**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**Звіт**

з самостійної роботи №2

З дисципліни “Soft skills, групова динаміка та комунікації”

На тему: “Командна розробка програмного забезпечення”

Виконали:

студент групи КНТ-114 Андрій ЖАРИК

студент групи КНТ-114 Ігор ШКУРІН

Прийняв:

доцент Валерій ЛЬОВКІН

**2025**

1. **КОМАНДНА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**
   1. **Мета роботи**

1.1.1 Навчитися розробляти програмне забезпечення, працюючи в команді.

1.1.2 Навчитися оформлювати програмну документацію.

**1.2 Завдання до роботи**

1.2.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи дані методичні вказівки, а також рекомендовану літературу.

1.2.2 Сформувати команду з двох студентів та отримати у викладача індивідуальне завдання.

1.2.3 Завести акаунт на сервісі GitHub та створити команду для роботи в Slack.

1.2.4 Узгодити розподіл зобов’язань для виконання завдання, використовуючи Slack.

1.2.5 Створити репозиторій для роботи над проектом та дозволити доступ до проекту обом користувачам.

1.2.6 Завантажити у віддалений репозиторій існуючі файли проекту.

1.2.7 Створити дві гілки проекту: для презентації ревізій та для відлагодження проекту.

1.2.8 Налаштувати доступ до Git-репозиторію в інтегрованому середовищі розробки Eclipse.

1.2.9 Виконати реалізацію проекту в Eclipse та зазначити, яка частина коду ким була розроблена.

1.2.10 Визначити додаткову функцію та реалізувати її, працюючи над нею одночасно вдвох.

1.2.11 Завершити роботу над проектом.

1.2.12 Відповідно до діючих стандартів індивідуально оформити програмний документ, узгоджений з викладачем, на розроблене програмне забезпечення.

1.2.13 Оформити звіт з роботи.

1.2.14 Відповісти на контрольні запитання.

**1.3 Короткі теоретичні відомості**

Для роботи з Git в інтегрованих середовищах розробки існують спеціальні плагіни, наприклад, EGit для Eclipse. Для того щоб інсталювати даний плагін, необхідно в Eclipse обрати пункт меню Help → Eclipse Marketplace та в рядку пошуку ввести EGit, після чого інсталювати його.

Спочатку потрібно встановити загальні налаштування підключення. Для цього з меню Window потрібно обрати пункт Preferences, далі обрати Version Control (Team) і Git. Далі необхідно всі потрібні налаштування для підключення до попередньо створеного Github-репозиторію, зокрема ім’я користувача (user.name), адресу електронної пошти (user.email).

Подальші кроки пов’язані зі створенням репозиторію для командної роботи над проєктом. Тому потрібно створити новий проєкт. Після цього з його контекстного меню обрати Team → Share Project, визначаючи потрібну конфігурацію репозиторію. Після того, як репозиторій створений або визначений, необхідно додати проєкт до виконання контролю версій, викликаючи з контекстного меню Team → Add to Index.

Звичні команди керування для систем контролю версій доступні через контекстне меню аналогічно описаним вище. Для того щоб зафіксувати зміни, потрібно викликати з контекстного меню Team → Commit.

Для того щоб встановити саме підключення на рівні репозиторію, необхідно з меню Team обрати далі пункти Remote, Push, після чого задати URL-адресу Github-репозиторію, задати облікові дані, обрати потрібну гілку.

**1.4 Текст програми**

Funcs.h – файл з заголовками функцій:

#ifndef FUNCS\_H

#define FUNCS\_H

#include <iostream>

using namespace std;

void fieldcreate(char\*\*&, int&, int&);

void fieldout(char\*\*, int, int);

void fielddelete(char\*\*, int);

void seekAliveCells(char\*\*, int, int);

bool askRestart();

void initConfig(char\*\* f, int w, int h);

int countAliveSusid(char\*\* f, int x, int y, int w, int h);

bool allCellsDead(char\*\* f, int w, int h);

void gameLoop(char\*\*& field, int w, int h);

#endif

Andriifuncs.cpp – файл з функціями розроблених Андрієм Жариком

#include "funcs.h"

void fieldcreate(char\*\*& f, int& w, int& h) {

cout << "Введіть розмір поля в довжину: ";

cin >> w;

cout << "Введіть розмір поля в висоту: ";

cin >> h;

f = new char\* [h];

for (int i = 0; i < h; i++) {

f[i] = new char[w];

}

for (int i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

f[i][j] = '\*';

}

}

}

void fieldout(char\*\* f, int w, int h) {

for (int i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

cout << f[i][j] << " ";

}

cout << endl;

}

}

void fielddelete(char\*\* f, int h) {

for (int i = 0; i < h; i++) {

delete[] f[i];

}

delete[] f;

}

void seekAliveCells(char\*\* f, int w, int h) {

char\*\* temp = new char\* [h];

for (int i = 0; i < h; i++) {

temp[i] = new char[w];

for (int j = 0; j < w; j++) {

temp[i][j] = f[i][j];

}

}

for (int i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

int count = countAliveSusid(temp, j, i, w, h);

if (temp[i][j] == 'o') {

if (count < 2 || count > 3) f[i][j] = '\*';

}

else {

if (count == 3) f[i][j] = 'o';

}

}

}

for (int i = 0; i < h; i++) {

delete[] temp[i];

}

delete[] temp;

}

ihorfuncs.cpp – файл з функціями розроблених Шкуріним Ігорем

#include "funcs.h"

#include <windows.h>

void initConfig(char\*\* f, int w, int h) { //користувач задає кількість живих клітин

int count;

cout << "Скільки живих клітин ви хочете ввести? ";

cin >> count;

for (int i = 0; i < count; i++) { //користувач вводить координати кожної клітини

int x, y;

cout << "Клітина #" << i + 1 << " — введіть x y: ";

cin >> x >> y;

//Перевірка того що координати в межах поля

if (x >= 0 && x < w && y >= 0 && y < h) {

f[y][x] = 'o'; //замінює мертву клітину \* на живу o

}

else {

cout << "Невірні координати, ця клітина буде пропущена." << endl;

}

}

}

int countAliveSusid(char\*\* f, int x, int y, int w, int h) {

int count = 0; // лічильник живих сусідів

//перебирає всі сусідні клітини навколо (x y)

for (int dx = -1; dx <= 1; dx++) {

for (int dy = -1; dy <= 1; dy++) {

if (dx == 0 && dy == 0) continue;

//пропускаємо саму клітину (x y)

int nx = x + dx; //нові координати по x

int ny = y + dy; //нові координати по y

//перевірка, чи не виходять координати за межі поля

if (nx >= 0 && nx < w && ny >= 0 && ny < h) {

//перевіряємо, чи ця сусідня клітина жива

if (f[ny][nx] == 'o') {

count++; //якщо клітина жива - додаємо до лічильника

}

}

}

}

return count;

}

bool allCellsDead(char\*\* f, int w, int h) {

for (int i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

if (f[i][j] == 'o') {

return false; // знайдена хоча б одна жива клітина

}

}

}

return true; // всі клітини мертві

}

void gameLoop(char\*\*& field, int w, int h) {

int generation = 0;

while (true) {

system("cls"); // очищення екрану

cout << "Покоління #" << generation << endl;

fieldout(field, w, h);

int aliveCount = 0;

for (int i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

if (field[i][j] == 'o') aliveCount++;

}

}

cout << "Живих клітин: " << aliveCount << endl;

if (aliveCount == 0) {

cout << "Усі клітини мертві. Гру завершено!" << endl;

break;

}

seekAliveCells(field, w, h);

generation++;

Sleep(750); // затримка перед наступним поколінням

}

}

main.cpp – основний файл

#include "funcs2.h"

#include <windows.h>

#include <cctype>

bool askRestart() {

char answer;

cout << "Бажаєте почати гру заново? (y/n): ";

cin >> answer;

cin.ignore(10000, '\n'); // очищає залишки вводу

return tolower(answer) == 'y';

}

int main() {

SetConsoleOutputCP(65001);

setlocale(LC\_ALL, "RU");

do {

int w, h;

char\*\* field = nullptr;

fieldcreate(field, w, h);

initConfig(field, w, h);

gameLoop(field, w, h);

fielddelete(field, h);

cout << endl;

} while (askRestart());

cout << "Дякуємо за гру!" << endl;

cout << "Натисніть Enter для виходу...";

cin.get();

return 0;

}

**1.5 Опис розподілу відповідальності в команді**

Команда складається з двох: Андрія Жарика та Ігоря Шкуріна. Обов’язки були розподілені рівномірно. Функції, які були розроблені Андрієм Жариком знаходяться у файлі під назвою “andriifuncs.cpp”, а ті які було розроблено Ігорем Шкуріним, знаходяться в файлі під назвою “ihorfuncs.cpp”. Робота над програмною документацією та звітом було виконано разом. Робота над “main.cpp” і додаткової функції – для перезапуску гри була виконана разом.

**2 КОПІЇ ЕКРАННИХ ФОРМ З РЕЗУЛЬТАТАМИ ВИКОНАННЯ ЗАВДАНЬ**

1.2.3 Завести акаунт на сервісі GitHub та створити команду для роботи в Slack.

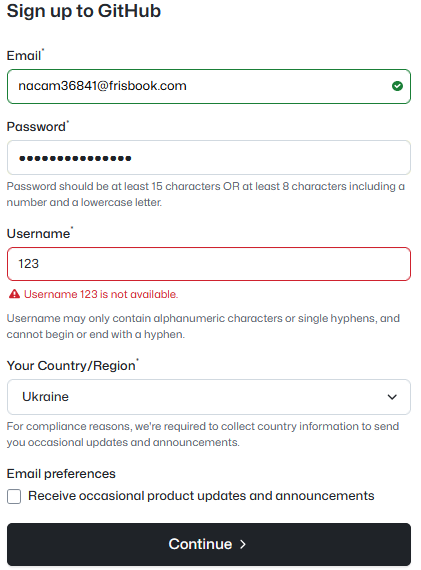
****

Рисунок 2.1 – створення акаунта на GitHub

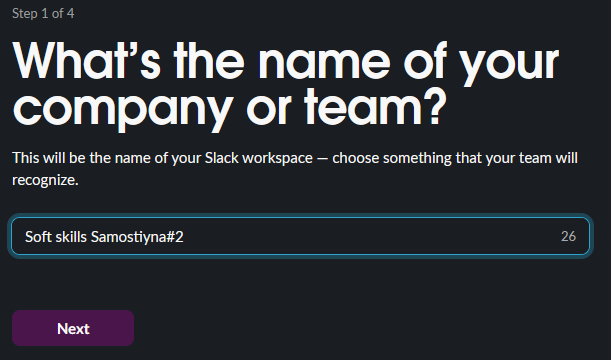
****

Рисунок 2.2 – перший крок створення команди для роботи в сервісі Slack

1.2.4 Узгодити розподіл зобов’язань для виконання завдання, використовуючи Slack.

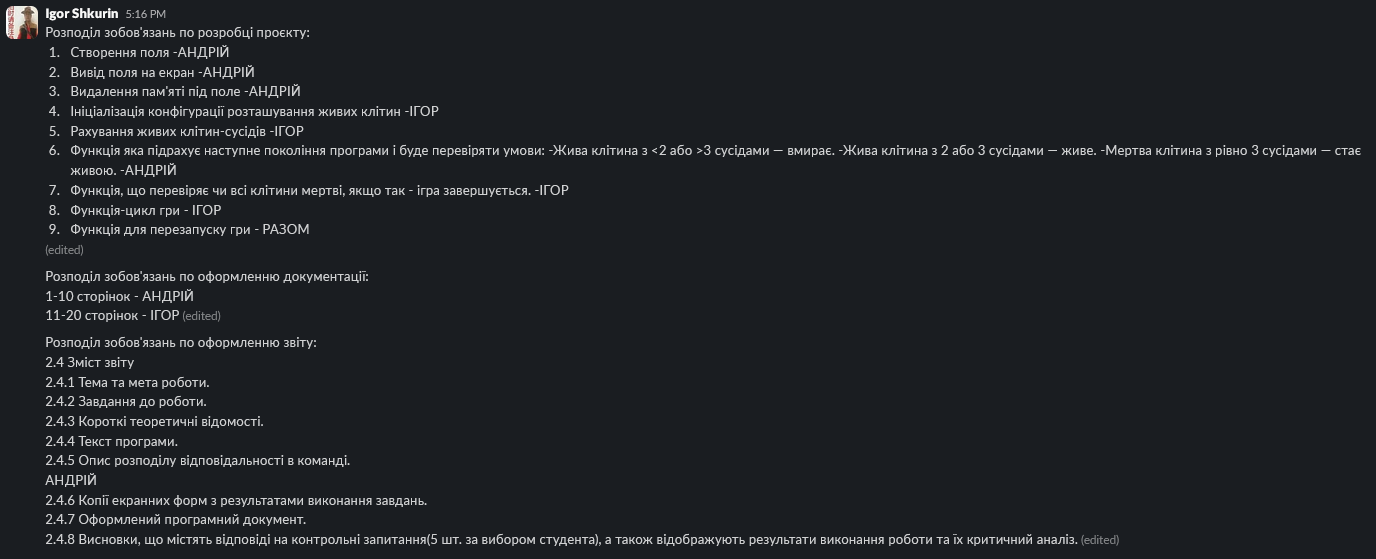


Рисунок 2.3 – розподіл зобов’язань для виконання завдання, використовуючи Slack

1.2.5 Створити репозиторій для роботи над проектом та дозволити доступ до проекту обом користувачам.

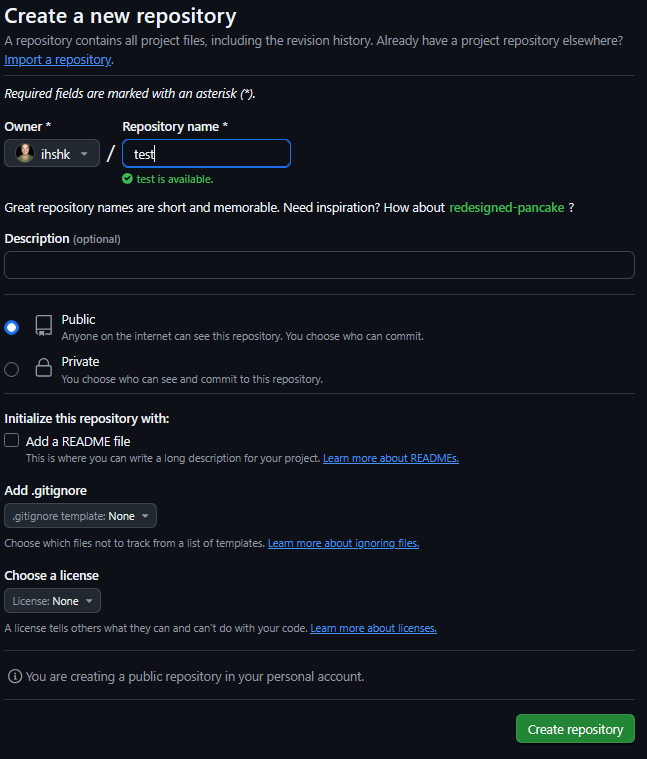


Рисунок 2.4 – створення репозиторію на сервісі GitHub для роботи над проєктом

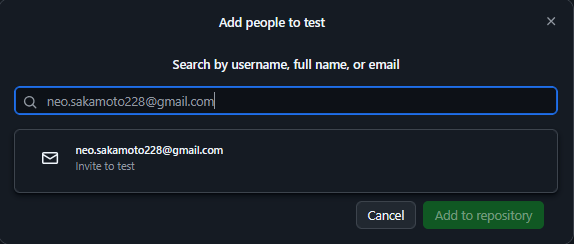


Рисунок 2.5 – дозволення доступу до репозиторію іншому користувачу

1.2.6 Завантажити у віддалений репозиторій існуючі файли проекту.



Рисунок 2.6 – завантаження існуючих файлів проекту до репозиторію за допомогою GitHub Desktop.

1.2.7 Створити дві гілки проекту: для презентації ревізій та для відлагодження проекту.

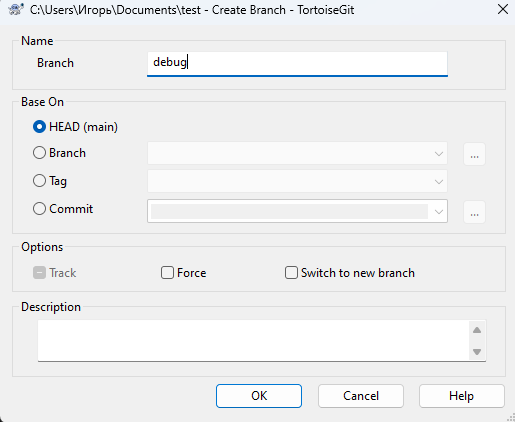


Рисунок 2.7 – створення гілки для відлагодження проекту за допомогою TortoiseGit

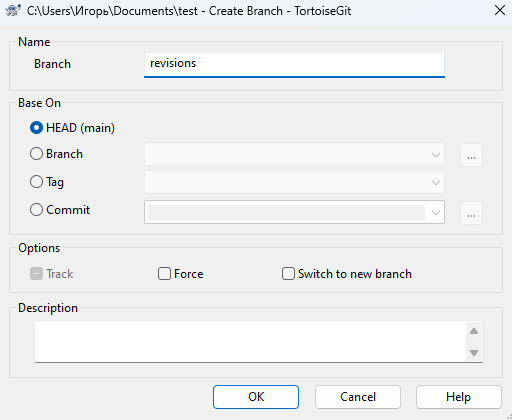


Рисунок 2.8 – створення гілки для презентацій ревізій за допомогою TortoiseGit

1.2.8 Налаштувати доступ до Git-репозиторію в інтегрованому середовищі розробки Eclipse.

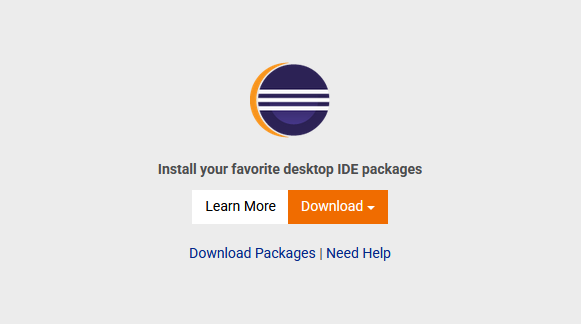


Рисунок 2.9 – завантаження Eclipse з інтернету

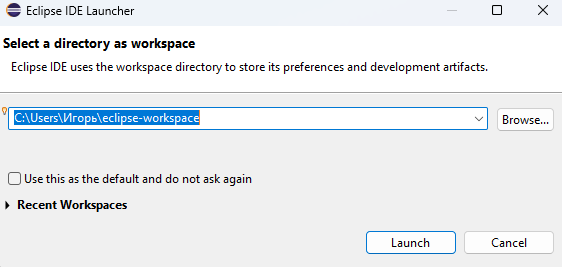


Рисунок 2.10 – першозначне налаштування Eclipse

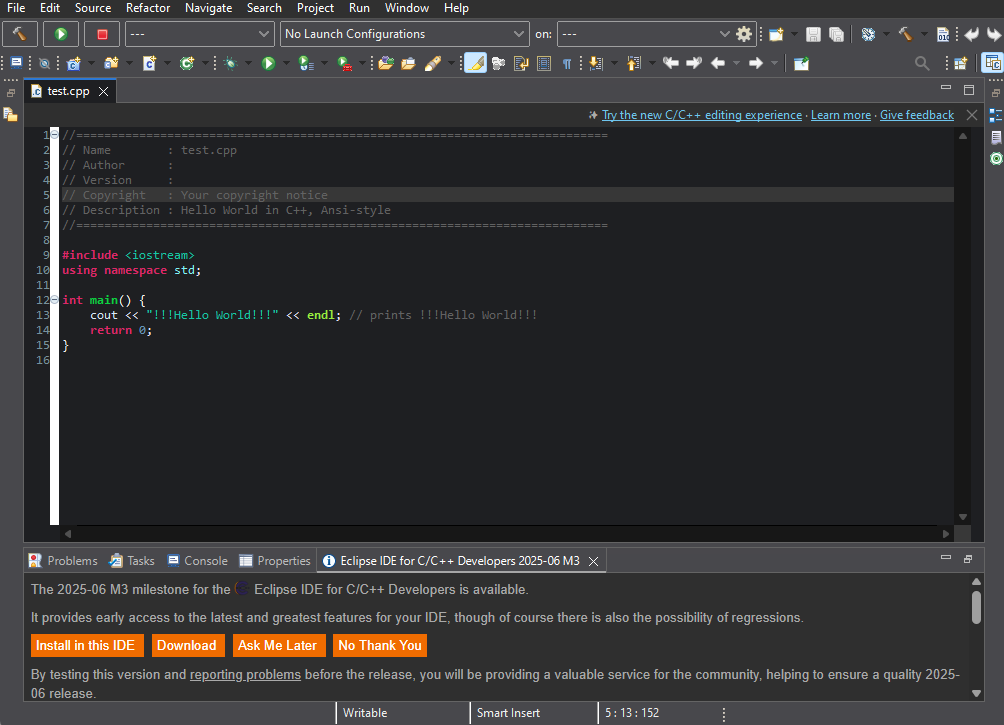


Рисунок 2.11 – інтерфейс Eclipse

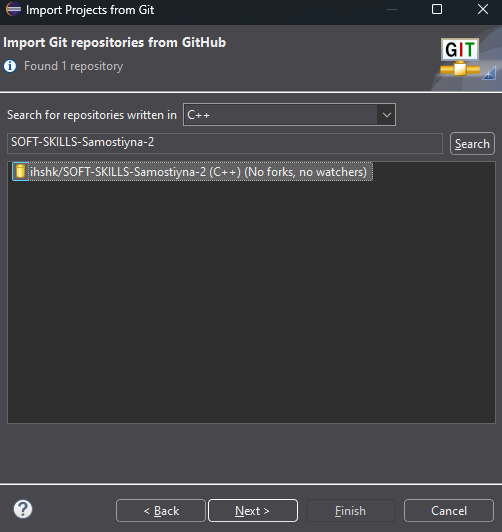


Рисунок 2.12 – налаштування доступу до Git-репозиторію в інтегрованому середовищі розробки Eclipse

1.2.9 Виконати реалізацію проекту в Eclipse та зазначити, яка частина коду ким була розроблена.



Рисунок 2.13 – реалізація одного з компонентних файлів проекту «main.cpp» в Eclipse, разом з зазнеченнями яку частину коду - хто розробив

1.2.10 Визначити додаткову функцію та реалізувати її, працюючи над нею одночасно вдвох.



Рисунок 2.14 – визначення додаткової функції за допомогою сервісу Slack

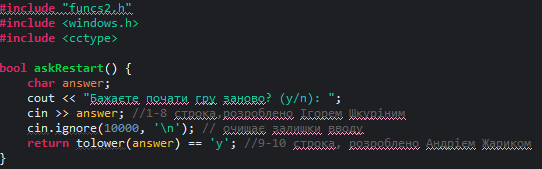


Рисунок 2.15 – розробка додаткової функції одночасно вдвох в Eclipse

1.2.12 Відповідно до діючих стандартів індивідуально оформити програмний документ, узгоджений з викладачем, на розроблене програмне забезпечення.

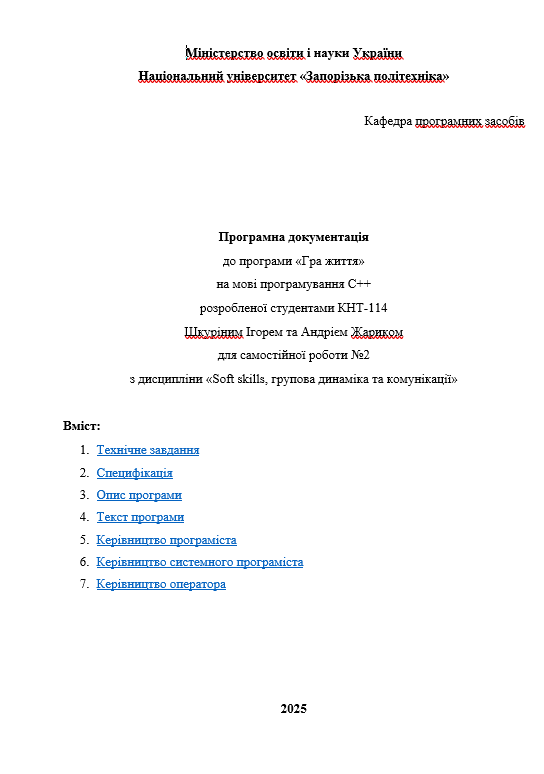


Рисунок 2.15 – титульний лист програмного документа на розроблене програмне забезпечення

**3 ВИСНОВКИ ТА КОНТРОЛЬНІ ЗАПИТАННЯ**

Висновки: У процесі виконання завдань було опрацьовано основні теоретичні відомості за темою роботи, що дозволило сформувати базове розуміння принципів командної розробки програмного забезпечення. Після формування команди з двох студентів та отримання індивідуального завдання від викладача, було створено акаунт на GitHub та організовано команду в Slack для зручного обміну інформацією. Було узгоджено розподіл зобов’язань між учасниками, що дало змогу ефективно координувати дії в межах проєкту. Створено репозиторій із наданим обом учасникам доступом, а також завантажено початкові файли у віддалене сховище. Надалі були створені дві гілки: одна — для демонстрації ревізій, інша — для відлагодження. Було успішно налаштовано підключення Git-репозиторію в середовищі Eclipse, у якому й здійснювалася основна розробка. У ході реалізації проєкту вказано, яка частина коду була написана кожним учасником, що забезпечило прозорість внеску кожного. Окремо було визначено додаткову функцію, над якою обидва учасники працювали одночасно, здійснюючи спільне планування, обговорення та програмування. Після завершення розробки програмного забезпечення кожен з учасників оформив свою частину програмного документу. Роботу було завершено складанням звіту, у якому відображено всі етапи виконання завдання, після чого було надано відповіді на контрольні запитання. Усі кроки були виконані з дотриманням вимог і термінів, що забезпечило успішне завершення роботи.

Контрольні запитання:

2.5.1 Обґрунтуйте необхідність оформлення програмної документації.

Оформлення програмної документації є необхідним етапом розробки програмного забезпечення, оскільки вона забезпечує чітке та зрозуміле описання функціональних і технічних аспектів проекту, полегшує підтримку, модифікацію та подальший розвиток системи, а також сприяє ефективній комунікації між учасниками розробки

2.5.2 Які основні документи оформлюються при розробленні програмного забезпечення?

– технічне завдання;

– специфікація;

– опис програми;

– текст програми;

– керівництво програміста;

– керівництво системного програміста;

– керівництво оператора.

2.5.5 Відповідно до якого стандарту оформляється документ «Технічне завдання»?

Згідно з ГОСТом 19.201–78

2.5.6 Які розділи містить технічне завдання?

– вступ;

– підстави для розробки;

– призначення розробки;

– вимоги до програми чи програмному виробу;

– вимоги до програмної документації;

– техніко-економічні показники;

– стадії та етапи розробки;

– порядок контролю та приймання [15].

2.5.7 Згідно з яким стандартом оформляється документ «Специфікація»?

Згідно з ГОСТом 19.202-78