib_LL_API const char LibLL::cALL_ErrSaveFile[]="\320\236\321\210\320\270\320\261\320\272\320\260 \321\201\320\276\321\205\321\200\320\260\320\275\320\270\320\32
MGE_Lib_LL_API const char LibLL::cALL_SaveChangesExit[]="\320\241\320\276\321\205\321\200\320\260\320\275\320\270\321\202\321\214 \320\267\320\260\320\272\320\275\320\275\320
MGE_Lib_LL_API const char LibLL::cFD3D_ErrCreateInterfaceDirect3D[]="\320\236\321\210\320\270\320\261\320\272\320\260 \321\201\320\276\320\267\320\264\320\260\320\275\320\270\320
MGE_Lib_LL_API const char LibLL::cFD3D_ErrReleaseInterfaceDirect3D[]="\320\236\321\210\320\270\320\261\320\272\320\260 \320\262\321\213\321\201\320\262\320\276\320\261\320\27
MGE_Lib_LL_API const char LibLL::cFD3D_ErrGetInfoVideoDevice[]="\320\236\321\210\320\261\320\272\320\260 \320\277\320\276\320\273\321\203\321\207\320\265\320\275\320\
MGE_Lib_LL_API const char LibLL::cFD3D_FormatPixelNotFound[]="\320\237\320\276\320\264\321\205\320\276\320\264\321\211\321\211\320\270\320\271 \321\204\320\276\321\200\320\27
MGE_Lib_LL_API const char LibLL::cFD3D_FormatDepthStencilNotFound[]="\320\237\320\276\320\264\321\205\320\276\320\264\321\211\321\211\320\270\320\271 \321\204\320\276\321\200\320\27
MGE_Lib_LL_API const char LibLL::cFD3D_ErrCreateDeviceDirect3D[]="\320\236\321\210\320\270\320\261\320\272\320\260 \321\201\320\276\320\267\320\264\320\260\320\275\320\270\320\32
MGE_Lib_LL_API const char LibLL::cFD3D_UseTypeVertexProcess_MVP[]="\320\230\321\201\320\277\320\276\320\273\321\214\320\267\320\276\320\262\320\260\320\275\320\270\320\265 \\3
MGE_Lib_LL_API const char LibLL::cFD3D_UseTypeVertexProcess_SVP[]="\320\230\321\201\320\277\320\276\320\273\321\214\320\267\320\276\320\262\320\260\320\275\320\270\320\265 \\3
MGE_Lib_LL_API const char LibLL::cFD3D_NotSupportShaderVersion30InHardVertexProcess[]="\320\235\321\202 \320\277\320\276\320\264\320\264\320\265\321\200\320\266\320\2
MGE_Lib_LL_API const char LibLL::cALL_WarReductionsPerformance[]="\320\237\321\200\320\265\320\264\321\203\320\277\321\200\320\265\320\266\320\264\320\265\320\275\320\270\320
MGE_Lib_LL_API const char LibLL::cCWE_Log[]="\320\226\321\203\321\200\320\273";
MGE_Lib_LL_API const char LibLL::cFSM_Log_FileMapNotFoundSetMappefault[]="War->\t\320\244\320\260\320\271\320\273 \320\263\320\265\320\276\320\274\320\265\321\202\321\200\320
MGE_Lib_LL_API const char LibLL::cFSM_IncorrectSizeFileMap[]="\320\235\320\265\320\272\320\276\321\200\320\265\320\275\321\213\320\271 \321\200\320\26
//END_SECTION_SALL
//Конец раздела автоматической генерации языковой локализации, изменять код внутри секции ЗАПРЕЩЕНО!, он просто будет перетёрт при автогенерации
```

Додаток В

Програмний код створення проекту в MGELauncher

```
void CWindowStartup::CGroup_CreateProject::Callback_Button_Create(void){
  //Проверка корректности ввода Проект->Имя проекта
  if(nAuxFunc::TestAllowedSymbolsNameFile_String(hInput_TabProject_NameProject->value())==0){
    //В случаи непрохода фильтров ввода
    hInput_TabProject_NameProject->color(FL_RED);
    hInput_TabProject_NameProject->redraw();
    hTabs->value(hGroup_TabProject);
    _Jump_IncorrectEnterName:
    LogMes(_Log_Typ_Fau,LibLL::cALL_Button_Ok,0,0,LibLL::cALL_AllowedSymbolsNameFile);
    return;}
  hInput_TabProject_NameProject->color(FL_BACKGROUND2_COLOR);
  hInput_TabProject_NameProject->redraw();

  //Проверка корректности ввода Проект->Имя приложения
  if(nAuxFunc::TestAllowedSymbolsNameFile_String(hInput_TabProject_NameApp->value())==0){
    //В случаи непрохода фильтров ввода
    hInput_TabProject_NameApp->color(FL_RED);
    hInput_TabProject_NameApp->redraw();
    hTabs->value(hGroup_TabProject);
    goto _Jump_IncorrectEnterName;}
  hInput_TabProject_NameApp->color(FL_BACKGROUND2_COLOR);
  hInput_TabProject_NameApp->redraw();

  INT LenPath=hOutput_TabProject_Dir->size()/*Без учёта 0 символа*/;
  //Проверка допустимой длины пути проекта
  if(hInput_TabProject_NameProject->size()+LenPath+MAX_PATH/2>MAX_PATH){
    LogMes(_Log_Typ_War,LibLL::cALL_Button_Ok,0,0,nILL::cCWSCGCP_ExceedLengthPathAndNameProject);
    return;}

  //Формирование директорий для нового проекта
  char szPath[MAX_PATH];
  memcpy(szPath,hOutput_TabProject_Dir->value(),LenPath);
  if(szPath[LenPath-1]!='\\')szPath[LenPath++]='\\';
  memcpy(&szPath[LenPath],hInput_TabProject_NameProject->value(),hInput_TabProject_NameProject->size())
  LenPath+=hInput_TabProject_NameProject->size()/*Без учёта 0 символа*/;
  szPath[LenPath++]='\\';
  szPath[LenPath]=0;

  //Проверка корректности ввода Проект->Директория. Создание директорий на файловой системе
  if(nAuxFunc::CreateDirectoryFile(szPath)){
    hOutput_TabProject_Dir->color(FL_RED);
    hOutput_TabProject_Dir->redraw();
    hTabs->value(hGroup_TabProject);
    LogMes(Log_CWSCGCP_ErrCreateDirectoryProject,LibLL::cALL_Button_Ok,0,0,
     cTypeAddCodeUI"%s \"%s\"",GetLastError(),LibLL::cALL_ErrCreateDirectory,szPath);
    return;}
  hOutput_TabProject_Dir->color(FL_BACKGROUND2_COLOR);
  hOutput_TabProject_Dir->redraw();
  //Проверка на указание типа карты Карта->Тип карты
  if(hChoice_TabMap_TypeMap->value()==0){
    //Если не выбрано из списка ничего
    hChoice_TabMap_TypeMap->textcolor(FL_RED);
    hChoice_TabMap_TypeMap->redraw();
    hTabs->value(hGroup_TabMap);
    return;}
  hChoice_TabMap_TypeMap->textcolor(FL_BACKGROUND2_COLOR);
  hChoice_TabMap_TypeMap->redraw();

  //Генерация файла проекта *.mge
  INT LenPathFileMGE=hInput_TabProject_NameProject->size()/*Без учёта 0 символа*/;
  memcpy(&szPath[LenPath],hInput_TabProject_NameProject->value(),LenPathFileMGE);
  LenPathFileMGE+=LenPath;//позиция будет установлена в начале начала мусора
  szPath[LenPathFileMGE]='.';
  szPath[LenPathFileMGE+1]='m';
  szPath[LenPathFileMGE+2]='g';
  szPath[LenPathFileMGE+3]='e';
  szPath[LenPathFileMGE+4]=0;
  //Проверка на существование файла
  { HANDLE hFile=CreateFileA(szPath,GENERIC_READ,NULL,NULL,OPEN_EXISTING,NULL,NULL);
    if(hFile==INVALID_HANDLE_VALUE){
      if(GetLastError()==ERROR_FILE_NOT_FOUND/*Файл не существует*/)goto Jump_OkCreateFileProject;
      LogMes(Log_CWSCGCP_ErrCreateFileProject,LibLL::cALL_Button_Ok,0,0,
       cTypeAddCodeUI"%s",GetLastError(),nILL::cCWSCGCP_ErrCreateProject);
      return;}
    CloseHandle(hFile);
    LogMes(_Log_Typ_Err,LibLL::cALL_Button_Ok,0,0,LibLL::cALL_ProjectWithThisNameAlreadyExists);
    return;}
  //Успешное выполнение
  Jump_OkCreateFileProject:
```

```
255
256   { //Формирование проекта
257     CGekBaseData_Cache_FA FileProject;
258     TVer tVer=Ver_Macro_Ver(hSpinner_TabProject_Ver_Basic->value(),
259                             hSpinner_TabProject_Ver_High->value(),
260                             hSpinner_TabProject_Ver_Low->value(),
261                             hSpinner_TabProject_Ver_Build->value());
262     TTypeProject tTypeProject=
263      (hChoice_TabSystem_OS->value()<<_TypeProjectM_Shift_OS)|
264      (hChoice_TabSystem_OSBit->value()<<_TypeProjectM_Shift_OSBit)|
265      (hChoice_TabSystem_GraphicsLibraryANDVersionShader->value()<<_TypeProjectM_Shift_GraphLibVerSha)|
266      (hChoice_TabSystem_TypeNetworkArchitecture->value()<<_TypeProjectM_Shift_TypeNetArch)|
267      (CurrentTypeMap<<_TypeProjectM_Shift_TypeMap)|
268      (CurrentTypeMap==0?(//Динамическая ограниченная карта высот
269 (TypeMap.DynamicLimitedHeightMap.hChoice_HeightMapResolution->value()<<_TypeProjectM_Shift_ResolutionHeightMap)
270        |(TypeMap.DynamicLimitedHeightMap.hChoice_MetersPerUnit->value()<<_TypeProjectM_Shift_MetersPerUnit)
271       ):NULL);
272     TProParam tProParam=
273       (hCheckButton_TabProject_Ver_Build_AutoIncrement->value()<<_ProParamF_Shift_AutoIncVerBuild);
274
275     //Сохранение данных
276     if(FileProject.WriteValueKeyV(nFileMGE::nSecHeader::cID_NameProject,
277         hInput_TabProject_NameProject->value(),hInput_TabProject_NameProject->size()+1/*Признак конца файла*/)||
278        FileProject.WriteValueKeyV(nFileMGE::nSecHeader::cID_NameApp,
279         hInput_TabProject_NameApp->value(),hInput_TabProject_NameApp->size()+1/*Признак конца файла*/)||
280        FileProject.WriteValueKeyLL(nFileMGE::nSecHeader::cID_VersionProject,tVer)|
281        FileProject.WriteValueKeyI(nFileMGE::nSecHeader::cID_TypeProject,tTypeProject)|
282        FileProject.WriteValueKeyI(nFileMGE::cID_ProParam,tProParam))
283        {LogMes(Log_CWSCGCP_ErrWriteFileProject,LibLL::cALL_Button_Ok,0,0,
284          cTypeAddCodeI"%s",FileProject.ErrWar(),nILL::cCWSCGCP_ErrCreateProject);
285         return;}
286
287     //Преписания пути
288     FileProject.SetPathFile(szPath,LenPathFileMGE+4);
289     if(FileProject.ErrWar()<CGekBaseData_Cache_FA::InfOk){
290       LogMes(Log_CWSCGCP_ErrSetPathFileProject,LibLL::cALL_Button_Ok,0,0,
291        cTypeAddCodeI"%s",FileProject.ErrWar(),nILL::cCWSCGCP_ErrCreateProject);
292       return;}
293
294     //Сохранение данных
295     FileProject.FlushDataToFile();
296     if(FileProject.ErrWar()<CGekBaseData_Cache_FA::InfOk){
297       LogMes(Log_CWSCGCP_ErrSaveFileProject,LibLL::cALL_Button_Ok,0,0,
298        cTypeAddCodeI"%s",FileProject.ErrWar(),nILL::cCWSCGCP_ErrCreateProject);
299       return;}}
300
301   //Запуск окна редактора
302   Fl_Group::current(NULL);//Корректно закрывает всю группу виджетов
303   if(OpenRunProject(szPath))return;
304
305   //Если успешно тогда текущее окно закрывается
306   delete static_cast<CWindowStartup*>(parent());
307 }
```

Додаток Г
Програмний код класу підсистеми менеджера даних

```
//###############################################################
//#                             CLASS                           #
//###############################################################
//Важное замечание!, использовать кеширование только в пределах открытого файла журналирования!

class CCacheMemory{
  private://Константы/Определения
    struct SHeaderData{                    //Заглавие данных
      static const uint32_t cShift_CounterLock=28u;         //28 //Смещение к битам начала счётчика блокировки
      static const uint32_t cMask_CounterLock=0xF0000000u;  //0x1111 0000 0000 0000  0000 0000 0000 0000 //Маска счётчика блокировки
      static const uint32_t cMask_LengthData=0xFFFFFFFu;    //0x0000 1111 1111 1111  1111 1111 1111 1111 //Маска длина данных
      uint32_t    CounterLock_LenData;    //Счётчик блокировки(4 бит)_Длина данных(28 бит)
      uint32_t    PosBeginData;           //Позиция начала данных
    };                                    //8 байт //static const int L=sizeof(SHeaderData);
    struct SFreeSpace{                    //Структура свободного пространства
      uint32_t    AmountFreeSpace;        //Количество свободного пространства
      uint32_t    PosBeginFreeSpace;      //Позиция начала свободного пространства
    };

  private://Константы/Определения
    static const uint32_t cMaxValueCounterLock=0xFu;//15 Максимальное значение счётчика блокировки
    static const uint32_t cMaxSizeData=0xFFFFFFFu;//268 435 455 Максимальный размер данный участка

  private://Переменные
    std::unordered_map<uint64_t,SHeaderData>ListHeaderData;//Список заглавий данных //x32=40, x64=80 байт static const int L1=sizeof(ListHeaderData);
    std::vector<SFreeSpace>vFreeSpace; //Список свободного пространства //x32=16, x64=32 байт static const int L2=sizeof(vFreeSpace);
    HANDLE hFile;//Дескриптор файла
    HANDLE hFileMap;//Дескриптор файла отображения
    char *pBeginAddress;//Адресс сопостовления отображения

  private://Функции
    inline const uint32_t TakeFreeSpace(const uint32_t &sizeSpace);//Взятие свободного пространства, возвращает позицию начала выделенного пространства
    void ReturnFreeSpace(const uint32_t &posBeginFreeSpace,const uint32_t &amountFreeSpace);//Возвращения свободной памяти(полос)

  public://Функции
    uint32_t Create(const char *szNameFileCache,uint32_t sizeCacheByte);//Создание кэш файла, возвращает выделеный размер что невсегда равен sizeCacheByte
    BOOL LockMemory(const uint64_t &id,const uint32_t &sizeByte);//Увеличивает счётчик блокировки памяти с резервированием указанного размера (Блокировка)
    BOOL LockMemory(const uint64_t &id);//Увеличивает счётчик блокировки памяти (Блокировка) выделеного участка, false - успех
    BOOL UnLockMemory(const uint64_t &id);//Уменьшает счётчик блокировки памяти (Разблокировка), false - успех
    void *PointBeginData(const uint64_t &id);//Указатель на начало данных, или NULL если ошибка
    uint32_t SizeDataByte(const uint64_t &id);//Размер данных в байтах
    BOOL Close(void);//Ракрытие файла кэширования и удаление данных кроме класса, в случаи ошибки будет предпринята попытка удалить всё что возможно
    CCacheMemory(void);//Конструктор по умолчанию
    ~CCacheMemory(void);//Деструктор по умолчанию
};//x32=68, x64=136 байт static const int L3=sizeof(CCacheMemory);

//###############################################################
//#                        EXTERN GLOBAL VAR                    #
//###############################################################
extern CCacheMemory g_CacheMem_System;//Системный файл кэширования, для разгрузки памяти от мало используемых данных

#endif
```

Додаток Д

Слайди комп'ютерної презентації

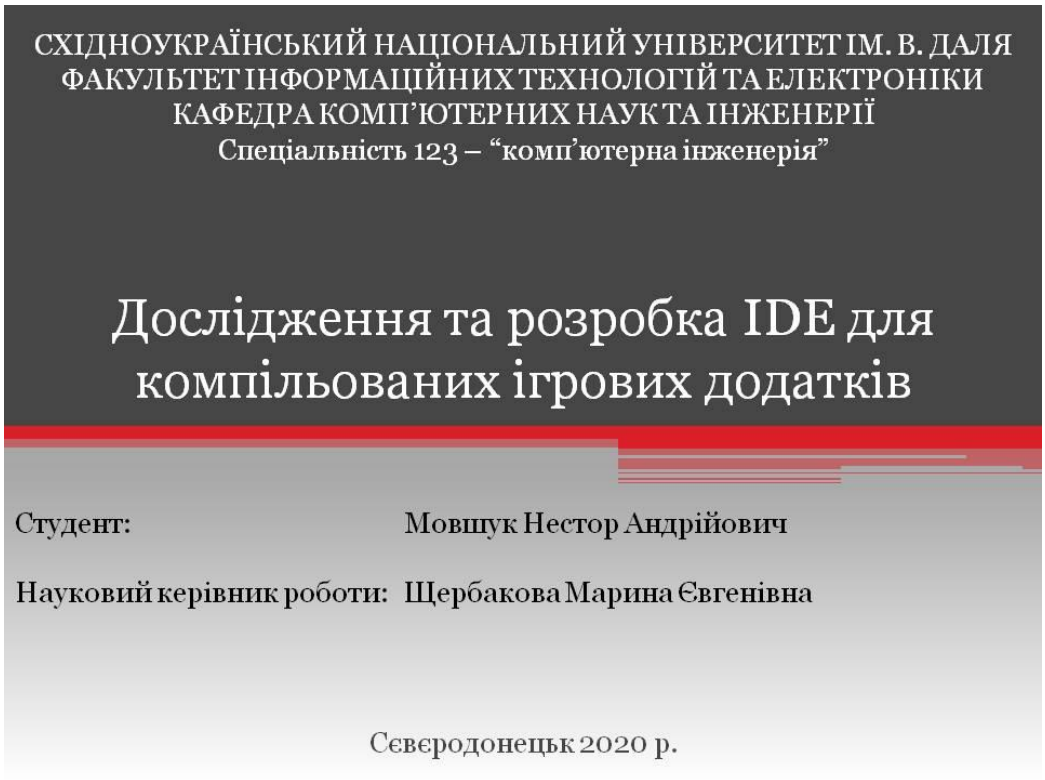

Рисунок Д.1 – Титульна сторінка

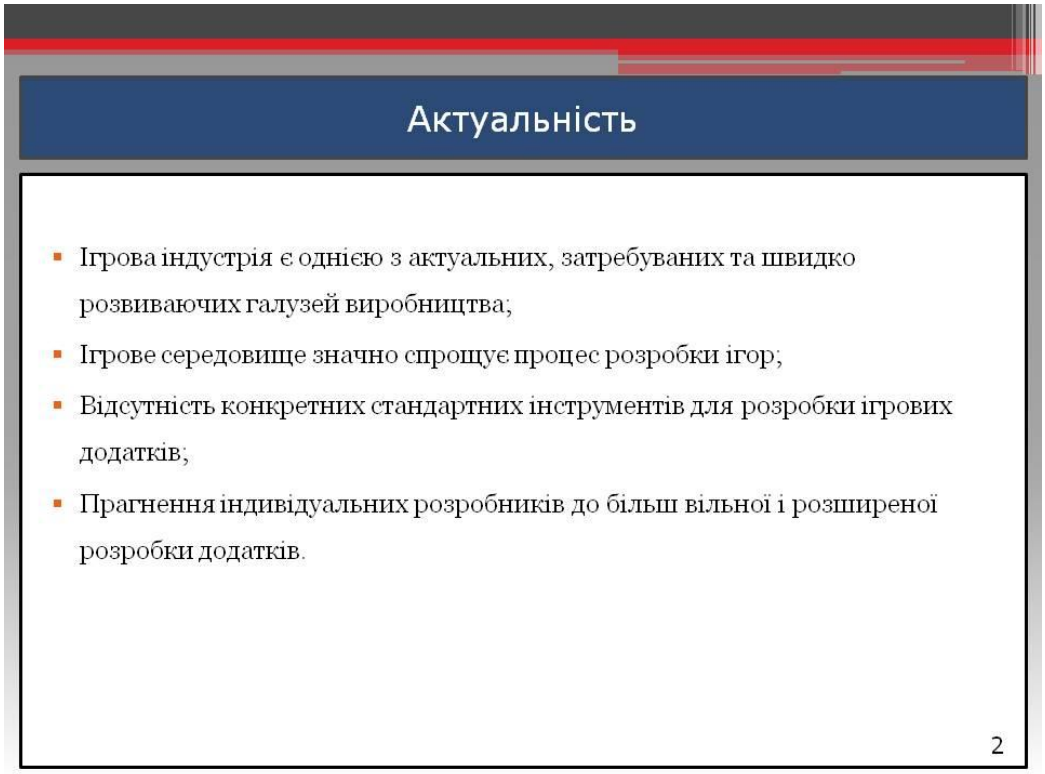

Рисунок Д.2 – Актуальність

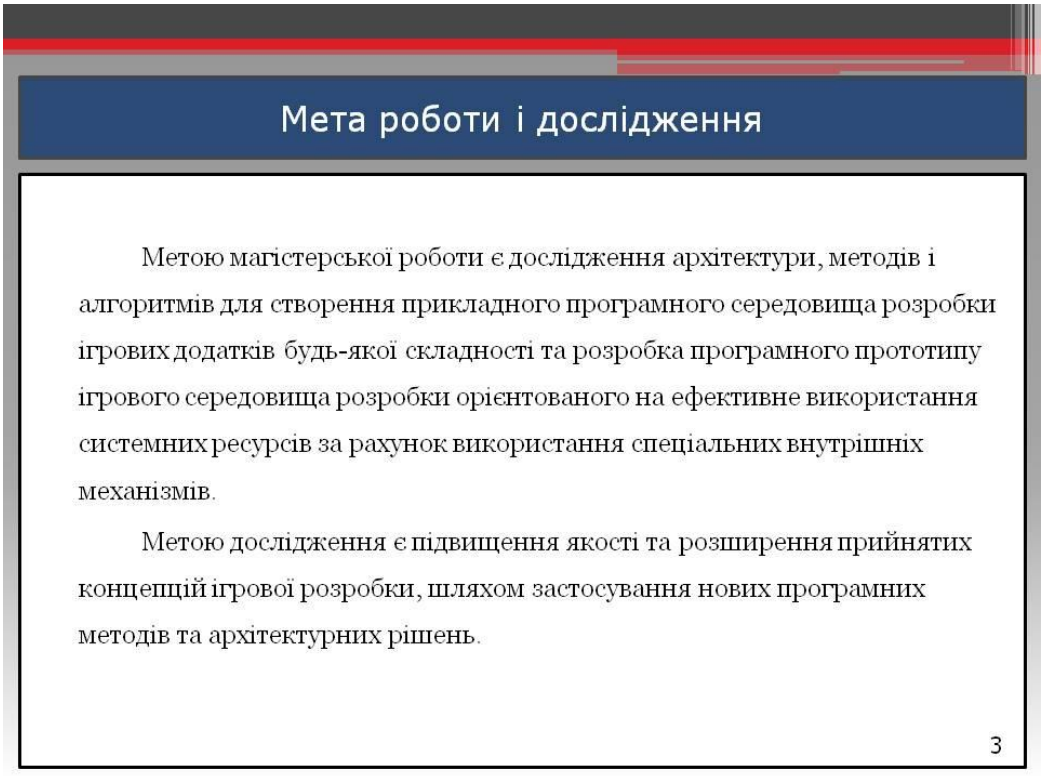

Рисунок Д.3 – Мета роботи і дослідження

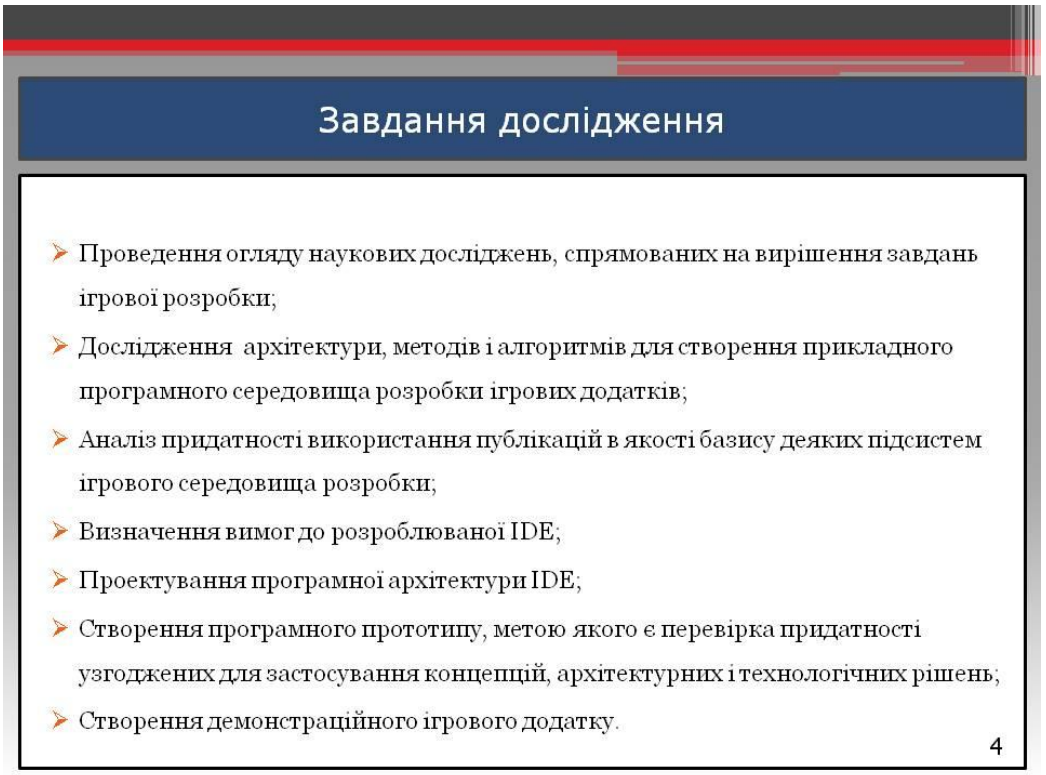

Рисунок Д.4 – Завдання дослідження

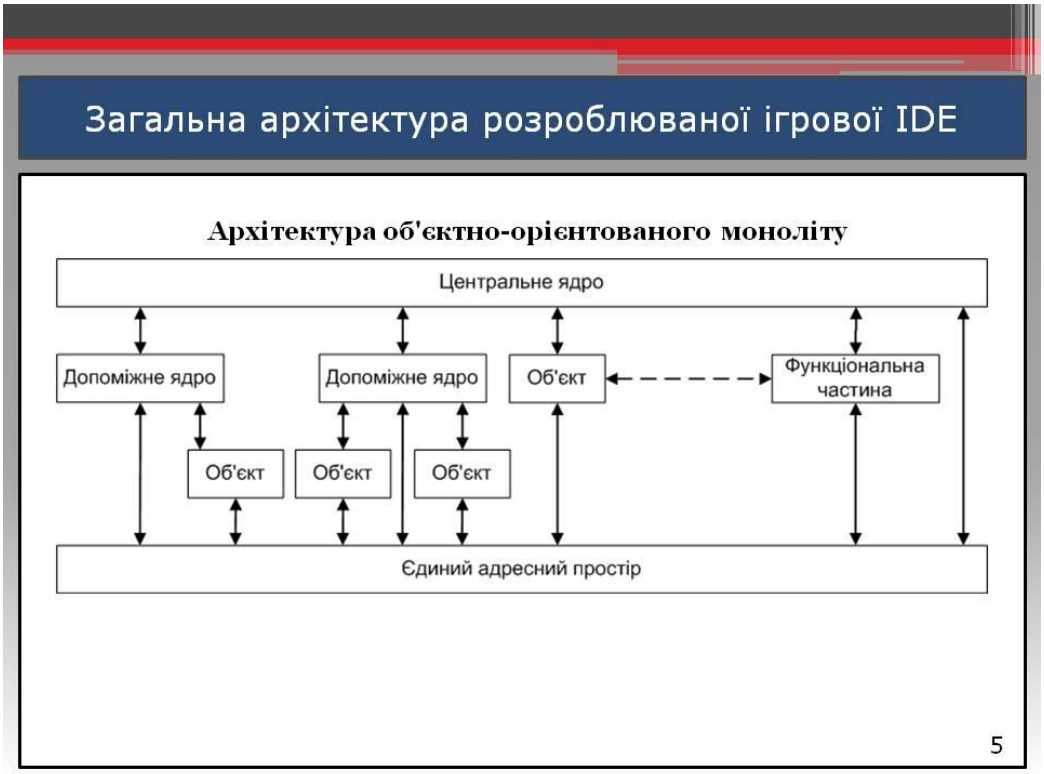

Рисунок Д.5 – Загальна архітектура розроблюваної ігрової IDE

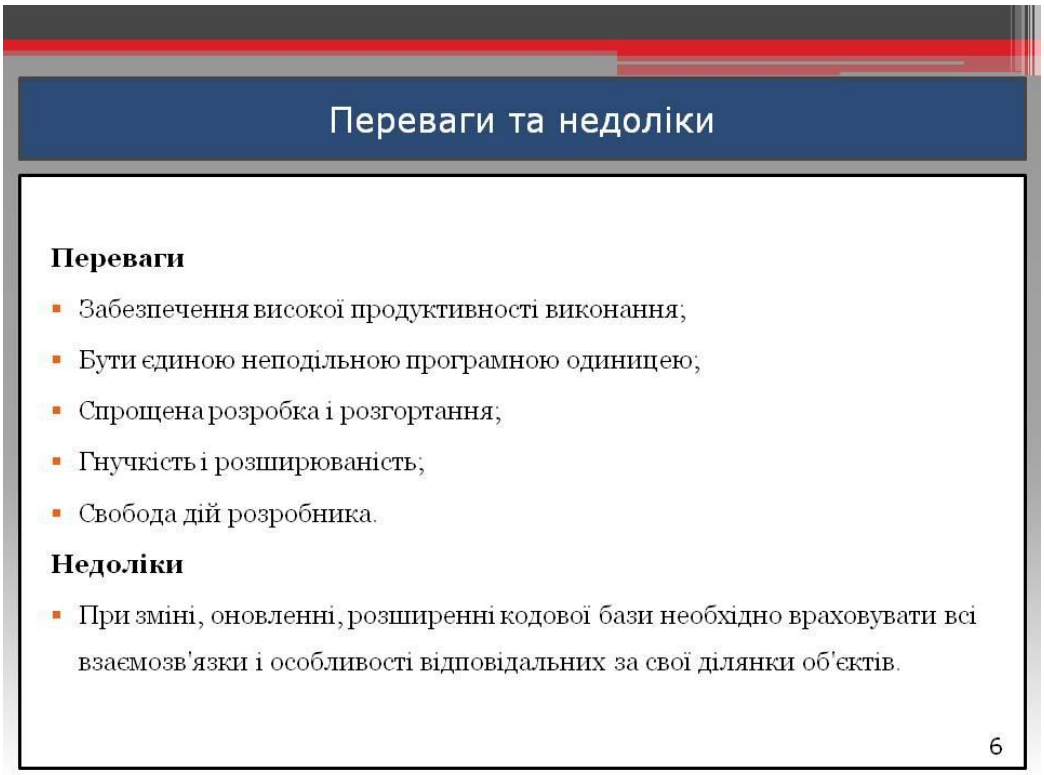

Рисунок Д.6 – Переваги та недоліки

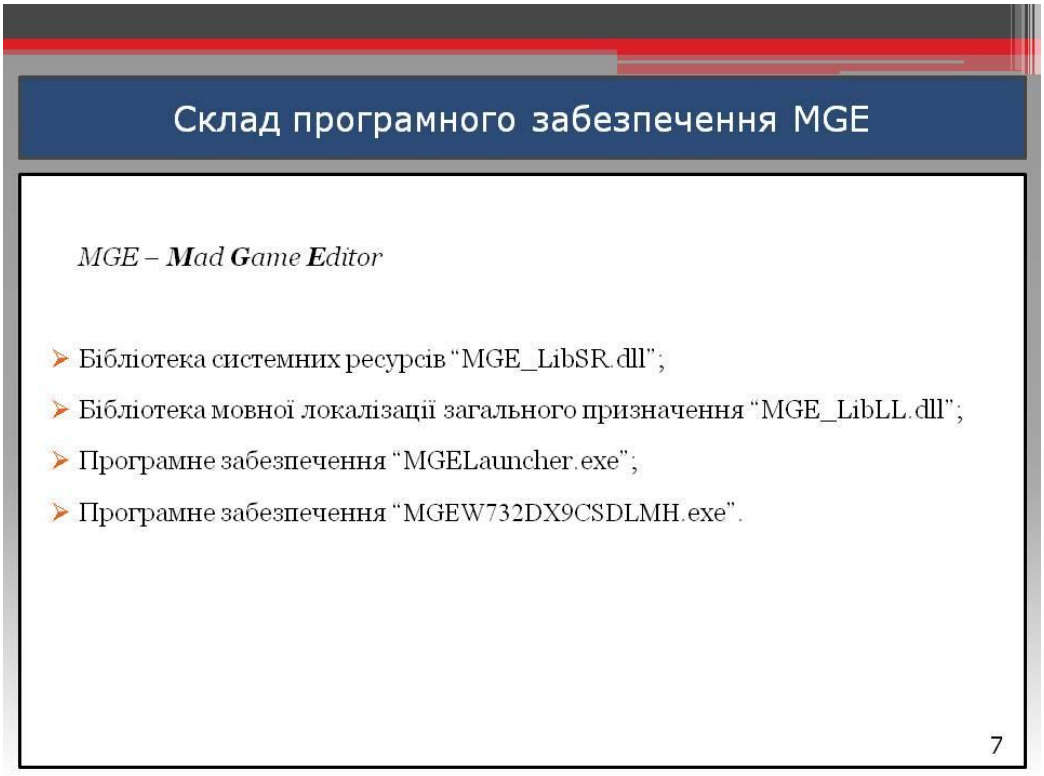

Рисунок Д.7 – Склад програмного забезпечення MGE

Рисунок Д.8 – Структура MGELauncher

Рисунок Д.9 – Структура типу шаблону проекту MGE

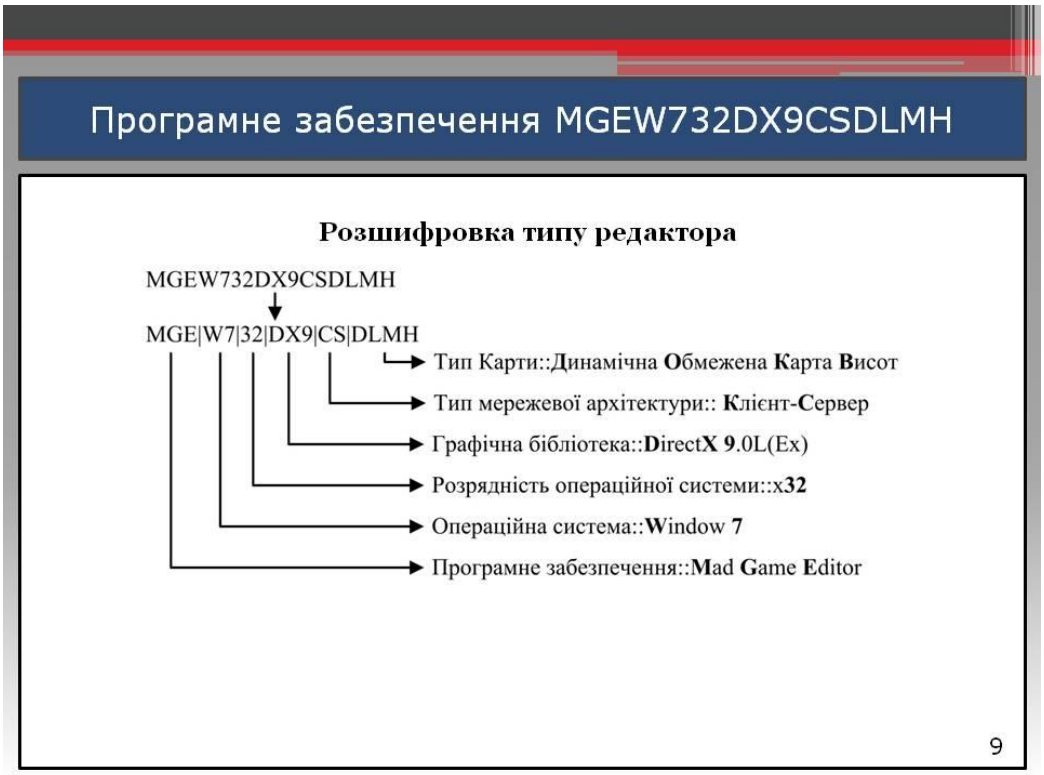

Рисунок Д.10 – Розшифровка типу редактора

Рисунок Д.11 – Архітектура прототипу редактора

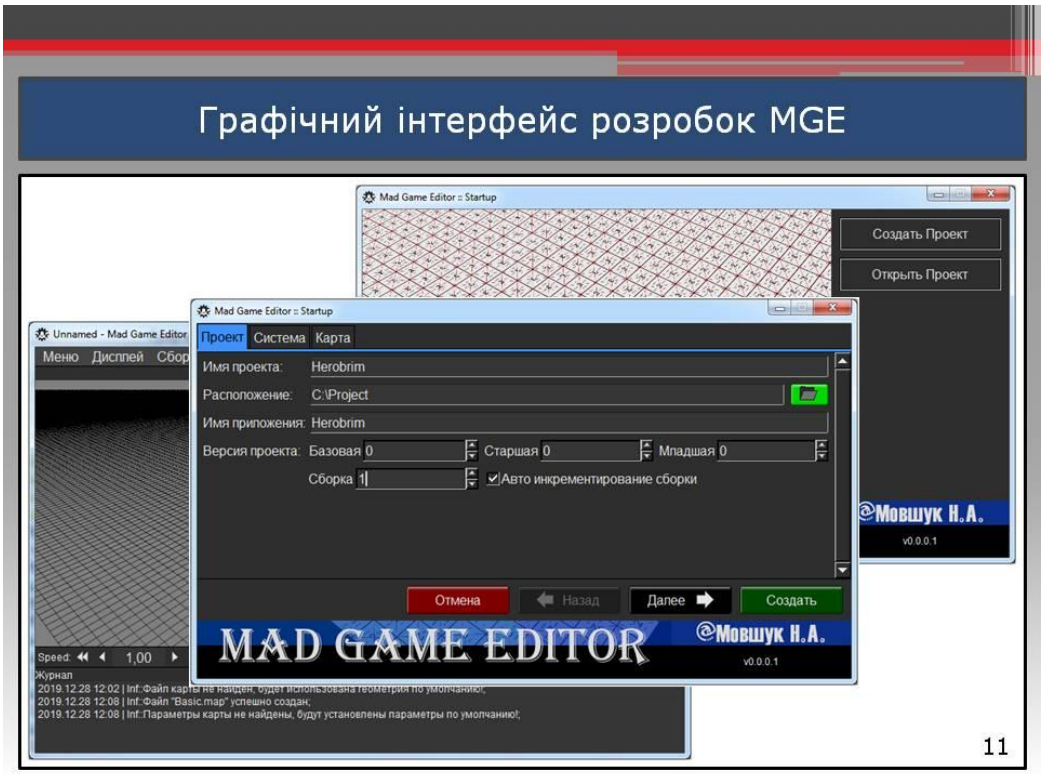

Рисунок Д.12 – Графічний інтерфейс розробок MGE

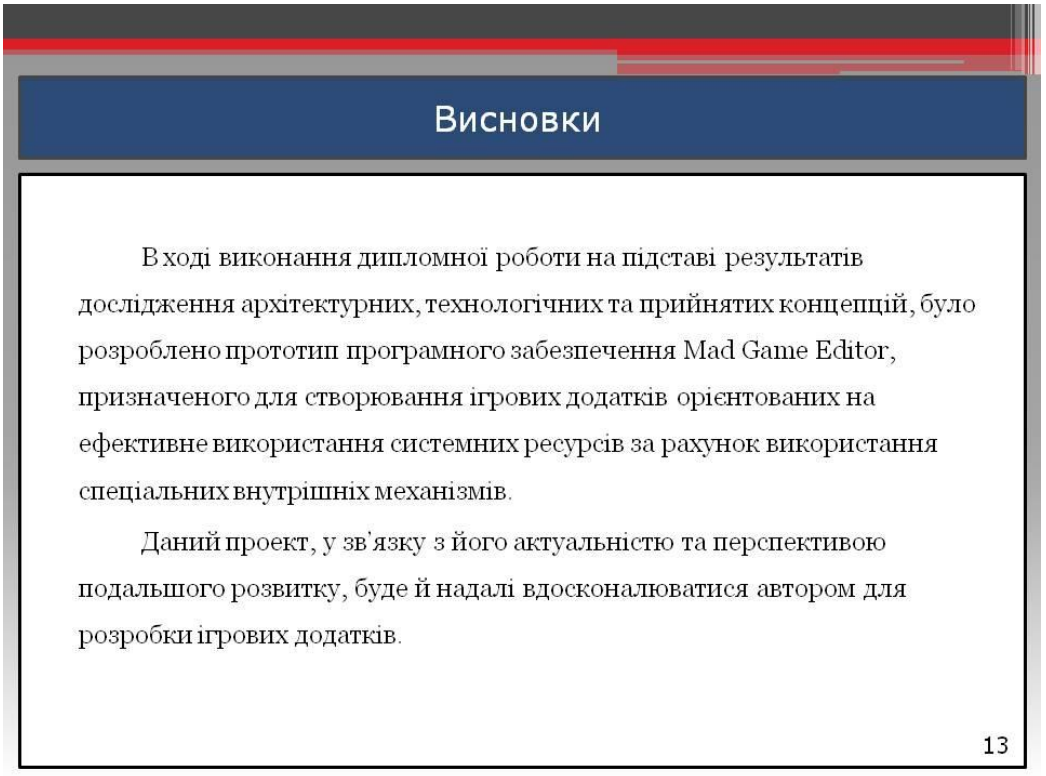

Рисунок Д.13 – Висновки