Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

|  |  |
| --- | --- |
| Факультет | Комп’ютерної інженерії та управління |
|  | (повна назва) |

|  |  |
| --- | --- |
| Кафедра | Електронних обчислювальних машин |
|  | (повна назва) |

**КУРСОВА РОБОТА**

|  |
| --- |
|  |
| (рівень вищої освіти) |

|  |  |
| --- | --- |
| з дисципліни: | Системне програмування |
|  |  |
| на тему: | Розробка програмного забезпечення з використанням |
|  | функцій WinAPI: гра “Літак” |
|  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Виконав: студент | | | 2 | курсу, групи | | | КІУКІ-20-3 |
| спеціальності | | 123 «Комп’ютерна інженерія» | | | | | |
|  | | (код і повна назва спеціальності) | | | | | |
| Освітньо-професійної програми: «Комп’ютерна інженерія» | | | | | | | |
| (повна назва освітньої програми) | | | | | | | |
|  |  | | | |  | Нальотов Д.Р. | |
|  | (підпис) | | | |  | (прізвище, ініціали) | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
| Голова комісії: |  | | | |  | проф. Волк М.О. | |
| Члени комісії: | (підпис) | | | |  | (посада, прізвище, ініціали)  ст. викл. Колтун Ю.М. | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали)  ас. Ярошевич Р.О. | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали) | |

2022 р.

Харківський національний університет радіоелектроніки

(найменування вищого навчального закладу)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кафедра: | ЕОМ | | | | |
| Дисципліна: | Системне програмування | | | | |
| Спеціальність: | Комп’ютерна інженерія | | | | |
| Курс: | 2 | Група: | КІУКІ-20-3 | Семестр: | 4 |

**З А В Д А Н Н Я**

на курсовий проект (роботу) студента

|  |
| --- |
| Нальотова Дмитра Романовича |
| (прізвище, ім’я, по батькові) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Тема роботи | | Розробка програмного забезпечення з використанням | | | |
| функцій WinAPI: гра “Літак” | | | | | |
|  | | | | | |
| 2. Строк здачі студентом закінченої роботи | | | | | 09.07.2022 р. |
| 3. Вихідні дані до роботи | | | | Microsoft Visual Studio 2022, | |
| мова програмування C, WinAPI | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| 4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): | | | | | |
| 1) Аналіз предметної області | | | | | |
| 2) Опис структури програми та алгоритмів роботи | | | | | |
| 3) Опис програми | | | | | |
| 4) Технічні вимоги та інструкція користувача | | | | | |
| 5) Тестування програми | | | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): | | | | | |
| Лістинг програми, файл з заголовками функцій, файл з реалізацією функцій програми | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| 6. Дата видачі завдання: | | | 14.02.2022 | | |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Найменування етапів**  **курсової роботи** | **Строк виконання**  **етапів роботи** | **Примітка** |
|  |  |  |  |
| 1 | Ознайомлення з літературними | 23.03.22 - 26.03.22 |  |
|  | джерелами, аналіз та вибір методу |  |  |
|  | вирішення поставленої задачі |  |  |
|  |  |  |  |
| 2 | Розробка алгоритмів рішення, | 02.04.22 – 21.04.22 |  |
|  | вибір системних засобів вирішення |  |  |
|  | задач курсового проектування |  |  |
|  |  |  |  |
| 3 | Проектування додатку | 21.04.22 - 01.05.22 |  |
|  |  |  |  |
| 4 | Налагодження та тестування | 01.05.22 - 19.05.22 |  |
|  | програми |  |  |
|  |  |  |  |
| 5 | Оформлення пояснювальної | 19.06.22 - 03.07.22 |  |
|  | записки |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | |  |  | Нальотов Д.Р. |
|  | | (підпис) |  | (прізвище, ім’я, по батькові) |
|  | |  |  | Волк М.О. |
| Керівник | | (підпис) |  | (прізвище, ім’я, по батькові) |
| « 14 » | лютого | 2022 р. |  | |

РЕФЕРАТ

Пояснювальна записка до курсової роботи містить 5 розділів, 25 с., 5 рисунків, 3 лістингів коду програми, 5 використаних джерел.

Об’єкт дослідження – гра “Літак”.

Мета роботи – розробка віконної програми, яка реалізовує правила гри “Літак”.

Метод дослідження – вивчення літератури, складання алгоритму та налагодження текстової програми на комп’ютері.

Додаток зручно використовувати в розважальних цілях, та в цілях розвитку реакції і моторики пальців.

Програма складена на мові C++ та за допомогою засобів Win32Api в середовищі Microsoft Visual Studio 2019.

ЛІТАК, СНАРЯДИ, АЛГОРИТМ, ВІКОННА ПРОЦЕДУРА, ПОВІДОМЛЕННЯ, СТРУКТУРА, ВІКНО, ФАЙЛ, МЕНЮ, ФУНКЦІЯ.

ЗМІСТ

[ВСТУП 6](#_Toc104987993)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8](#_Toc104987994)

[2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМИ 9](#_Toc104987995)

[3 ПРОГРАМНА РЕАЛІЗАЦІЯ 12](#_Toc104987996)

4 ТЕХНІЧНІ ВИМОГИ І ТЕСТУВАННЯ ПРОГРАМИ 16

5 ІНСТРУКЦІЯ КОРИСТУВАЧА………………………………………...…….17

ВИСНОВКИ……..……………………………………………………………….19

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.……………………………………………[2](#_Toc104988000)0

ДОДАТОК А………………………..……………………………………………21

# ВСТУП

З початку 80-х почалося широке впровадження ЕОМ у діяльність людини.

ЕОМ дозволяє повсюдно автоматизувати людську діяльність та звільнити інтелектуальні ресурси для творчої праці. У зв'язку з широким використанням ЕОМ виникає у своєрідної “мінімалізації” витрат часу освоєння цієї техніки максимально широким колом користувачів.

Слово «комп'ютер» означає «вирахувати», тобто пристрій для обчислень. Це з тим, що перші комп'ютери створювалися як пристрої обчислень.

Хоча комп'ютери створювалися для чисельних розрахунків, незабаром виявилося, що можуть обробляти та інші види інформації - адже майже всі вони можуть бути представлені в числовій формі. Нині з допомогою комп'ютерів як проводяться числові розрахунки, а й готуються до друку книжки, створюються малюнки, кінофільми, музика, здійснюється управління заводами і космічними кораблями тощо. Комп'ютери перетворилися на універсальні засоби обробки всіх видів інформації, використовуваних людиною.[1]

Відносно високі можливості IBM PC - сумісних комп'ютерів з переробки інформації дозволили використовувати їх (а не більш потужні комп'ютери) як для вирішення більшості завдань у бізнесі, так і для багатьох особистих потреб користувача таких, як розвага та комп'ютерні ігри.

Як наслідок розвитку комп'ютерних обчислень з'явилися комп'ютерні ігри. Комп'ютерна гра – комп'ютерна програма, служить для організації ігрового процесу, зв'язку з партнерами по грі, або сама виступаюча як партнер [2].

В сучасному світі існує багато способів та підходів до створення ігор. Зараз будь-хто може за своїм бажанням створити невеличку гру, на відміну від великих ігрових проектів, її дуже просто створити.

В даній записці метою її створення є опис гри “Літак”. Це гра, в якій уявний літак літає поміж уявних снарядів. З часом швидкість снарядів підвіщується, тобто ще й піднімається складність гри. Час, який користувач зможе провести у цій грі,залежить лише від гравця,а саме його реакції на обставини та складності,що будуть виникати протягом гри.

Комп'ютерні ігри так само існують для того, щоб перетворити дозвілля або вільну хвилинку людини на розважальний, а іноді навіть повчальний час. Багато комп'ютерних ігор мають не тільки розважальну функцію, але й так само здатні розвивати реакцію, швидкість мислення. Ця гра “Літак” з комп'ютером спрямована на розвиток швидкості реакції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В даній роботі ми аналізуємо створення гри “Літак”. Мета роботи – реалізувати за допомогою бібліотеки WinApi та мови програмування С++ (рисунок 1.1).

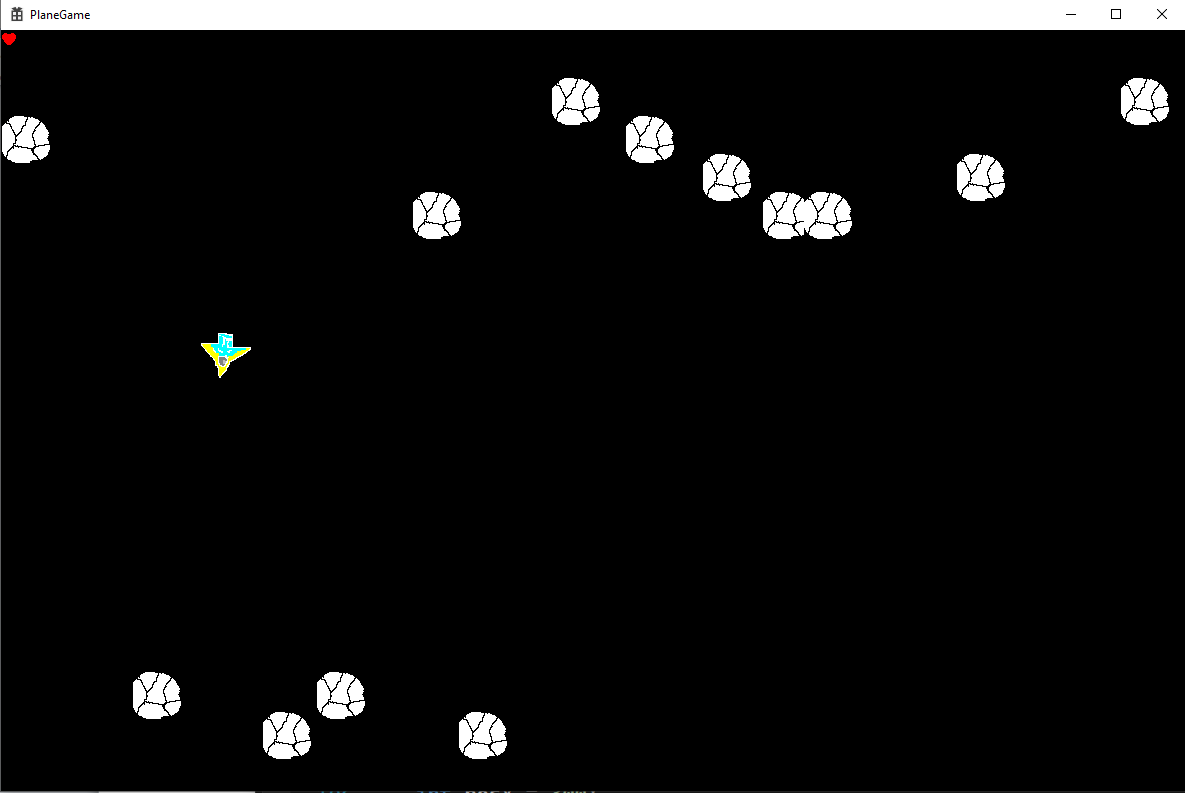


Рисунок 1.1 – Початковий вигляд вікна гри

Комп’ютерні ігри для людини – це не лише засіб розважитись. Вони допомагають розвинути людську реакцію, інтелект, знання в певних галузях і областях . У грі “Літак” вміння грати довго та не витрачати життя залежить лише від швидкості реакції гравця на поточні обставини гри, які динамічно змінюватимуться в сторону збільшення, що також сприятиме збільшенню реакції та її поліпшенню.

Правила гри дуже спрощенні та зрозумілі: за допомогою стрілок на клавіатурі ліворуч та праворуч ви керуєте літаком та рухаєтесь у лівий та правий бік відповідно. Головна задача це ухилятися від снарядів. Якщо ухилитися від снаряда було неможливо, то відбудеться зіткнення літака зі снарядом й гра закінчиться, сповістивши про це гравця відповідним діалоговим вікном

# Гра складається з “ігрового поля” та його об'єктів,а саме масив з 15 об’єктів “снаряд”.

# Об'єкти переміщуються за таймером. Таймер спрацьовує кожні 100 мс, тим самим приводячи снаряди в рух і викликає перевірку зіткнення снаряда з літаком. Літаки та снаряди при перетинанні краю екрана переносяться на інший кінець клієнтської області. При генерації снарядів вони випадковим чином можуть з'явитися як згори, так і знизу екрана і почати своє переміщення і до протилежного краю. У кожного снаряда є своя швидкість, яка генерується в межах максимальної та мінімальної. При перетині краю екрана краю вікна прискорюється, якщо ще не досяг максимальної швидкості. 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМИ

Алгоритм – послідовність команд, призначена виконавцю, у результаті якої він має вирішити поставлене завдання. Для комп’ютерних програм алгоритм є списком деталізованих інструкцій, у процесі виконання яких система з початкового стану переходить у кінцевий [3].

В процесі розробки гри було розроблено і використано показану на рисунку 2.1 структуру проекту.

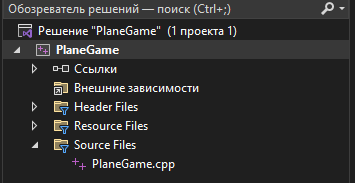


Рисунок 2.1 – Структура проекту

Проект складається з файлу Source.cpp – головного файлу програми, який реалізовує логіку роботи додатку. Усі функціі описані та реалізовані у початковому файлі .cpp.

Для розробки програми обрано бібліотеку WinApi а саме “Класичний додаток Windows”, та мову програмування C++.

Робота мого проекту може бути описана таким чином: з’являється літак, потім снаряди починають свій рух, та користувач повинен за допомогою клавіш “стрілка ліворуч” та “стрілка праворуч” ухилятися від снарядів.

Алгоритм роботи додатку, що розроблено можливо уявити наступними кроками:

Крок 1: запуск програми, відображення вікна зі грою.

Крок 2: гравцю потрібно переміщувати літак по полю і ухилятися від снарядів до тих пір, доки снаряд не влучить у літак.

Крок 3: Якщо гравець не уникнув зіткнення зі снарядом, гра закінчується,сповістивши про це гравця.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

В розділі програмної реалізації буде коротко описаний склад і функціонал cpp. файлу під назвою PlaneGame. Буде описаний зміст файлу, логіку виконання функцій, деякі рішення з приводу вирішення окремих задач, функцій.

Нижче наведено лістинги коду з поясненнями усіх логічних рішень, функцій.

RECT fieldSize - це структура, яка зберігає розміри ігрового поля

int planeSize – змінна яка вказує розмір літачка

int AsteroidSize – змінна яка вказує розмір снаряда

int posX – поточні положення літака на ігровому полі за координатою Х

int posY – поточні положення літака на ігровому полі за координатою Y

int dash – на скільки переміщується літак координаті Х за одне натискання клавіші

int maxAsteroidSpeed – максимальна швидкість снаряда

int minAsteroidSpeed – мінімальна швидкість снаряда

int asteroidCount – кількість снарядів

int LifeCount – кількість життів

int HeartSize – розмір картинки серця

int NeedToColision – необхідна кількість пікселів для зіткнення об'єктів

HDC hdcStone – контекст пристрою, в якому зберігається растрове зображення снаряда

HDC hdcPlane – контекст пристрою в якому зберігається растрове зображення літака

HDC hdcHeart – контекст пристрою, в якому зберігається растрове зображення серця

HBITMAP BitmapHeart – растрове зображення серця завантаженого з ресурсів

Class Object – це клас об’єкту снаряда. У ньому задається швидкість снаряда по координатам Х та Y, відбувається генерація випадкової швидкості по координатам Х та Y в межах максимальної і мінімальної. Також у цьому класі є метод, який переміщує снаряд по координатам Х та Y,якщо вони торкаються краю ігрового екрану;збільшує швидкість,після того,якщо вона ще не досягла максимальної; переміщує снаряд у протилежний край екрану – вгору чи донизу.

Клас об’єкту снаряда Class Object представлений у лістингу 3.1

class Object

{

public:

float x, y, SpeedX, SpeedY;

Object(float x, float y) :

x(x), y(y) {

SpeedY = (rand() % maxAsteroidSpeed \* 2) - maxAsteroidSpeed;

SpeedX = (rand() % maxAsteroidSpeed \* 2) - maxAsteroidSpeed;

}

void update()

{

x += SpeedX;

y += SpeedY;

if (x > fieldSize.right) x = 0;

if (x < 0) x = fieldSize.right;

if (y < 0) {

y = fieldSize.bottom;

if (SpeedY < maxAsteroidSpeed)

{

SpeedY -= 0.5;

}

}

if (y > fieldSize.bottom)

{

y = 0;

if (SpeedY < maxAsteroidSpeed)

{

SpeedY += 0.5;

}

}

}

};

Лістинг 3.1 – клас об’єкту снаряда

std::vector<Object\*> Asteroids – це масив снарядів

Функція GameOver() , яка використовується тоді, коли відбулося зіткнення снаряду з літаком. Вона зупиняє таймер через те, що снаряди більше не переміщаються, виводить повідомлення про те, що гравець програв, та гру закінченно

Функція GameOver() представлена нижче у лістингу 3.2

void GameOver() {

KillTimer(hWnd, 100);

MessageBox(hWnd, L"Игра окончена", L"Уведомление", MB\_OK);

}

Лістинг 3.2 - Функція GameOver()

Функція Colision() , яка перевіряє зіткнення літака зі снарядом. Вона має цикл по усім снарядам: перевірка збігу пікселів зі снарядами, що знаходяться згори,знизу,ліворуч,праворуч від літака.

Функція Colision() представлена нижче у лістингу 3.3

void Colision() {

for (int i = 0; i < Asteroids.size(); i++)

{

if ((Asteroids[i]->x + AsteroidSize > posX + NeedToColision && Asteroids[i]->x + AsteroidSize < posX + planeSize)

|| (posX + planeSize - NeedToColision > Asteroids[i]->x && Asteroids[i]->x > posX )) {

if ((Asteroids[i]->y < posY + NeedToColision && posY + NeedToColision < Asteroids[i]->y + AsteroidSize)

|| (Asteroids[i]->y < posY + planeSize + NeedToColision && posY + planeSize < Asteroids[i]->y + AsteroidSize - NeedToColision))

{

LifeCount--;

break;

}

}

}

}

Лістинг 3.2 – Функція Colision()

Основна віконна процедура програми обробляє такі повідомлення:

WM\_CREATE при обробці повідомлення спочатку завантажуються в растрові зображення: літака, снаряда, серця; після чого вони завантажуються окремі контексти пристроїв. Генеруються випадковим чином снаряди на ігровому полі, заповнюючи масив снарядом, після чого запускається таймер.

WM\_TIMER під час обробки повідомлення кожен снаряд переміщається на задану ним швидкість. Перевіряється чи зіткнувся снаряд із саме влітку. Якщо зіткнення сталося, то гра закінчується, повідомивши про це гравця.

WM\_KEYDOWN при обробці повідомлення переміщає літак у вказаному користувачем напрямку. Якщо при переміщенні літак перетнув край клієнтської області, тоді він переміститься в протилежний бік.

WM\_PAINT при обробці повідомлення створюється пензель на основі растрового з оброблення серця, попередньо завантаженого при створенні вікна, виводиться кількість життів гравця, використовуючи цю пензель, далі на екрані розташовуються снаряди в їх текучих позиціях, картинка снаряда копіюється з контексту пристрою, у власних координатах, картинка літачка береться з підготовленого заздалегідь контексту пристрою.

# 4 ТЕХНІЧНІ ВИМОГИ І ТЕСТУВАННЯ ПРОГРАМИ

Тестування програми проведено на операційній системі Microsoft Windows 10 x64, процесор Ryzen 5 1600x. Даний проект може бути використаний на будь-якій ОС Windows від XP до Windows 11 із архітектурою х64 та х86.

При розробці гри виникла проблема з перевіркою, суперечки астероїдів та літаків. Ця проблема була вирішена створенням функції калижн, в якій була наведена перевірка суперечки координат літака і кожного астероїда.

Можливий візуальний некоректний момент зображення зіткнення через те, що астероїд і літак є растровими зображеннями, у яких є пусті місця в кутах, а перевірка відбувається між квадратами, в яких знаходиться картинка. Ця проблема біла частина вирішена, тим, що зіткнення зараховується якщо два квадрати стикаються хоча б на 15 пікселів, тим самим зіткнення стали візуально зрозумілі для користувачів.

Також у програмі була виявлена ​​помилка в тому, що переміщення астероїдів і літаків виконано в одному потоці, через що салон може врізатися в астероїд, якщо користувач натиснув кнопку переміщення прямо перед моментом зіткнення, але в цей момент вже відбувається перевірка зіткнення літака з астероїдом.

Для покращення візуального оформлення гри було використано бітові образи растрових зображень, замість звичайних прямокутників.

Інших проблем чи помилок під час виконання роботи виявлено не було.

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

При запуску програми, користувач бачить початковий вигляд вікна, показаний на рисунку 5.1.

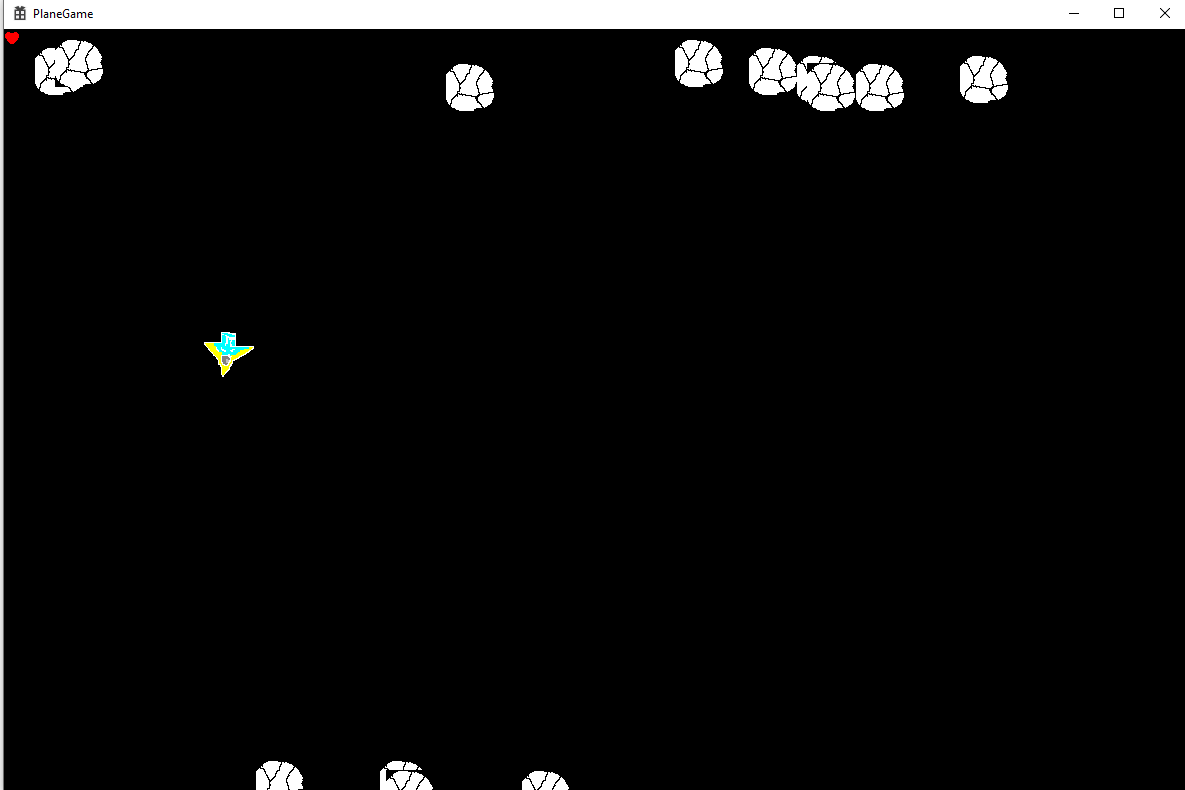


Рисунок 5.1 – Початковий вигляд вікна програми

На рисунку 5.2 ми можемо побачити приклад геймплею гри. Ми бачимо ігрове поле, по якому за допомогою стрілок на клавіатурі ми можемо рухати нашу літак, також на скріншоті є снаряди, властивості яких описані вище.

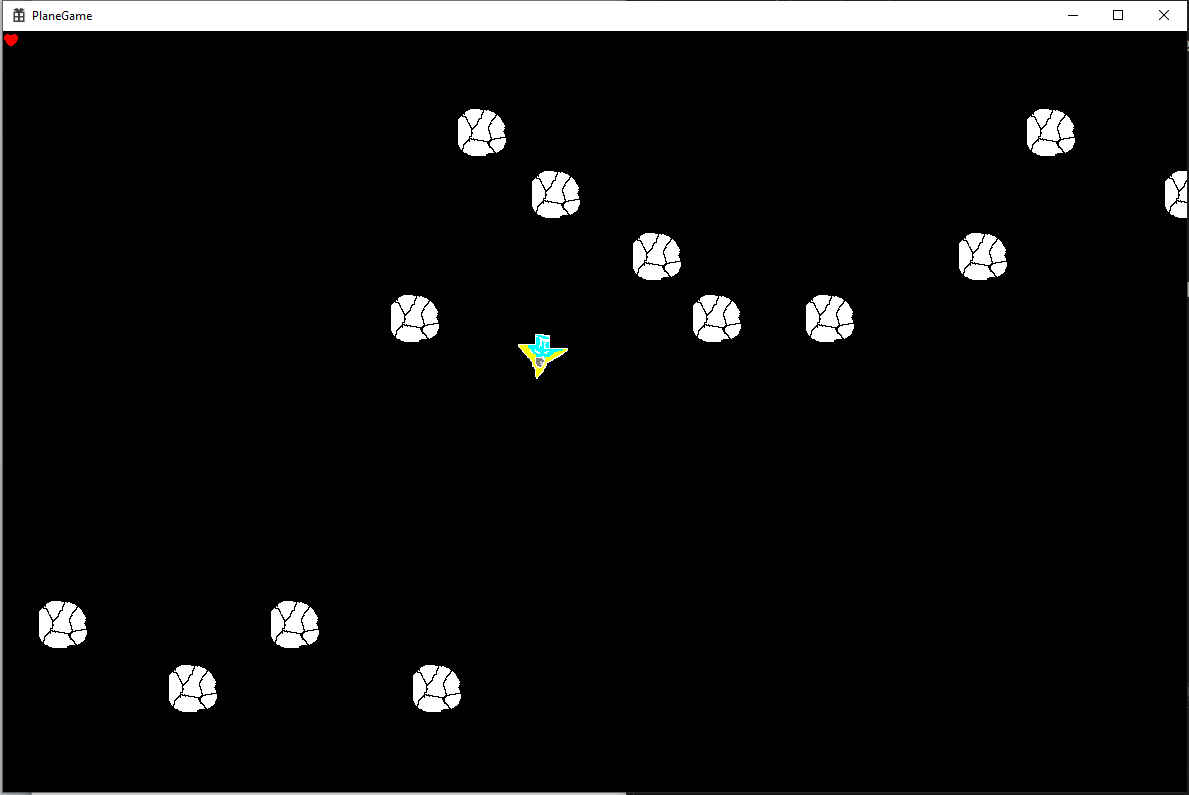


Рисунок 5.2 – Приклад геймплею

Якщо користувач зіткнеться зі снарядом, то гра буде завершена з відповідним повідомленням (рисунок 5.3).

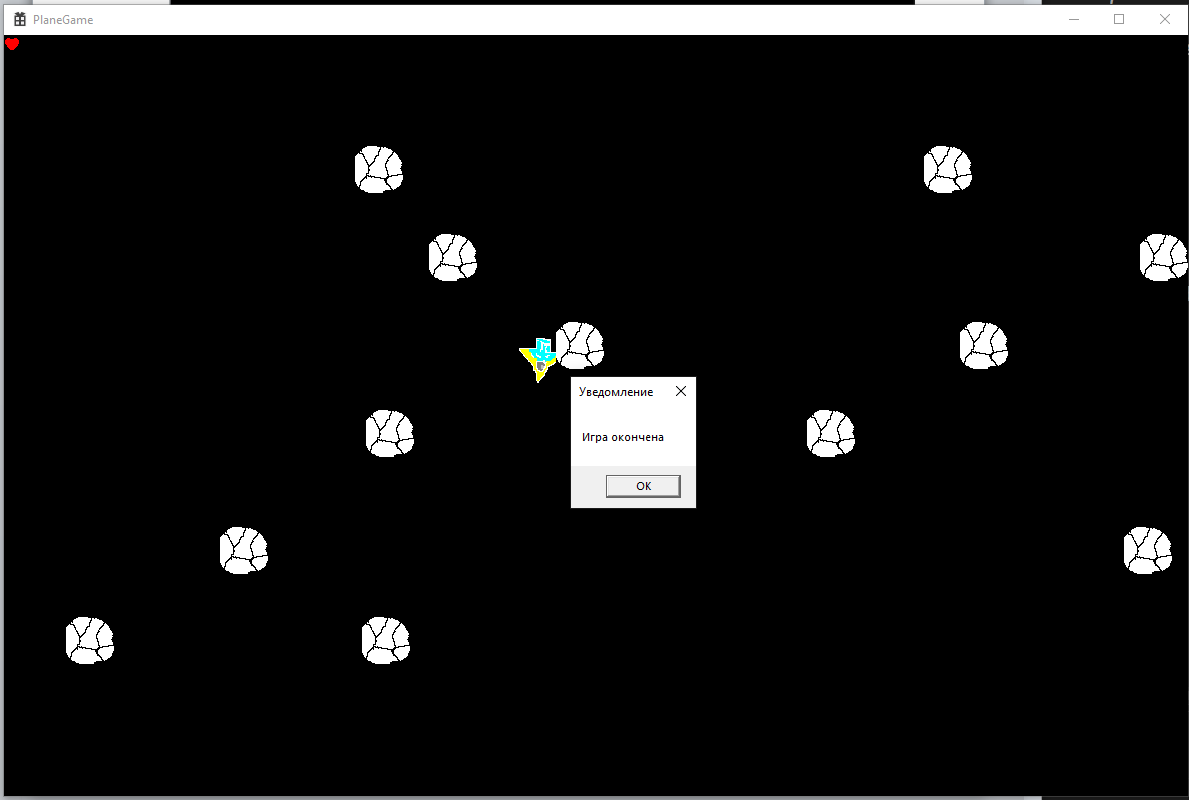


Рисунок 5.3 – Приклад кінця гри

# ВИСНОВКИ

В ході курсового проектування було проаналізовано задану предметну область і розроблено гру “Літак”. Для реалізації поставленої задачі використано інтегроване середовище розробки Microsoft Visual Studio 2022 бібліотеку WinApi. Код програми написано мовою програмування C++.

Після узгодження програми з викладачем, показана програма була вдосконалена і дороблена до кінця з усіми поставленими вимогами викладача.

Тестування програми показало деякі помилки, які були виявлені та були виправлені, усе зазначено у пункті 4 (Технічні вимоги і тестування програми). Програма працює справно, згідно з інструкцією користувача, яка зазначена в 5 пункті роботи (Інструкція користувача).

Програма може бути вдосконалена шляхом додавання різних інтерфейсів, фонів і растрових зображень, додавання рахунку очків в самій програмі і зміною кольору змійки на вибір користувача.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Прата С. Мова програмування C++. Лекції та вправи. 6-е видання. М.: “Діалектика-Вільямс”. 2012, 1248 с.

2. Лаптев В. В. C++. Объектно-ориентированное программирование: Учебное пособие / В. В Лаптев. — СПб.: Питер, 2008. —464 с Кравець П.О. Об'єктно-орієнтоване програмування. Начальний посібник. Л., 2012.

3. Бхаргава А. [Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих](https://www.piter.com/collection/all/product/grokaem-algoritmy-illyustrirovannoe-posobie-dlya-programmistov-i-lyubopytstvuyuschih-2): Питер. 2017, 288 с.

4. Николай Литвиненко. Технология программирования на С++ Win32 API-приложения: БХВ-Петербург. 2003, 572 с.

5. Посилання на проект на Google Drive. URL: https://drive.google.com/drive/folders/10PN6gRmRj2d4KfKjS8xVayfsUzPVdkbc

# ДОДАТОК А

Лістинг програми

// PlaneGame.cpp : Defines the entry point for the application.

//

#include "framework.h"

#include "resource.h"

#include "PlaneGame.h"

#include <vector>

#define MAX\_LOADSTRING 100

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_PLANEGAME, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_PLANEGAME));

MSG msg;

// Main message loop:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_PLANEGAME));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+3);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

HWND hWnd;

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, 1200, 800, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

RECT fieldSize;

int planeSize = 50;

int AsteroidSize = 48;

int posX = 200;

int posY = 300;

int dash = 15;

int maxAsteroidSpeed = 10;

int minAsteroidSpeed = 4;

int asteroidCount = 15;

int LifeCount = 1;

int HeartSize = 16;

int NeedToColision = 15;

HDC hdcStone;

HDC hdcPlane

HDC hdcHeart;

HBITMAP BitmapHeart;

class Object

{

public:

float x, y, SpeedX, SpeedY;

Object(float x, float y) :

x(x), y(y) {

SpeedY = (rand() % maxAsteroidSpeed \* 2) - maxAsteroidSpeed;

SpeedX = (rand() % maxAsteroidSpeed \* 2) - maxAsteroidSpeed;

}

void update()

{

x += SpeedX;

y += SpeedY;

if (x > fieldSize.right) x = 0;

if (x < 0) x = fieldSize.right;

if (y < 0) {

y = fieldSize.bottom;

if (SpeedY < maxAsteroidSpeed)

{

SpeedY -= 0.5;

}

}

if (y > fieldSize.bottom)

{

y = 0;

if (SpeedY < maxAsteroidSpeed)

{

SpeedY += 0.5;

}

}

}

};

std::vector<Object\*> Asteroids;

void GameOver() {

KillTimer(hWnd, 100);

MessageBox(hWnd, L"Игра окончена", L"Уведомление", MB\_OK);

}

void Colision() {

for (int i = 0; i < Asteroids.size(); i++)

{

if ((Asteroids[i]->x + AsteroidSize > posX + NeedToColision && Asteroids[i]->x + AsteroidSize < posX + planeSize)

|| (posX + planeSize - NeedToColision > Asteroids[i]->x && Asteroids[i]->x > posX )) {

if ((Asteroids[i]->y < posY + NeedToColision && posY + NeedToColision < Asteroids[i]->y + AsteroidSize)

|| (Asteroids[i]->y < posY + planeSize + NeedToColision && posY + planeSize < Asteroids[i]->y + AsteroidSize - NeedToColision))

{

LifeCount--;

break;

}

}

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

HBITMAP BitmapStone = LoadBitmap(hInst, MAKEINTRESOURCEW(Stone));

HBITMAP BitmapPlane = LoadBitmap(hInst, MAKEINTRESOURCEW(Plane));

BitmapHeart = LoadBitmap(hInst, MAKEINTRESOURCEW(Heart));

HDC hdc = GetDC(hWnd);

hdcStone = CreateCompatibleDC(hdc);

hdcPlane = CreateCompatibleDC(hdc);

hdcHeart = CreateCompatibleDC(hdc);

SelectObject(hdcStone, BitmapStone);

SelectObject(hdcPlane, BitmapPlane);

SelectObject(hdcHeart, BitmapHeart);

ReleaseDC(hWnd, hdc);

GetClientRect(hWnd, &fieldSize);

for (int i = 0; i < asteroidCount; i++)

{

Object\* temp = new Object(rand() % fieldSize.right, fieldSize.bottom);

Asteroids.push\_back(temp);

}

SetTimer(hWnd, 100, 100, NULL);

break;

}

case WM\_TIMER:

{

for (int i = 0; i < Asteroids.size(); i++)

{

Asteroids[i]->update();

}

Colision();

if (LifeCount == 0)

{

GameOver();

}

InvalidateRect(hWnd, NULL, true);

break;

}

case WM\_KEYDOWN: {

switch (wParam)

{

case VK\_RIGHT: {

if (posX < fieldSize.right - planeSize - dash)

{

posX += dash;

}

else

{

posX = 0;

}

break;

}

case VK\_LEFT: {

if (posX > dash)

{

posX -= dash;

}

else {

posX = fieldSize.right - planeSize - 1;

}

break;

}

default:

break;

}

break;

}

case WM\_COMMAND: {

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

HBRUSH HeartBrush = CreatePatternBrush(BitmapHeart);

StretchBlt(hdc, posX, posY,

planeSize,

planeSize, hdcPlane,

0, 0, planeSize, planeSize, SRCCOPY);

for (int i = 0; i < Asteroids.size(); i++)

{

StretchBlt(hdc, Asteroids[i]->x, Asteroids[i]->y,

AsteroidSize,

AsteroidSize, hdcStone,

0, 0, AsteroidSize, AsteroidSize, SRCCOPY);

}

SelectObject(hdc, HeartBrush);

for (int i = 0; i < LifeCount; i++)

{

Rectangle(hdc, HeartSize \* i, 0, HeartSize \* i + HeartSize, HeartSize);

}

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}