Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх»

Виконав: студент 2-го курсу, групи 1ПІ-20м спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Наумовський А.Ю.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ
Майданюк В.П.
(прізвище та ініціали)
«____» _____ 2021 р.

Опонент: к.т.н., доц. каф. КН
Крилик Л.В.
(прізвище та ініціали)
«____» _____ 2021 р.

Допущено до захисту

Завідувач кафедри ПЗ

д.т.н., проф. Романюк О.Н.

(прізвище та ініціали)

«____» _____ 2021 р.

ВНТУ - 2021 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 – Інформаційні технології
Спеціальність – 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

О.Н. Романюк

" 13 " вересня 2021 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Наумовському Андрію Юрійовичу

1. Тема роботи – Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх.
керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від " 24 " вересня 2021 року № 277
2. Строк подання студентом роботи 1 грудня
3. Вихідні дані до роботи: програмних код, який використовує функції та структури, які реалізовані у програмній бібліотеці, середовище розробки – Microsoft Visual Studio Code, мова розробки – C, стандарт C11, операційні системи та платформи – Windows, Linux, MacOS, Android та HTML5.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): аналіз завдання та вибір методу вирішення, розробка методу та моделей системи, розробка бібліотеки, тестування бібліотеки, економічні розрахунки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): мета, об'єкт та предмет дослідження; основні задачі; обгрунтування актуальності; порівняльний аналіз аналогів; метод та моделі системи; зображення архітектури; результати роботи; апробація і публікації, наукова новизна.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н., доц. каф. ПЗ Майданюк В.П.		
5	к.т.н., доц. каф. ЕПВМ Ратушняк О. Г.		

7. Дата видачі завдання 14 вересня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз, вибір та обґрунтування актуальності розробки	15.09.21 – 26.09.21	Вик.
2	Аналіз існуючих аналогів	27.09.21 – 03.10.21	Вик.
3	Аналіз інформаційного забезпечення	04.10.21 – 10.10.21	Вик.
4	Аналіз методів та засобів реалізації бібліотеки	11.10.21 – 19.10.21	Вик.
5	Розробка методів та моделей -реалізації бібліотеки	20.10.21 – 25.10.21	Вик.
6	Програмна реалізація модулів бібліотеки	26.10.21 – 18.11.21	Вик.
7	Тестування роботи бібліотеки	19.11.21 – 24.11.21	Вик.
8	Економічна частина	25.11.21- 30.11.21	Вик.

Студент _____ Наумовський А.Ю.

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Майданюк В.П.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.5

Наумовський А. Ю. Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп’ютерних іграх. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. 156 с. На укр. мові. Бібліогр.: 34 назв; рис.: 58; табл. 13.

Магістерська кваліфікаційна робота присвячена удосконаленню методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх. Під час виконання магістерської кваліфікаційної роботи було проаналізовано предметну область, проаналізовано основні аналоги, виявлено їх головні недоліки та переваги, на основі яких було обрано основні функції для бібліотеки. Також розроблено метод та модель реалізації графічного інтерфейсу та розглянуто їх застосування під час розробки комп’ютерних ігор. Особливість розроблених засобів полягає у компілюванні додатків для різних систем з однієї кодової бази, що дозволить пришвидшити розробку комп’ютерних ігор.

Розроблено основні модулі бібліотеки, що призначені для керування графічним контекстом, потоками, вікнами та взаємодією з користувачем. Також розроблені модулі для відображення графічних примітивів, зображень, текстур, матеріалів та шейдерів.

За допомогою розробленої програмної бібліотеки можна полегшити створення комп’ютерних ігор, адже бібліотека має велику кількість реалізованих функцій та структур, які необхідні при розробці графічних інтерфейсів користувача. Запропонована програмна бібліотека має зрозумілу інструкцію користувача, працює на більшості сучасних платформ.

Бібліотеку розроблено мовою C з використанням програмних засобів Microsoft Visual Studio Code.

Ключові слова: бібліотека, користувацький інтерфейс, графічний інтерфейс користувача, комп’ютерна графіка, комп’ютерна гра.

ABSTRACT

Naumovskyi A.Y. Improving methods of implementing user interfaces for their use in computer games. Master's thesis in specialty 121 - Software engineering. Vinnitsa: VNTU, 2021. – 156 p.

In Ukrainian language. Bibliographer: 34 titles; fig.: 58; tabl. 13.

The master's thesis is devoted to improvement of methods of implementation of user interfaces for their use in computer games. During the master's thesis the subject area was analyzed, the main analogues were analyzed, their main disadvantages and advantages were revealed, on the basis of which the main functions for the library were chosen. The method and model of graphic interface implementation have also been developed and their application in the development of computer games has been considered. A special feature of the developed tools is the compilation of applications for different systems from a single code base, which will speed up the development of computer games.

The main modules of the library have been developed, which are designed to manage the graphic context, flows, windows and interaction with the user. Modules have also been developed to display graphic primitives, images, textures, materials and shaders.

With the help of the developed software library you can facilitate the creation of computer games, because the library has a large number of implemented functions and structures that are needed when developing graphical user interfaces. The proposed software library has clear user instructions, works on most modern platforms.

The library is developed in C using Microsoft Visual Studio Code software.

Keywords: library, user interface, graphical user interface, computer graphics, computer game.

ЗМІСТ

ВСТУП 8

1 АНАЛІЗ РОЗВИТКУ БІБЛІОТЕК ДЛЯ РОЗРОБКИ ІНТЕРФЕЙСУ КОРИСТУВАЧА ПРИ СТВОРЕННІ КОМП'ЮТЕРНИХ ІГОР 12

1.1 Аналіз розвитку бібліотек для розробки інтерфейсу користувача .. 12

1.2 Порівняльний аналіз аналогів 17

1.3 Аналіз методів і засобів реалізації бібліотеки 20

1.4 Постановка задач дослідження 27

1.5 Висновки 29

2 РОЗРОБКА МЕТОДУ ТА МОДЕЛЕЙ БІБЛІОТЕКИ ДЛЯ РОЗРОБКИ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА 30

2.1 Аналіз інформаційного забезпечення бібліотеки 30

2.2 Розробка моделі графічного інтерфейсу 35

2.3 Розробка методу реалізація графічних інтерфейсів для кросплатформенного використання 38

2.4 Висновки 41

3 РОЗРОБКА БІБЛІОТЕКИ 43

3.1 Варіантний аналіз і обгрунтування вибору засобів розробки 43

3.2 Вибір середовища розробки 48

3.3 Розробка модулів бібліотеки 52

3.4 Розробка модуля для роботи з вікнами 58

3.5 Розробка модуля рендерингу 65

3.6 Розробка модуля відображення графічних примітивів 71

3.7 Розробка модуля роботи з зображеннями, текстурами, матеріалами та шейдерами 79

3.8 Висновки 89

4 ТЕСТУВАННЯ БІБЛІОТЕКИ 90
4.1 Аналіз методів і засобів тестування 90
4.2 Тестування бібліотеки 93
4.3 Розробка інструкції користувача 97
4.4 Висновки 99
5 ЕКОНОМІЧНА ЧАСТИНА 100
5.1 Оцінювання комерційного потенціалу розробки 100
5.2 Прогнозування витрат на виконання науково-дослідної роботи ... 106
5.3 Розрахунок економічної ефективності науково-технічної розробки ..
..... 112
5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності 113
5.5 Висновки до економічного розділу 115
ВИСНОВКИ 117
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 119
Додаток А. Технічне завдання 124
Додаток Б. Протокол перевірки роботи на плагіат 129
Додаток В. Лістинг коду 131
Додаток Г. Ілюстративна частина 154

ВСТУП

Обґрунтування вибору теми дослідження. Інтерфейс користувача – одна з найважливіших частин практично будь-якого програмного комплексу, що забезпечує взаємодію людини з комп’ютером. За останні десятиліття було досягнуто значного прогресу як у розширенні можливостей введення та виведення інформації, так і в методах розробки інтерфейсів користувача. Розробники інтерфейсів почали активно переходити від програмування взаємодії з пристроями вводу/виводу в машинних кодах для кожної окремої програми до більш високих рівнів абстракції (використання драйверів до спеціальних бібліотек компонент інтерфейсу користувача Motif, MFC, Qt, GTKht.h).

Інтерфейс користувача в іграх полягає в тому, щоб дозволити гравцеві виконувати завдання в ігровому світі шляхом прямого введення або дії на Heads Up Display (HUD). Багато варіацій інтерфейсу надають користувачеві змогу якомога краще налаштувати взаємодію с програмним додатком за допомогою кнопок, меню, повзунків та інших додаткових віджетів. Мета інтерфейсу користувача завжди полягає в покращенні користувацького досвіду та спрощенні взаємодії.

У розглянутій літературі [19] було оглянуто методи реалізації графічних інтерфейсів користувача для різних платформ, проте не оглянуто можливості створення графічних інтерфейсів для різних платформ з однієї кодової бази без адаптації під окрему систему.

Тому актуальною є розробка бібліотеки, яка б удосконалила методи реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета та задачі дослідження. Метою роботи є удосконалення методів реалізації користувацьких інтерфейсів шляхом розробки спеціалізованих бібліотек та адаптації їх до використання при створенні комп'ютерних ігор, що дозволить пришвидшити розробку подібних програмних продуктів.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

а) аналіз існуючих рішень;

б) аналіз особливостей реалізації графічного інтерфейсу користувача в комп’ютерних іграх;

в) розробка методу та моделі реалізації компонентів бібліотеки;

г) розробка програмних модулів бібліотеки;

д) тестування програмної бібліотеки.

Об’єктом дослідження є процес розробки графічного інтерфейсу користувача в комп’ютерній грі з використання спеціалізованих бібліотек.

Предметом дослідження є методи та засоби розробки програмної бібліотеки.

Методи дослідження*:* Для реалізації поставлених задач були використані: методи створення програмних бібліотек – для розробки бібліотеки графічного інтерфейсу, методи графічної реалізації об’єктів – для розробки графічних елементів користувацького інтерфейсу, методи графічної візуалізації об’єктів – для реалізації графічних об’єктів, методи аналізу інформаційного забезпечення – для вводу та обробки даних, які були отримані від користувача, методи теорії інформації – для аналізу інформаційного контенту програми.

Наукова новизна одержаних результатів:

а) подальшого розвитку дістав метод реалізації графічних інтерфейсів у комп'ютерних іграх на різних платформах, який, на відміну від існуючих, дозволить компілювати додатки для різних систем з однієї кодової бази, що дозволить пришвидшити розробку комп'ютерних ігор;

б) подальшого розвитку дістала модель бібліотеки для графічних інтерфейсів, яка, на відміну від існуючих, акумулює функції керування вікнами, взаємодії з користувачем, відображення графічних примітивів, зображень, текстур та шейдерів, що дозволяє використання створених компонент в процесі компілювання та інтеграції інтерфейсу комп'ютерних ігор забезпечувати кросплафторменну реалізацію і збільшити швидкість обробки даних.

Практичне значення одержаних результатів. Розроблена програмна бібліотека, функції та структури можуть використовуватися в комп'ютерних іграї і дозволяють спростити процес розробки графічного інтерфейсу користувача під час розробки комп'ютерних ігор для різних платформ.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У роботах, що опубліковані у співавторстві, здобувачу належить: розроблений метод релізації графічних інтерфейсів у комп'ютерних іграх та модель, що описує структуру бібліотеки у [15], а також визначення функціоналу бібліотеки та склад її модулей, комп'ютерна програма для прискореного зафарбування Гуро [17], контекстно-адаптивна модель представлення зображень [34].

Апробація матеріалів магістерської кваліфікаційної роботи. Результати роботи доповідалися на:

а) міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (Вінниця 2021);

б) міжнародній науково-практичній конференції молодих вчених та студентів «Молодь у світі сучасних технологій» (Херсон 2020).

Публікації. Результати роботи були опубліковано в 4 наукових працях, у тому числі 3 статті у матеріалах конференцій [15, 22, 34], 1 свідоцтво про реєстрацію авторського права на комп’ютерну програму [17].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 34 найменувань, 4 додатки. Робота містить 43 ілюстрацій, 13 таблиць.

1 АНАЛІЗ РОЗВИТКУ БІБЛІОТЕК ДЛЯ РОЗРОБКИ ІНТЕРФЕЙСУ КОРИСТУВАЧА ПРИ СТВОРЕННІ КОМП'ЮТЕРНИХ ІГОР

1.1 Аналіз розвитку бібліотек для розробки інтерфейсу користувача

Графічний інтерфейс користувача (ГІК, англ. GUI, Graphical user interface) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації [5].

Виконання дій у ГІК – це безпосередня маніпуляція з графічними елементами. Окрім комп'ютерів, GUI використовують у мобільних пристроях, таких, як мобільні телефони, планшети, електронні книги, портативні медіапрогравачі тощо. Термін ГІК зазвичай не вживають стосовно інтерфейсів з низькою роздільною здатністю. Наприклад, у відеоіграх використовують інтерфейс HUD. HUD (Head-Up Display) зазвичай є прозорим інтерфейсом, який накладається на гру [12]. Він повідомляє гравцеві важливу інформацію, без потреби відволікатися від геймплею шляхом відкриття меню або вспливаючими вікнами [10].

З винаходом сучасних комп'ютерів ранні програми та пов'язані з ними інтерфейси були спрямовані на програміста. Отже, використання комп'ютерів було обмежено тими, хто мав здатність розробляти складні серії інструкцій мовою, спеціально розробленою для комп'ютера. Програмісти генерували інструкції за допомогою кабелів, розташованих на дротяних платах, призначених для переміщення електронних значень з регістра в регістр за допомогою складних суматорів і віднімувачів, доки математичні функції, які

вони логічно побудували, не забезпечуватимуть маніпулювання даними в бажаній формі. Часто результати або просто відображалися на цифровій панелі, або виводилися у форматі звичайного паперу.

З появою технології перфокарт і стандартизованих компіляторів інструкції, розроблені програмістом, стали портативними. Програмісти змогли розробити стандартизовані програми, які легко використовуються, але зрозумілі лише комп'ютерним фахівцям зі знанням мов програмування високого рівня. Розвиток цих мов програмування дозволив послідовно створювати стандартизовані звіти у форматі, корисному для користувачів.

Оскільки була вимога забезпечити безпосередню взаємодію між користувачем і комп'ютером, були розроблені нові інтерфейси реального часу. В кінцевому підсумку це призвело до того, що дисплей з електронно-променевою трубкою (ЕПТ) забезпечував можливість монохромного відображення символів із підключеною клавіатурою, що дозволяє оператору взаємодіяти з процесором. Програмісти, знайомі з додатками, задовольнялися короткими підказками та блимаючими курсорами, щоб повідомити їх про те, що комп'ютер очікує введення. Оскільки цей підхід був неочевидним для тих, хто не знайомий з конкретною програмою, потреба в розробці більш узгоджених наборів відповідей і підказок стала вимогою в дизайні інтерфейсу.

Оскільки була вимога забезпечити безпосередню взаємодію між користувачем і комп'ютером, були розроблені нові інтерфейси реального часу. В кінцевому підсумку це призвело до того, що дисплей з електронно-променевою трубкою (ЕПТ) забезпечував можливість монохромного відображення символів із підключеною клавіатурою, що дозволяє оператору взаємодіяти з процесором. Програмісти, знайомі з додатками, задовольнялися

короткими підказками та блимаючими курсорами, щоб повідомити їх про те, що комп’ютер очікує введення. Оскільки цей підхід був неочевидним для тих, хто не знайомий з конкретною програмою, потреба в розробці більш узгоджених наборів відповідей і підказок стала вимогою в дизайні інтерфейсу [1].

З широким поширенням персонального комп’ютера (ПК) все більше користувачів використовували додатки для різноманітних обчислювальних вимог. Основними програмами, доступними для ПК, були обробка текстів, обчислення електронних таблиць і деяка проста обробка бази даних. Впровадження настільного комп’ютера корпорації Apple, керованого піктограмами, різко змінило взаємодію людини з комп’ютером. Тепер користувачі мали графічний інтерфейс для взаємодії. З випуском операційної системи Microsoft Windows для користувачів стали доступні сотні нових програм, починаючи від комп’ютерних ігор і закінчуючи графічними презентаційними програмами і візуальними інструментами комп’ютерного програмування.

Сьогодні інтерфейс у стилі Windows широко прийнято серед користувачів, а його зовнішній вигляд є галузевим стандартом для розробки додатків [4]. Графічний інтерфейс, хоча і є значним покращенням у порівнянні з інтерфейсами на основі символів, все ще вимагає підходу, орієнтованого на користувача під час його розвитку.

Інтерфейс користувача зазвичай належить до інструментів, за допомогою яких користувач взаємодіє з програмним забезпеченням або системою за допомогою графічного зв’язку, керування голосом або введення руху. Як поводиться система, як користувач взаємодіє з системою, як виглядають

значки, кнопки та текст, а макет визначається за допомогою дизайну інтерфейсу користувача, який створюється на основі принципів дизайну інтерфейсу користувача, щоб забезпечити безперебійну роботу користувача [6]. У той час як дизайн UI зосереджується на конкретних об’єктах системи, дизайн User Experience (UX) [2] визначає загальний досвід користувача; відчуття, які користувач отримує під час взаємодії та потік всередині системи, і часто базується на користувацьких тестах та ітераціях для досягнення високої зручності використання.

Структуру інтерфейсу можна розглядати як кластер мікровзаємодій, і його можна пояснити як взаємодії, про які користувач зазвичай не думає – їх легко зрозуміти, їх можна виконувати швидко і щоразу отримувати той самий очікуваний ефект, тому вони ефективно виконують задумане. завдання користувача. Прикладом поширеної взаємодії з інтерфейсом може бути лайк зображення в соціальних мережах або вимкнення звуку телефону; дизайн того, як користувач взаємодіє з кнопкою, як вона має форму та колір, де вона розміщена та як вона поводить себе під час взаємодії з усіма, залежить від того, чи розуміє користувач, використовувати взаємодію чи ні. Однак деталі дизайну взаємодії не є причиною того, що користувач купив телефон або провів час у додатку, а деталі – це те, на чому побудований продукт і що сприяє загальному відчуттю продукту. Загалом можна сказати, що взаємодії, які виконують те, чого від них очікує користувач [7], і створюють відчуття задоволення, посилять відчуття цінності в продукті, але не будуть активно сприйматися користувачем. Однак взаємодія, яка виконує щось несподіване у плоівгяггі з тим, що задумав користувач, вважається поганою і може викликати розчарування та

роздратування, тому вони частіше визнаються користувачем і знижують відчуття цінності продукту.

Взаємодія гравця з грою повинна бути передбачуваними, а це означає, що гравець повинен мати можливість очікувати, що станеться при використанні введення. Контроль ніколи не повинен відчувати себе випадковим або давати випадковий результат, якого гравець не може очікувати. Це може бути важко, оскільки різні гравці очікують різних речей, але можна змінити очікування гравців за допомогою зображення та графіки. Важливо дати гравцеві зрозуміти, чому щось сталося, щоб гравець міг відтворити дію знову, це можна зробити за допомогою зворотного зв’язку, такого як анімація, ефекти частинок, звук тощо. Передбачуваність також дозволяє гравцеві зрозуміти негативні наслідки і дає їм знання про те, як їх уникнути [11]. Це йде рука об руку з принципом миттєвої реакції, коли важливо дати гравцеві зворотній зв’язок, як тільки він виконує взаємодію, щоб змусити гру реагувати. Сама подія може тривати тривалий час, але гравець повинен отримати відгук про те, що щось з’являється менше ніж за секунду після активації події. Новизна в грі важлива, щоб не зробити введення та взаємодії нудними; введення має бути передбачуваним, але все одно цікавим, щоб не повторюватися протягом кількох годин. Цього можна уникнути, додавши варіації в анімацію, середовище або інші механіки під час гри. Іншим рішенням є додавання в гру фізики, яка унеможливлює те, щоб одне й те саме відбувалося однаково кілька разів, що дозволяє гравцеві експериментувати з тією ж взаємодією довше [9].

Хоча система інтерфейсу користувача має на меті існувати і діяти без несподіванок, легко зрозуміти і не бути складною, домінуюча тенденція полягає в тому, що ігрові системи мають на меті бути веселими, іноді

складними, важкодоступними та з несподіванок [18]. Однак у порівнянні відповідних принципів є багато подібності; сама взаємодія повинна прояснити користувачеві, який ефект вона матиме, важливо, щоб взаємодії відчувалися чуйними через негайний ефект, який можна посилити за допомогою аудіо, візуальних зображень і відчуття, взаємодії повинні бути розроблені так, щоб вказувати, для чого воно використовується і як використовувати його, і система повинна відчувати себе задовільною у використанні.

1.2 Порівняльний аналіз аналогів

У якості аналогів для порівняльного аналізу були обрані: «SFML», «AGKSharp», «LibGDX».

«SFML» – надає простий інтерфейс до різних компонентів комп’ютера, щоб полегшити розробку ігор та мультимедійних додатків. Він складається з п’яти модулів:

а) System – керування часом та потоками, він є обов'язковим, оскільки всі модулі залежать від нього.;

б) Window – керування вікнами та взаємодією з користувачем.;

в) Graphics – відображення графічних примітивів, зображень, текстур, матеріалів та шейдерів;

г) Audio – надає інтерфейс для керування звуком;

д) Network – надає функціонал для розробки програм, що використовують мережу.

На рисунку 1.1 зображено офіційний сайт SFML.

Рисунок 1.1 – Офіційний сайт SFML

«AGKSharp» – надає можливість писати програми AppGameKit на мові C# та VisualBasic. Також надає шаблон, який дозволяє розробляти додатки AppGameKit з WinForms. Логотип AGKSharp зображено на рисунку 1.2.

Рисунок 1.2 – Логотип AGKSharp

«LibGDX» – це безкоштовна бібліотека для розробки ігор із відкритим вихідним кодом, написана мовою програмування Java. Це дозволяє розробляти настільні та мобільні ігри, використовуючи ту саму кодову базу. На рисунку 1.3 зображено офіційний сайт LibGDX.

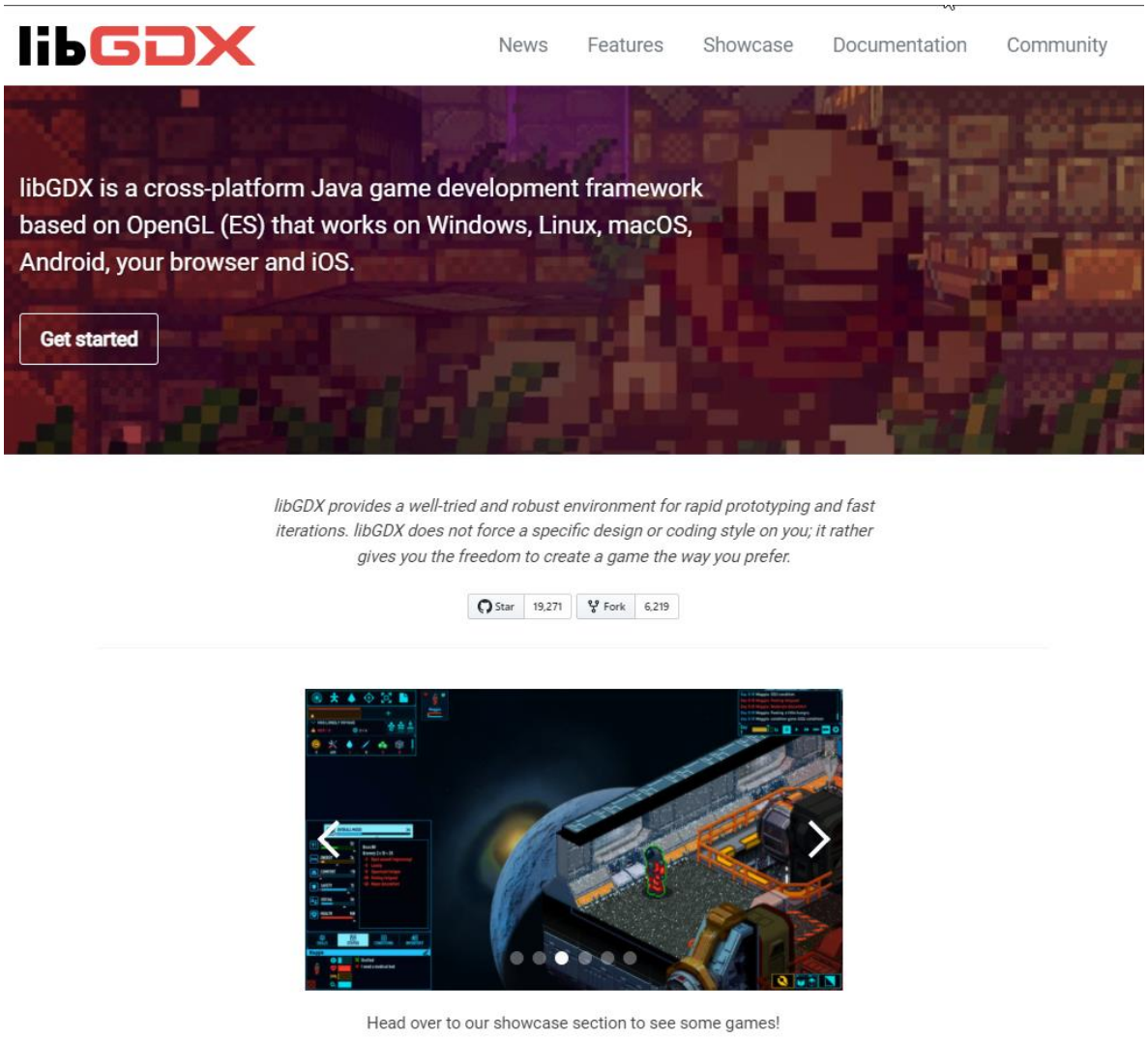


Рисунок 1.3 – Офіційний сайт LibGDX

У таблиці 1.1 наведена порівняльна характеристика аналогів із розроблюваним програмним продуктом.

Таблиця 1.1 – Порівняльний аналіз з аналогами

Критерій	«SFML»	«AGKSharp»	«LibGDX»	«temp»
Швидкодія	+	-	-	+
Мультиплат-форменість	+	-	-	+
Компілювання додатку для мобільних пристроїв	-	-	+	+
Компілювання додатку для Web (HTML5)	-	-	-	+

Таким чином, розроблювана програмна бібліотека для реалізації графічного інтерфейсу є кращим за існуючі аналоги за низкою критеріїв, що обумовлює актуальність і доцільність його розробки.

1.3 Аналіз методів і засобів реалізації бібліотеки

Бібліотека (англ. library) – збірник підпрограм або об’єктів, використовуваних для розробки програмного забезпечення. Вони можуть включати дані конфігурації, документацію, довідкові дані, шаблони повідомлень, попередньо написаний код і підпрограми, класи, значення або специфікації типу.

Інакше кажучи бібліотека – це колекція реалізацій поведінки, написаних мовою програмування, яка викликає певну поведінку через певний інтерфейс.

Тому для написання програми високого рівня, розробники можуть використовувати бібліотеку для визову системних викликів, замість того щоб самостійно програмувати код для використання цих системних викликів. Такий підхід до реалізації функціональності дозволить повторно використовувати написаний код для використання декількома програмами. Програма викликає надану бібліотекою поведінкуза допомогою механізмами мови. Наприклад, у мові С, яка відноситься до простих імперативних мов, поведінка в реалізованих у функціях бібліотеки викликається за допомогою звичайного виклику функції С.

Бібліотечний код організовано для використання в кількох не пов’язаних програмах, але код, який є частиною однієї програми, організований для використання в одній програмі. Ця різниця може бути ієрархічною в міру зростання програми, наприклад, багатомільйонна програма. У цьому випадку може існувати внутрішня бібліотека, яка повторно використовується незалежною частиною великої програми. Відмінною особливістю є те, що бібліотека організована так, що її можна повторно використовувати окремою програмою або підпрограмою. Користувачеві потрібно знати лише інтерфейс, а не внутрішні деталі бібліотеки.

Цінність бібліотеки полягає у повторному використанні стандартизованих програмних елементів. Коли програма викликає бібліотеку, вона отримує поведінку, реалізовану в цій бібліотеці, без необхідності реалізовувати її самостійно. Бібліотека сприяє обміну базовим кодом і полегшує розповсюдження коду.

Поведінка, реалізована бібліотекою, може бути пов’язана з програмою-тригером на різних етапах життєвого циклу програми. Якщо доступ до коду

бібліотеки здійснюється під час компіляції програми, що викликає, бібліотека називається статичною бібліотекою.

Щоб покращити модульність і можливість повторного використання, зазвичай, часто використовувані функції включаються в бібліотеки. Традиційна бібліотека — це статична бібліотека, яка являє собою набір об’єднаних об’єктних файлів подібних типів. Статична бібліотека зберігається на диску як архів («.a» файл). Існує індекс, який забезпечує асоціацію символів з об’єктними файлами, які надають визначення символів у статичній бібліотеці [13]. Цей індекс буде використовуватися редактором посилань (ld) для прискорення пошуку символів. Для створення цього індексу можна використовувати команду ranlib, вона еквівалентна передачі параметра «-s» команді «ar».

Коли у зв’язуванні бере участь статична бібліотека, за замовчуванням редактор посилань виконує вибіркове вилучення об’єктних файлів у статичну бібліотеку. Він проходить через бібліотеку ітераційно і витягує лише об’єктні файли, які містять визначення символів невизначених символів у внутрішній таблиці символів редактора посилань. Редактор посилань об’єднує всі розділи програмних даних, такі як розділи «.text», «.data» і «.bss» вибраних об’єктних файлів, щоб утворити нові розділи в наступному вихідному об’єктному файлі. Розділи інформації про посилання, такі як розділи «.symtab», «.synstr» тощо, будуть використовуватися редактором посилань для зміни інших розділів. Ці інформаційні розділи також використовуються для створення нових розділів інформації про посилання в наступному вихідному об’єктному файлі. На рисунку 1.4 зображено процес зв’язування об’єктних файлів зі статичною бібліотекою, а також створення виконуваного файлу.

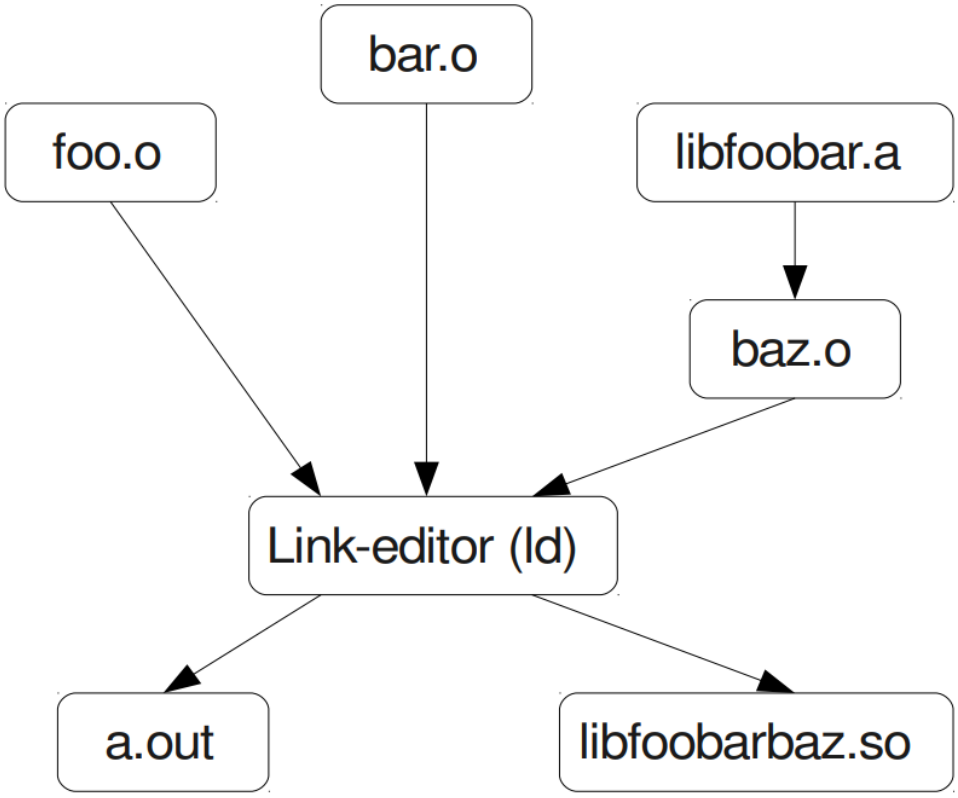


Рисунок 1.4 – Процес зв’язування об’єктних файлів зі статичною бібліотекою

У якості альтернативи до статичної бібліотеки можна створити виконуваний файл програми, яка викликається, і поширити його откремо від реалізації бібліотеки. Поведінка бібліотеки стає активною як частина процесу запуску виконання або в середині виконання після того, як виконуваний файл був викликаний для виконання [31]. У цьому випадку бібліотека називається динамічною бібліотекою (вона завантажується під час виконання). Динамічна бібліотека може бути використана та зв'язана під час підготовки коду до виконання компонувальником. Крім того, в середині виконання програма може явно запросити завантаження модуля.

Для вирішення проблем із втратою ресурсів та обслуговуванням статичних бібліотек операційні системи використовують динамічні бібліотеки. Ідея динамічних бібліотек полягає в тому, щоб мати лише одну копію часто використовуваних функцій у фізичній пам’яті, іншні компоненти, сервіси чи програми мають змогу використовувати функції, яка надає бібліотека замість того, щоб мати власні копії.

Динамічна бібліотека – це неподільна одиниця, яка створюється з одного або кількох переміщуваних об’єктних файлів. Коли динамічна бібліотека бере участь у зв’язуванні, на відміну від зв’язування зі статичною бібліотекою, розділи програмних даних динамічної бібліотеки та більшість розділів інформації про посилання не будуть використовуватися редактором посилань. Однак для запису динамічної бібліотеки як залежності вихідного об’єктного файлу буде створено деяку іншу інформацію. Ця помітна різниця між зв’язуванням зі статичними та динамічними бібліотеками вказує на те, що розмір виконуваних файлів може бути значно зменшений шляхом зв’язування зі динамічними бібліотеками, таким чином заощаджується дисковий простір. Інша основна відмінність полягає в тому, що фактичний процес зв'язування динамічної бібліотеки виконується під час виконання компонувальником під час виконання (ld-linux.so), який сам є динамічною бібліотекою. Компонувальник часу виконання відповідає за завантаження динамічних бібліотек і прив’язування символів до програм до того, як програми отримають контроль від операційних систем для виконання. Компонувальник під час виконання також може бути викликаний під час процесу виконання програми, щоб завантажити додаткові динамічні бібліотеки за допомогою функції dlopen. Символи, надані цими динамічними бібліотеками, можна зв’язати за

допомогою функції Symbols dlsym, і цей процес часто називають динамічним завантаженням. Після того, як динамічна бібліотека завантажується у фізичну пам’ять, її функції та змінні можуть спільно використовуватися всіма процесами, які посилаються на неї, тому обсяг використовуваної фізичної пам’яті може бути зменшений.

Щоб створити ефективну динамічну бібліотеку, компілятору слід передати спеціальну опцію (-fPIC/-fpic). На рисунку 1.5 показано процес зв'язування об'єктних файлів із динамічною бібліотекою, а також створення виконуваних файлів.

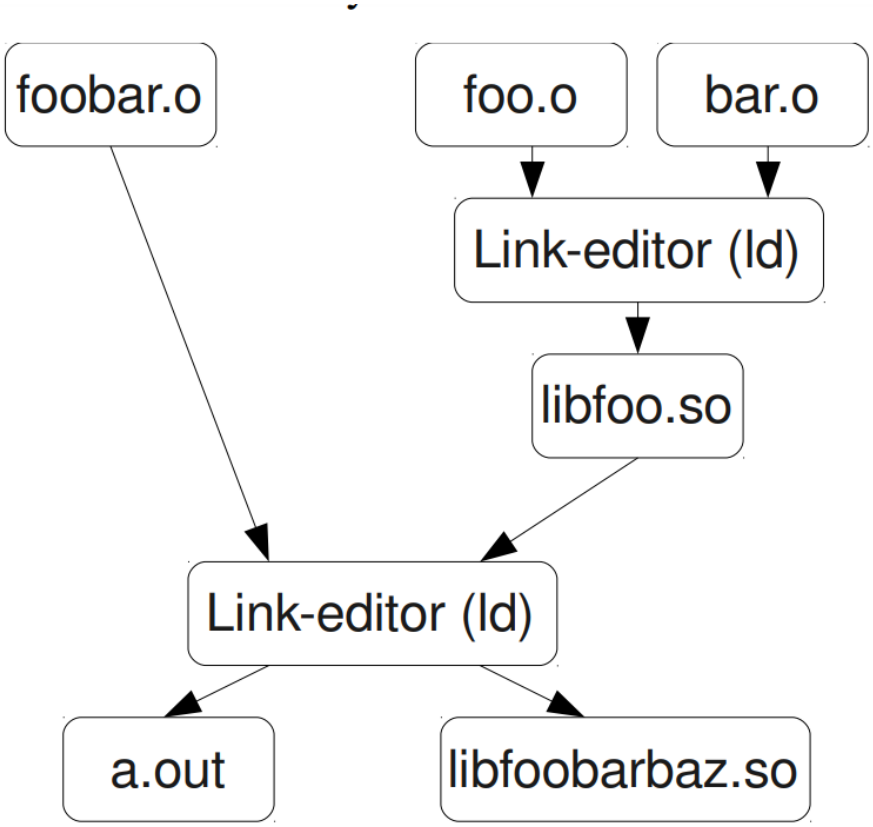


Рисунок 1.5 – Процес зв'язування об'єктних файлів із динамічною бібліотекою

Більшість скомпільованих мов мають стандартну бібліотеку, хоча розробники моють змогу створювати власні бібліотеки, у яких буду зберігати функції та структури, які найбільш важливі для них. Більшість сучасних програмних систем надають бібліотеки, які реалізують більшість системних служб. Такі бібліотеки організовують послуги, необхідні для сучасних програм. Тому більшість коду, що використовується в сучасних програмах, міститься в цих системних бібліотеках.

1.4 Постановка задач дослідження

Розробити метод реалізації графічних інтерфейсів у комп’ютерних іграх на різних платформах, який компілює додатки для різних систем з однієї кодової бази, дозволить пришвидшити розробку комп’ютерних ігор.

Розробити модель бібліотеки для графічних інтерфейсів, яка акумулює функції керування вікнами, взаємодії з користувачем, відображення графічних примітивів, зображень, текстур та шейдерів, що дозволяє використання створених компонент в процесі компілювання та інтеграції інтерфейсу комп’ютерних ігор забезпечувати кросплафтормену реалізацію та збільшити швидкість обробки даних.

Під час розробки графічного інтерфейсу користувача перед розробниками програмного забезпечення постає ціла низка проблем у реалізації взаємодії з користувачем:

а) розробнику доводиться досліджувати велику кількість різних бібліотек, призначених для роботи у різних середовищах (пристроях, операційних системах);

б) потреба в узгодженні та комплексному використанні кількох різних бібліотек;

в) потреба у розробці однієї системи для кількох різних середовищ, що потребує повторного здійснення робіт;

Саме на вирішення вказаних проблем направлено удосконалення методів реалізації користувацьких інтерфейсів, що є актуальним для їх використання в комп'ютерних іграх. Для досягнення цієї мети було поставлено такі задачі, які потрібно зробити:

а) проаналізувати предметну область;

б) проаналізувати існуючі рішення;

в) проаналізувати існуючі методи та засоби реалізації програмних бібліотек;

г) розробити моделі та загальну структуру розроблювальної програмної бібліотеки;

д) розробити алгоритми функцій, які міститиме бібліотека;

е) реалізувати функції та організувати їх у вигляді структурних блоків архітектури системи;

ж) провести тестування усіх створених структурних блоків архітектури системи;

з) створити документацію до розробленої програмної бібліотеки.

Технічне завдання на розробку наведено у Додатку А.

1.5 Висновки

Проаналізовано стан питання та предметну область. Визначено основні типи інтерфейсу користувача при створенні комп’ютерних ігор, вимоги до них та особливості їх створення. Визначено принципи роботи програмних бібліотек, вимоги до них та особливості їх створення.

Проведено порівняльний аналіз існуючих аналогів, а саме «SFML», «AGKSharp», «LibGDX». Визначенно основні недоліки аналогів: швидкість роботи та кросплатформенність, і переваги розроблювальної системи перед ними. Було підтверджено актуальність розробки.

Проведено аналіз методів та засобів реалізації поставленої задачі. Проаналізовано особливості створення графічного інтерфейсу користувача для різних платформ. Було прийнято рішення створити програмну бібліотеку, що підходить для мультиплатформенної розробки.

Було визначено перелік задач для виконання.

2 РОЗРОБКА МЕТОДУ ТА МОДЕЛЕЙ БІБЛІОТЕКИ ДЛЯ РОЗРОБКИ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА

2.1 Аналіз інформаційного забезпечення бібліотеки

Формат виконуваного файлу та зв’язування спочатку був розроблений та опублікований UNIX System Laboratories (USL) як частина бінарного інтерфейсу додатків (ABI). На цей момент ELF є стандартним форматом об’єктного файлу в Linux. За різними способами використання об’єктні файли ELF можна розділити на три основні типи:

а) файл, який містить код і дані, придатні для зв’язування з іншими об’єктними файлами для створення виконуваного або спільного об’єктного файлу. Компіляція вихідного файлу створює файл, який можна переміщувати. Цей файл має розширення «.o», «.dll», т.д.;

б) виконуваний файл містить програму, придатну для виконання. Цей файл має розширення «.exe» або може його не мати у UNIX, Linux або GNU - подібних системах;

в) динамічних об’єктний файл містить код і дані, придатні для зв’язування в двох контекстах. По-перше, редактор посилань може обробити його з іншими переміщуваними та спільними об’єктними файлами, щоб створити інший об’єктний файл. По-друге, динамічний компонувальник поєднує його з виконуваним файлом та іншими спільними об’єктами для створення образу процесу. Розширення файлу спільного об’єктного файлу – «.so» [27].

Модель програмування на основі бібліотеки повинна зосередитися на кількох аспектах розробки, які відрізняються від точки зору розробки на основі

мови. Програмування не можна просто розглядати як діяльність, пов’язану зі створенням коду на основі вивченого набору мовних конструкцій. Така модель програмування на основі бібліотеки містить такі аспекти, як:

а) множинність – програмісти використовують кілька бібліотек з різних джерел, які не мають спільної системи виробництва і не є повністю DSL (доменною мовою) для створення складніших подій;

б) інтегрований – бібліотеки можуть бути навіть не з однієї мови програмування;

в) еволюція – бібліотеки є еволюційними елементами програмування, які продовжують розвиватися протягом усього часу використання. Перший публічний випуск може бути альфа-релізом або, швидше за все, бета-релізом. Бібліотеки також повинні оновлюватися, щоб включати нові потреби користувачів і дотримуватися технологічних стандартів і рамок, щоб залишатися конкурентоспроможними.

г) легасі – бібліотеки будуть супроводжуватися спадщиною старих версій і застарілими джерелами інформації, що ускладнить кардинальні зміни. Якщо концепція дизайну змінюється в потоці версій, старі рішення залишаться на задньому плані, що ускладнює розуміння та розробку за допомогою бібліотеки [28];

д) дебати – бібліотечне програмування включає технічні дебати, тобто участь розвитку бібліотеки в спільноті користувачів. Потреби та бажання користувачів бібліотеки потрапляють у списки розсилки, дискусійні форуми, а також іноді як запити на функції [3]. Звіти про помилки також належать до цієї категорії. Дизайн бібліотеки повинен сприяти цій діяльності, забезпечуючи належну підтримку для участі та водночас витягуючи цінність з цієї дискусії.

У порівнянні з мовним програмуванням, бібліотечне програмування включає зміни в технічному середовищі і, отже, постійне включення нового та зміненого матеріалу. Спілкування стає більш важливим аспектом програмування. По суті, бібліотеки слід розглядати як послуги громади, що полегшують програмування прикладних програм. Користувачі бібліотеки з усього світу соціально пов’язані через бібліотеку. Активне товариство із соціальною структурою користувачів і досвідчених користувачів, де відбувається обмін знаннями та фактичним кодом, має значення для популярності бібліотек.

Загалом бібліотеки вважаються не графічними інструментами. Однак бібліотеки мають користувацькі інтерфейси, за допомогою яких користувачі мають доступ до своїх функцій. Онлайн-документація є найпоширенішим прикладом інтерфейсу бібліотеки, за допомогою якого програмісти дізнаються про синтаксис і семантику бібліотек. Дискусійні форуми та списки розсилки в Інтернеті є іншими прикладами підтримки комунікаційних завдань, які мають користувачі бібліотеки та розробники бібліотек щодо бібліотеки. Крім того, часто надаються системи обробки помилок для обробки запитів на помилки та функцій щодо бібліотек. Крім того, функціональність завершення коду в середовищах розробки можна розглядати як графічні інтерфейси, за допомогою яких програмісти можуть отримати доступ до функціональних можливостей бібліотеки шляхом прямого маніпулювання.

Існує багато різноманітних бібліотечних служб, розподілених у великій кількості інтерфейсів. Немає загального інтерфейсу користувача, за допомогою якого користувач виконує пов’язані з бібліотекою завдання. Постачання загального інтерфейсу користувача, звичайно, полегшить використання

бібліотеки за рахунок автоматизації кроків, зроблених вручну, щоб перетинати межі інтерфейсу користувача, але це також може протидіяти ролі бібліотек в інших інструментах, таких як редактор, і робити занадто великий акцент на бібліотеці. Зробимо основне порівняння між інтерфейсами бібліотеки та інтерфейсом звичайного текстового процесора (в цьому випадку приклади взяті з MS Word). Текстовий процесор, який працює як довідкова документація, вимагатиме, щоб ви прочитали про функцію збереження в посібнику та знайшли правильний синтаксис для введення в підказці для введення команди. Доступні функції описані в текстовому форматі і повинні бути скопійовані вручну у вихідний код. Копіювання навіть заважає використання гіперпосилань у документації, оскільки гіперпосилання важко вибрати як текст (покажчик постійно зміщується на руку, яку не можна використовувати для копіювання). З іншого боку, функція завершення коду підтримує копіювання та правильність синтаксису. Крім того, усі команди представлені з однаковою релевантністю, тобто без змістовної структури. Час, необхідний для пошуку та введення команди відкрити файл, дорівнює чи навіть довший, ніж час, необхідний для відкриття веб-сторінки в офісі в Інтернеті. Команди організовані без урахування ймовірності використання. Команди перераховані в алфавітному порядку, що в текстовому процесорі розмістило б автотекст угорі, а скасування — у нижній частині системи меню. Порівняння зі звичайним інтерфейсом інтерфейсу настільних додатків виявляє відсутність орієнтованого на користувача дизайну в комунікації з бібліотекою. Проте, справедливості кажучи, також важливо зазначити, що користувачі бібліотеки – це певна марка користувачів програмного забезпечення, які, можливо, потребують підтримки іншого стилю, ніж звичайні користувачі.

Користувачі бібліотек, як правило, є програмістами і, таким чином, відрізняються від традиційних користувачів досить чітко в деяких аспектах:

а) розуміння коду – користувачі бібліотеки мають або повинні вільно володіти мовою програмування, на якій базується бібліотека;

б) консольний інтерфейс – користувачі бібліотек звикли до інтерфейсу мов програмування в виді консолі і, можливо, більше звикли саме до такого стилю інтерфейсу користувача, ніж до моделей взаємодії GUI;

в) досвідчені користувачі комп'ютерів – багато користувачів бібліотеки є досвідченими користувачами комп’ютерів, достатньо компетентними, щоб використовувати гнучкі відкриті системи, які надають їм достатньо місця для побудови індивідуально розроблених систем підтримки. Занадто жорсткі конструкції можуть стати перешкодою в їх робочих процесах.

Хоча для користувачів бібліотеки потрібен дизайн взаємодії, орієнтований на використання, це не обов’язково означає тип дизайну, який підходить для звичайного користувача настільного комп’ютера [33].

Представлена тут модель бібліотечної комунікації розглядає бібліотечне спілкування як діяльність спільноти, де незалежні розробники та користувачі з різними цілями використовують і розвивають бібліотеки.

Бібліотечне спілкування – це акт спілкування як частина бібліотечного програмування. У порівнянні з мовами, бібліотеки набагато більші, безперервно змінюються, рідко повні, менш добре перевірені та менш формально визначені набори конструкцій. Технічна еволюція під час програмування проектів, спадок опублікованої та непослідовної документації, а також дебати є елементами бібліотечної комунікації. Програмісти використовують кілька бібліотек від незалежних розробників, практично не

співпрацюючи в процесі комунікації. Розрізняти розробників і користувачів не так актуально, оскільки розробники часто є користувачами інших бібліотек. Члени бібліотечних спільнот – це просто програмісти з різною технічною компетенцією. Важливіше розрізняти ролі, такі як екзаменатори та студенти, щоб допомогти програмісту виконувати завдання [32].

Зрештою, програмісти мають таку ж потребу в дизайні, орієнтованому на використання, в інтерфейсах бібліотеки, як і користувачі в цілому. Однак програмісти також відрізняються від традиційних користувачів. Вони вільно володіють кодом, використовуються для інтерфейсів на основі команд і є досвідченими користувачами комп’ютерів. Підтримка описаної тут моделі програмування з умовами програміста як користувача та мережі взаємопов’язаних бібліотек і програмістів, на мою думку, є важливим кроком у розвитку глобальної інженерії програмного забезпечення. У розділі 4 обговорюється дизайн електронних мережевих інструментів і процесів у зв'язку з цією моделлю.

Отже, було вирішено обрати статичний варіант бібліотеки, оскільки його розробка є швидшою та дозволяє отримати робочий продукт, який можна буде вдосконалити надалі.

2.2 Розробка моделі графічного інтерфейсу

При розробці програмного забезпечення, як правило, компонент розглядається як частина програмного забезпечення або системи. У цьому випадку компонент інтерфейсу користувача належить до візуальних частин системи і може бути, серед іншого, кнопками, полями введення, текстом і значками. Компоненти діють як будівельні блоки і зазвичай проектуються і

розробляються з урахуванням можливості повторного використання та масштабованості.

Щоб забезпечити можливість повторного використання та масштабованості та полегшити використання компонентів, їх часто збирають і роблять доступними в одному місці, яке називається бібліотекою інтерфейсу користувача, з документацією та прикладами для використання розробниками.

Бібліотеки інтерфейсу користувача складаються з надійного набору компонентів, доступних для використання [16]. Ці компоненти зазвичай спроектовані та розроблені для забезпечення найкращої швидкодії, мають високий показник згуртованості, низький показник зв’язності, добре протестовані та мають хорошу читабельність.

Оскільки метрика згуртованості підказує принцип єдиної відповідальності, згуртованість у сучасних структурах і компонентах інтерфейсу користувача визначається шляхом оцінки його обов’язків та кількості спільних властивостей. Наприклад, компонент, який отримує властивості, який виявляється невикористаним або відповідає за обчислення двох незалежних значень, свідчить про низьку згуртованість. Визначення рівня зв'язку проводиться шляхом вивчення кількості залежностей. Наприклад, компонент, необхідний для зміни іншого компонента, щоб змінити себе, вказує на високу зв’язність. Іншими словами, зміна однієї частини системи не повинна впливати на інші. Висока згуртованість та низька зв’язність часто вказують на хорошу читабельність; однак, можна визначити читабельність за узгодженістю між вихідним кодом і коментарями. Іншим показником читабельності є визначення того, як читачі, крім автора, самі розуміють і сприймають код.

Для реалізації необхідної функціональності для розробки гри було сформовано список модулів, які будуть містити структури та функції для:

а) роботи з вікнами, графічним контекстом та який буде забезпечувати мультиплатформеність;

б) рендерингу елементів інтерфейсу на дисплей;

в) введення та дотику – цей модуль має надавати інформацію про введені користувачем кнопки клавіатури, миші або дотики на тачскріні;

г) таймеру та керування потоками;

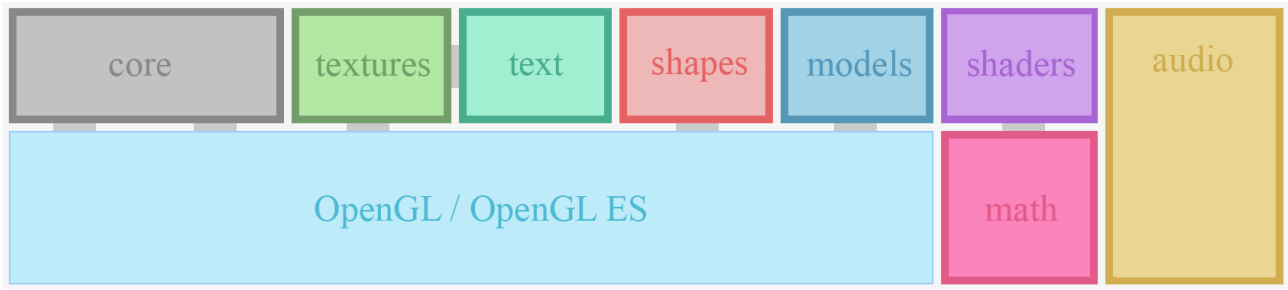
д) камери;

е) відображення графічних примітивів;

ж) роботи з зображеннями, текстурами, матеріалами та шейдерами;

з) роботи зі звуком.

На рисунку 2.1 зображено загальну архітектуру програмної бібліотеки, що розробляється.



Рисунку 2.1 – Узагальнена модель графічного інтерфейсу з урахуванням архітектури бібліотеки

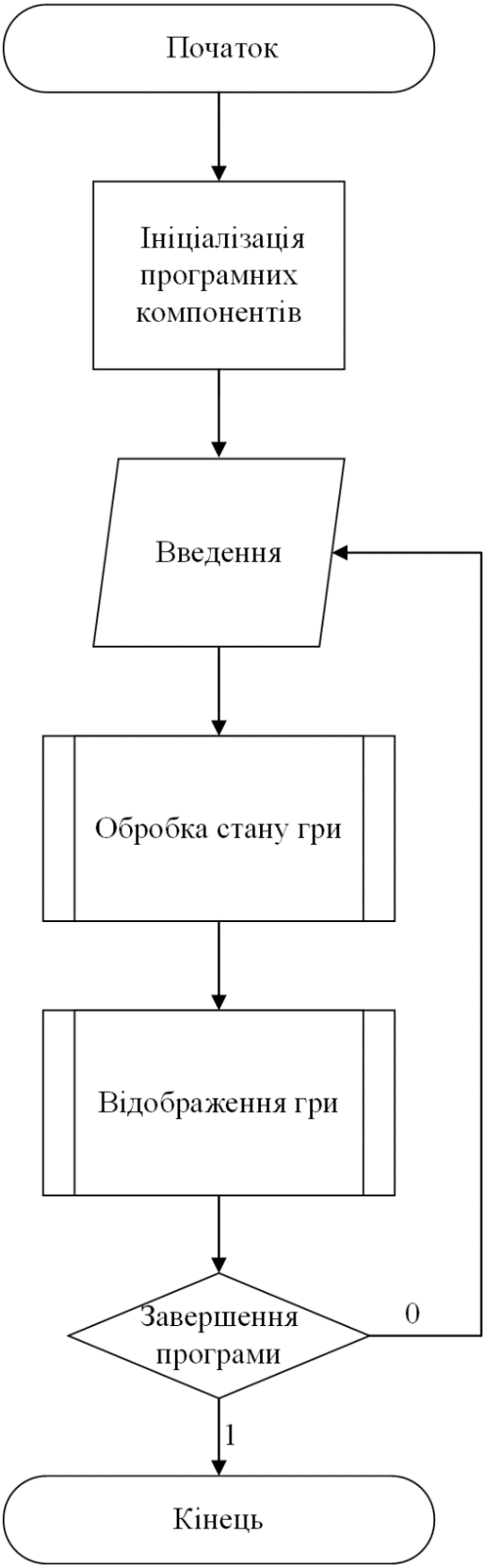
Таким чином, розроблювана програмна бібліотека має містити модулі, які забезпечать загальну структуру гри, всі інструменти взаємодії користувача зі грою.

2.3 Розробка методу реалізація графічних інтерфейсів для кросплатформенного використання

Ігровий цикл – це шаблон проектування, який керує процесом введення, внутрішнім оновленням статусу програми, відображенням та іншими процесами гри. Він є основним процесом усіх потоків рендерингу гри та реалізований у всіх іграх [25].

Ігровий цикл виконується безперервно під час гри. Кожен такт циклу обробляє введення користувача без блокування, оновлює стан гри та відображає елементи графічного інтерфейсу [29]. Оскільки кожен цикл циклу оновлює дисплей, важливо, щоб цикл виконувався з регулярними інтервалами, тому цей шаблон гарантує, що ігровий час буде відбуватися з однаковою швидкістю на різних налаштуваннях обладнання.

При запуску гри відбувається ініціалізація програмних компонентів та запуск ігрового циклу. До ініціалізацій програмних компонентів можна віднести десеріалізація збереженого стану компонентів програми, створення вікна та графічного контексту, ініціалізація пристроїв введення та виведення. Після цього програма зчитує введення користувача, оновлює стан програми та відправляє запити на відображення елементів гри. Узагальнений алгоритм роботи гри зображений на рисунку 2.2.

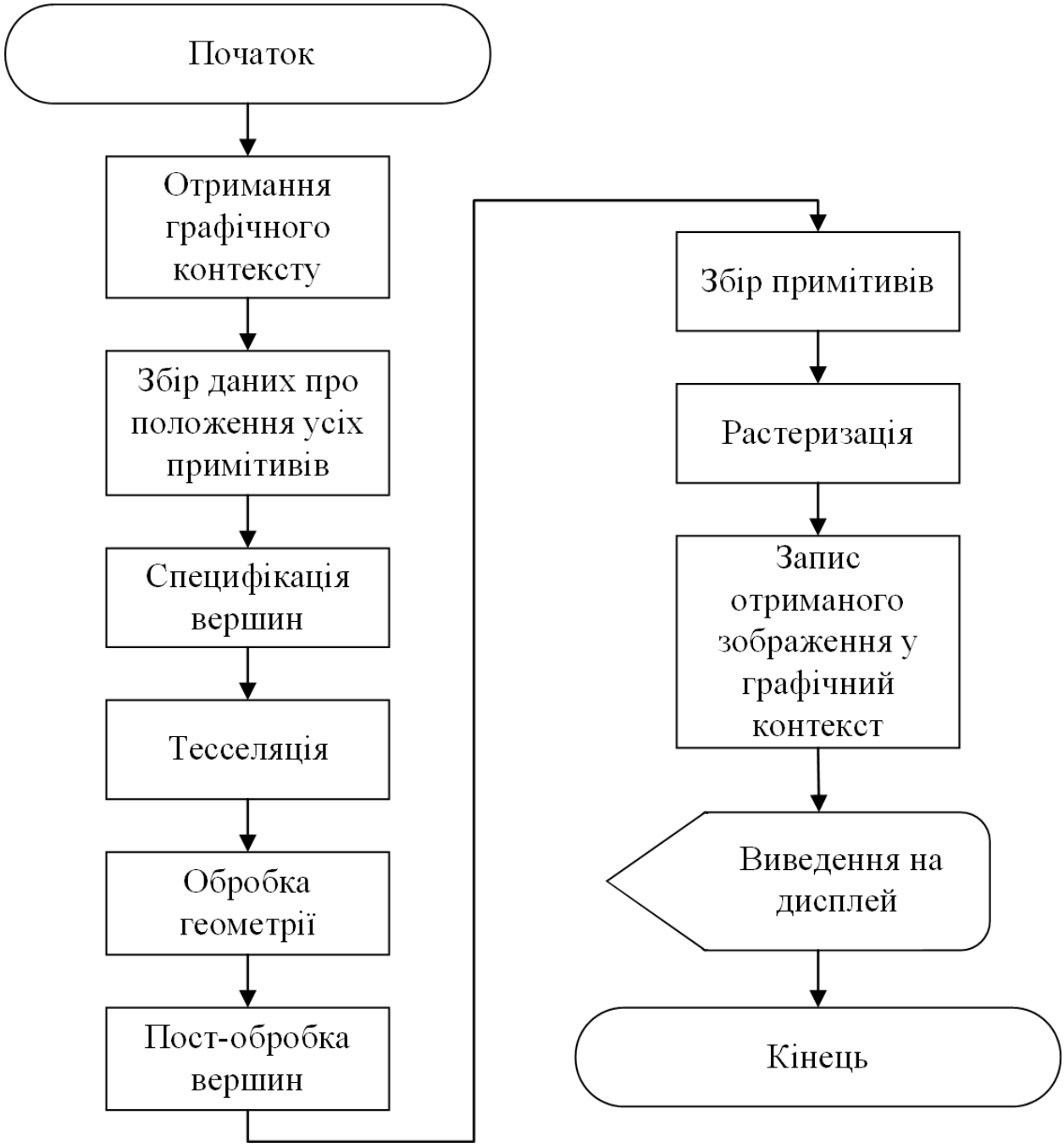


Рисунку 2.2 – Алгоритм роботи гри

Метод реалізації графічних інтерфейсів у комп’ютерних іграх на різних платформах відрізняється тим, що дозволить компілювати додатки для різних систем з однієї кодової бази, що значно пришвидшить розробку комп’ютерних ігор, полягає у такій послудовності дій:

1. отримання графічного контексту від поточного вікна;
2. збір даних про положення усіх примітивів;
3. специфікація вершин: на цьому етапі впорядковується список вершин, які визначають межі примітиву. Також визначаються інші атрибути вершин, такі як колір, координати текстури тощо;
4. тесселяція: на цьому етапі примітиви перетворюються у мозайку пікселей, тобто поділяються на більш гладку сітку трикутників;
5. обробка геометрії: на цьому етапі вхідні примітиви перетворюються на серію трикутників;
6. пост-обробка вершин: на цьому етапі відбувається відрізання. Відрізання відкидає область примітивів, що лежать за межами області перегляду;
7. збір примітивів: на цьому етапі дані вершин упорядковуються в послідовність простих примітивів (лінії, точки або трикутники);
8. растеризація: перетворення примітивів у дані про фрагменти пікселей;
9. запис отриманого зображення у графічний контекст;
10. виведення вмісту графічного контексту на дисплей.

На рисунку 2.3 наведений алгоритм роботи розробленого методу.



Рисунку 2.3 – Алгоритм графічного відображення гри

2.4 Висновки

Проаналізовано інформаційне забезпечення. Проаналізовано принципи створення програмних бібліотек. Обрано статичний варіант бібліотеки для розробки.

Розроблено метод графічного інтерфейсу орієнтований на кросплатформенне використання та модель інтерфейсу з урахуванням архітектури бібліотеки, що дозволяють пришвидшити процес створення користувацького інтерфейсу комп’ютерної гри та забезпечити його використання на різних ігрових платформах. Створено деталізовану модель системи. Вона складається з програмних модулів, що забезпечують створення графічних елементів гри та елементів графічного інтерфейсу користувача.

Проаналізовано елементи, що забезпечують графічну взаємодію.

3 РОЗРОБКА БІБЛІОТЕКИ

3.1 Варіантний аналіз і обгрунтування вибору засобів розробки

Перед початком нового програмного проекту необхідно вирішити, яку мову програмування використовувати. Низькорівневі мови гарні для оптимізації швидкості виконання або розміру програми, в той час як високорівневі мови гарні для створення ясного і добре структурованого коду. Серед мов, які можуть підійти під завдання роботи з зображеннями можна виділити три найбільш популярних мови програмування: C, Java та C# [21].

C – компільована, статично типізована мова програмування для загального призначення.

Мова C використовується вже більше ніж 40 років. Хоча це не найдавніша комп'ютерна мова, вона є найдавнішою та найпоширенішою, тому вона може адаптуватися до сьогоднішнього технологічного віку.

Підтримує парадигми програмування, такі як процедурне програмування, функціональне програмування, узагальнене програмування, абстрагування даних, віртуальних функцій, модульність та індивідуальна компіляція. Стандартна бібліотека, серед іншого, включає часто використовувані контейнери та алгоритми. C поєднує властивості мов високого та низького рівня.

Стандартні бібліотеки C це здебільшого набори стандартних бібліотек C. Більшість бібліотек C включають стандартну бібліотеку шаблонів. бібліотеку шаблонів надає корисні інструменти, такі як ітератор (високорівневий покажчик) та контейнери (подібні до масивів, які можуть автоматично зростати, додаючи нові елементи). Як і у випадку з C, особливістю доступу до

бібліотеки є використання #include вхідної директиви для розміщення стандартних файлів заголовків.

Переваги мови С :

а) обчислювальна продуктивність;

б) підтримка різних стилів програмування: структурне, узагальнене програмування, функціональне програмування;

в) видалення об’єктів з пам’яті, що спрощує і підвищує надійність управління пам'яттю та іншими ресурсами;

г) перевантаження операторів;

д) шаблони дають можливість побудови узагальнених контейнерів і алгоритмів для різних типів даних;

е) доступність для мови С існує величезна кількість навчальної літератури.

Недоліки мови С :

а) необхідність стежити за пам’яттю;

б) наявність безлічі можливостей, які порушують принципи тіпобезопасності призводить до того, що у програмі може легко з’явитися важковловима помилка.

Мова Java з'явилася в рамках розробки передових програмних продуктів для різних домашніх пристроїв. За допомогою мови програмування С був розпочат цей проект, але незабаром з'явилася група бар'єрів, ідеальним способом боротьби з ним була заміна самого інструмента - мови програмування. Стало очевидно, що нам потрібна незалежна від платформи мова програмування, яка не повинна індивідуально збиратися для кожної

архітектури і може запускати програми, які можна використовувати на різних процесорах в різних операційних системах.

Програми Java переводяться в байт-код, який виконує віртуальна java машина (Java Machine). Вона оброблює байтовий потік і віддає інструкції устаткуванню як інтерпретатор, але з тією різницею, що байтовий код, на відміну від тексту, обробляється значно швидше. Перевага способу запуску програми полягає в абсолютній незалежності багатого коду від ОС та обладнання, що дозволяє машині Java запускати програми Java на будь-якому пристрої, що забезпечує віртуальну машину.

Додатки переносяться на безліч різних платформ. Одного разу написаний додаток не доведеться переписувати під інші платформи: він буде працювати без будь-яких змін на різних операційних системах і апаратних архітектурах. Мова Java — об'єктно-орієнтована й одночасно досить проста мова програмування. Процес розроблення програмного забезпечення з використанням Java значно скорочується в силу того, що Java — інтерпретована мова. Довгий процес компіляції-збіркизавантаження застарів, тепер програму тільки треба відкомпілювати та відразу використовувати.

Автоматична збірка сміття була додана в Java, що спрощує процес програмування, тому що не потрібно слідкувати за очисткою пам'яті, але дещо ускладнює систему в цілому. Завжди викликало масу проблем управління пам'яттю в C і C++, тепер же про це не доведеться багато піклуватися.

Переваги мови Java:

а) сувора статична типізація. Вона значно скорочує чисельність помилок і поліпшує здійсненність підтримки коду, тим паче при застосуванні статичного аналізатора;

б) кросплатформеність;

Недоліки мови Java:

а) платне комерційне використання;

б) багатослівний і складний код.

Мова програмування C# суміщає об'єктно-орієнтовані і контекстно-орієнтовані концепції програмування. Вона розроблялась в 1998-2001 роках в компанії Microsoft як головна мова розроблення додатків для платформи Microsoft .NET групою інженерів під керівництвом Андерса Хейлсберга. В стандартну установку .NET входить компілятор C#, тому програми на ньому можна створювати й компілювати навіть без інструментальних засобів Microsoft Visual Studio.

Майже 75% синтаксичних можливостей мови C# подібні мові програмування Java, як і Java, C# спочатку призначався для веб-розробки. Його крім того також називають «очищеною версією Java». Біля 10% синтаксичних можливостей запозичено з мови C++ та 5% — з Visual Basic. І близько 10% — це реалізація власних ідей розробників C#.

Всупереч на дуже суттєві розбіжності між компонентною об'єктною моделлю COM (головного стандарту Microsoft для компонентного проектування і реалізації 41 програмного забезпечення) і моделлю Java, мова програмування має досить багато спільного. Загальне середовище виконання програм засноване на використанні проміжної мови IL (Intermediate Language — проміжна мова), яка здійснює майже ту ж роль, що і байт-код віртуальної машини Java. Застосовані в рамках технології .NET компілятори з різних мов транслюють програми в IL-код. Так само, як і байт-код Java, IL-код являє собою

команди гіпотетичної стекової обчислювальної машини. Але є і різниця в пристрої й застосування IL.

В основу проектування лягла легкість застосування при розробці мови, домінуючи над потужністю мови й швидкістю виконання. З цього місця і збирач сміття з контрольованими об’єктними посиланнями, який автоматично звільняє пам’ять, відбираючи при цьому процесорний час. На думку багатьох розробників, другим найважливішим чинником уникнення помилок при розробці є безпека роботи з типами.

Переваги мови C#:

а) лямбда-функції. Як і анонімні внутрішні класи Java, C# гарантує більш продуктивні лямбда-функції;

б) підтримка ключового слова «yield». За допомогою «yield» можна встановити генератор. Використовуючи генератор, можна створювати обчислення послідовностей «на льоту»;

в) підтримка оператора «?» пропонує досить простий синтаксис для отримання значення посилального типу.

Недоліки мови C#:

а) пріоритетна орієнтованість на Windows платформу;

б) відсутність покажчиків та адресної арифметики.

В таблиці 3.1 наведено результати порівняння розглянутих мов програмування за обраними критеріями.

Таблиця 3.1 – Порівняння мов програмування

Назва критерію	C	C#	Java
Глобальні змінні	1	1	0
Ручне управління пам'яттю	1	1	0
Препроцесорна обробка	1	1	1
Метапрограмування	1	1	1
Наявність властивостей	1	0	0
Сумарний коефіцієнт	5	4	2

Враховуючи всі вищенаведені переваги, можна сказати, що мови програмування С є найбільш доцільними для розробки програмних засобів для ущільнення зображень.

3.2 Вибір середовища розробки

Не менш важливим для ефективної розробки програмного забезпечення є обрання інтегрованого середовища розробки. IDE (An integrated development environment) – це програма, яке забезпечує комплексні засоби для комп'ютерних програмістів для розробки програмного забезпечення. IDE зазвичай складається щонайменше з редактора вихідного коду, засобів автоматизації побудови та налагоджувача.

Оскільки С є одним з найбільш поширених мов програмування, для нього існує велика кількість інтегрованих середовищ розробки, що володіють різним функціоналом.

Серед найрозповсюдженіших, які підтримують С є:

а) Visual Studio Code;

б) JetBrains Clion;

в) Qt Creator;

г) Eclipse SDK;

Microsoft Visual Studio Code – інтегрованим середовищем призначеної для розробки ПО, що володіє рядом додаткових інструментальних засобів. Функціонал MS Visual Studio Code дозволяє розробляти консольні додатки, додатки з графічним інтерфейсом, веб-сайти, веб-додатки та веб-служби для платформ.

MS Visual Studio Code містить вбудований інструмент для редагування вихідного коду програми і дозволяє виробляти рефакторинг коду. Вбудований в MS Visual Studio Code відладчик працює має два рівні роботи: рівень налагодження вихідного коду, і рівень налагодження машинного коду. Додаткові вбудовані інструментальні засоби містять форми графічного інтерфейсу і зручний редактор для створення додатків з інтерфейсами, містять веб-редактор і редактор класів. Дане середовище розробки підтримує можливість створення і підключення плагінів (доповнень), які використовуються з метою розширення функціоналу. MS Visual Studio Code підтримує системи контролю версій вихідного коду (наприклад, Visual SourceSafe або Subversion).

JetBrains CLion - розумна інтегроване середовище розробки програмного забезпечення, призначена для розробки додатків на мовах C і C ++. JetBrains Clion дозволяє розробляти програмне забезпечення на платформах Windows, OS X і Linux.

Багатофункціональне середовище розробки - JetBrains CLion включає в себе вбудований відладчик, багато шаблонів готового коду і інтерфейс для роботи з популярними системами контролю версій (наприклад, Subversion, Git, GitHub, Mercurial, CVS, Perforce і TFS). CLion надає можливості автодоповнення і автоформатирования коду, виконує аналіз коду на стрічку з підсвічуванням потенційних проблем і пропонує способи їх виправлення, підтримує систему збирання кроссплатформенних проектів CMake, а також різні рефакторингом коду.

Qt Creator - повністю інтегроване середовище розробки програмного забезпечення, яка містить інструменти для проектування і розробки складних додатків під різними операційними системами.

Qt Creator інтегрований з кроссплатформенную системами автоматизації збирання: qmake і CMake, містить редактор коду Qt Designer, що дозволяє проектувати і збирати графічні інтерфейси користувача (GUI) з віджетів Qt. Дане середовище містить багато корисних інструментів, таких як підтримка систем управління версіями і емулятор Qt. Qt Creator містить внутрішній відладчик для налагодження звичайних додатків на C, а також включає можливість підключити мобільні пристрої до свого комп'ютера і налагоджувати запущені на них програми.

Eclipse SDK - вільна інтегроване середовище розробки програмного забезпечення з відкритим вихідним кодом. Eclipse SDK має велику розгалужену систему плагінів (доповнень), яка забезпечує роботи середу з такими мовами програмування, як C, C ++, PHP, Perl, Python, Ruby, Ada.

Eclipse SDK дозволяє створювати багатоплатформність для Microsoft Windows, Mac OS X, Linux дистрибутивів і навіть Solaris. Дання середовище

включає в себе Eclipse Java development tools (JDT), що містить компілятор Java. Програма використовує інструменти, які утворюють собою систему Eclipse Rich Client Platform.

Така система допомагає розробникам створювати потужні функціональні користувальницькі додатки з гарним графічним інтерфейсом на основі CSS (Cascading Style Sheets - каскадні таблиці стилів). Інтегроване середовище розробки Eclipse SDK надає Java IDE програму з усім необхідним набором інструментів для розробки якісних програм. В таблиці 3.2 наведено результати порівняння засобів розробки на мові С.

Таблиця 3.2 – Порівняння засобів розробки

Назва критерію	MS Visual Studio Code	JetBrains Clion	Eclipse	Qt Creator
Крос-платформеність	+	+	+	+
Можливість розробки графічних інтерфейсів	+	-	-	+
Автодоповнення	+	+	+	+
Готові бібліотеки	+	+	+	+
Надійність	+	-	-	-
Функціональність	+	+	+	-
Сумарний коефіцієнт	6	4	4	4

В результаті проведення аналізу засобів розробки програмного додатку сумарний коефіцієнт Microsoft Visual Studio Code – 6, переважає коефіцієнти

інших аналогів, тому дане середовище є оптимальним для даної розробки та вирішення поставленої задачі.

3.3 Розробка модулів бібліотеки

Для реалізації необхідної функціональності для розробки гри було сформовано список модулів, які будуть містити структури та функції для:

а) роботи з вікнами, графічним контекстом та який буде забезпечувати мультиплатформеність;

б) рендерингу елементів інтерфейсу на дисплей;

в) введення та дотику – цей модуль має надавати інформацію про введені користувачем кнопки клавіатури, миші або дотики на тачскріні;

г) таймеру та керування потоками;

д) камери;

е) відображення графічних примітивів;

ж) роботи з зображеннями, текстурами, матеріалами та шейдерами;

роботи зі звуком;

Для пришвидшення розробки у бібліотеці розроблено такі структури даних (рисунок 3.1):

а) Vector2 – містить 2 значення о положенні елемента на двовимірний площині;

б) Vector3 – містить 3 значення о положенні елемента на тривимірний площині;

в) Rectangle – містить 4 значення о положенні прямокутника на площині та його висоту та ширину;

г) Matrix – містить 16 значень про значення переміщення, поворту та масштабування елементу у графічном контексті;

д) Color– містить 4 значень про RGBA представлення коліру;

е) Image – містить значення про шлях до зображення та його представлення у пам’яті;

ж) Texture2D – містить значення про номер текстури у пам’яті, значення висоти, ширини та формату;

з) Camera – містить значення про тип камери та позицію у тривимірому просторі;

и) Camera2D – містить значення про позицію камери у двовимірний просторі;

к) BoundingBox – містить дані про границі колізії елементів інтерфейсу;

л) Mesh – містить данні про точки, з яких складається меш, їх нормалі та колір;

м) Shader – містить дані про положення точок, ModelView-Projection матриці, карти текстур;

н) Material – містить дані про текстуру, карту нормалей та шейдер;

о) Model – містить дані про меш, його матеріал та матрицю трансформації;

п) Ray – містить 2 Vector3 структури позиції та напрямку;

р) Sound – містить дані про шлях до звуку та буфер;

с) Wave – містить дані про буфер, об’єм буферу, кількості бітів на семпл звуку та значення каналу.

На рисунку 3.1 зображено структури Vector2 та Vector3, які відповідають за збереження положення елементів у просторі, Rectange – о положенні прямокутника на площині та його висоту та ширину, Matrix –

значення переміщення, повороту та масштабування елементу, Color – представлення кольору, Image – шлях до зображення та його представлення у пам’яті.

```c
// Vector2 type
typedef struct Vector2 {
    float x;
    float y;
} Vector2;

// Vector3 type
typedef struct Vector3 {
    float x;
    float y;
    float z;
} Vector3;

// Rectangle type
typedef struct Rectangle {
    int x;
    int y;
    int width;
    int height;
} Rectangle;

// Matrix type
typedef struct Matrix {
    float m0, m4, m8, m12;
    float m1, m5, m9, m13;
    float m2, m6, m10, m14;
    float m3, m7, m11, m15;
} Matrix;

// Color type, RGBA (32bit)
typedef struct Color {
    unsigned char r;
    unsigned char g;
    unsigned char b;
    unsigned char a;
} Color;

// Image type, bpp always RGBA (32bit)
typedef struct Image {
    void *data;             // Image raw data
    int width;              // Image base width
    int height;             // Image base height
    int mipmaps;            // Mipmap levels, 1 by default
    int format;             // Data format (TextureFormat)
} Image;
```

Рисунок 3.1 – Реалізація структур Vector2, Vector3, Rectangel, Matrix, Color та Image бібліотеки

На рисунку 3.2 зображено структури Texture, яка зберігає значення про номер текстури у пам’яті, висоти, ширини та формату, Camera – про тип камери та позицію у тривимірому просторі, Camera2D – про позицію камери у двовимірний просторі, BoundingBox – про границі колізії елементів інтерфейсу, Mesh –про точки, з яких складається меш, їх нормалі та колір.

```
// Texture2D type, bpp always RGBA (32bit)
typedef struct Texture2D {
    unsigned int id;        // OpenGL texture id
    int width;              // Texture base width
    int height;             // Texture base height
    int mipmaps;            // Mipmap levels, 1 by default
    int format;             // Data format (TextureFormat)
} Texture2D;

// Camera type, defines a camera position/orientation in 3d space
typedef struct Camera {
    Vector3 position;       // Camera position
    Vector3 target;         // Camera target it looks-at
    Vector3 up;             // Camera up vector (rotation over its axis)
    float fovy;             // Camera field-of-view apperture in Y (degrees)
} Camera;

// Camera2D type, defines a 2d camera
typedef struct Camera2D {
    Vector2 offset;         // Camera offset (displacement from target)
    Vector2 target;         // Camera target (rotation and zoom origin)
    float rotation;         // Camera rotation in degrees
    float zoom;             // Camera zoom (scaling), should be 1.0f by default
} Camera2D;

// Bounding box type
typedef struct BoundingBox {
    Vector3 min;            // minimum vertex box-corner
    Vector3 max;            // maximum vertex box-corner
} BoundingBox;

// Vertex data definning a mesh
typedef struct Mesh {
    int vertexCount;        // number of vertices stored in arrays
    int triangleCount;      // number of triangles stored (indexed or not)
    float *vertices;        // vertex position (XYZ - 3 components per vertex) (shader-location = 0)
    float *texcoords;       // vertex texture coordinates (UV - 2 components per vertex) (shader-location = 1)
    float *texcoords2;      // vertex second texture coordinates (useful for lightmaps) (shader-location = 5)
    float *normals;         // vertex normals (XYZ - 3 components per vertex) (shader-location = 2)
    float *tangents;        // vertex tangents (XYZ - 3 components per vertex) (shader-location = 4)
    unsigned char *colors;  // vertex colors (RGBA - 4 components per vertex) (shader-location = 3)
    unsigned short *indices;// vertex indices (in case vertex data comes indexed)

    unsigned int vaoId;     // OpenGL Vertex Array Object id
    unsigned int vboId[7];  // OpenGL Vertex Buffer Objects id (7 types of vertex data)
} Mesh;
```

Рисунок 3.2 – Реалізація структур Texture, Camera, Camera2D, BoundingBox та Mesh бібліотеки

На рисунку 3.3 зображено структури Shader містить дані про положення точок, ModelView-Projection матриці, карти текстур [14], Material –про текстуру, карту нормалей та шейдер, Model –про меш, його матеріал та матрицю трансформації.

```
// Shader type (generic shader)
typedef struct Shader {
    unsigned int id;        // Shader program id

    // Vertex attributes locations (default locations)
    int vertexLoc;          // Vertex attribute location point    (default-location = 0)
    int texcoordLoc;        // Texcoord attribute location point  (default-location = 1)
    int texcoord2Loc;       // Texcoord2 attribute location point (default-location = 5)
    int normalLoc;          // Normal attribute location point    (default-location = 2)
    int tangentLoc;         // Tangent attribute location point   (default-location = 4)
    int colorLoc;           // Color attibute location point      (default-location = 3)

    // Uniform locations
    int mvpLoc;             // ModelView-Projection matrix uniform location point (vertex shader)
    int tintColorLoc;       // Diffuse color uniform location point (fragment shader)

    // Texture map locations (generic for any kind of map)
    int mapTexture0Loc;     // Map texture uniform location point (default-texture-unit = 0)
    int mapTexture1Loc;     // Map texture uniform location point (default-texture-unit = 1)
    int mapTexture2Loc;     // Map texture uniform location point (default-texture-unit = 2)
} Shader;

// Material type
typedef struct Material {
    Shader shader;          // Standard shader (supports 3 map textures)

    Texture2D texDiffuse;   // Diffuse texture  (binded to shader mapTexture0Loc)
    Texture2D texNormal;    // Normal texture   (binded to shader mapTexture1Loc)
    Texture2D texSpecular;  // Specular texture (binded to shader mapTexture2Loc)

    Color colDiffuse;       // Diffuse color
    Color colAmbient;       // Ambient color
    Color colSpecular;      // Specular color

    float glossiness;       // Glossiness level (Ranges from 0 to 1000)
} Material;

// Model type
typedef struct Model {
    Mesh mesh;              // Vertex data buffers (RAM and VRAM)
    Matrix transform;       // Local transform matrix
    Material material;      // Shader and textures data
} Model;
```

Рисунок 3.3 – Реалізація структур Shader, Material та Model бібліотеки

На рисунку 3.4 зображено структури Ray– містить 2 Vector3 структури позиції та напрямку, Sound –про шлях до звуку та буфер,Wave – про буфер, об’єм буферу, кількості бітів на семпл звуку та значення каналу.

```
// Ray type (useful for raycast)
typedef struct Ray {
    Vector3 position;       // Ray position (origin)
    Vector3 direction;      // Ray direction
} Ray;

// Sound source type
typedef struct Sound {
    unsigned int source;    // Sound audio source id
    unsigned int buffer;    // Sound audio buffer id
} Sound;

// Wave type, defines audio wave data
typedef struct Wave {
    void *data;                 // Buffer data pointer
    unsigned int dataSize;      // Data size in bytes
    unsigned int sampleRate;    // Samples per second to be played
    short bitsPerSample;        // Sample size in bits
    short channels;
} Wave;
```

Рисунок 3.4 – Реалізація структур Ray, Sound та Wave бібліотеки

3.4 Розробка модуля для роботи з вікнами

Модуль складається з 8 основних функцій:

а) InitWindow – ініціалізує вікно;

б) InitGraphicsDevice – ініціалізує графічний конектс;

в) CloseWindow – закриває вікно;

г) WindowShouldClose – перевіряє чи натиснута кнопка закриття програми;

д) IsWindowMinimized – повертає значення стану вікна;

е) ToggleFullscreen – змінує стан вікна;

ж) GetScreenWidth – повертає значення ширини вікна;

з) GetScreenHeight – повертає значення висоти вікна;

Список функцій з поясненнями у коментарях у заголовочному файлу зображенно на рисунку 3.5.

```
#if defined(PLATFORM_ANDROID)
// Init Android Activity and OpenGL Graphics
void InitWindow(int width, int height, struct android_app *state);
#elif defined(PLATFORM_DESKTOP) || defined(PLATFORM_RPI) || defined(PLATFORM_WEB)
// Initialize Window and OpenGL Graphics
void InitWindow(int width, int height, const char *title);
#endif

// Close Window and Terminate Context
void CloseWindow(void);
// Detect if KEY_ESCAPE pressed or Close icon pressed
bool WindowShouldClose(void);
// Detect if window has been minimized (or lost focus)
bool IsWindowMinimized(void);
// Fullscreen toggle (only PLATFORM_DESKTOP)
void ToggleFullscreen(void);
// Get current screen width
int GetScreenWidth(void);
// Get current screen height
int GetScreenHeight(void);
```

Рисунок 3.5 – Список функція модуля роботи з вікнами

На рисунку 3.6 зображено реалізція функції InitWindow для Desktop та Web платформ.

```
void InitWindow(int width, int height, const char *title)
{
    // Store window title (could be useful...)
    windowTitle = title;

    // Init graphics device (display device and OpenGL context)
    InitGraphicsDevice(width, height);

    LoadDefaultFont();

    // Init hi-res timer
    InitTimer();

#if defined(PLATFORM_WEB)
    emscripten_set_fullscreenchange_callback(0, 0, 1, EmscriptenFullscreenChangeCallback);

    // NOTE: Some code examples
    //emscripten_set_touchstart_callback(0, NULL, 1, Emscripten_HandleTouch);
    //emscripten_set_touchend_callback("#canvas", data, 0, Emscripten_HandleTouch);
    emscripten_set_touchstart_callback("#canvas", NULL, 1, EmscriptenInputCallback);
    emscripten_set_touchend_callback("#canvas", NULL, 1, EmscriptenInputCallback);
    emscripten_set_touchmove_callback("#canvas", NULL, 1, EmscriptenInputCallback);
    emscripten_set_touchcancel_callback("#canvas", NULL, 1, EmscriptenInputCallback);

    // TODO: Add gamepad support (not provided by GLFW3 on emscripten)
    //emscripten_set_gamepadconnected_callback(NULL, 1, EmscriptenInputCallback);
    //emscripten_set_gamepaddisconnected_callback(NULL, 1, EmscriptenInputCallback);
#endif

    mousePosition.x = (float)screenWidth/2.0f;
    mousePosition.y = (float)screenHeight/2.0f;
}
```

Рисунок 3.6 – Реалізація InitWindow для Desktop та Web платформ

Оскільки мобільні платформи потребують додактової ініціалізації таких компонентів як тачскрін, кнопок багатозадачності, зміни гучності, пристроїв вібрації, гіроскопу т.д. реалізація буде інакшою. На рисунку 3.7 зображено реалізція функції InitWindow для для Android платформи.

```
// Android activity initialization
void InitWindow(int width, int height, struct android_app *state){
    screenWidth = width;
    screenHeight = height;

    // Set desired windows flags before initializing anything
    ANativeActivity_setWindowFlags(app->activity, AWINDOW_FLAG_FULLSCREEN, 0);  //AWINDOW_FLAG_SCALED, AWINDOW_FLAG_DITHER

    int orientation = AConfiguration_getOrientation(app->config);

    if (orientation == ACONFIGURATION_ORIENTATION_PORT) TraceLog(INFO, "PORTRAIT window orientation");
    else if (orientation == ACONFIGURATION_ORIENTATION_LAND) TraceLog(INFO, "LANDSCAPE window orientation");

    // TODO: Automatic orientation doesn't seem to work
    if (width <= height){
        AConfiguration_setOrientation(app->config, ACONFIGURATION_ORIENTATION_PORT);
    }
    else{
        AConfiguration_setOrientation(app->config, ACONFIGURATION_ORIENTATION_LAND);
    }

    InitAssetManager(app->activity->assetManager);

    // Wait for window to be initialized (display and context)
    while (!windowReady)
    {
        // Process events loop
        while ((ident = ALooper_pollAll(0, NULL, &events,(void**)&source)) >= 0)
        {
            // Process this event
            if (source != NULL) process(app, source);

            // NOTE: Never close window, native activity is controlled by the system!
            //if (app->destroyRequested != 0) windowShouldClose = true;
        }
    }
}
```

Рисунок 3.7 – Реалізація InitWindow для Android платформи

При створенні вікна на платформах Desktop та Web викликається фукнція InitGraphicsDevice. Вона використовується для ініціалізації вікна та параметрів вікна таких, як ширина, висота, назва, іконка, чи буде вікно з можливістю змінити розмір. Після створення вікна та визначення його характеристик функція InitGraphicsDevice створює графічний контекст для відображення елементів гри та графічного інтерфейсу. На рисунку 3.8 зображено код реалізації функції InitGraphicsDevice на операційних системах Windows, Linux, MacOS та платформі Web (HTML5).

```
static void InitGraphicsDevice(int width, int height)
{
    screenWidth = width;        // User desired width
    screenHeight = height;      // User desired height

    downscaleView = MatrixIdentity();

#if defined(PLATFORM_DESKTOP) || defined(PLATFORM_WEB)
    glfwSetErrorCallback(ErrorCallback);
#if defined(PLATFORM_DESKTOP)
    // Find monitor resolution
    const GLFWvidmode *mode = glfwGetVideoMode(glfwGetPrimaryMonitor());

    displayWidth = mode->width;
    displayHeight = mode->height;

    // Screen size security check
    if (screenWidth <= 0) screenWidth = displayWidth;
    if (screenHeight <= 0) screenHeight = displayHeight;
#endif  // defined(PLATFORM_DESKTOP)

#if defined(PLATFORM_WEB)
    displayWidth = screenWidth;
    displayHeight = screenHeight;
#endif  // defined(PLATFORM_WEB)

    glfwDefaultWindowHints();                   // Set default windows hints

    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);     // Avoid window being resizable

    if (configFlags & FLAG_MSAA_4X_HINT)
    {
        glfwWindowHint(GLFW_SAMPLES, 4);       // Enables multisampling x4 (MSAA), default is 0
    }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);        // Choose OpenGL major version (just hint)
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);        // Choose OpenGL minor version (just hint)
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE); // Profiles Hint: Only 3.3 and above!
                                                                   // Other values: GLFW_OPENGL_ANY_PROFILE, GLFW_OPENGL_COMPAT_PROFILE
#ifdef __APPLE__
        glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);  // OSX Requires
#else
        glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_FALSE); // Fordward Compatibility Hint: Only 3.3 and above!
#endif
        //glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, GL_TRUE);
    }
```

Рисунок 3.8 – Реалізація функції InitGraphicsDevice

Після натиснення кнопки завершення програми необхідно очистити надані пристроїм ресурси, знищити графічний контекст та вікно. Функція CloseWindow знищую графічний контекст та вікно (рисунок 3.9).

```
void CloseWindow(void)
{
    UnloadDefaultFont();

    rlglClose();                // De-init rlgl

#if defined(PLATFORM_DESKTOP) || defined(PLATFORM_WEB)
    glfwDestroyWindow(window);
    glfwTerminate();
#endif

#if defined(PLATFORM_ANDROID) || defined(PLATFORM_RPI)
    // Close surface, context and display
    if (display != EGL_NO_DISPLAY)
    {
        eglMakeCurrent(display, EGL_NO_SURFACE, EGL_NO_SURFACE, EGL_NO_CONTEXT);

        if (surface != EGL_NO_SURFACE)
        {
            eglDestroySurface(display, surface);
            surface = EGL_NO_SURFACE;
        }

        if (context != EGL_NO_CONTEXT)
        {
            eglDestroyContext(display, context);
            context = EGL_NO_CONTEXT;
        }

        eglTerminate(display);
        display = EGL_NO_DISPLAY;
    }
#endif

#if defined(PLATFORM_RPI)
    // Wait for mouse and gamepad threads to finish before closing
    // NOTE: Those threads should already have finished at this point
    // because they are controlled by windowShouldClose variable
    pthread_join(mouseThreadId, NULL);
    pthread_join(gamepadThreadId, NULL);
#endif
}
```

Рисунок 3.9 – Реалізація функція Close Window

До допоміжних функцій можна віднести функції: WindowShouldClose – перевіряє чи бути програма зачиненою, підготовлює програму до завершення, IsWindowMinimized – повертає значення про стан вікна, ToggleFullscreen – перемикає стан вікна від «на весь екран» до мінімізованого, GetScreenWidth – повертає значення ширини вікна для подальшого використання у відображені елементів, GetScreenHeight – повертає значення висоти вікна для подальшого використання у відображені елементів [24]. Допоміжні функції зображено на рисунку 3.10.

```
bool WindowShouldClose(void){
#if defined(PLATFORM_DESKTOP) || defined(PLATFORM_WEB)
    while (windowMinimized) glfwPollEvents();

    return (glfwWindowShouldClose(window));
#endif
#if defined(PLATFORM_ANDROID) || defined(PLATFORM_RPI)
    return windowShouldClose;
#endif
}
// Detect if window has been minimized (or lost focus)
bool IsWindowMinimized(void){
#if defined(PLATFORM_DESKTOP) || defined(PLATFORM_WEB)
    return windowMinimized;
#else
    return false;
#endif
}

// Fullscreen toggle
void ToggleFullscreen(void){
#if defined(PLATFORM_DESKTOP)
    fullscreen = !fullscreen;          // Toggle fullscreen flag

    if (fullscreen) glfwSetWindowMonitor(window, glfwGetPrimaryMonitor(), 0, 0, screenWidth, screenHeight, GLFW_DONT_CARE);
    else glfwSetWindowMonitor(window, NULL, 0, 0, screenWidth, screenHeight, GLFW_DONT_CARE);
#endif
}

// Get current screen width
int GetScreenWidth(void){
    return screenWidth;
}

// Get current screen height
int GetScreenHeight(void){
    return screenHeight;
}
```

Рисунок 3.10 – Реалізація допоміжних функцій модуля

3.5 Розробка модуля рендерингу

Модуль складається з 9 основних функцій:

а) ClearBackground – очищення дисплей перед відмалювання наступного кадра гри;

б) BeginDrawing – відкриває загальний контекст для відображення елементів та примітивів;

в) EndDrawing;

г) Begin2dMod – завершує головний контекст відмалювання елементів гри;

д) End2dMode – завершує контекст для відображення двовимірних елементів та примітивів;

е) Begin3dMode – відкриває контекст для відображення тривимірних елементів та примітивів;

ж) End3dMode – завершує контекст для відображення тривимірних елементів та примітивів;

з) BeginTextureMode – відкриває контекст для відображення текстур;

и) EndTextureMode – завершує контекст для відображення текстур;

Список функцій з поясненнями у коментарях у заголовочному файлу зображенно на рисунку 3.11.

```
// Sets Background Color
void ClearBackground(Color color);
// Setup drawing canvas to start drawing
void BeginDrawing(void);
// End canvas drawing and Swap Buffers (Double Buffering)
void EndDrawing(void);

// Initialize 2D mode with custom camera
void Begin2dMode(Camera2D camera);
// Ends 2D mode custom camera usage
void End2dMode(void);
// Initializes 3D mode for drawing (Camera setup)
void Begin3dMode(Camera camera);
// Ends 3D mode and returns to default 2D orthographic mode
void End3dMode(void);
// Initializes render texture for drawing
void BeginTextureMode(RenderTexture2D target);
// Ends drawing to render texture
void EndTextureMode(void);
```

Рисунок 3.11 – Список функція модуля рендирингу

Функція ClearBackground використовується для того, щоб очистити дисплей перед відмалювання наступного кадра гри у переданий у функцію колір. Функція BeginDrawing відкриває загальний контекст для відображення примитивів та елементів гри та графічного інтерфейсу [30]. Функція EndDrawing завершує головний контекст відмалювання елементів. Ці функції зображені на рисунку 3.12.

```
// Sets Background Color
void ClearBackground(Color color){
    rlClearColor(color.r, color.g, color.b, color.a);
}

// Setup drawing canvas to start drawing
void BeginDrawing(void){
    currentTime = GetTime();            // Number of elapsed seconds since InitTimer() was called
    updateTime = currentTime - previousTime;
    previousTime = currentTime;

    rlClearScreenBuffers();             // Clear current framebuffers
    rlLoadIdentity();                   // Reset current matrix (MODELVIEW)
    rlMultMatrixf(MatrixToFloat(downscaleView));       // If downscale required, apply it here
}

// End canvas drawing and Swap Buffers (Double Buffering)
void EndDrawing(void){
    rlglDraw();                     // Draw Buffers (Only OpenGL 3+ and ES2)

    SwapBuffers();                  // Copy back buffer to front buffer
    PollInputEvents();              // Poll user events

    // Frame time control system
    currentTime = GetTime();
    drawTime = currentTime - previousTime;
    previousTime = currentTime;

    frameTime = updateTime + drawTime;

    double extraTime = 0.0;

    while (frameTime < targetTime)
    {
        // Implement a delay
        currentTime = GetTime();
        extraTime = currentTime - previousTime;
        previousTime = currentTime;
        frameTime += extraTime;
    }
}
```

Рисунок 3.12 – Основні функції рендирингу

Функція BeginTextureMode відкриває контекст для відображення текстур. Функція EndTextureMode завершує відмалювання текстур. Ці функції зображені на рисунку 3.13.

```
void BeginTextureMode(RenderTexture2D target)
{
    rlglDraw();                            // Draw Buffers (Only OpenGL 3+ and ES2)

    rlEnableRenderTexture(target.id);      // Enable render target

    rlClearScreenBuffers();                // Clear render texture buffers

    // Set viewport to framebuffer size
    rlViewport(0, 0, target.texture.width, target.texture.height);

    rlMatrixMode(RL_PROJECTION);           // Switch to PROJECTION matrix
    rlLoadIdentity();                      // Reset current matrix (PROJECTION)

    // Set orthographic projection to current framebuffer size
    // NOTE: Configured top-left corner as (0, 0)
    rlOrtho(0, target.texture.width, target.texture.height, 0, 0.0f, 1.0f);

    rlMatrixMode(RL_MODELVIEW);            // Switch back to MODELVIEW matrix
    rlLoadIdentity();                      // Reset current matrix (MODELVIEW)

    //rlScalef(0.0f, -1.0f, 0.0f);         // Flip Y-drawing (?)
}

// Ends drawing to render texture
void EndTextureMode(void)
{
    rlglDraw();                            // Draw Buffers (Only OpenGL 3+ and ES2)

    rlDisableRenderTexture();              // Disable render target

    // Set viewport to default framebuffer size (screen size)
    // TODO: consider possible viewport offsets
    rlViewport(0, 0, GetScreenWidth(), GetScreenHeight());

    rlMatrixMode(RL_PROJECTION);           // Switch to PROJECTION matrix
    rlLoadIdentity();                      // Reset current matrix (PROJECTION)

    // Set orthographic projection to current framebuffer size
    // NOTE: Configured top-left corner as (0, 0)
    rlOrtho(0, GetScreenWidth(), GetScreenHeight(), 0, 0.0f, 1.0f);

    rlMatrixMode(RL_MODELVIEW);            // Switch back to MODELVIEW matrix
    rlLoadIdentity();                      // Reset current matrix (MODELVIEW)
}
```

Рисунок 3.13 – Функції рендирингу текстур

Функція Begin2dMode відкриває контекст для відображення двовимірних елементів та примітивів. Ця функція потребує структуру Camera2D для визначення границь для відображення елементів, типу проекції та іншого. Під час рендирингу ця функція створює матрицю перетворення, повороту та масштабу для визначення положення примітивів та елементів інтерфейсу. Функція End2dMode завершує відмалювання двовимірних елементів. Ці функції зображені на рисунку 3.14.

```
// Initialize 2D mode with custom camera
void Begin2dMode(Camera2D camera)
{
    rlglDraw();                            // Draw Buffers (Only OpenGL 3+ and ES2)

    rlLoadIdentity();                      // Reset current matrix (MODELVIEW)

    // Camera rotation and scaling is always relative to target
    Matrix matOrigin = MatrixTranslate(-camera.target.x, -camera.target.y, 0.0f);
    Matrix matRotation = MatrixRotate((Vector3){ 0.0f, 0.0f, 1.0f }, camera.rotation*DEG2RAD);
    Matrix matScale = MatrixScale(camera.zoom, camera.zoom, 1.0f);
    Matrix matTranslation = MatrixTranslate(camera.offset.x + camera.target.x, camera.offset.y + camera.target.y, 0.0f);

    Matrix matTransform = MatrixMultiply(MatrixMultiply(matOrigin, MatrixMultiply(matScale, matRotation)), matTranslation);

    rlMultMatrixf(MatrixToFloat(matTransform));
}

// Ends 2D mode custom camera usage
void End2dMode(void)
{
    rlglDraw();                            // Draw Buffers (Only OpenGL 3+ and ES2)

    rlLoadIdentity();                      // Reset current matrix (MODELVIEW)
}
```

Рисунок 3.14 – Функції рендирингу двовимірних елементів

На рисунку 3.15 розбражена функція Begin3dMode, яка відкриває контекст для відображення тривимірних елементів та примітивів. Ця функція потребує структуру тривимірної Camera для визначення границь для відображення елементів, типу проекці, положення камери в просторі та іншого.

```
void Begin3dMode(Camera camera)
{
    rlglDraw();                         // Draw Buffers (Only OpenGL 3+ and ES2)

    if (IsVrDeviceReady()) BeginVrDrawing();

    rlMatrixMode(RL_PROJECTION);        // Switch to projection matrix

    rlPushMatrix();                     // Save previous matrix, which contains the settings for the 2d ortho projection
    rlLoadIdentity();                   // Reset current matrix (PROJECTION)

    // Setup perspective projection
    float aspect = (float)screenWidth/(float)screenHeight;
    double top = 0.01*tan(camera.fovy*PI/360.0);
    double right = top*aspect;

    // NOTE: zNear and zFar values are important when computing depth buffer values
    rlFrustum(-right, right, -top, top, 0.01, 1000.0);

    rlMatrixMode(RL_MODELVIEW);         // Switch back to modelview matrix
    rlLoadIdentity();                   // Reset current matrix (MODELVIEW)

    // Setup Camera view
    Matrix cameraView = MatrixLookAt(camera.position, camera.target, camera.up);
    rlMultMatrixf(MatrixToFloat(cameraView));      // Multiply MODELVIEW matrix by view matrix (camera)

    rlEnableDepthTest();                // Enable DEPTH_TEST for 3D
}
```

Рисунок 3.15 – Реалізація функції Begin3dMode

На рисунку 3.16 розбражена функція End3dMode, яка завершує контекст для відображення тривимірних елементів та примітивів.

```
// Ends 3D mode and returns to default 2D orthographic mode
void End3dMode(void)
{
    rlglDraw();                         // Process internal buffers (update + draw)

    if (IsVrDeviceReady()) EndVrDrawing();

    rlMatrixMode(RL_PROJECTION);        // Switch to projection matrix
    rlPopMatrix();                      // Restore previous matrix (PROJECTION) from matrix stack

    rlMatrixMode(RL_MODELVIEW);         // Get back to modelview matrix
    rlLoadIdentity();                   // Reset current matrix (MODELVIEW)

    //rlTranslatef(0.375, 0.375, 0);      // HACK to ensure pixel-perfect drawing on OpenGL (after exiting 3D mode)

    rlDisableDepthTest();               // Disable DEPTH_TEST for 2D
}
```

Рисунок 3.16 – Реалізація функції End3dMode

3.6 Розробка модуля відображення графічних примітивів

Модуль складається з 18 основних функцій:

а) DrawPixel – функція приймає 2 значення положення пікселя у просторі та кольору, яким будет зафарбований;

б) DrawPixelV – функція приймає 1 значення положення пікселя у просторі Vector2 та кольору, яким будет зафарбований;

в) DrawLine – функція приймає 4 значення положення почок початку та кінця у просторі та кольору, яким будет зафарбована лінія;

г) DrawLineV – функція приймає 2 значення положення почок початку та кінця у просторі Vector2 та кольору, яким будет зафарбована лінія;

д) DrawCircle – приймає 2 значення про точку центра кола у просторі, радіус та колір, яким необхідно зафарбувати коло;

е) DrawCircleGradient – приймає 2 значення про точку центра кола у просторі, радіус та 2 кольори, яким необхідно зафарбувати коло;

ж) DrawCircleV – приймає значення Vector2 про точку центра кола у просторі, радіус та колір, яким необхідно зафарбувати коло;

з) DrawCircleLines – приймає 2 значення про точку центра кола у просторі, радіус та колір, яким необхідно зафарбувати коло та створює коло за допомогою примітивів ліній;

и) DrawRectangle – приймає 4 значення про точку початку прямокутника у просторі, значення висота та ширини та колір, яким необхідно зафарбувати прямокутник;

к) DrawRectangleRec – приймає значення Rectangle про положення прямокутника у просторі, висоти та ширини та колір, яким необхідно зафарбувати прямокутник;

л) DrawRectangleGradient – приймає 4 значення про точку початку прямокутника у просторі, значення висота та ширини та 2 кольори, яким необхідно зафарбувати прямокутник градієнтом;

м) DrawRectangleV – приймає 2 Vector2 значення про точку початку прямокутника у просторі, Vector2 висоти та ширини, та колір, яким необхідно зафарбувати прямокутник;

н) DrawRectangleLines – приймає 4 значення про точку початку прямокутника у просторі, висоти, ширини, та колір, яким необхідно зафарбувати прямокутник та створює його за допомогою примітивів ліній;

о) DrawTriangle – приймає 3 значення Vector2 про точки трикутника та колір, яким необхідно зафарбувати трикутник;

п) DrawTriangleLines – приймає 3 значення Vector2 про точки трикутника та колір, яким необхідно зафарбувати трикутник;

р) DrawPoly – приймає Vector2 про точки полігона та колір, яким необхідно зафарбувати полігон;

Список функцій для відображення графічних примітивів з поясненнями у коментарях у заголовочному файлу зображенно на рисунку 3.17.

```
// Draw a pixel
void DrawPixel(int posX, int posY, Color color);
// Draw a pixel (Vector version)
void DrawPixelV(Vector2 position, Color color);
// Draw a line
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color);
// Draw a line (Vector version)
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color);
// Draw a color-filled circle
void DrawCircle(int centerX, int centerY, float radius, Color color);
// Draw a gradient-filled circle
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2);
// Draw a color-filled circle (Vector version)
void DrawCircleV(Vector2 center, float radius, Color color);
// Draw circle outline
void DrawCircleLines(int centerX, int centerY, float radius, Color color);
// Draw a color-filled rectangle
void DrawRectangle(int posX, int posY, int width, int height, Color color);

// Draw a gradient-filled rectangle
void DrawRectangleGradient(int posX, int posY, int width, int height, Color color1, Color color2);
// Draw a color-filled rectangle (Vector version)
void DrawRectangleV(Vector2 position, Vector2 size, Color color);
// Draw rectangle outline
void DrawRectangleLines(int posX, int posY, int width, int height, Color color);
// Draw a color-filled triangle
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
// Draw triangle outline
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
// Draw a regular polygon (Vector version)
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color);
// Draw a closed polygon defined by points
void DrawPolyEx(Vector2 *points, int numPoints, Color color);
// Draw polygon lines      You, seconds ago • Uncommitted changes
void DrawPolyExLines(Vector2 *points, int numPoints, Color color);
```

Рисунок 3.17 – Список функція модуля відображення графічних примітивів

Функція DrawPixel приймає 2 значення положення пікселя у просторі для відображення та колір, яким необхідно зафарбувати його. Функція DrawPixelV приймає значення Vector2 положення пікселя у просторі для відображення та колір, яким необхідно зафарбувати його. Функція DrawLine приймає 4 значення про початкову та кінцеву точку у просторі та колір, яким необхідно зафарбувати лінію. Функція DrawLineV приймає 2 значення про початкову та кінцеву точку у просторі та колір, яким необхідно зафарбувати лінію [8]. Функція DrawCircle приймає 2 значення про точку центра кола у просторі,

радіус та колір, яким необхідно зафарбувати коло. Реалізація цих функцій зображена на рисунку 3.18.

```
void DrawPixel(int posX, int posY, Color color)
{
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2i(posX, posY);
        rlVertex2i(posX + 1, posY + 1);
    rlEnd();
}

// Draw a pixel (Vector version)
void DrawPixelV(Vector2 position, Color color)
{
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2f(position.x, position.y);
        rlVertex2i(position.x + 1, position.y + 1);
    rlEnd();
}

// Draw a line
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color)
{
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2i(startPosX, startPosY);
        rlVertex2i(endPosX, endPosY);
    rlEnd();
}

// Draw a line  (Vector version)
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color)
{
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2f(startPos.x, startPos.y);
        rlVertex2f(endPos.x, endPos.y);
    rlEnd();
}

// Draw a color-filled circle
void DrawCircle(int centerX, int centerY, float radius, Color color)
{
    DrawCircleV((Vector2){ centerX, centerY }, radius, color);
}
```

Рисунок 3.18 – Реалізація функцій Пікселя та Лінії

Функція DrawCircleV приймає значення Vector2 про точку центра кола у просторі, радіус та колір, яким необхідно зафарбувати коло. Функція DrawCircleGradient приймає 2 значення про точку центра кола у просторі, радіус та 2 кольори, яким необхідно зафарбувати коло. Реалізація цих функцій зображена на рисунку 3.19.

```
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2)
{
    rlBegin(RL_TRIANGLES);
        for (int i = 0; i < 360; i += 10)
        {
            rlColor4ub(color1.r, color1.g, color1.b, color1.a);
            rlVertex2i(centerX, centerY);
            rlColor4ub(color2.r, color2.g, color2.b, color2.a);
            rlVertex2f(centerX + sin(DEG2RAD*i)*radius, centerY + cos(DEG2RAD*i)*radius);
            rlColor4ub(color2.r, color2.g, color2.b, color2.a);
            rlVertex2f(centerX + sin(DEG2RAD*(i + 10)) * radius, centerY + cos(DEG2RAD*(i + 10))*radius);
        }
    rlEnd();
}

// Draw a color-filled circle (Vector version)
// NOTE: On OpenGL 3.3 and ES2 we use QUADS to avoid drawing order issues (view rlglDraw)
void DrawCircleV(Vector2 center, float radius, Color color)
{
    rlEnableTexture(GetDefaultTexture().id); // Default white texture

    rlBegin(RL_QUADS);
        for (int i = 0; i < 360; i += 20)
        {
            rlColor4ub(color.r, color.g, color.b, color.a);

            rlVertex2i(center.x, center.y);
            rlVertex2f(center.x + sin(DEG2RAD*i)*radius, center.y + cos(DEG2RAD*i)*radius);
            rlVertex2f(center.x + sin(DEG2RAD*(i + 10)) * radius, center.y + cos(DEG2RAD*(i + 10)) * radius);
            rlVertex2f(center.x + sin(DEG2RAD*(i + 20)) * radius, center.y + cos(DEG2RAD*(i + 20)) * radius);
        }
    rlEnd();

    rlDisableTexture();
}
```

Рисунок 3.19 – Реалізація функцій відображення Кола

Функція DrawCircleLines 2 значення про точку центра кола у просторі, радіус та колір, яким необхідно зафарбувати коло та створює коло за допомогою примітивів ліній. Функція DrawRectangle приймає 4 значення про точку початку прямокутника у просторі, значення висота та ширини та колір,

яким необхідно зафарбувати прямокутник. Функція DrawRectangleRec приймає значення Rectangle про положення прямокутника у просторі, висоти та ширини та колір, яким необхідно зафарбувати прямокутник. Функція DrawRectangleGradient приймає 4 значення про точку початку прямокутника у просторі, значення висота та ширини та 2 кольори, яким необхідно зафарбувати прямокутник градієнтом. Реалізація цих функцій зображена на рисунку 3.20.

```
// Draw circle outline
void DrawCircleLines(int centerX, int centerY, float radius, Color color){
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);

        // NOTE: Circle outline is drawn pixel by pixel every degree (0 to 360)
        for (int i = 0; i < 360; i += 10)
        {
            rlVertex2f(centerX + sin(DEG2RAD*i)*radius, centerY + cos(DEG2RAD*i)*radius);
            rlVertex2f(centerX + sin(DEG2RAD*(i + 10)) * radius, centerY + cos(DEG2RAD*(i + 10))*radius);
        }
    rlEnd();
}

// Draw a color-filled rectangle
void DrawRectangle(int posX, int posY, int width, int height, Color color){
    Vector2 position = { (float)posX, (float)posY };
    Vector2 size = { (float)width, (float)height };

    DrawRectangleV(position, size, color);
}

// Draw a color-filled rectangle
void DrawRectangleRec(Rectangle rec, Color color){
    DrawRectangle(rec.x, rec.y, rec.width, rec.height, color);
}

// Draw a gradient-filled rectangle
// NOTE: Gradient goes from bottom (color1) to top (color2)
void DrawRectangleGradient(int posX, int posY, int width, int height, Color color1, Color color2){
    rlBegin(RL_TRIANGLES);
        rlColor4ub(color1.r, color1.g, color1.b, color1.a); rlVertex2i(posX, posY);
        rlColor4ub(color2.r, color2.g, color2.b, color2.a); rlVertex2i(posX, posY + height);
        rlColor4ub(color2.r, color2.g, color2.b, color2.a); rlVertex2i(posX + width, posY + height);

        rlColor4ub(color1.r, color1.g, color1.b, color1.a); rlVertex2i(posX, posY);
        rlColor4ub(color2.r, color2.g, color2.b, color2.a); rlVertex2i(posX + width, posY + height);
        rlColor4ub(color1.r, color1.g, color1.b, color1.a); rlVertex2i(posX + width, posY);
    rlEnd();
}
```

Рисунок 3.20 – Реалізація функцій відображення Кола лініями та Прямокутника

Функція DrawRectangleV приймає 2 Vector2 значення про точку початку прямокутника у просторі, Vector2 висоти та ширини, та колір, яким необхідно зафарбувати прямокутник. Реалізація цієї функції зображена на рисунку 3.21.

```
void DrawRectangleV(Vector2 position, Vector2 size, Color color)
{
    rlEnableTexture(GetDefaultTexture().id); // Default white texture
    rlBegin(RL_QUADS);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlNormal3f(0.0f, 0.0f, 1.0f);
        rlTexCoord2f(0.0f, 0.0f);
        rlVertex2f(position.x, position.y);
        rlTexCoord2f(0.0f, 1.0f);
        rlVertex2f(position.x, position.y + size.y);
        rlTexCoord2f(1.0f, 1.0f);
        rlVertex2f(position.x + size.x, position.y + size.y);
        rlTexCoord2f(1.0f, 0.0f);
        rlVertex2f(position.x + size.x, position.y);
    rlEnd();
    rlDisableTexture();
}
```

Рисунок 3.21– Реалізація функцій відображення Прямокутника

Функція DrawRectangleLines приймає 4 значення про точку початку прямокутника у просторі, висоти, ширини, та колір, яким необхідно зафарбувати прямокутник та створює його за допомогою примітивів ліній. Функція DrawTriangle приймає 3 значення Vector2 про точки трикутника та колір, яким необхідно зафарбувати трикутник. Реалізація цієї функції зображена на рисунку 3.22.

```
void DrawRectangleV(Vector2 position, Vector2 size, Color color)
{
    rlEnableTexture(GetDefaultTexture().id); // Default white texture
    rlBegin(RL_QUADS);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlNormal3f(0.0f, 0.0f, 1.0f);
        rlTexCoord2f(0.0f, 0.0f);
        rlVertex2f(position.x, position.y);
        rlTexCoord2f(0.0f, 1.0f);
        rlVertex2f(position.x, position.y + size.y);
        rlTexCoord2f(1.0f, 1.0f);
        rlVertex2f(position.x + size.x, position.y + size.y);
        rlTexCoord2f(1.0f, 0.0f);
        rlVertex2f(position.x + size.x, position.y);
    rlEnd();
    rlDisableTexture();
}

// Draw rectangle outline
// NOTE: On OpenGL 3.3 and ES2 we use QUADS to avoid drawing order issues (view rlglDraw)
void DrawRectangleLines(int posX, int posY, int width, int height, Color color)
{
    DrawRectangle(posX, posY, width, 1, color);
    DrawRectangle(posX + width - 1, posY + 1, 1, height - 2, color);
    DrawRectangle(posX, posY + height - 1, width, 1, color);
    DrawRectangle(posX, posY + 1, 1, height - 2, color);
}

// Draw a triangle
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color)
{
    rlEnableTexture(GetDefaultTexture().id); // Default white texture
    rlBegin(RL_QUADS);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2f(v1.x, v1.y);
        rlVertex2f(v2.x, v2.y);
        rlVertex2f(v2.x, v2.y);
        rlVertex2f(v3.x, v3.y);
    rlEnd();
    rlDisableTexture();
}
```

Рисунок 3.22 – Реалізація функцій відображення Поямлкуьгтка та Трикутника

Функція DrawTriangleLines приймає 3 значення Vector2 про точки трикутника та колір, яким необхідно зафарбувати трикутник. Функція DrawPoly приймає Vector2 про точки полігона та колір, яким необхідно зафарбувати полігон. Реалізація цих функцій зображена на рисунку 3.23.

```
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color){
    rlBegin(RL_LINES);
        rlColor4ub(color.r, color.g, color.b, color.a);
        rlVertex2f(v1.x, v1.y);
        rlVertex2f(v2.x, v2.y);
        rlVertex2f(v2.x, v2.y);
        rlVertex2f(v3.x, v3.y);
        rlVertex2f(v3.x, v3.y);
        rlVertex2f(v1.x, v1.y);
    rlEnd();
}

// Draw a regular polygon of n sides (Vector version)
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color){
    if (sides < 3) sides = 3;

    rlPushMatrix();
        rlTranslatef(center.x, center.y, 0.0);
        rlRotatef(rotation, 0, 0, 1);
        rlBegin(RL_TRIANGLES);
            for (int i = 0; i < 360; i += 360/sides)
            {
                rlColor4ub(color.r, color.g, color.b, color.a);

                rlVertex2i(0, 0);
                rlVertex2f(sin(DEG2RAD*i)*radius, cos(DEG2RAD*i)*radius);
                rlVertex2f(sin(DEG2RAD*(i + 360/sides))*radius, cos(DEG2RAD*(i + 360/sides))*radius);
            }
        rlEnd();
    rlPopMatrix();
}
```

Рисунок 3.23 – Реалізація функцій відображення Трикутника та Полігона

3.7 Розробка модуля роботи з зображеннями, текстурами, матеріалами та шейдерами

Модуль складається з 24 основних функцій:

а) LoadImage – ініціалізує та зберігає представлення зображення у пам’яті;

б) LoadTexture – ініціалузує текстуру з переданого шляху;

в) LoadTextureFromImage – приймає значення шляху до зображення, ініціалізує зображення, потім з ньому створює текстуру;

г) UnloadImage – приймає значення зображення та ініціалізує текстуру напряму;

д) UnloadTexture – видаляє представлення зображення з пам’яті;

е) GetTextureData – повертає властивості переданої текстури;

ж) UpdateTexture – відправляє запити до графічного контексту на обновлення текстури;

з) DrawTexture – відображає передану текстуру на дисплеї;

и) ImageCopy – створює копію представлення переданого зображення у пам’ті;

к) ImageResize – змінює значення ширини та висоти переданого зображення;

л) ImageDraw – відображає передане зображення на дисплей;

м) LoadMaterial – ініціалізує матеріал;

н) LoadDefaultMaterial – створює матеріал по замовчуванню;

о) UnloadMaterial – видаляє представлення матеріалу з пам’яті;

п) LoadShader – ініціалізує шейдер;

р) GetDefaultShader – створює шейдер по замовчуванню;

с) UnloadShader – видаляє представлення шейдеру з пам’яті.

Список функцій модуля для роботи з зображеннями, текстурами, матеріалами та шейдерами з поясненнями у коментарях у заголовочному файлу зображенно на рисунку 3.24.

```
Image LoadImage(const char *fileName);
// Load an image as texture into GPU memory
Texture2D LoadTexture(const char *fileName);
// Load a texture from image data
Texture2D LoadTextureFromImage(Image image);
// Unload image from CPU memory (RAM)
void UnloadImage(Image image);
// Unload texture from GPU memory
void UnloadTexture(Texture2D texture);
// Get pixel data from GPU texture and return an Image
Image GetTextureData(Texture2D texture);
// Create an image duplicate (useful for transformations)
Image ImageCopy(Image image);
// Crop an image to a defined rectangle
void ImageCrop(Image *image, Rectangle crop);
// Resize and image (bilinear filtering)
void ImageResize(Image *image, int newWidth, int newHeight);
// Draw a source image within a destination image
void ImageDraw(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec);
// Flip image vertically
void ImageFlipVertical(Image *image);
// Flip image horizontally
void ImageFlipHorizontal(Image *image);
// Modify image color: tint
void ImageColorTint(Image *image, Color color);
// Modify image color: invert
void ImageColorInvert(Image *image);
// Modify image color: contrast (-100 to 100)
void ImageColorContrast(Image *image, float contrast);
// Modify image color: brightness (-255 to 255)
void ImageColorBrightness(Image *image, int brightness);
// Update GPU texture with new data
void UpdateTexture(Texture2D texture, void *pixels);

// Draw a Texture2D
void DrawTexture(Texture2D texture, int posX, int posY, Color tint);

// Load material data (from file)
Material LoadMaterial(const char *fileName);
// Load default material (uses default models shader)
Material LoadDefaultMaterial(void);
// Unload material textures from VRAM
void UnloadMaterial(Material material);

// Load a custom shader and bind default locations
Shader LoadShader(char *vsFileName, char *fsFileName);
// Unload a custom shader from memory
void UnloadShader(Shader shader);
```

Рисунок 3.24 – Список функція модуля

Функція LoadImage приймає значення шляху до зображення, ініціалізує та зберігає представлення зображення у пам’яті. Реалізація зображена на рисунку 3.25.

```
Image LoadImage(const char *fileName){
    Image image;
    // Initialize image default values
    image.data = NULL;
    image.width = 0;
    image.height = 0;
    image.mipmaps = 0;
    image.format = 0;
    if ((strcmp(GetExtension(fileName),"png") == 0) ||
        (strcmp(GetExtension(fileName),"bmp") == 0) ||
        (strcmp(GetExtension(fileName),"tga") == 0) ||
        (strcmp(GetExtension(fileName),"jpg") == 0) ||
        (strcmp(GetExtension(fileName),"gif") == 0) ||
        (strcmp(GetExtension(fileName),"psd") == 0) ||
        (strcmp(GetExtension(fileName),"pic") == 0))
    {
        int imgWidth = 0;
        int imgHeight = 0;
        int imgBpp = 0;

        // NOTE: Using stb_image to load images (Supports: BMP, TGA, PNG, JPG, ...)
        image.data = stbi_load(fileName, &imgWidth, &imgHeight, &imgBpp, 0);

        image.width = imgWidth;
        image.height = imgHeight;
        image.mipmaps = 1;

        if (imgBpp == 1) image.format = UNCOMPRESSED_GRAYSCALE;
        else if (imgBpp == 2) image.format = UNCOMPRESSED_GRAY_ALPHA;
        else if (imgBpp == 3) image.format = UNCOMPRESSED_R8G8B8;
        else if (imgBpp == 4) image.format = UNCOMPRESSED_R8G8B8A8;
    }
    return image;
}
```

Рисунок 3.25 – Реалізація функції LoadImage

Функція LoadTexture приймає значення шляху до зображення, ініціалізує зображення, потім з ньому створює текстуру. Функція LoadTextureFromImage приймає значення зображення та ініціалізує текстуру напряму. Функція

UnloadImage видаляє представлення зображення з пам’яті. Реалізація функцій роботи з зображеннями та текстурами зображено на рисунку 3.26.

```
Texture2D LoadTexture(const char *fileName){
    Texture2D texture;

    Image image = LoadImage(fileName);

    if (image.data != NULL)
    {
        texture = LoadTextureFromImage(image);
        UnloadImage(image);
    }
    else
    {
        texture.id = 0;
    }

    return texture;
}

Texture2D LoadTextureFromImage(Image image){
    Texture2D texture;

    // Init texture to default values
    texture.id = 0;
    texture.width = 0;
    texture.height = 0;
    texture.mipmaps = 0;
    texture.format = 0;

    texture.id = rlglLoadTexture(image.data, image.width, image.height, image.format, image.mipmaps);

    texture.width = image.width;
    texture.height = image.height;
    texture.mipmaps = image.mipmaps;
    texture.format = image.format;

    return texture;
}

// Unload image from CPU memory (RAM)
void UnloadImage(Image image)
{
    free(image.data);
}
```

Рисунок 3.26 – Реалізація функції роботи з зображення та текстурами

Функція UnloadTexture видаляє представлення текстури з пам’яті. Функція GetTextureData повертає властивості переданої текстури. Функція UpdateTexture відправляє запити до графічного контексту на обновлення

текстури. Функція DrawTexture відображає передану текстуру на дисплеї. Реалізація функцій роботи з текстурами зображено на рисунку 3.27.

```
// Unload texture from GPU memory
void UnloadTexture(Texture2D texture)
{
    if (texture.id != 0)
    {
        rlDeleteTextures(texture.id);
    }
}

Image GetTextureData(Texture2D texture){
    Image image;
    image.data = NULL;

    if (texture.format < 8)
    {
        image.data = rlglReadTexturePixels(texture);

        if (image.data != NULL)
        {
            image.width = texture.width;
            image.height = texture.height;
            image.mipmaps = 1;

            image.format = texture.format;
        }
    }

    return image;
}

// Draw a Texture2D
void DrawTexture(Texture2D texture, int posX, int posY, Color tint){
    DrawTextureEx(texture, (Vector2){ (float)posX, (float)posY }, 0, 1.0f, tint);
}
```

Рисунок 3.27 – Реалізація функції роботи з текстурами

Функція ImageCopy створює копію представлення переданого зображення. Функція ImageResize приймає 3 значення: оригінальне зображення, нове значення ширини та нове значення висоти та змінює значення ширини та

висоти переданого зображення. Реалізація функцій роботи з зображеннями зображено на рисунку 3.28.

```
Image ImageCopy(Image image){
    Image newImage;
    int size = image.width*image.height;
    switch (image.format)
    {
        case UNCOMPRESSED_GRAYSCALE: newImage.data = (unsigned char *)malloc(size); break;              // 8 bit per pixel (no alpha)
        case UNCOMPRESSED_GRAY_ALPHA: newImage.data = (unsigned char *)malloc(size*2); size *= 2; break; // 16 bpp (2 channels)
        case UNCOMPRESSED_R5G6B5: newImage.data = (unsigned short *)malloc(size); size *= 2; break;      // 16 bpp
        case UNCOMPRESSED_R8G8B8: newImage.data = (unsigned char *)malloc(size*3); size *= 3; break;     // 24 bpp
        case UNCOMPRESSED_R5G5B5A1: newImage.data = (unsigned short *)malloc(size); size *= 2; break;    // 16 bpp (1 bit alpha)
        case UNCOMPRESSED_R4G4B4A4: newImage.data = (unsigned short *)malloc(size); size *= 2; break;    // 16 bpp (4 bit alpha)
        case UNCOMPRESSED_R8G8B8A8: newImage.data = (unsigned char *)malloc(size*4); size *= 4; break;   // 32 bpp
    }
    if (newImage.data != NULL)
    {
        memcpy(newImage.data, image.data, size);
        newImage.width = image.width;
        newImage.height = image.height;
        newImage.mipmaps = image.mipmaps;
        newImage.format = image.format;
    }

    return newImage;
}

void ImageResize(Image *image, int newWidth, int newHeight) {
    // Get data as Color pixels array to work with it
    Color *pixels = GetImageData(*image);
    Color *output = (Color *)malloc(newWidth*newHeight*sizeof(Color));

    // NOTE: Color data is casted to (unsigned char *), there shouldn't been any problem...
    stbir_resize_uint8((unsigned char *)pixels, image->width, image->height, 0, (unsigned char *)output, newWidth, newHeight, 0, 4);

    int format = image->format;

    UnloadImage(*image);

    *image = LoadImageEx(output, newWidth, newHeight);
    ImageFormat(image, format);  // Reformat 32bit RGBA image to original format

    free(output);
    free(pixels);
}
```

Рисунок 3.28 – Реалізація функції роботи з зображеннями

Функція ImageDraw відображає передане зображення на дисплей. Реалізація функцій ImageDraw зображено на рисунку 3.29.

```
void ImageDraw(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec) {
    if (srcRec.x < 0) srcRec.x = 0;
    if (srcRec.y < 0) srcRec.y = 0;
    if ((srcRec.x + srcRec.width) > src.width){
        srcRec.width = src.width - srcRec.x;
    }
    if ((srcRec.y + srcRec.height) > src.height){
        srcRec.height = src.height - srcRec.y;
    }
    if (dstRec.x < 0) dstRec.x = 0;
    if (dstRec.y < 0) dstRec.y = 0;

    if ((dstRec.x + dstRec.width) > dst->width){
        dstRec.width = dst->width - dstRec.x;
    }

    if ((dstRec.y + dstRec.height) > dst->height){
        dstRec.height = dst->height - dstRec.y;
    }

    // Get dstination image data as Color pixels array to work with it
    Color *dstPixels = GetImageData(*dst);

    Image srcCopy = ImageCopy(src);     // Make a copy of source image to work with it
    ImageCrop(&srcCopy, srcRec);        // Crop source image to desired source rectangle

    // Scale source image in case destination rec size is different than source rec size
    if ((dstRec.width != srcRec.width) || (dstRec.height != srcRec.height))
    {
        ImageResize(&srcCopy, dstRec.width, dstRec.height);
    }

    Color *srcPixels = GetImageData(srcCopy);

    UnloadImage(srcCopy);

    free(srcPixels);
    free(dstPixels);
}
```

Рисунок 3.29 – Реалізація функції ImageDraw

Функція LoadMaterial приймає значення шлаху матеріалу та ініціалізує для подальшого використання. Функція LoadDefaultMaterial ініціалізує матеріал по заповчуванню. Функція UnloadMaterial видаляє представлення матеріалу з пам’яті. Реалізація функцій роботи з матеріалами зображено на рисунку 3.30.

```
// Load material data (from file)
Material LoadMaterial(const char *fileName){
    Material material = { 0 };

    if (strcmp(GetExtension(fileName), "mtl") == 0) material = LoadMTL(fileName);

    return material;
}

// Load default material (uses default models shader)
Material LoadDefaultMaterial(void){
    Material material = { 0 };

    material.shader = GetDefaultShader();
    material.texDiffuse = GetDefaultTexture();      // White texture (1x1 pixel)
    //material.texNormal;           // NOTE: By default, not set
    //material.texSpecular;         // NOTE: By default, not set

    material.colDiffuse = WHITE;    // Diffuse color
    material.colAmbient = WHITE;    // Ambient color
    material.colSpecular = WHITE;   // Specular color

    material.glossiness = 100.0f;   // Glossiness level

    return material;
}

// Unload material from memory
void UnloadMaterial(Material material){
    rlDeleteTextures(material.texDiffuse.id);
    rlDeleteTextures(material.texNormal.id);
    rlDeleteTextures(material.texSpecular.id);
}
```

Рисунок 3.30 – Реалізація функції роботи з матеріалами

Функція LoadShader приймає значення шлаху точок та шейдеру та ініціалізує для подальшого використання. Функція GetDefaultShader ініціалізує шейдер по заповчуванню. Функція UnloadShader видаляє представлення шейдеру з пам’яті. Реалізація функцій роботи з матеріалами зображено на рисунку 3.31.

```
Shader LoadShader(char *vsFileName, char *fsFileName){
    Shader shader = { 0 };

    // Shaders loading from external text file
    char *vShaderStr = ReadTextFile(vsFileName);
    char *fShaderStr = ReadTextFile(fsFileName);

    if ((vShaderStr != NULL) && (fShaderStr != NULL))
    {
        shader.id = LoadShaderProgram(vShaderStr, fShaderStr);

        // After shader loading, we try to load default location names
        if (shader.id != 0) LoadDefaultShaderLocations(&shader);

        // Shader strings must be freed
        free(vShaderStr);
        free(fShaderStr);
    }

    return shader;
}

// Unload a custom shader from memory
void UnloadShader(Shader shader)
{
    if (shader.id != 0)
    {
        rlDeleteShader(shader.id);
    }
}

Shader GetDefaultShader(void)
{
#if defined(GRAPHICS_API_OPENGL_33) || defined(GRAPHICS_API_OPENGL_ES2)
    return defaultShader;
#else
    Shader shader = { 0 };
    return shader;
#endif
}
```

Рисунок 3.31 – Реалізація функції роботи з шейдерами

3.8 Висновки

У цьому розділі було проаналізовано мови програмування та інтегровані середовища розробки. Було обрано Microsoft Visual Studio Code та C для розробки програмної бібліотеки.

Створено основні програмні модулі бібліотеки, які містять необхідні функціональність для розробки ігор: модуль керуванням вікнами, модуль рендерингу на дисплей, модуль відображення графічних примітивів та модуль роботи з зображенями, текстурами, матеріалами та шейдерами.

4 ТЕСТУВАННЯ БІБЛІОТЕКИ

4.1 Аналіз методів і засобів тестування

Тестування програмного забезпечення – перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином.

Виділяють три рівні тестування:

а) модульне;

б) інтеграційне;

в) системне.

Модульне тестування – тестування, яке має на меті перевірити працездатність окремих модулів (функції або класу). Модульне тестування зазвичай виконується незалежно для кожного програмного модуля і є, мабуть, найбільш поширеним видом тестування, особливо для систем малих і середніх розмірів. Як критерій повноти використовується відсоток покриття тестами ключових елементів модуля (оператори, гілки логічних умов і т.д.).

Модульні тести перевіряють, що певні дії на модуль призводять до бажаного результату. Як правило. Модульні тести створюються з використанням методу «білого ящика». При наявності залежностей модуля, що тестується від інших модулів замість них використовуються так звані mock-об’єкти, що надають фіктивну реалізацію їх інтерфейсів.

Модульне тестування дозволяє програмістам безпечно проводити рефакторинг, будучи впевненими, що змінений модуль як і раніше працює коректно. Модульні тести можуть використовуватися в якості специфікації, так як при тестуванні фактично перевіряється їх очікувану поведінку.

Інтеграційне тестування (integration testing) – одна з фаз тестування ПО, при якому окремі програмні модулі об’єднуються і тестуються в комплексі. Зазвичай інтеграційне тестування проводиться після модульного тестування і передує системному тестування. Метою даного виду тестування є знаходження проблем взаємодії модулів взаємодії модулів (компонент, підсистем).

Системне тестування – це тестування повною, інтегрованої системи з метою перевірки її відповідності вимогам до системи і показниками якості. Системні тести враховують такі аспекти системи, як стійкість в роботі, продуктивність, відповідність системи очікуванням користувача і т.п. Для визначення повноти системного тестування оцінюються виконання всіх можливих сценаріїв роботи (як штатних, так і позаштатних), різні методи взаємодії системи з зовнішнім середовищем і т.д.

Важливо розуміти, що тестування ПЗ включає не тільки власне проведення тестів, але і багато інших дій, які пов’язані з процесом забезпечення якості:

а) аналіз і планування;

б) розробку тестових сценаріїв;

в) оцінку критеріїв закінчення тестування;

г) написання звітів;

д) рецензування документації (в тому числі і вихідного коду);

е) проведення статичного аналізу.

Характеристики якості ПЗ:

а) функціональність;

б) надійність;

в) зручність використання;

г) ефективність;

д) зручність супроводу;

е) портативність.

Технології тестування використовуються при застосуванні тих чи інших методів тестування. Серед методів тестування зазвичай виділяють два найбільш поширених:

а) метод «чорної скриньки»;

б) метод «білої скриньки».

Тестування чорної скриньки – це вид тестування програмного забезпечення, коли від тестувальників не вимагається знання про код або внутрішню структуру програмного забезпечення. Метод тестування чорної скриньки заснований на тестуванні ПЗ з різними входами і порівнянні результатів з очікуваними.

Тестування білої скриньки – це метод тестування, що призначений для тестування зі знанням внутрішньої роботи ПЗ. Цей метод використовується в модульному тестуванні. Тестування білої скриньки призначене для тестування коду, тестів, гілок, шляхи, рішень і потоку даних тестованої програмі. Тестування білої скриньки і тестування чорної скриньки доповнюють одне одного, оскільки кожен з підходів до тестування може виявити певну категорію помилок.

4.2 Тестування бібліотеки

Тестування програмного модуля проводиться за методикою «чорної скриньки». Вона базується на використанні шаблонів тестування (тест-кейсів). Це означає, що буде створено декілька тест-кейсів для перевірки правильності роботи основних функцій програмного додатку. Розроблені для перевірки правильності роботи програмного додатку тест-кейси описано в таблиці 4.1.

Таблиця 4.1 – Тест-кейси

<table>
<tr><td rowspan="3">Код тест-кейса</td><td colspan="3">Опис тест-кейса</td></tr>
<tr><td colspan="2">Хід тестування</td><td>Очікуваний результат</td></tr>
<tr><td>Дата тестування</td><td colspan="2">Результат</td></tr>
<tr><td rowspan="3">1</td><td colspan="3">Тестування створення вікна</td></tr>
<tr><td colspan="2">1. Ініціалізація вікна. Передамо значення назви вікна як «Test01».
2. Реалізувати ігровий цикл.
3. Реалізувати функція очищення ресурсів після закінчення ігрового циклу</td><td>В результаті запуску програми було створенно вікно з назвою «Test01» (рисунок 4.1).</td></tr>
<tr><td>19.11.21</td><td colspan="2">Пройдено</td></tr>
<tr><td rowspan="3">2</td><td colspan="3">Тестування відображення графічний примітивів</td></tr>
<tr><td colspan="2">1. Ініціалізація вікна. Передамо значення назви вікна як «Test02».
2. Реалізувати ігровий цикл.
3. У ігровому циклі реалізуємо створення «Пікселя», «Лінії», «Прямокутника», «Трикутника» та «Коло» по середині вікна
4. Реалізувати функція очищення ресурсів після закінчення ігрового циклу</td><td>В результаті запуску програми було створенно вікно з назвою «Test01» та усі передані графічні примітиви по середині дисплею (рисунок 4.2).</td></tr>
<tr><td>20.11.21</td><td colspan="2">Пройдено</td></tr>
</table>

Продовження табл. 4.1

3	Тестування відображення текстури	
	1. Ініціалізація вікна. Передамо значення назви вікна як «Test03». 2. Реалізувати ігровий цикл. 3. У ігровому циклі реалізуємо відображення текстури. 4. Реалізувати функція очищення ресурсів після закінчення ігрового циклу	В результаті запуску програми було створенно вікно з назвою «Test03» та передана текстура по середині дисплею (рисунок 4.3).
	21.11.21	Пройдено
4	Тестування відображення мешу	
	1. Ініціалізація вікна. Передамо значення назви вікна як «Test04». 2. Реалізувати ігровий цикл. 3. У ігровому циклі реалізуємо створення мешу по середині вікна 4. Реалізувати функція очищення ресурсів після закінчення ігрового циклу	В результаті запуску програми було створенно вікно з назвою «Test04» та відображено меш по середині дисплею (рисунок 4.4).
	23.11.21	Пройдено
5	Тестування відображення тривимірної моделі	
	1. Ініціалізація вікна. Передамо значення назви вікна як «Test05». 2. Реалізувати ігровий цикл. 3. У ігровому циклі реалізуємо створення моделі з матеріалами по замовчуванню по середині вікна 4. Реалізувати функція очищення ресурсів після закінчення ігрового циклу	В результаті запуску програми було створенно вікно з назвою «Test05» та відображено моделі з матеріалами по замовчуванню по середині дисплею (рисунок 4.5).
	24.11.21	Пройдено

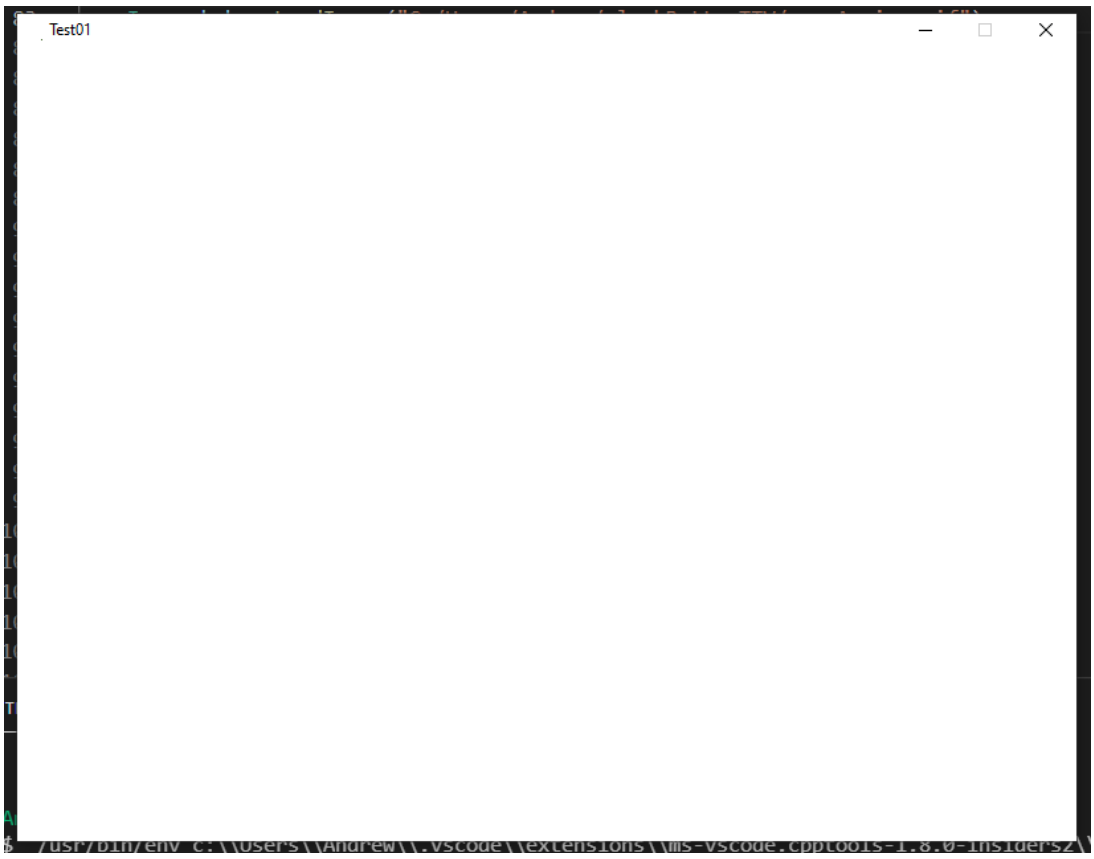


Рисунок 4.1 – Результат тестування створення вікна

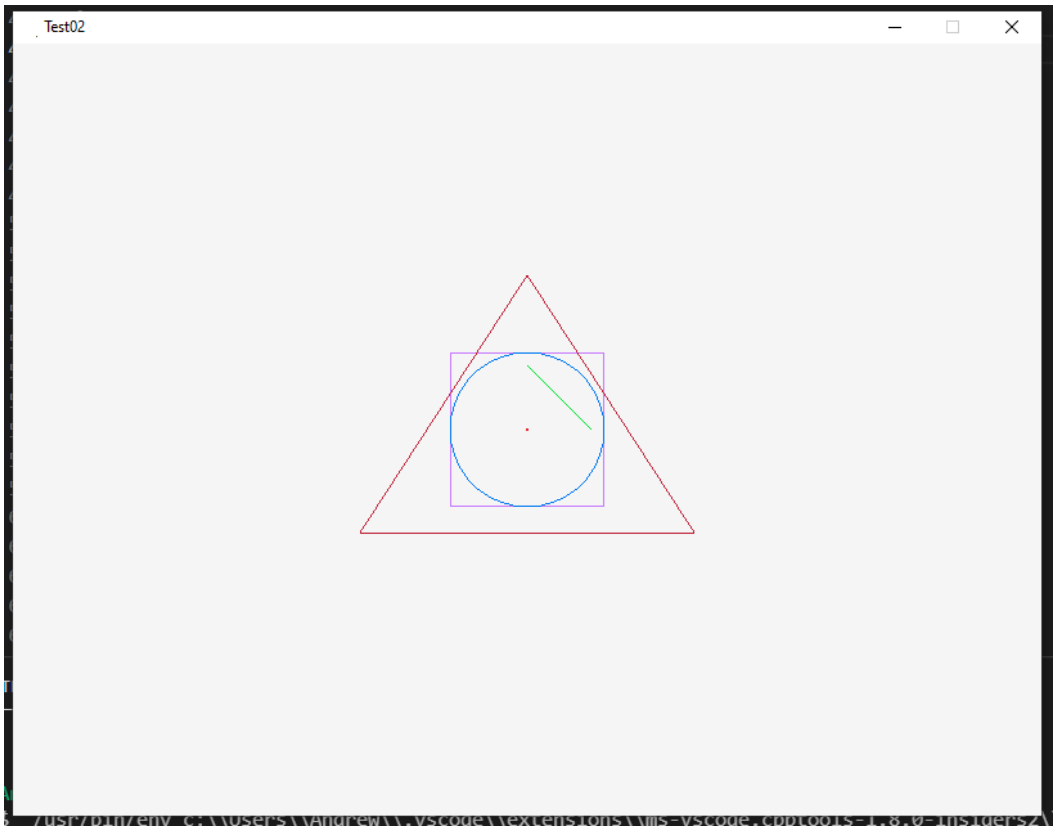


Рисунок 4.2 – Результат тестування відображення графічний примітивів

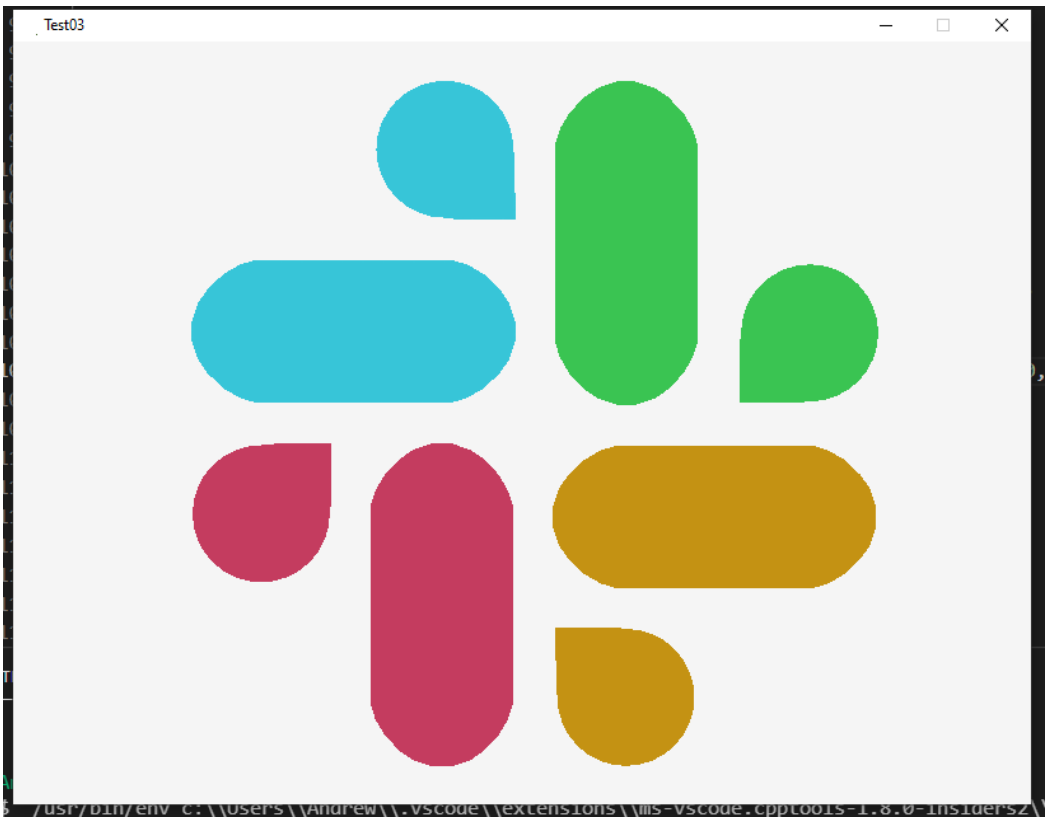


Рисунок 4.3 – Результат тестування відображення текстури

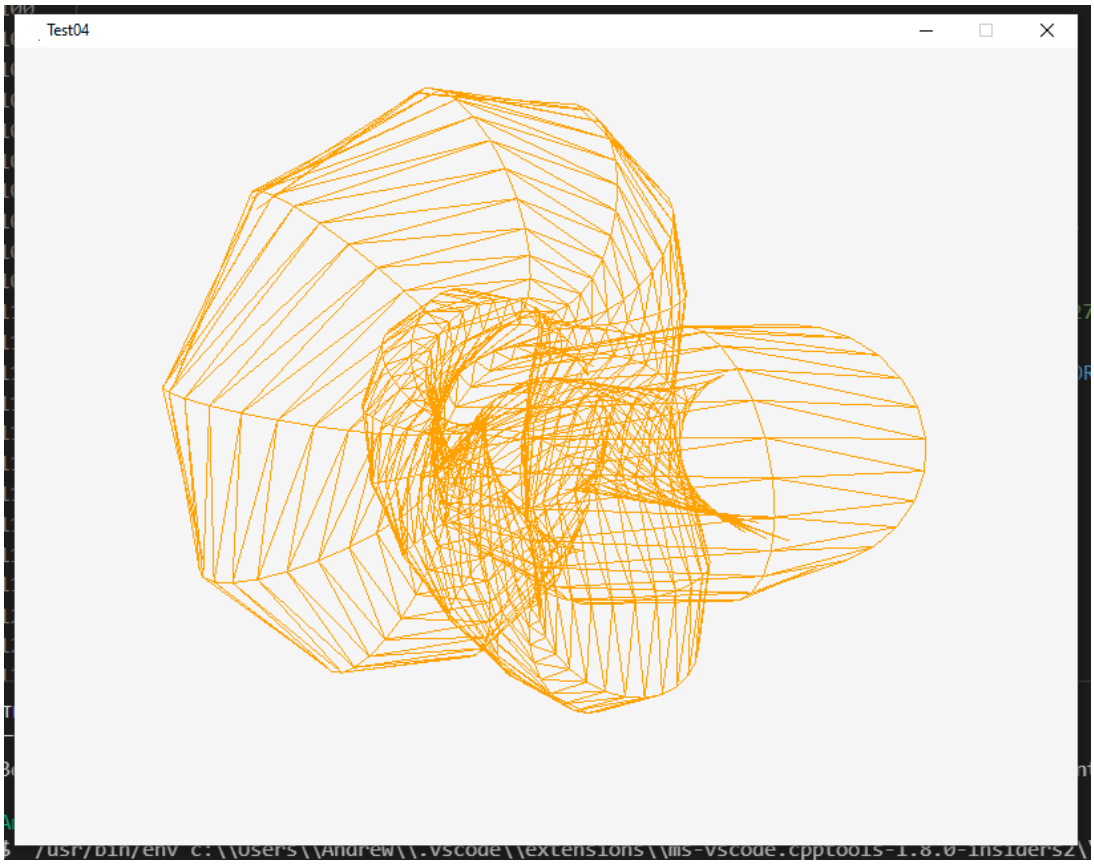


Рисунок 4.4 – Результат тестування відображення мешу

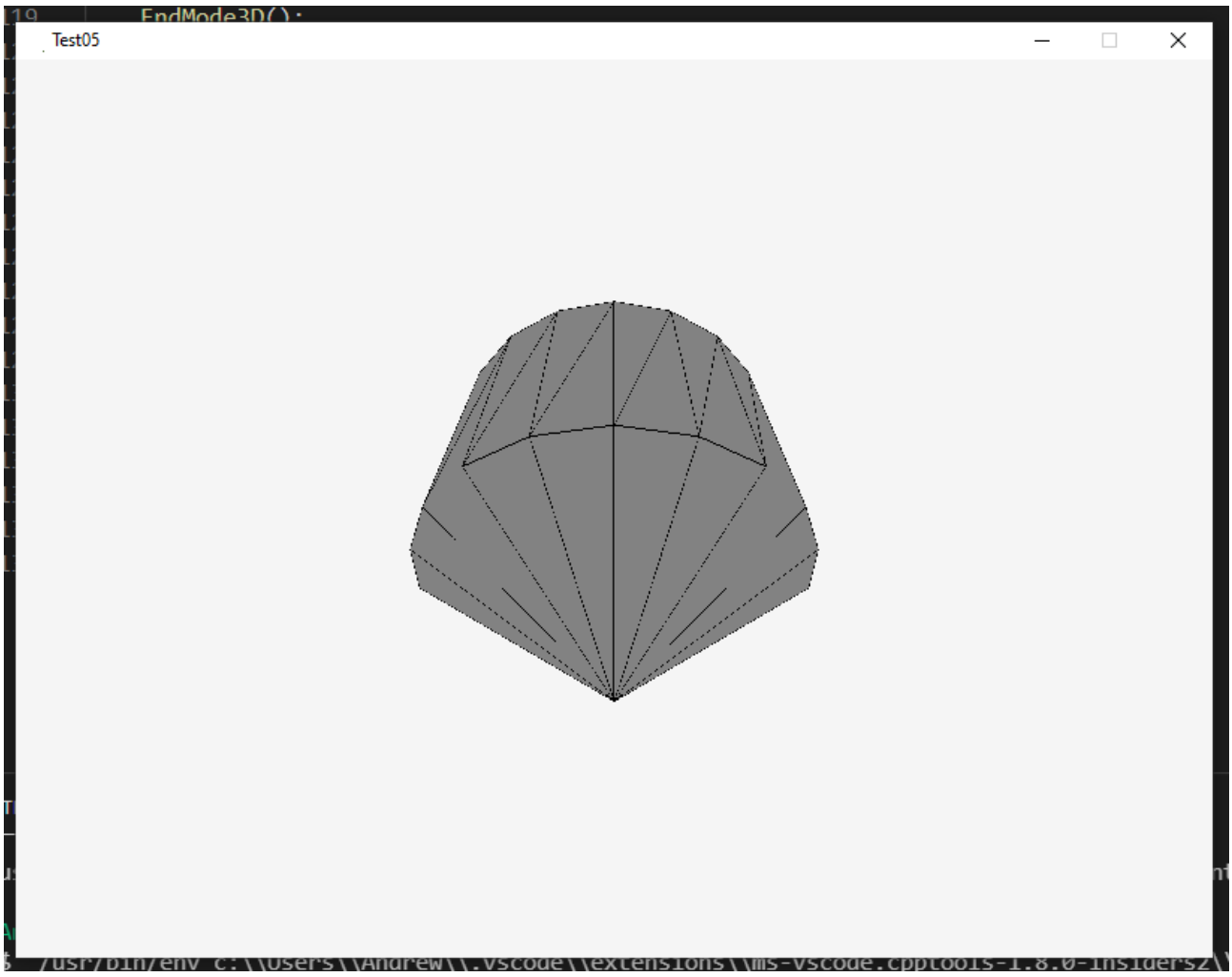


Рисунок 4.5 – Результат тестування відображення моделі

4.3 Розробка інструкції користувача

Для покращення користувацького досвіду та пришвидшення навчанню з бібліотекою необхідно розробити інструкції користувача, як б давала змогу розробника при необхідності знаходити функції та зрозуміти їх функціонал [19].

Розроблювальная бібліотека має файл заголовку, у якому міститься усі функція та роз’яснення їх функцоналу (рисунок 4.6), так що, при необхідності кожен користувач має змогу відкрити цей файл та знайти необхідну інформацію.

```
void CloseWindow(void);                                 // Close Window and Terminate Context
bool WindowShouldClose(void);                           // Detect if KEY_ESCAPE pressed or Close icon pressed
bool IsWindowMinimized(void);                           // Detect if window has been minimized (or lost focus)
void ToggleFullscreen(void);                            // Fullscreen toggle (only PLATFORM_DESKTOP)
int GetScreenWidth(void);                               // Get current screen width
int GetScreenHeight(void);                              // Get current screen height

void ShowCursor(void);                                  // Shows cursor
void HideCursor(void);                                  // Hides cursor
bool IsCursorHidden(void);                              // Returns true if cursor is not visible
void EnableCursor(void);                                // Enables cursor
void DisableCursor(void);                               // Disables cursor

void ClearBackground(Color color);                      // Sets Background Color
void BeginDrawing(void);                                // Setup drawing canvas to start drawing
void EndDrawing(void);                                  // End canvas drawing and Swap Buffers (Double Buffering)

void Begin2dMode(Camera2D camera);                      // Initialize 2D mode with custom camera
void End2dMode(void);                                   // Ends 2D mode custom camera usage
void Begin3dMode(Camera camera);                        // Initializes 3D mode for drawing (Camera setup)
void End3dMode(void);                                   // Ends 3D mode and returns to default 2D orthographic mode
void BeginTextureMode(RenderTexture2D target);          // Initializes render texture for drawing
void EndTextureMode(void);                              // Ends drawing to render texture
```

Рисунок 4.6 – Вміст файлу заголовка бібліотеки та роз’яснення функцій

У таблиці 4.2 та 4.3 наведено мінімальну та рекомендовану конфігурацію персонального комп’ютера для запуску програми.

Таблиця 4.2 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) або ARM процесор з тактовою частотою 1 ГГц
Об’єм оперативної пам’яті	512 МБ оперативної пам’яті
Розмір жорсткого диску	8 МБ
Графічний пристрій	OpenGL v. 1-2.1

Таблиця 4.3 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86), 64-розрядний (x64) або ARM процесор з тактовою частотою 2 ГГц
Об’єм оперативної пам’яті	1 ГБ
Розмір жорсткого диску	8 МБ
Графічний пристрій	OpenGL v. ES – 4.1

Програмна бібліотека працює під управлінням сімейства операційних систем Windows, Linux та Android, а також для платформи Web (HTML5).

4.4 Висновки

Серед розглянутих метидів тестування для тестування програмної бібліотеки було обрано стратегію тестування «чорної скриньки». Результат тестування показав повну працездатність програмного продукту та відповідність поставленому технічному завданню.

Визначено мінімальну та рекомендовану конфігурації персональних комп’ютерів необхідні для коректної роботи бібліотеки.

Розроблено інструкцію користувача по встановленню програмного продукту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх, що дозволить пришвидшити розробку подібних програмних продуктів у майбутньому.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету кафедри програмного забезпечення: Войтко Вікторія Володимирівна, Майданюк Володимир Павлович, Коваленко Олена Олександрівна. Для проведення технологічного аудиту було використано таблицю 5.1 [23] в якій за п'ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 5.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Актив на конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовую-ться у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

	виробництво та реалізацію продукту	вимагає значних коштів та часу	та часу		

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Войтко В. В.	Майданюк В. П.	Коваленко О. О.
	Бали, виставлені експертами:		
1	3	3	3
2	2	2	3
3	2	2	4
4	4	3	3
5	4	4	3
6	3	4	2
7	1	1	3
8	4	4	3
9	3	3	2
10	4	4	3
11	4	3	4
12	4	4	4
Сума балів	$СБ_1$=38	$СБ_2$=33	$СБ_3$=37
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{38 + 33 + 37}{3} = 36$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 36 бали, що згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Нову розробку можна реалізувати підприємствам, які розробляють гру з графічним інтерфейсом.

Порівняємо нову розробку з аналогом, який існує на ринку. В якості аналога для розробки було обрано Cocos2d. Основними недоліками аналога є неможливість використання при розробці тривимірних ігор. Також до недоліків можна віднести необхідність сплачувати відсоток від прибутку, якщо кількість користувачів програмного продукту, розробленого за допомогою Cocos2D, перевищує 10,000.

На відміну від аналога власна розробка дозволяє створювати програмні продукти з використанням і тривимірної графіки, а оплата ліцензії здійснюється один раз і є фіксованою.

Також система випереджає аналог за такими параметрами як: можливість розробки інтерфейсу користувача у тривимірному графічному контексті; кількість систем, які підтримуються; швидкодія; компілювання додатку для Web/HTML5; компілювання додатку для мобільних пристроїв.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Використання ОЗУ, ГБ	2	1,5	1,3	30%
Використання ЦП, %	30	23	1,3	30%
Середня кількість кадрів за секунду	60	80	1,3	30%
Середня швидкість компілювання додатку, с.	120	80	1,5	10%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{2}{1,5} = 1,3;$$

$$q_2 = \frac{30}{23} = 1,3;$$

$$q_3 = \frac{80}{60} = 1,3;$$

$$q_4 = \frac{120}{80} = 1,5.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.3}$$

$$К_{я.в.} = 1,3 \cdot 0,3 + 1,3 \cdot 0,3 + 1,3 \cdot 0,3 + 1,5 \cdot 0,1 = 1,32$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 5.5 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Таблиця 5.5 – Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Використання ОЗУ, ГБ	2	1,5
Використання ЦП, %	30	23
Середня кількість кадрів за секунду	60	80
Середня швидкість компілювання додатку, с.	120	80
Використання ОЗУ, ГБ	2	1,5
2. Економічні показники		
Ціна придбання, грн.	500	300

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \quad (5.5)$$

де P_{Hei}, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$\text{I}_{\text{е.п.}} = \frac{300}{500} = 0{,}6;$$

$$\text{К} = \frac{1{,}32}{0{,}6} = 2{,}2.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможніше, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$\text{З}_\text{О} = \frac{\text{М}}{\text{Т}_\text{р}} * t \text{ (грн)} \quad (5.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_P – число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Для розробки програмні засоби для реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх необхідно залучити програміста з посадовим окладом 8000 грн. Кількість робочих днів у місяці

складає 22, а кількість робочих днів програміста складає 22. Зведемо сумарні розрахунки до таблиця 5.6.

Таблиця 5.6 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	10000	454,5	5	2273
Програміст	8000	363,6	22	8000
Всього				10273

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 – 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (5.7)$$

$$З_д = 0{,}11 * 10273 = 1130 \text{ (грн)}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.3):

$$Н_{ЗП} = (З_о + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (10273 + 1130\ ) * \frac{22}{100} = 2508{,}6\ (грн)$$

4. Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \quad грн., \qquad (5.9)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.7 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	125	1	125
Ручка	11	1	11
CD-диск	12	1	12
Флешка	135	1	135
Всього			283
З врахуванням коефіцієнта транспортування			311,3

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного

забезпечення необхідного для проведення дослідження. В роботі використовувалось безкоштовне програмне забезпечення тому витрати на нього не враховуються.

6. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.10}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 20000 грн.

$$А = \frac{20000 \cdot 1}{2 \cdot 12} = 833{,}33$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_і} \tag{5.11}$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0,3 \cdot 160 \cdot 4,1 \cdot 0,5}{0,8} = 123$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 10273 \cdot \frac{100}{100\%} = 10273 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 10273+1130+2508,6+311,3+833,33+123+10273=25451,7$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,9.

Звідси:

$$ЗВ = \frac{25451,7}{0,9} = 28279,65 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_{1}^{n} \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N \right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100} \right) \qquad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

Р – коефіцієнт, який враховує рентабельність продукту. Р = 0,25;

х – ставка податку на прибуток. У 2021 році – 18%.

Припустимо, що при впроваджені результатів наукової розробки покращується якість програмного продукту для формування індивідуальних тренувань. Припустимо, що ціна від зросте на 50 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 500 шт., протягом другого року – на 700 шт., протягом третього року на 800 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до

складає 300 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [50 \cdot 1 + (300 + 50) \cdot 500] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 29903{,}18 \text{ грн.}$$

$$\Delta П_2 = [50 \cdot 1 + (300 + 50) \cdot (500 + 700)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 71797{,}13 \text{ грн.}$$

$$\Delta П_3 = [50 \cdot 1 + (300 + 50) \cdot (500 + 700 + 800)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 119628{,}55 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (5.15)$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 28279{,}6 = 56559{,}31$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \qquad (5.16)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t},$$

(5.17)

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

τ – t – період часу (в роках).

$$\text{ПП} = \frac{29903{,}18}{(1+0{,}2)^1} + \frac{71797{,}13}{(1+0{,}2)^2} + \frac{119628{,}55}{(1+0{,}2)^3} = 144329{,}92 \text{ грн.}$$

$$\text{Е}_{\text{абс}} = (144329{,}92 - 144630{,}06\ ) = 87770{,}61 \text{грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

(5.18)

$T_ж$ – життєвий цикл наукової розробки, роки.

$$\mathrm{E_в} = \sqrt[3]{1 + \frac{87770{,}61}{56559{,}3}} - 1 = 0{,}6 = 60\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.19}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d= (0,14…0,2);

f − показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0{,}18 + 0{,}05 = 0{,}23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{5.20}$$

$$\mathrm{T_{ок}} = \frac{1}{0{,}6} = 1{,}7 \text{ роки}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

5.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу методів реалізації користувацьких інтерфейсів для їх використання в комп''терних іграх, який є на вище середньому рівні. При порівнянні нової розробки з аналогом виявлено,

що вона є якіснішою і конкурентоспроможнішою відносно аналога, а також краще по технічним і економічним показникам.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 25451,7 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 28279,65 грн.

Вкладені інвестиції в даний проект окупляться через 1,7 роки при прогнозованому прибутку 144329,92 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі удосконалено методи реалізації користувацьких інтерфейсів для їх використання в комп’ютерних іграх та розроблено програмну бібліотеку. Роботу оформлено відповідно до вимог [26]. Ілюстративний матеріал наведено у додатку Г. Розроблена бібліотека для графічних інтерфейсів, акумулює функції керування вікнами, взаємодії з користувачем, відображення графічних примітивів, зображень, текстур та шейдерів, що дозволяє використання створених компонент в процесі компілювання та інтеграції інтерфейсу комп’ютерних ігор забезпечувати кросплафтормену реалізацію та збільшити швидкість обробки даних.

Перед розробкою було проаналізовано інформаційне забезпечення бібліотеки, проведено аналіз основних аналогів, виявлено їхні переваги та недоліки. На основі отриманих результатів було визначено функціонал власної бібліотеки, який вирішує проблеми, наявні в аналогах. Ключовими показниками є швидкодія та мультиплатформенність.

У результаті проведення варіантного аналізу було обрано мову програмування C та IDE – Microsoft Visual Studio Code.

Запропоновано метод реалізації графічних інтерфейсів у комп’ютерних іграх на різних платформах. Метод полягає у тому, що дозволить компілювати додатки для різних систем з однієї кодової бази, що дозволить пришвидшити розробку комп’ютерних ігор. Використання даного методу дозволяє прискорити розробку ігор, за рахунок відсутності потреби адаптувати код гри під кожну платформу.

Створено модель бібліотеки для графічних інтерфейсів, яка складається з модулів роботи з вікнами та графічним контекстом, рендерингу, таймеру та керування потоками, введення та дотику, камери, відображення графічних примітивів, роботи з зображеннями, текстурами, матеріалами, шейдерами та роботи зі звуком. Особливість моделі полягає у тому, що вона акумулює функції реалізовані у модулях, що дозволяє використання створених компонент в процесі компілювання та інтеграції інтерфейсу комп'ютерних ігор забезпечувати кросплафтормену реалізацію та збільшити швидкість обробки даних.

У процесі розробки було реалізовано основні модулі програмної бібліотеки. Для ефективної розробки бібілотеки було обрано архітектуру системи загалом.

Для тестування програмної бібліотеки було обрано стратегію тестування «чорної скриньки». Результат тестування показав повну працездатність програмного продукту та відповідність поставленому технічному завданню. Розроблено інструкцію користувача.

Було проведено розрахунки витрат та прибутків від впровадження розробки, термінів окупності, визначено економічний ефект. Розрахунки показали, що розробка є конкурентоспроможною на IT-ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Абрас К., Мелоні-Крічмар Д., Пріс Дж. Орієнтований на користувача дизайн. Таузанд-Оакс : Sage Publications, 2004. – 445 с.

2. Вайншенк С. Вайншенк Ю., Сара С. Enterprise-Wide GUI Design. Ньо-Джерсі : John Wiley and Sons. 1995. – 420 с.

3. Вуд Дж., Райт Х., Бродлі К. Покращення візуалізації завдяки співпраці контрибуторів. Мілан : In Proceedings of Eurographics Workshop on Scientific Visualization. 1995. – 124 с.

4. Галіц В. Настав час очистити Windows. Ньо-Джерсі : Wiley publishing. 1994. – 40 с.

5. Галіц В. Основний посібник із дизайну інтерфейсу користувача. Ознайомлення з принципами та методами проектування графічного інтерфейсу. Ньо-Джерсі : Wiley publishing, 2007. – 888с.

6. Голгофа Г., Кутаз Дж., Тевенін Д. Уніфікована довідкова структура для багатоцільових інтерфейсів користувача. Оксфорд : Interacting with Computers, 2003. – 308 с.

7. Гулліксен Дж. Доменний дизайн інтерфейсів користувача. International Journal of Human–Computer Interaction, 1995. – 151 с.

8. Джонсон Дж. Проектування з розумом: простий посібник із розуміння правил дизайну інтерфейсу користувача. Амстердам : Elsevier Science & Technology, 2010. – 231 с.

9. Енгель Дж., Гердін К. Оцінка підходів до розробки інтерфейсу користувача на основі моделі. Аугсбург : In Human-Computer Interaction. 2014. – 307 с.

10. Ермі Л., Майра Ф. Основні компоненти ігрового процесу. Ліон : Proceedings of DiGRA, 2005. – 124 с.

11. Журнал Gamasutra. Notebook: Bad Game Designer, No Twinkie [Електронний ресурс]. URL : http://wwwgamasutra.com/view/feature/3812/the _designers_notebook_bad_game _.php?print=1

12. Журнал Gamasutra. Off With Their HUDs!: Rethinking the Heads-Up Display in Game Design. [Електронний ресурс]. URL : http://www.gamasutra.com/view/feature/130948/off_with_their_huds_rethinking.php

13. Журнал LinuxFoundation. Executable and Linking Format Specification [Електронний ресурс]. URL: https://refspecs.linuxfoundation.org/elf/elf.pdf

14. Журнал SemanticScholar. Леблан М. MDA: Формальний підхід до ігрового дизайну та дослідження ігор. [Електронний ресурс]. URL: https://www.semanticscholar.org/paper/2b134e5c46eec50f69c702c0b4aa29687d5d8f

15. Войтко В.В., Майданюк В.П., Денисюк А.В., Наумовський А.Ю. Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх / Електронні інформаційні ресурси: створення, використання, доступ. Пам’яті Олексія Петровича Стахова. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2021 р. – Суми/Вінниця: НІКО/ВНТУ, 2021. С. 56-58

16. Клеміш К., Вебер І., Бенаталлах Б. Повторне використання компонентів інтерфейсу користувача з урахуванням контексту. Advanced Information Systems Engineering, 2013. – 83 с.

17. Комп'ютерна програма "для прискореного зафарбування за методом Гуро" : а. с. №89438 Україна / О. Н. Романюк, А. Ю. Наумовський. - заяв. 06.06.2019; опубл. 26.07.2019, Бюл. №53

18. Костер Р. Теорія розваги для ігрового дизайну. Норвіч : Paraglyph Press, 2005. – 300 с.

19. Ксін Ю. Хуанг Дж. Лаі Ю. Дослідження та аналіз інтерфейсних фреймворків і бібліотек у розвитку електронного бізнесу. Пекін : Association for Computing Machinery, 2019. – 72 с.

20. Майданюк В. П. Методи і засоби комп'ютерних інформаційних технологій. Кодування зображень. Вінниця : ВНТУ, 2001. – 63 с.

21. Маккензі С. Закон Фіттса як інструмент дослідження та проектування у взаємодії людини та комп'ютера. Йорк : Human-computer interaction. 1992. – 139 с.

22. Матеріали L науково-технічної конференції підрозділів Вінницького національного технічного університету. Вінниця : НТКП ВНТУ–2021, 2021. – 3061 с.

23. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

24. Олсен Д. Розробка інтерфейсів користувача. Берлінгтон : Morgan Kaufmann Pub. 1998. – 414 с.

25. Офіційна web-сторінка фірми Nielsen Norman Group. The Difference Between Web Design and GUI Design. [Електронний ресурс]. URL : https://www.nngroup.com/articles/the-differencebetween-web-design-and-gui-design/

26. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти / Уклад. : А. О. Семенов, Л. П. Громова, Т. В. Макарова, О. В. Сердюк. – Вінниця : ВНТУ, 2021. 60 с.

27. Райс Дж., Буаверт Р. Від бібліотек програмного забезпечення до середовищ для вирішення проблем. Оксфорд : Interacting with Computers. 1996. – 53 с.

28. Саідіан Х. Дейл Р. Розробка вимог: встановлення зв'язку між розробником програмного забезпечення та замовником. Information and Software Technology, 2000. – 428 с.

29. Свінк С. Відчуття гри: Посібник дизайнера з віртуальних відчуттів. Лондон : Elsevier, Inc, 2009. – 376 с.

30. Танненбаум Дж., Лін Б., Бізоччі Дж. Ігри, оповідання та дизайн інтерфейсу. Неаполь : Int. J. of Arts and Technology. 2011. – 479 с.

31. Ульріх Д. Як писати динамічні бібліотеки. Red Hat, Inc, 2006. – 47 с.

32. Черні Т., Донаху Дж., Сонг І. До ефективного адаптивного дизайну інтерфейсу користувача. Пекін : Association for Computing Machinery, 2013. – 60с.

33. Черні Т., Чалупа В., Донаху Дж. На шляху до розумного дизайну інтерфейсу користувача. Лондон : ICISA, 2012. – 46 с.

34. Майданюк В.П., Наумовський А.Ю. Розробка програмного забезпечення ущільнення зображень без втрат на основі алгоритму арифметичного кодування / Матеріали міжнародної науково-практичної конференції молодих вчених та студентів «Молодь у світі сучасних технологій». – Херсон, 2020. [Електронний ресурс]. Режим доступу до ресурсу: http://kntu.net.ua/ukr/content/view/full/58984.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Кафедра програмного забезпечення

ЗАТВЕРДЖУЮ
д.т.н., проф. О.Н. Романюк
“___” _____2021 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп’ютерних іграх»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. В.П. Майданюк
“___”_____2021 р.
Виконав:
_____ ст. гр.1ПІ-20м А.Ю. Наумовський
“___”_____2021 р.

Вінниця – 2021

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх».

Галузь застосування – графічний інтерфейс користувача, комп’ютерні ігри.

2. Підстава для розробки

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № __ ректора по ВНТУ про закріплення тем МКР

3. Мета та призначення розробки

Метою роботи є удосконалення методів реалізації користувацьких інтерфейсів шляхом розробки спеціалізованих бібліотек та адаптації їх до використання при створенні комп'ютерних ігор, що дозволить пришвидшити розробку подібних програмних продуктів.

Програмна бібліотека призначена для використання розробниками ігор, що бажають пришвидшити процес розробки графічного інтерфейсу для використання в комп’ютерній грі на більшості систем.

4. Вхідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР):

а) Абрас К., Мелоні-Крічмар Д., Пріс Дж. Орієнтований на користувача дизайн. Таузанд-Оакс : Sage Publications, 2004. – 445 с.

б) Голгофа Г., Кутаз Дж., Тевенін Д. Уніфікована довідкова структура для багатоцільових інтерфейсів користувача. Оксфорд : Interacting with Computers, 2003. – 308 с.

в) Джонсон Дж. Проектування з розумом: простий посібник із розуміння правил дизайну інтерфейсу користувача. Амстердам : Elsevier Science & Technology, 2010. – 231 с.

г) Ксін Ю. Хуанг Дж. Лаі Ю. Дослідження та аналіз інтерфейсних фреймворків і бібліотек у розвитку електронного бізнесу. Пекін : Association for Computing Machinery, 2019. – 72 с.

д) Райс Дж., Буаверт Р. Від бібліотек програмного забезпечення до середовищ для вирішення проблем. Оксфорд : Interacting with Computers. 1996. – 53 с.

е) Черні Т., Донаху Дж., Сонг І. До ефективного адаптивного дизайну інтерфейсу користувача. Пекін : Association for Computing Machinery, 2013. – 60с.

5. Технічні вимоги

а) Середовища розробки – Microsoft Visual Studio Code;

б) Мови програмування – C;

в) Допоміжні засоби – OpenGL.

6. Конструктивні вимоги

Функціонал програмної бібліотеки повинен забезпечувати усі потреби розробників при створенні комп’ютерних ігор.

Текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

а) пояснювальна записка до МКР;

б) технічне завдання;

в) лістинги програми.

8. Вимоги до рівня стандартизації та уніфікації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії і етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз, вибір та обґрунтування актуальності розробки	15.09.21 – 26.09.21
2	Аналіз існуючих аналогів	27.09.21 – 03.10.21
3	Аналіз інформаційного забезпечення	04.10.21 – 10.10.21
4	Аналіз методів та засобів реалізації бібліотеки	11.10.21 – 19.10.21
5	Розробка методів та моделей -реалізації бібліотеки	20.10.21 – 25.10.21
6	Програмна реалізація модулів бібліотеки	26.10.21 – 18.11.21
7	Тестування роботи бібліотеки	19.11.21 – 24.11.21
8	Економічна частина	25.11.21- 30.11.21

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки роботи на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх.**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 20м

Науковий керівник: к.т.н. доц. Майданюк В.П.

Unicheck	
Оригінальність	**93,2 %**
Схожість	6,8 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлена з повним звітом подібності, який був згенерований Системою щодо роботи «Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх».

Автор _____ Наумовський Андрій Юрійович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток В. Лістинг коду

Лістинг коду заголовний файл бібліотеки
temp.h

```
#ifndef PI
#define PI 3.14159265358979323846
#endif

#define DEG2RAD (PI / 180.0)
#define RAD2DEG (180.0 / PI)

// Keyboard Function Keys
#define KEY_SPACE            32
#define KEY_ESCAPE          256
#define KEY_ENTER           257
#define KEY_RIGHT           262
#define KEY_LEFT            263
#define KEY_DOWN            264
#define KEY_UP              265
#define KEY_F1              290
#define KEY_F2              291
#define KEY_F3              292
#define KEY_F4              293
#define KEY_F5              294
#define KEY_F6              295
#define KEY_F7              296
#define KEY_F8              297
#define KEY_F9              298
#define KEY_F10             299
#define KEY_LEFT_SHIFT      340
#define KEY_LEFT_CONTROL    341
#define KEY_LEFT_ALT        342
#define KEY_RIGHT_SHIFT     344
#define KEY_RIGHT_CONTROL   345
#define KEY_RIGHT_ALT       346

// Mouse Buttons
#define MOUSE_LEFT_BUTTON     0
#define MOUSE_RIGHT_BUTTON    1
#define MOUSE_MIDDLE_BUTTON   2

// Gamepad Number
#define GAMEPAD_PLAYER1       0
#define GAMEPAD_PLAYER2       1
#define GAMEPAD_PLAYER3       2
#define GAMEPAD_PLAYER4       3

// Some Basic Colors
#define LIGHTGRAY  (Color){ 200, 200, 200, 255 }   // Light Gray
#define GRAY       (Color){ 130, 130, 130, 255 }   // Gray
#define DARKGRAY   (Color){ 80, 80, 80, 255 }      // Dark Gray
#define YELLOW     (Color){ 253, 249, 0, 255 }     // Yellow
#define GOLD       (Color){ 255, 203, 0, 255 }     // Gold
#define ORANGE     (Color){ 255, 161, 0, 255 }     // Orange
#define PINK       (Color){ 255, 109, 194, 255 }   // Pink
```

```
#define RED        (Color){ 230, 41, 55, 255 }     // Red
#define MAROON     (Color){ 190, 33, 55, 255 }     // Maroon
#define GREEN      (Color){ 0, 228, 48, 255 }      // Green
#define LIME       (Color){ 0, 158, 47, 255 }      // Lime
#define DARKGREEN  (Color){ 0, 117, 44, 255 }      // Dark Green
#define SKYBLUE    (Color){ 102, 191, 255, 255 }   // Sky Blue
#define BLUE       (Color){ 0, 121, 241, 255 }     // Blue
#define DARKBLUE   (Color){ 0, 82, 172, 255 }      // Dark Blue
#define PURPLE     (Color){ 200, 122, 255, 255 }   // Purple
#define VIOLET     (Color){ 135, 60, 190, 255 }    // Violet
#define DARKPURPLE (Color){ 112, 31, 126, 255 }    // Dark Purple
#define BEIGE      (Color){ 211, 176, 131, 255 }   // Beige
#define BROWN      (Color){ 127, 106, 79, 255 }    // Brown
#define DARKBROWN  (Color){ 76, 63, 47, 255 }      // Dark Brown

#define WHITE      (Color){ 255, 255, 255, 255 }   // White
#define BLACK      (Color){ 0, 0, 0, 255 }         // Black
#define BLANK      (Color){ 0, 0, 0, 0 }           // Blank (Transparent)
#define MAGENTA    (Color){ 255, 0, 255, 255 }     // Magenta

//----------------------------------------------------------------------------------
// Types and Structures Definition
//----------------------------------------------------------------------------------

// Boolean type
typedef enum { false, true } bool;

// Color type, RGBA (32bit)
typedef struct Color {
    unsigned char r;
    unsigned char g;
    unsigned char b;
    unsigned char a;
} Color;

// Rectangle type
typedef struct Rectangle {
    int x;
    int y;
    int width;
    int height;
} Rectangle;

// Image type, bpp always RGBA (32bit)
typedef struct Image {
    Color *pixels;
    int width;
    int height;
} Image;

// Texture2D type, bpp always RGBA (32bit)
typedef struct Texture2D {
    unsigned int glId;
    int width;
    int height;
} Texture2D;

// SpriteFont one Character (Glyph) data, defined in text module
typedef struct Character Character;
```

```
// SpriteFont type, includes texture and charSet array data
typedef struct SpriteFont {
    Texture2D texture;
    int numChars;
    Character *charSet;
} SpriteFont;

// Vector2 type
typedef struct Vector2 {
    float x;
    float y;
} Vector2;

// Vector3 type
typedef struct Vector3 {
    float x;
    float y;
    float z;
} Vector3;

// Camera type, defines a camera position/orientation in 3d space
typedef struct Camera {
    Vector3 position;
    Vector3 target;
    Vector3 up;
} Camera;

// Basic 3d Model type
typedef struct Model {
    int numVertices;
    Vector3 *vertices;
    Vector2 *texcoords;
    Vector3 *normals;
} Model;

// Basic Sound source and buffer
typedef struct Sound {
    unsigned int source;
    unsigned int buffer;
} Sound;

#ifdef __cplusplus
extern "C" {
#endif

//------------------------------------------------------------------------------------
// Window and Graphics Device Functions (Module: core)
//------------------------------------------------------------------------------------
void InitWindow(int width, int height, const char *title);  // Initialize Window and Graphics
Context (OpenGL)
void InitWindowEx(int width, int height, const char* title, // Initialize Window and Graphics
Context (OpenGL),...
                  bool resizable, const char *cursorImage); // ...define if windows-resizable and
custom cursor
void CloseWindow();                                         // Close Window and Terminate Context
bool WindowShouldClose();                                      // Detect if KEY_ESCAPE pressed or
Close icon pressed
void ToggleFullscreen();                                    // Fullscreen toggle (by default F11)
void SetCustomCursor(const char *cursorImage);              // Set a custom cursor icon/image
```

```
void SetExitKey(int key);                                   // Set a custom key to exit program
(default is ESC)

void ClearBackground(Color color);                          // Sets Background Color
void BeginDrawing();                                            // Setup drawing canvas to start
drawing
void EndDrawing();            // End canvas drawing and Swap Buffers (Double Buffering)

void Begin3dMode(Camera cam);  // Initializes 3D mode for drawing (Camera setup)
void End3dMode();              // Ends 3D mode and returns to default 2D orthographic mode

void SetTargetFPS(int fps);    // Set target FPS (maximum)
float GetFPS();                // Returns current FPS
float GetFrameTime();          // Returns time in seconds for one frame

Color GetColor(int hexValue);       // Returns a Color struct from hexadecimal value
int GetHexValue(Color color);       // Returns hexadecimal value for a Color

int GetRandomValue(int min, int max);      // Returns a random value between min and max
Color Fade(Color color, float alpha);     // Color fade-in or fade-out, alpha 0.0 to 1.0

//------------------------------------------------------------------------------------
// Input Handling Functions (Module: core)
//------------------------------------------------------------------------------------
bool IsKeyPressed(int key);            // Detect if a key has been pressed once
bool IsKeyDown(int key);               // Detect if a key is being pressed
bool IsKeyReleased(int key);           // Detect if a key has been released once
bool IsKeyUp(int key);                 // Detect if a key is NOT being pressed

bool IsMouseButtonPressed(int button);    // Detect if a mouse button has been pressed once
bool IsMouseButtonDown(int button);       // Detect if a mouse button is being pressed
bool IsMouseButtonReleased(int button);  // Detect if a mouse button has been released once
bool IsMouseButtonUp(int button);        // Detect if a mouse button is NOT being pressed
int GetMouseX();                         // Returns mouse position X
int GetMouseY();                         // Returns mouse position Y
Vector2 GetMousePosition();              // Returns mouse position XY

//------------------------------------------------------------------------------------
// Basic Shapes Drawing Functions (Module: shapes)
//------------------------------------------------------------------------------------
void DrawPixel(int posX, int posY, Color color);           // Draw a pixel
void DrawPixelV(Vector2 position, Color color);             // Draw a pixel (Vector version)
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color);
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color);
void    DrawCircle(int    centerX,    int    centerY,    float    radius,    Color    color);
// Draw a color-filled circle
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2);
// Draw a gradient-filled circle
void DrawCircleV(Vector2 center, float radius, Color color);   // Draw a color-filled circle
void    DrawCircleLines(int    centerX,    int    centerY,    float    radius,    Color    color);
// Draw circle outline
void   DrawRectangle(int   posX,   int   posY,   int   width,   int   height,   Color   color);
// Draw a color-filled rectangle
void DrawRectangleRec(Rectangle rec, Color color);       // Draw a color-filled rectangle
void DrawRectangleGradient(int posX, int posY, int width, int height, Color color1, Color
color2); // Draw a gradient-filled rectangle
void      DrawRectangleV(Vector2      position,      Vector2      size,      Color      color);
// Draw a color-filled rectangle (Vector version)
void   DrawRectangleLines(int   posX,   int   posY,   int   width,   int   height,   Color   color);
// Draw rectangle outline
```

```
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
// Draw a color-filled triangle
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
// Draw triangle outline
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color);
// Draw a regular polygon (Vector version)
void DrawPolyEx(Vector2 *points, int numPoints, Color color);
// Draw a closed polygon defined by points
void DrawPolyExLines(Vector2 *points, int numPoints, Color color);
// Draw polygon lines

bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2);
// Check collision between two rectangles
bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2);
// Check collision between two circles
bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec);
// Check collision between circle and rectangle
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2);
// Get collision rectangle for two rectangles collision
bool CheckCollisionPointRec(Vector2 point, Rectangle rec);
// Check if point is inside rectangle
bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius);
// Check if point is inside circle
bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3);
// Check if point is inside a triangle

//------------------------------------------------------------------------------------
// Texture Loading and Drawing Functions (Module: textures)
//------------------------------------------------------------------------------------
Image LoadImage(const char *fileName);         // Load an image into CPU memory (RAM)
Texture2D LoadTexture(const char *fileName);    // Load an image as texture into GPU memory
Texture2D LoadTextureFromRES(const char *rresName, int resId);
// Load an image as texture from rRES file
Texture2D CreateTexture2D(Image image);      // Create a Texture2D from Image data
void UnloadImage(Image image);               // Unload image from CPU memory (RAM)
void UnloadTexture(Texture2D texture);       // Unload texture from GPU memory

void DrawTexture(Texture2D texture, int posX, int posY, Color tint);
// Draw a Texture2D
void DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint);
// Draw a Texture2D with extended parameters
void DrawTextureRec(Texture2D texture, Rectangle sourceRec, Vector2 position, Color tint);
// Draw a part of a texture defined by a rectangle
void DrawTexturePro(Texture2D texture, Rectangle sourceRec, Rectangle destRec, Vector2 origin,
// Draw a part of a texture defined by a rectangle with 'pro' parameters
                    float rotation, Color tint);

//------------------------------------------------------------------------------------
// Font Loading and Text Drawing Functions (Module: text)
//------------------------------------------------------------------------------------
SpriteFont GetDefaultFont();                       // Get the default SpriteFont
SpriteFont LoadSpriteFont(const char *fileName);   // Load a SpriteFont image into GPU
void UnloadSpriteFont(SpriteFont spriteFont);      // Unload SpriteFont from GPU memory
void DrawText(const char *text, int posX, int posY, int fontSize, Color color);
// Draw text (using default font)
void DrawTextEx(SpriteFont spriteFont, const char* text, Vector2 position,
// Draw text using SpriteFont and additional parameters
                int fontSize, int spacing, Color tint);
int MeasureText(const char *text, int fontSize);   // Measure string width for default font
```

```
Vector2 MeasureTextEx(SpriteFont spriteFont, const char *text, int fontSize, int spacing);
// Measure string size for SpriteFont
int GetFontBaseSize(SpriteFont spriteFont);         // Returns the base size for a SpriteFont
(chars height)
void DrawFPS(int posX, int posY);                // Shows current FPS on top-left corner

//------------------------------------------------------------------------------------
// Basic 3d Shapes Drawing Functions (Module: models)
//------------------------------------------------------------------------------------
void DrawCube(Vector3 position, float width, float height, float lenght, Color color);
// Draw cube
void DrawCubeV(Vector3 position, Vector3 size, Color color);  // Draw cube (Vector version)
void DrawCubeWires(Vector3 position, float width, float height, float lenght, Color color);
// Draw cube wires
void DrawSphere(Vector3 centerPos, float radius, Color color);
// Draw sphere
void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color);
// Draw sphere with extended parameters
void DrawSphereWires(Vector3 centerPos, float radius, Color color);
// Draw sphere wires
void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float height, int
slices, Color color); // Draw a cylinder/cone
void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int
slices, Color color); // Draw a cylinder/cone wires
void DrawPlane(Vector3 centerPos, Vector2 size, Vector3 rotation, Color color);
// Draw a plane
void DrawPlaneEx(Vector3 centerPos, Vector2 size, Vector3 rotation, int slicesX, int slicesZ,
Color color); // Draw a plane with divisions
void DrawGrid(int slices, float spacing);         // Draw a grid (centered at (0, 0, 0))
void DrawGizmo(Vector3 position, bool orbits);    // Draw gizmo (with or without orbits)

//------------------------------------------------------------------------------------
// Model 3d Loading and Drawing Functions (Module: models)
//------------------------------------------------------------------------------------
Model LoadModel(const char *fileName);       // Load a 3d model (.OBJ)
void UnloadModel(Model model);               // Unload 3d model from memory
void DrawModel(Model model, Vector3 position, float scale, Color color);
// Draw a model
void DrawModelEx(Model model, Texture2D texture, Vector3 position, float scale, Color tint);
// Draw a textured model
void DrawModelWires(Model model, Vector3 position, float scale, Color color);
// Draw a model wires

//------------------------------------------------------------------------------------
// Audio Loading and Playing Functions (Module: audio)
//------------------------------------------------------------------------------------
void InitAudioDevice();                    // Initialize audio device and context
void CloseAudioDevice();                   // Close the audio device and context
Sound LoadSound(char *fileName);           // Load sound to memory

void UnloadSound(Sound sound);             // Unload sound

void PlaySound(Sound sound);               // Play a sound
void PauseSound(Sound sound);              // Pause a sound
void StopSound(Sound sound);               // Stop playing a sound
bool IsPlaying(Sound sound);               // Check if a sound is currently playing
void SetVolume(Sound sound, float volume);   // Set volume for a sound (1.0 is base level)
void SetPitch(Sound sound, float pitch);     // Set pitch for a sound (1.0 is base level)

#ifdef __cplusplus
```

```
}
#endif
#endif
```

Лістинг коду основного модуля

core.c

```
#include "temp.h"

#include <GLFW/glfw3.h>     // GLFW3 lib: Windows, OpenGL context and Input management
//#include <GL/gl.h>        // OpenGL functions (GLFW3 already includes gl.h)
#include <stdio.h>          // Standard input / output lib
#include <stdlib.h>         // Declares malloc() and free() for memory management, rand()
#include <time.h>           // Useful to initialize random seed
#include <math.h>           // Math related functions, tan() on SetPerspective
#include "vector3.h"        // Basic Vector3 functions
#include "utils.h"          // WritePNG() function

//----------------------------------------------------------------------------------
// Types and Structures Definition
//----------------------------------------------------------------------------------
typedef Color pixel;

//----------------------------------------------------------------------------------
// Global Variables Definition
//----------------------------------------------------------------------------------
static GLFWwindow* window;                  // Main window
static bool fullscreen;                     // Fullscreen mode track

static double currentTime, previousTime;    // Used to track timmings
static double updateTime, drawTime;         // Time measures for update and draw
static double frameTime;                    // Time measure for one frame
static double targetTime = 0;               // Desired time for one frame, if 0 not applied

static int windowWidth, windowHeight;       // Required to switch between windowed/fullscren mode
(F11)
static const char *windowTitle;             // Required to switch between windowed/fullscren mode
(F11)
static int exitKey = GLFW_KEY_ESCAPE;

static bool customCursor = false;           // Tracks if custom cursor has been set
static bool cursorOnScreen = false;         // Tracks if cursor is inside client area
static Texture2D cursor;                    // Cursor texture

static char previousKeyState[512] = { 0 };  // Required to check if key pressed/released once
static char currentKeyState[512] = { 0 };   // Required to check if key pressed/released once

static char previousMouseState[3] = { 0 };  // Required to check if mouse btn pressed/released
once
static char currentMouseState[3] = { 0 };   // Required to check if mouse btn pressed/released
once

static char previousGamepadState[32] = {0}; // Required to check if gamepad btn pressed/released
once
static char currentGamepadState[32] = {0};  // Required to check if gamepad btn pressed/released
once

//----------------------------------------------------------------------------------
```

```
// Other Modules Functions Declaration (required by core)
//------------------------------------------------------------------------------------
extern void LoadDefaultFont();       // [Module: text] Loads default font on InitWindow()
extern void UnloadDefaultFont();     // [Module: text] Unloads default font from GPU memory

//------------------------------------------------------------------------------------
// Module specific Functions Declaration
//------------------------------------------------------------------------------------
static void InitGraphicsDevice();      // Initialize Graphics Device (OpenGL stuff)
static void ErrorCallback(int error, const char *description);
// GLFW3 Error Callback, runs on GLFW3 error
       static void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods);
       // GLFW3 Keyboard Callback, runs on key pressed
static void CursorEnterCallback(GLFWwindow* window, int enter);
// GLFW3 Cursor Enter Callback, cursor enters client area
static void WindowSizeCallback(GLFWwindow* window, int width, int height);
// GLFW3 WindowSize Callback, runs when window is resized
static void CameraLookAt(Vector3 position, Vector3 target, Vector3 up);
 // Setup camera view (updates MODELVIEW matrix)
       static void SetPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);
       // Setup view projection (updates PROJECTION matrix)
static void TakeScreenshot();
// Takes a bitmap (BMP) screenshot and saves it in the same folder as executable

//------------------------------------------------------------------------------------
// Module Functions Definition - Window and OpenGL Context Functions
//------------------------------------------------------------------------------------

// Initialize Window and Graphics Context (OpenGL)
void InitWindow(int width, int height, const char *title)
{
    InitWindowEx(width, height, title, true, NULL);
}

// Initialize Window and Graphics Context (OpenGL) with extended parameters
void  InitWindowEx(int  width,  int  height,  const  char*  title,  bool  resizable,  const  char
*cursorImage)
{
    glfwSetErrorCallback(ErrorCallback);

    if (!glfwInit()) exit(1);

    if (!resizable) glfwWindowHint(GLFW_RESIZABLE, GL_FALSE); // Avoid window being resizable

    window = glfwCreateWindow(width, height, title, NULL, NULL);

    windowWidth = width;
    windowHeight = height;
    windowTitle = title;

    if (!window)
    {
        glfwTerminate();
        exit(1);
    }

    glfwSetWindowSizeCallback(window, WindowSizeCallback);
    glfwSetCursorEnterCallback(window, CursorEnterCallback);

    glfwMakeContextCurrent(window);
```

```
    glfwSetKeyCallback(window, KeyCallback);
    glfwSwapInterval(0);            // Disables GPU v-sync (if set), so frames are not limited to
screen refresh rate (60Hz -> 60 FPS)
    // If not set, swap interval uses GPU v-sync configuration
    // Framerate can be setup using SetTargetFPS()
    InitGraphicsDevice();

    previousTime = glfwGetTime();

    LoadDefaultFont();

    if (cursorImage != NULL)
    {
        cursor = LoadTexture(cursorImage);

        glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);
        customCursor = true;
    }

    srand(time(NULL));      // Initialize random seed
}

// Close Window and Terminate Context
void CloseWindow()
{
    UnloadDefaultFont();

    glfwDestroyWindow(window);
    glfwTerminate();
}

// Set a custom cursor icon/image
void SetCustomCursor(const char *cursorImage)
{
    if (customCursor) UnloadTexture(cursor);

    cursor = LoadTexture(cursorImage);

    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);
    customCursor = true;
}

// Set a custom key to exit program
void SetExitKey(int key)
{
    exitKey = key;
}

// Detect if KEY_ESCAPE pressed or Close icon pressed
bool WindowShouldClose()
{
    return (glfwWindowShouldClose(window));
}

// Fullscreen toggle (by default F11)
void ToggleFullscreen()
{
    if (glfwGetKey(window, GLFW_KEY_F11))
    {
        fullscreen = !fullscreen;      // Toggle fullscreen flag
```

```
        glfwDestroyWindow(window);     // Destroy the current window (we will recreate it!)

        if (fullscreen) window = glfwCreateWindow(windowWidth, windowHeight, windowTitle,
glfwGetPrimaryMonitor(), NULL);    // Fullscreen mode
        else window = glfwCreateWindow(windowWidth, windowHeight, windowTitle, NULL, NULL);

        if (!window)
        {
            glfwTerminate();
            exit(1);
        }

        glfwMakeContextCurrent(window);
        glfwSetKeyCallback(window, KeyCallback);

        InitGraphicsDevice();
    }
}

// Sets Background Color
void ClearBackground(Color color)
{
    float r = (float)color.r / 255;
    float g = (float)color.g / 255;
    float b = (float)color.b / 255;
    float a = (float)color.a / 255;

    glClearColor(r, g, b, a);
}

// Setup drawing canvas to start drawing
void BeginDrawing()
{
    currentTime = glfwGetTime();
    updateTime = currentTime - previousTime;
    previousTime = currentTime;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();                   // Reset current matrix (MODELVIEW)

    glTranslatef(0.375, 0.375, 0);      // HACK to have 2D pixel-perfect drawing on OpenGL
}

// End canvas drawing and Swap Buffers (Double Buffering)
void EndDrawing()
{
    if (customCursor && cursorOnScreen) DrawTexture(cursor, GetMouseX(), GetMouseY(), WHITE);

    glfwSwapBuffers(window);
    glfwPollEvents();

    currentTime = glfwGetTime();
    drawTime = currentTime - previousTime;
    previousTime = currentTime;

    frameTime = updateTime + drawTime;

    double extraTime = 0;
```

```
    while (frameTime < targetTime)
    {
        currentTime = glfwGetTime();
        extraTime = currentTime - previousTime;
        previousTime = currentTime;
        frameTime += extraTime;
    }
}

// Initializes 3D mode for drawing (Camera setup)
void Begin3dMode(Camera camera)
{
    glMatrixMode(GL_PROJECTION);        // Switch to projection matrix

    glPushMatrix();                      // Save previous matrix, which contains the settings for
the 2d ortho projection
    glLoadIdentity();                   // Reset current matrix (PROJECTION)

    SetPerspective(45.0f, (GLfloat)windowWidth/(GLfloat)windowHeight, 0.1f, 100.0f);
// Setup perspective projection

    glMatrixMode(GL_MODELVIEW);         // Switch back to modelview matrix
    glLoadIdentity();                   // Reset current matrix (MODELVIEW)

    CameraLookAt(camera.position, camera.target, camera.up);        // Setup Camera view
}
// Ends 3D mode and returns to default 2D orthographic mode
void End3dMode()
{
    glMatrixMode(GL_PROJECTION);        // Switch to projection matrix
    glPopMatrix();               // Restore previous matrix (PROJECTION) from matrix stack

    glMatrixMode(GL_MODELVIEW);         // Get back to modelview matrix
    glLoadIdentity();                   // Reset current matrix (MODELVIEW)

    glTranslatef(0.375, 0.375, 0);      // HACK to ensure pixel-perfect drawing on OpenGL
}
// Set target FPS for the game
void SetTargetFPS(int fps)
{
    targetTime = 1 / (float)fps;

    printf("TargetTime per Frame: %f seconds\n", (float)targetTime);
}

// Returns current FPS
float GetFPS()
{
    return (1/(float)frameTime);
}

// Returns time in seconds for one frame
float GetFrameTime()
{
    double roundedFrameTime =  round(frameTime*10000) / 10000;

    return (float)roundedFrameTime;    // Time in seconds to run a frame
}
```

```
// Returns a Color struct from hexadecimal value
Color GetColor(int hexValue)
{
    Color color;

    color.r = (unsigned char)(hexValue >> 24) & 0xFF;
    color.g = (unsigned char)(hexValue >> 16) & 0xFF;
    color.b = (unsigned char)(hexValue >> 8) & 0xFF;
    color.a = (unsigned char)hexValue & 0xFF;

    return color;
}

// Returns hexadecimal value for a Color
int GetHexValue(Color color)
{
    return ((color.a << 24) + (color.r << 16) + (color.g << 8) + color.b);
}

// Returns a random value between min and max (both included)
int GetRandomValue(int min, int max)
{
    if (min > max)
    {
        int tmp = max;
        max = min;
        min = tmp;
    }

    return (rand()%(abs(max-min)+1) + min);
}

// Fades color by a percentadge
Color Fade(Color color, float alpha)
{
    if (alpha < 0.0) alpha = 0.0;
    else if (alpha > 1.0) alpha = 1.0;

    return (Color){color.r, color.g, color.b, color.a*alpha};
}

//----------------------------------------------------------------------------------
// Module Functions Definition - Input (Keyboard, Mouse, Gamepad) Functions
//----------------------------------------------------------------------------------

// Detect if a key has been pressed once
bool IsKeyPressed(int key)
{
    bool ret = false;

    currentKeyState[key] = IsKeyDown(key);

    if (currentKeyState[key] != previousKeyState[key])
    {
        if (currentKeyState[key]) ret = true;
        previousKeyState[key] = currentKeyState[key];
    }
    else ret = false;

    return ret;
```

```
}

// Detect if a key is being pressed (key held down)
bool IsKeyDown(int key)
{
    if (glfwGetKey(window, key) == GLFW_PRESS) return true;
    else return false;
}

// Detect if a key has been released once
bool IsKeyReleased(int key)
{
    bool ret = false;

    currentKeyState[key] = IsKeyUp(key);

    if (currentKeyState[key] != previousKeyState[key])
    {
        if (currentKeyState[key]) ret = true;
        previousKeyState[key] = currentKeyState[key];
    }
    else ret = false;

    return ret;
}

// Detect if a key is NOT being pressed (key not held down)
bool IsKeyUp(int key)
{
    if (glfwGetKey(window, key) == GLFW_RELEASE) return true;
    else return false;
}

// Detect if a mouse button has been pressed once
bool IsMouseButtonPressed(int button)
{
    bool ret = false;

    currentMouseState[button] = IsMouseButtonDown(button);

    if (currentMouseState[button] != previousMouseState[button])
    {
        if (currentMouseState[button]) ret = true;
        previousMouseState[button] = currentMouseState[button];
    }
    else ret = false;

    return ret;
}

// Detect if a mouse button is being pressed
bool IsMouseButtonDown(int button)
{
    if (glfwGetMouseButton(window, button) == GLFW_PRESS) return true;
    else return false;
}

// Detect if a mouse button has been released once
bool IsMouseButtonReleased(int button)
{
```

```
    bool ret = false;

    currentMouseState[button] = IsMouseButtonUp(button);

    if (currentMouseState[button] != previousMouseState[button])
    {
        if (currentMouseState[button]) ret = true;
        previousMouseState[button] = currentMouseState[button];
    }
    else ret = false;

    return ret;
}

// Detect if a mouse button is NOT being pressed
bool IsMouseButtonUp(int button)
{
    if (glfwGetMouseButton(window, button) == GLFW_RELEASE) return true;
    else return false;
}

// Returns mouse position X
int GetMouseX()
{
    double mouseX;
    double mouseY;

    glfwGetCursorPos(window, &mouseX, &mouseY);

    return (int)mouseX;
}

// Returns mouse position Y
int GetMouseY()
{
    double mouseX;
    double mouseY;

    glfwGetCursorPos(window, &mouseX, &mouseY);

    return (int)mouseY;
}

// Returns mouse position XY
Vector2 GetMousePosition()
{
    double mouseX;
    double mouseY;

    glfwGetCursorPos(window, &mouseX, &mouseY);

    Vector2 position = { (float)mouseX, (float)mouseY };

    return position;
}

//----------------------------------------------------------------------------------
// Module specific Functions Definition
//----------------------------------------------------------------------------------
```

```
// GLFW3 Error Callback, runs on GLFW3 error
static void ErrorCallback(int error, const char *description)
{
    printf(description);
}

// GLFW3 Keyboard Callback, runs on key pressed
static void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == exitKey && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    else if (key == GLFW_KEY_F11 && action == GLFW_PRESS)
    {
        ToggleFullscreen();
    }
    else if (key == GLFW_KEY_F12 && action == GLFW_PRESS)
    {
        TakeScreenshot();
    }
}

static void CursorEnterCallback(GLFWwindow* window, int enter)
{
    if (enter == GL_TRUE) cursorOnScreen = true;
    else cursorOnScreen = false;
}

// GLFW3 WindowSize Callback, runs when window is resized
static void WindowSizeCallback(GLFWwindow* window, int width, int height)
{
    InitGraphicsDevice();   // If window is resized, graphics device is re-initialized
}

// Initialize Graphics Device (OpenGL stuff)
static void InitGraphicsDevice()
{
    int fbWidth, fbHeight;

    glfwGetFramebufferSize(window, &fbWidth, &fbHeight);     // Get framebuffer size of current
window

    glViewport(0, 0, fbWidth, fbHeight);     // Set viewport width and height

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);      // Clear used buffers, depth buffer
is used for 3D
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);                   // Set background color (black)
    glClearDepth(1.0f);                                     // Clear depth buffer

    glEnable(GL_DEPTH_TEST);         // Enables depth testing (required for 3D)
    glDepthFunc(GL_LEQUAL);          // Type of depth testing to apply

    glEnable(GL_BLEND);              // Enable color blending
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);      // Color blending function

    glHint(GL_PERSPECTIVE_CORRECTION_HINT,  GL_NICEST);         // Improve quality of color and
texture coordinate interpolation (Deprecated in OGL 3.0)
     // Other options: GL_FASTEST, GL_DONT_CARE (default)
```

```
    glMatrixMode(GL_PROJECTION);                // Switch to PROJECTION matrix
    glLoadIdentity();                           // Reset current matrix (PROJECTION)
    glOrtho(0, fbWidth, fbHeight, 0, 0, 1);    // Config orthographic mode: top-left corner
    glMatrixMode(GL_MODELVIEW);                 // Switch back to MODELVIEW matrix
    glLoadIdentity();                           // Reset current matrix (MODELVIEW)

    glDisable(GL_LIGHTING);                     // Lighting Disabled...


    glShadeModel(GL_SMOOTH);                //  Smooth  shading  between  vertex  (vertex  colors
interpolation)
}

// Setup camera view (updates MODELVIEW matrix)
static void CameraLookAt(Vector3 position, Vector3 target, Vector3 up)
{
    float rotMatrix[16];            // Matrix to store camera rotation

    Vector3 rotX, rotY, rotZ;      // Vectors to calculate camera rotations X, Y, Z (Euler)

    // Construct rotation matrix from vectors
    rotZ = VectorSubtract(position, target);
    VectorNormalize(&rotZ);
    rotY = up;                                  // Y rotation vector
    rotX = VectorCrossProduct(rotY, rotZ);    // X rotation vector = Y cross Z
    rotY = VectorCrossProduct(rotZ, rotX);    // Recompute Y rotation = Z cross X
    VectorNormalize(&rotX);                   // X rotation vector normalization
    VectorNormalize(&rotY);                   // Y rotation vector normalization

    rotMatrix[0] = rotX.x;
    rotMatrix[1] = rotY.x;
    rotMatrix[2] = rotZ.x;
    rotMatrix[3] = 0.0f;
    rotMatrix[4] = rotX.y;
    rotMatrix[5] = rotY.y;
    rotMatrix[6] = rotZ.y;
    rotMatrix[7] = 0.0f;
    rotMatrix[8] = rotX.z;
    rotMatrix[9] = rotY.z;
    rotMatrix[10] = rotZ.z;
    rotMatrix[11] = 0.0f;
    rotMatrix[12] = 0.0f;
    rotMatrix[13] = 0.0f;
    rotMatrix[14] = 0.0f;
    rotMatrix[15] = 1.0f;

    glMultMatrixf(rotMatrix);    // Multiply MODELVIEW matrix by rotation matrix

    glTranslatef(-position.x, -position.y, -position.z);    // Translate eye to position
}

// Setup view projection (updates PROJECTION matrix)
static void SetPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
{
   double xmin, xmax, ymin, ymax;

   ymax = zNear * tan(fovy * PI / 360.0);
   ymin = -ymax;
   xmin = ymin * aspect;
   xmax = ymax * aspect;
```

```
   glFrustum(xmin, xmax, ymin, ymax, zNear, zFar);
}
```

Лістинг коду модуля графічних примітивів

shapes.c

```
#include "temp.h"

#include <GL/gl.h>      // OpenGL functions
#include <stdlib.h>     // Required for abs() function
#include <math.h>       // Math related functions, sin() and cos() used on DrawCircle*
                        // sqrt() and pow() and abs() used on CheckCollision*

// Draw a pixel
void DrawPixel(int posX, int posY, Color color)
{
    glBegin(GL_POINTS);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2i(posX, posY);
    glEnd();
}

// Draw a pixel (Vector version)
void DrawPixelV(Vector2 position, Color color)
{
    glBegin(GL_POINTS);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2f(position.x, position.y);
    glEnd();
}

// Draw a line
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color)
{
    glBegin(GL_LINES);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2i(startPosX, startPosY);
        glVertex2i(endPosX, endPosY);
    glEnd();
}

// Draw a line  (Vector version)
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color)
{
    glBegin(GL_LINES);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2f(startPos.x, startPos.y);
        glVertex2f(endPos.x, endPos.y);
    glEnd();
}

// Draw a color-filled circle
void DrawCircle(int centerX, int centerY, float radius, Color color)
{
    glEnable(GL_POLYGON_SMOOTH);
    glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);  // Deprecated on OGL 3.0

    DrawPoly((Vector2){centerX, centerY}, 360, radius, 0, color);
```

```
    glDisable(GL_POLYGON_SMOOTH);
}

// Draw a gradient-filled circle
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2)
{
    glBegin(GL_TRIANGLE_FAN);
        glColor4ub(color1.r, color1.g, color1.b, color1.a);
        glVertex2i(centerX, centerY);
        glColor4ub(color2.r, color2.g, color2.b, color2.a);

        for (int i=0; i <= 360; i++)        //i++ --> Step = 1.0 pixels
        {
            glVertex2f(centerX + sin(DEG2RAD*i) * radius, centerY + cos(DEG2RAD*i) * radius);
        }
    glEnd();
}

// Draw a color-filled circle (Vector version)
void DrawCircleV(Vector2 center, float radius, Color color)
{
    glEnable(GL_POLYGON_SMOOTH);
    glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);

    glBegin(GL_TRIANGLE_FAN);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2f(center.x, center.y);

        for (int i=0; i <= 360; i++)        //i++ --> Step = 1.0 pixels
        {
            glVertex2f(center.x + sin(DEG2RAD*i) * radius, center.y + cos(DEG2RAD*i) * radius);
        }
    glEnd();

    glDisable(GL_POLYGON_SMOOTH);
}

// Draw circle outline
void DrawCircleLines(int centerX, int centerY, float radius, Color color)
{
    glEnable(GL_LINE_SMOOTH);                       // Smoothies circle outline (anti-aliasing
applied)
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);       // Best quality for line smooth (anti-aliasing
best algorithm)

    glBegin(GL_LINE_LOOP);
        glColor4ub(color.r, color.g, color.b, color.a);

        // NOTE: Circle outline is drawn pixel by pixel every degree (0 to 360)
        for (int i=0; i < 360; i++)
        {
            glVertex2f(centerX + sin(DEG2RAD*i) * radius, centerY + cos(DEG2RAD*i) * radius);
        }
   glEnd();

   glDisable(GL_LINE_SMOOTH);
}

// Draw a color-filled rectangle
void DrawRectangle(int posX, int posY, int width, int height, Color color)
```

```
{
    glBegin(GL_QUADS);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2i(posX, posY);
        glVertex2i(posX + width, posY);
        glVertex2i(posX + width, posY + height);
        glVertex2i(posX, posY + height);
    glEnd();
}

// Draw a color-filled rectangle
void DrawRectangleRec(Rectangle rec, Color color)
{
    DrawRectangle(rec.x, rec.y, rec.width, rec.height, color);
}

// Draw a gradient-filled rectangle
void DrawRectangleGradient(int posX, int posY, int width, int height, Color color1, Color color2)
{
    glBegin(GL_QUADS);
        glColor4ub(color1.r, color1.g, color1.b, color1.a);
        glVertex2i(posX, posY);
        glVertex2i(posX + width, posY);
        glColor4ub(color2.r, color2.g, color2.b, color2.a);
        glVertex2i(posX + width, posY + height);
        glVertex2i(posX, posY + height);
    glEnd();
}

// Draw a color-filled rectangle (Vector version)
void DrawRectangleV(Vector2 position, Vector2 size, Color color)
{
    glBegin(GL_QUADS);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2i(position.x, position.y);
        glVertex2i(position.x + size.x, position.y);
        glVertex2i(position.x + size.x, position.y + size.y);
        glVertex2i(position.x, position.y + size.y);
    glEnd();
}

// Draw rectangle outline
void DrawRectangleLines(int posX, int posY, int width, int height, Color color)
{
    glBegin(GL_LINE_LOOP);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2i(posX, posY);
        glVertex2i(posX + width - 1, posY);
        glVertex2i(posX + width - 1, posY + height - 1);
        glVertex2i(posX, posY + height - 1);
    glEnd();
}

// Draw a triangle
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color)
{
    glBegin(GL_TRIANGLES);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2f(v1.x, v1.y);
        glVertex2f(v2.x, v2.y);
```

```
        glVertex2f(v3.x, v3.y);
    glEnd();
}

void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color)
{
    glBegin(GL_LINE_LOOP);
        glColor4ub(color.r, color.g, color.b, color.a);
        glVertex2f(v1.x, v1.y);
        glVertex2f(v2.x, v2.y);
        glVertex2f(v3.x, v3.y);
    glEnd();
}

// Draw a regular polygon of n sides (Vector version)
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color)
{
    if (sides < 3) sides = 3;

    glPushMatrix();
        glTranslatef(center.x, center.y, 0);
        glRotatef(rotation, 0, 0, 1);

        glBegin(GL_TRIANGLE_FAN);
            glColor4ub(color.r, color.g, color.b, color.a);
            glVertex2f(0, 0);

            for (int i=0; i <= sides; i++)
            {
                glVertex2f(radius*cos(i*2*PI/sides), radius*sin(i*2*PI/sides));
            }
        glEnd();
    glPopMatrix();
}

// Draw a closed polygon defined by points
void DrawPolyEx(Vector2 *points, int numPoints, Color color)
{
    if (numPoints >= 3)
    {
        glEnable(GL_POLYGON_SMOOTH);
        glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);

        glBegin(GL_POLYGON);
            glColor4ub(color.r, color.g, color.b, color.a);

            for (int i = 0; i < numPoints; i++)
            {
                glVertex2f(points[i].x, points[i].y);
            }
        glEnd();

        glDisable(GL_POLYGON_SMOOTH);
    }
}

// Draw polygon lines
void DrawPolyExLines(Vector2 *points, int numPoints, Color color)
{
    if (numPoints >= 2)
```

```
    {
        glBegin(GL_LINE_LOOP);
            glColor4ub(color.r, color.g, color.b, color.a);

            for (int i = 0; i < numPoints; i++)
            {
                glVertex2f(points[i].x, points[i].y);
            }
        glEnd();
    }
}

// Check if point is inside rectangle
bool CheckCollisionPointRec(Vector2 point, Rectangle rec)
{
    bool collision = false;

    if ((point.x >= rec.x) && (point.x <= (rec.x + rec.width)) && (point.y >= rec.y) && (point.y
<= (rec.y + rec.height))) collision = true;

    return collision;
}

// Check if point is inside circle
bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius)
{
    return CheckCollisionCircles(point, 0, center, radius);
}

// Check if point is inside a triangle defined by three points (p1, p2, p3)
bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3)
{
    bool collision = false;

    float alpha = ((p2.y - p3.y)*(point.x - p3.x) + (p3.x - p2.x)*(point.y - p3.y)) /
                  ((p2.y - p3.y)*(p1.x - p3.x) + (p3.x - p2.x)*(p1.y - p3.y));

    float beta = ((p3.y - p1.y)*(point.x - p3.x) + (p1.x - p3.x)*(point.y - p3.y)) /
                 ((p2.y - p3.y)*(p1.x - p3.x) + (p3.x - p2.x)*(p1.y - p3.y));

    float gamma = 1.0f - alpha - beta;

    if ((alpha > 0) && (beta > 0) & (gamma > 0)) collision = true;

    return collision;
}

// Check collision between two rectangles
bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2)
{
    bool collision = false;

    int dx = abs((rec1.x + rec1.width / 2) - (rec2.x + rec2.width / 2));
    int dy = abs((rec1.y + rec1.height / 2) - (rec2.y + rec2.height / 2));

    if ((dx <= (rec1.width / 2 + rec2.width / 2)) && ((dy <= (rec1.height / 2 + rec2.height /
2)))) collision = true;

    return collision;
}
```

```
// Check collision between two circles
bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2)
{
    bool collision = false;

    float dx = center2.x - center1.x;      // X distance between centers
    float dy = center2.y - center1.y;      // Y distance between centers

    float distance = sqrt(dx*dx + dy*dy);  // Distance between centers

    if (distance <= (radius1 + radius2)) collision = true;

    return collision;
}

// Check collision between circle and rectangle
bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec)
{
    bool collision = false;

    float dx = abs((rec.x + rec.width / 2) - center.x);
    float dy = abs((rec.y + rec.height / 2) - center.y);

    if ((dx <= (rec.width / 2 + radius)) && (dy <= (rec.height / 2 + radius))) collision = true;

    return collision;
}

// Get collision rectangle for two rectangles collision
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2)
{
    Rectangle retRec = { 0, 0, 0, 0 };

    if (CheckCollisionRecs(rec1, rec2))
    {
        int dxx = abs(rec1.x - rec2.x);
        int dyy = abs(rec1.y - rec2.y);

        if (rec1.x <= rec2.x)
        {
            if (rec1.y <= rec2.y)
            {
                retRec.x = rec2.x;
                retRec.y = rec2.y;
                retRec.width = rec1.width - dxx;
                retRec.height = rec1.height - dyy;
            }
            else
            {
                retRec.x = rec2.x;
                retRec.y = rec1.y;
                retRec.width = rec1.width - dxx;
                retRec.height = rec2.height - dyy;
            }
        }
        else
        {
            if (rec1.y <= rec2.y)
            {
```

```
                retRec.x = rec1.x;
                retRec.y = rec2.y;
                retRec.width = rec2.width - dxx;
                retRec.height = rec1.height - dyy;
            }
            else
            {
                retRec.x = rec1.x;
                retRec.y = rec1.y;
                retRec.width = rec2.width - dxx;
                retRec.height = rec2.height - dyy;
            }
        }

        if (retRec.width >= rec2.width) retRec.width = rec2.width;
        if (retRec.height >= rec2.height) retRec.height = rec2.height;
    }

    return retRec;
}
```

Додаток Г.

Ілюстративна частина

УДОСКОНАЛЕННЯ МЕТОДІВ РЕАЛІЗАЦІЇ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ ДЛЯ ЇХ ВИКОРИСТАННЯ В КОМП’ЮТЕРНИХ ІГРАХ

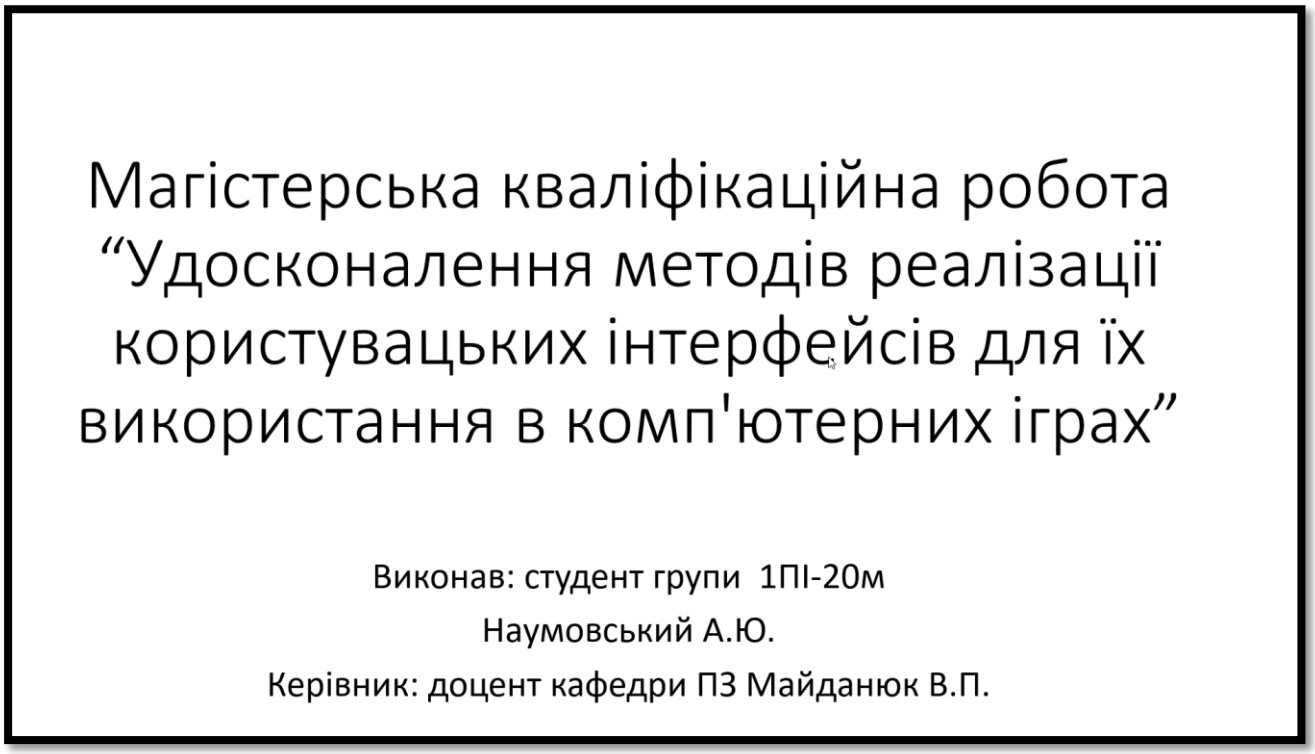


Рисунок Г.1 – Назва роботи

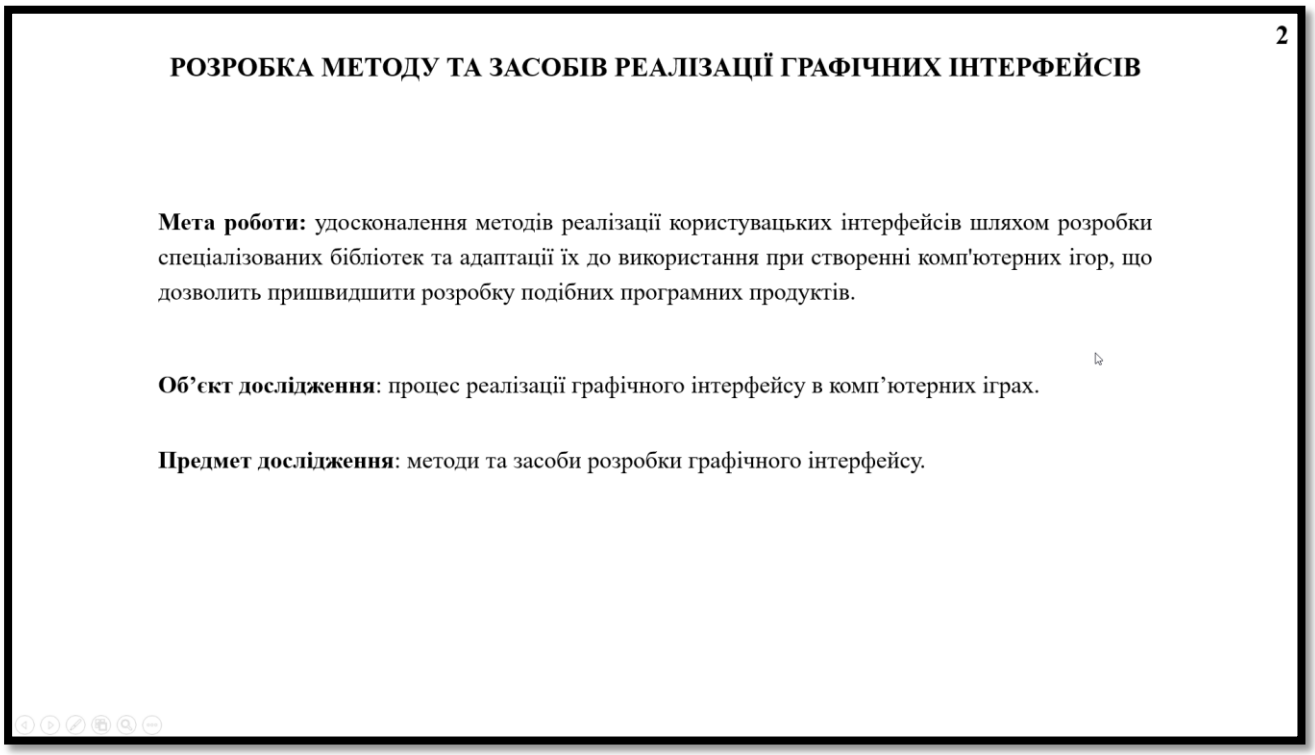


Рисунок Г.2 – Мета, об’єкт і предмет дослідження

3

ЗАДАЧІ

- аналіз існуючих рішень;

- аналіз особливостей реалізації графічного інтерфейсу користувача в комп'ютерних іграх;

- розробка методу та моделі реалізації компонентів бібліотеки;

- розробка програмних модулів бібліотеки;

- провести тестування програмного продукту для перевірки всіх можливих варіантів використання.

Рисунок Г.3 – Задачі

4

АНАЛІЗ АНАЛОГІВ

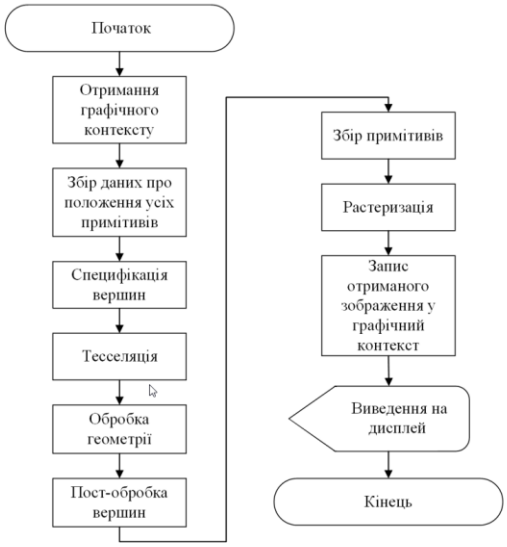
Критерій	SFML	AGKSharp	LibGDX	temp
Швидкодія	+	-	-	+
Мультиплат-форменість	+	-	-	+
Компілювання додатку для мобільних пристроїв	-	-	+	+
Компілювання додатку для Web (HTML5)	-	-	-	+

Рисунок Г.4 – Аналіз аналогів

5

ЗАГАЛЬНІ АЛГОРИТМИ

Узагальнений алгоритм роботи гри



Алгоритм графічного відображення гри

Рисунок Г.5 – Загальні алгоритми

6

МЕТОД РЕАЛІЗАЦІЇ ГРАФІЧНИХ ІНТЕРФЕЙСІВ У КОМП'ЮТЕРНИХ ІГРАХ

1. отримання графічного контексту від поточного вікна;
2. збір даних про положення усіх примітивів;
3. специфікація вершин: на цьому етапі впорядковується список вершин, які визначають межі примітиву. Також визначаються інші атрибути вершин, такі як колір, координати текстури тощо;
4. тесселяція: на цьому етапі примітиви перетворюються у мозайку пікселей, тобто поділяються на більш гладку сітку трикутників;
5. обробка геометрії: на цьому етапі вхідні примітиви перетворюються на серію трикутників;.
6. пост-обробка вершин: на цьому етапі відбувається відрізання. Відрізання відкидає область примітивів, що лежать за межами області перегляду;
7. збір примітивів: на цьому етапі дані вершин упорядковуються в послідовність простих примітивів (лінії, точки або трикутники);
8. растеризація: перетворення примітивів у дані про фрагменти пікселей;
9. запис отриманого зображення у графічний контекст;
10. виведення вмісту графічного контексту на дисплей

Рисунок Г.6 – Метод реалізації графічних інтерфейсів у комп'ютерних іграх

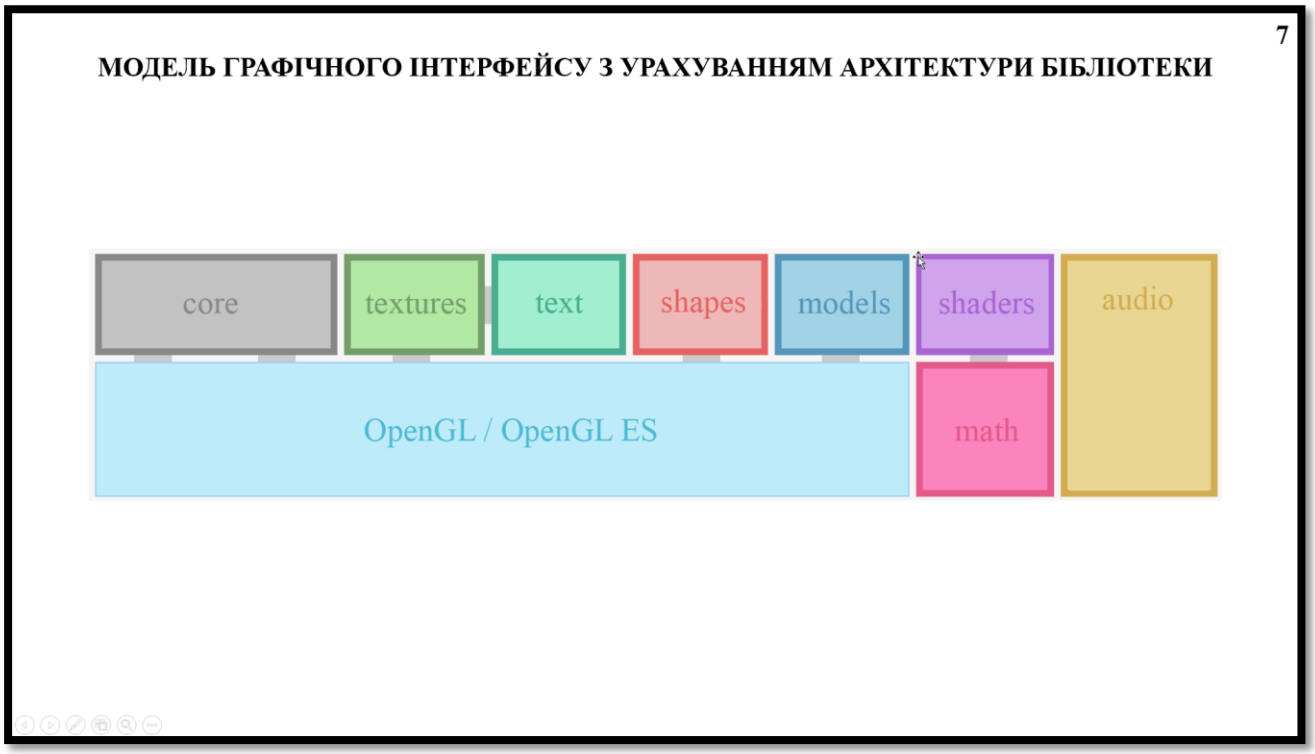


Рисунок Г.7 – Архітектура бібліотеки

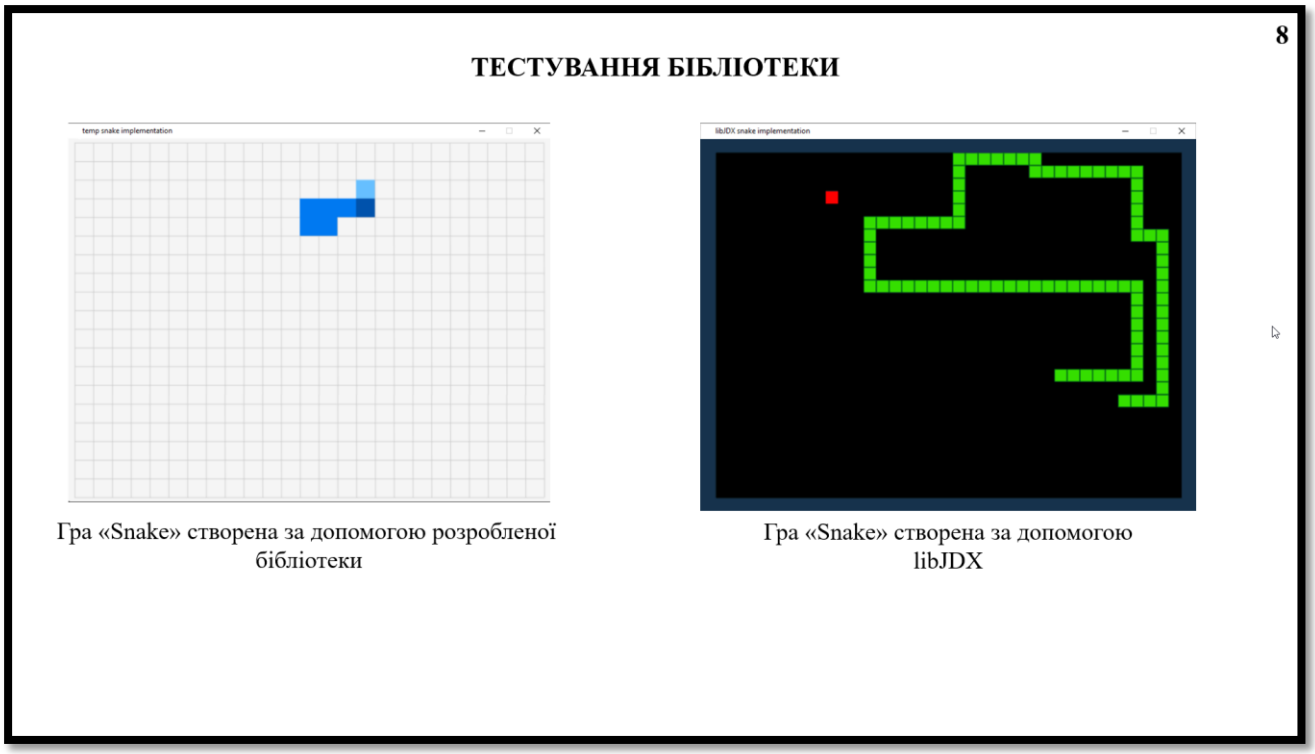
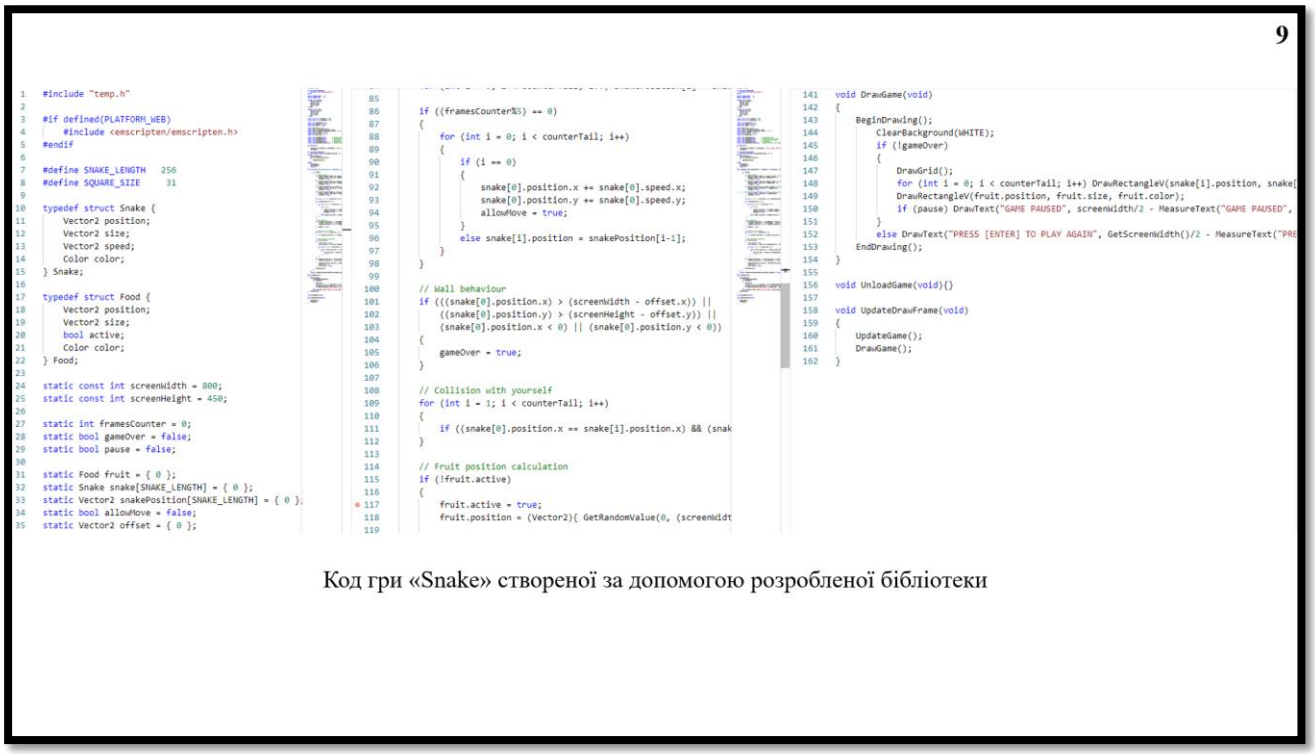
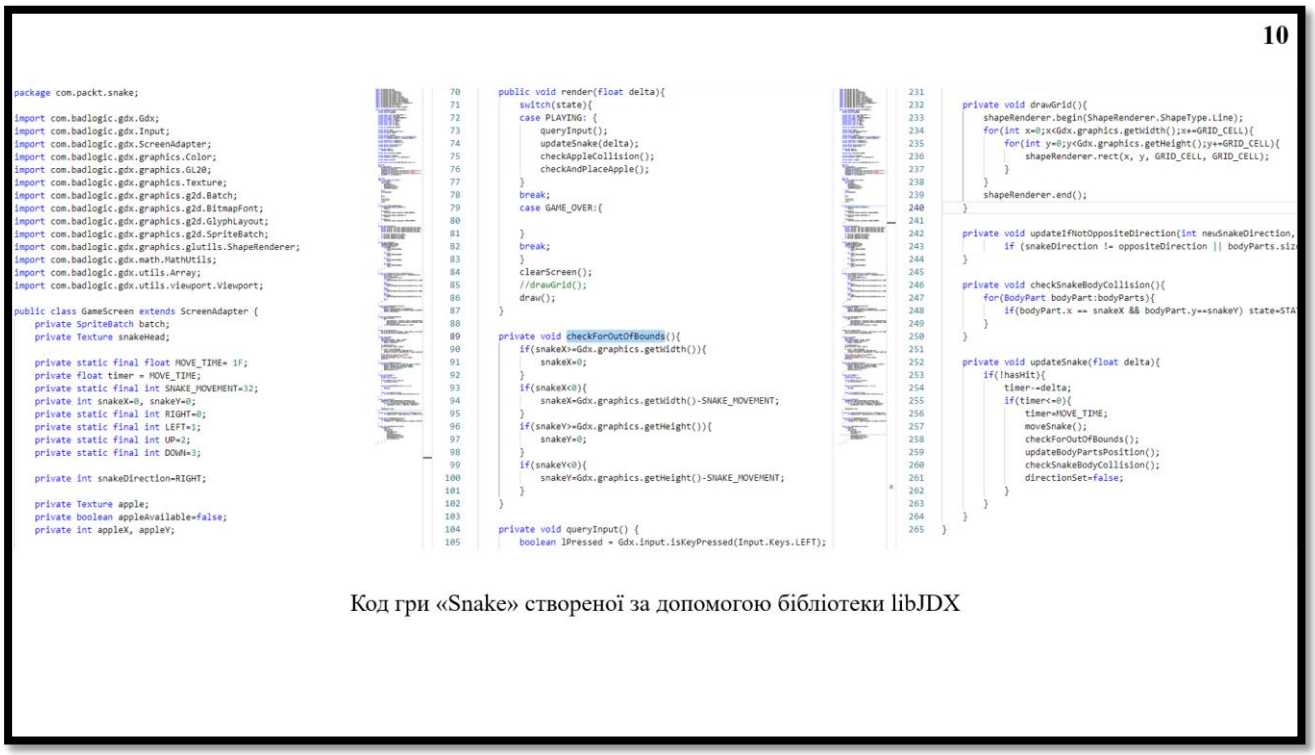


Рисунок Г.8 – Тестування бібліотеки



Код гри «Snake» створеної за допомогою розробленої бібліотеки

Рисунок Г.9 – Код гри «Snake» створеної за допомогою розробленої бібліотеки



Код гри «Snake» створеної за допомогою бібліотеки libJDX

Рисунок Г.10 – Код гри «Snake» створеної за допомогою бібліотеки libJDX

11

Наукова новизна:

- подальшого розвитку дістав метод реалізації графічних інтерфейсів у комп'ютерних іграх на різних платформах, який, на відміну від існуючих, дозволить компілювати додатки для різних систем з однієї кодової бази, що дозволить пришвидшити розробку комп'ютерних ігор;
- подальшого розвитку дістала модель бібліотеки для графічних інтерфейсів, яка, на відміну від існуючих, акумулює функції керування вікнами, взаємодії з користувачем, відображення графічних примітивів, зображень, текстур та шейдерів, що дозволяє використання створених компонент в процесі компілювання та інтеграції інтерфейсу комп'ютерних ігор забезпечувати кросплафторменну реалізацію і збільшити швидкість обробки даних.

Практична цінність:

- Розроблена програмна бібліотека, функції та структури можуть використовуватися в комп'ютерних іграї і дозволяють спростити процес розробки графічного інтерфейсу користувача під час розробки комп'ютерних ігор для різних платформ.

Рисунок Г.11 – Наукова новизна та практична цінність

12

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було:

- проведено аналіз існуючих аналогів, ключовими показниками є швидкодія та мультиплатформенність
- проаналізовано особливості реалізації графічного інтерфейсу користувача в комп'ютерних іграх;
- розроблено метод та моделі реалізації компонентів графічного інтерфейсу;
- розроблено основні програмні модулі бібліотеки, які містять необхідний функціонал для розробки ігор;
- проведено тестування бібліотеки.

Рисунок Г.12 – Висновки

13

АПРОБАЦІЯ

Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на:

- міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (Вінниця 2021);
- міжнародній науково-практичній конференції молодих вчених та студентів «Молодь у світі сучасних технологій» (Херсон 2020).

Рисунок Г.13 – Апробації

14

ПУБЛІКАЦІЇ

Результати дослідження магістерської кваліфікаційної роботи опубліковані в наступних наукових роботах:

- Войтко В.В., Майданюк В.П., Денисюк А.В., Наумовський А.Ю. Удосконалення методів реалізації користувацьких інтерфейсів для їх використання в комп'ютерних іграх / Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2021 р. – Суми/Вінниця: НІКО/ВНТУ, 2021. С. 56-58
- Майданюк В.П., Наумовський А.Ю. Розробка програмного забезпечення ущільнення зображень без втрат на основі алгоритму арифметичного кодування / Матеріали міжнародної науково-практичної конференції молодих вчених та студентів «Молодь у світі сучасних технологій». – Херсон, 2020.
- Майданюк В.П., Наумовський А.Ю. Розробка програмного забезпечення для реалізації користувацький інтерфейсів / Матеріали L науково-технічної конференції підрозділів Вінницького національного технічного університету 10-12 березня 2021 року – Вінниця : НТКП ВНТУ, 2021. С. 1026-1027
- Комп'ютерна програма "для прискореного зафарбування за методом Гуро" : а. с. №89438 Україна / О. Н. Романюк, А. Ю. Наумовський. - заяв. 06.06.2019; опубл. 26.07.2019, Бюл. №53

Рисунок Г.14 – Публікації