МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра моделювання і програмного забезпечення

К У Р С О В А Р О Б О Т А

з дисципліни “Об’єктно-орієнтоване програмування”
на тему: “Розробка класу-моделі польоту снаряда”
студента 2 курсу ФІТ групи ІПЗ-21-2
Губарєва Ростислава Вадимовича

Керівник курсової роботи
д.т.н., доц. Котов І.А.

Кривий Ріг
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра моделювання і програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедрою

к.п.н., доц. А.М. Стрюк

12 березня 2023р.

З А В Д А Н Н Я

На курсову роботу

з дисципліни “Об’єктно-орієнтоване програмування”

студента 2 курсу ФІТ групи ІПЗ-21-2

Губарєва Ростислава Вадимовича

Тема курсової роботи “Розробка класу-моделі польоту снаряда”

Керівник курсової роботи доц. Котов І.А.

Мета роботи:

Розробка класу-моделі польоту снаряда. Клас має містити параметри снаряда та гармати (вага снаряда, кут нахилу гармати, стартова швидкість). Методи класу розраховують параметри траєкторії польоту снаряда (висота, дальність).

ЗМІСТ

ВСТУП

Моделювання польоту снарядів є важливою задачею в багатьох галузях, таких як військова техніка, авіація, космічна техніка та інші. Розробка класу-моделі польоту снаряда є важливим етапом у процесі розробки нових технологій і поліпшення існуючих.

Метою даної курсової роботи є розробка класу-моделі польоту снаряда, яка дозволить відтворювати поведінку снаряда в різних умовах. У роботі буде досліджено основні закономірності, які впливають на політ снаряда, такі як динамічний стан снаряда, аеродинамічні характеристики, вплив середовища, траєкторію польоту та інші.

Для класу-моделі буде використано середу розробки Visual Studio 2022 мовою програмування C#, яка дозволить реалізувати модель та провести чисельні експерименти. Результатом роботи буде клас-модель, яка буде включати в себе основні характеристики снаряда та дозволить проводити аналіз його поведінки в різних умовах.

Ця курсова робота має велике значення для розвитку технічного прогресу та вдосконалення сучасної техніки. Розробка ефективної класу-моделі польоту снаряда може сприяти створенню нових технологій, які в подальшому можуть використовуватись в різних галузях промисловості та науки.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ РОБОТИ

1.1. Аналіз професійної галузі проблеми

Аналіз професійної галузі показує, що розробка моделі польоту снарядів є важливою задачею для багатьох галузей, таких як військова техніка, авіація, космічна техніка та інші. Для успішного виконання цих задач необхідно мати досконалі моделі польоту снарядів, які дозволяють точно передбачити поведінку снарядів у різних умовах.

У військовій техніці модель польоту снаряду є важливою задачею для розробки і вдосконалення систем зброї та засобів оборони. Точність та ефективність використання зброї залежить від того, наскільки точно можна передбачити поведінку снаряду. У авіації та космічній техніці ця модель дозволяє вивчити аеродинамічні характеристики снаряду та його поведінку у різних атмосферних умовах. Крім того, розробка моделі польоту снарядів може знайти застосування в галузі безпілотних літальних апаратів (БПЛА). Клас-модель польоту дозволяє створити точну симуляцію польоту БПЛА, що дозволяє забезпечити безпеку польотів та уникнути аварій.

Також модель польоту снаряду може використовуватись у спортивних дисциплінах, наприклад, у стрільбі з лука або з пневматичної зброї. В цих видах спорту дуже важливо мати точну інформацію про політ снаряду, щоб передбачити його місце падіння та забезпечити максимальну влучність удару.

Модель польоту снарядів також має важливе значення для наукових досліджень, зокрема у фізиці, математиці та інших наукових галузях. За допомогою цієї моделі можна вивчати закономірності руху об'єктів в атмосфері та в космосі, досліджувати вплив різних факторів на поведінку об'єктів, визначати їх параметри та характеристики.

1.2. Аналіз існуючих аналогів

В інтернеті є готові рішення в області проектування моделі польоту снаряду. Зазвичай це онлайн калькулятори для розрахунку формули, за якою летить снаряд, та дослідження впливу різних умов на траєкторію снаряда.

Як приклад, є онлайн-застосунок Projectile Motion від компанії PhET.

https://phet.colorado.edu/en/simulations/projectile-motion

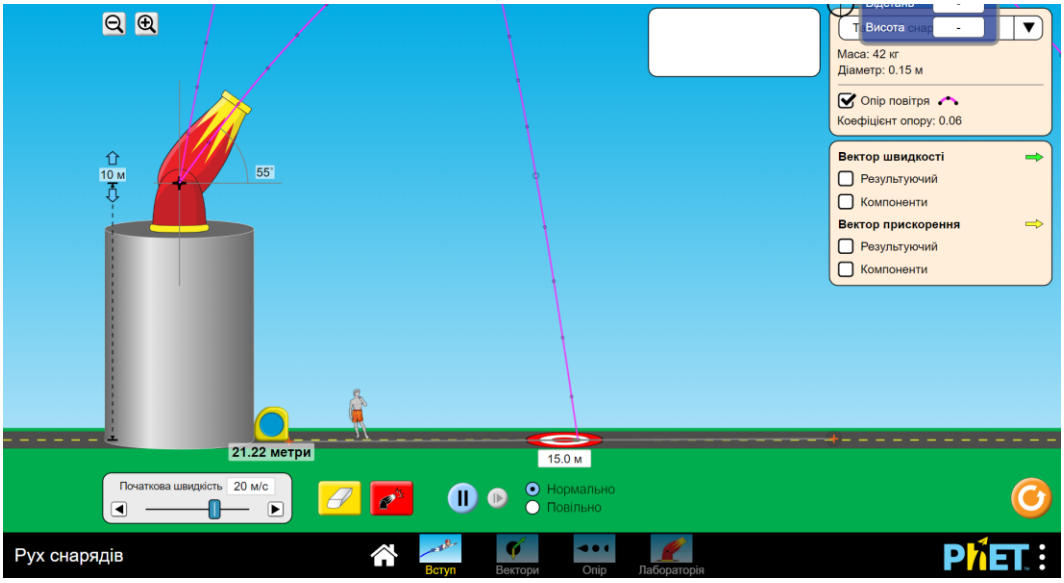


Рисунок 1.1 - Приклад інтерфейсу Projectile Motion від компанії PhET

За допомогою цього застосунку можна:

1. Визначити, як кожен параметр (початкова висота, початковий кут, початкова швидкість, маса, діаметр і висота) впливає на траєкторію об’єкта з опором повітря та без нього;
2. Передбачити, як зміна початкових умов вплине на траєкторію снаряда;
3. Оцінити, куди приземлиться об’єкт, враховуючи його початкові умови;
4. Описати вплив опору на швидкість і прискорення.

Компанія PhET дуже відповідально підійшла до створення симуляції польоту снаряду, додавши в неї розрахунок векторів швидкості, прискорення і сили, а також розрахунок впливу коефіцієнту опору і висоти над рівнем моря.

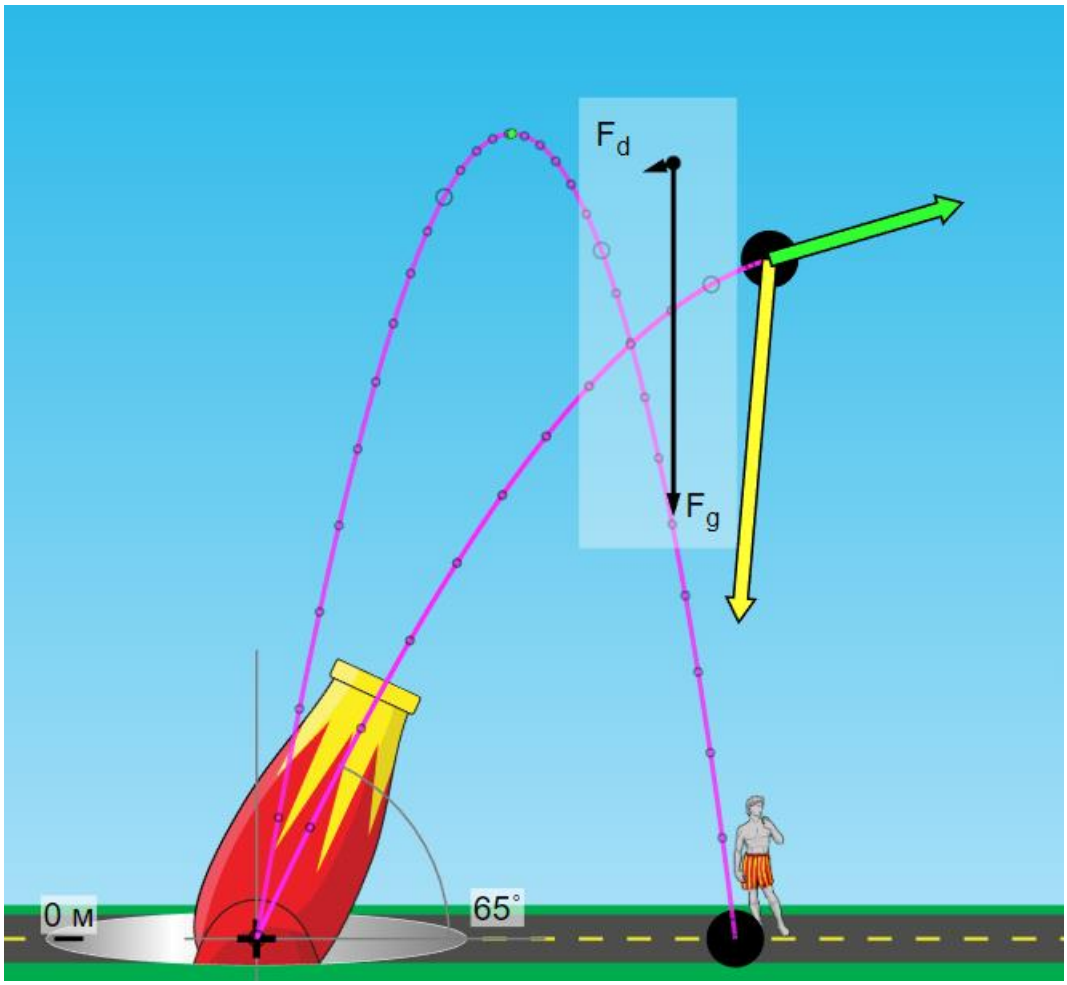


Рисунок 1.2 - Обчислення векторів швидкості, прискорення і сили

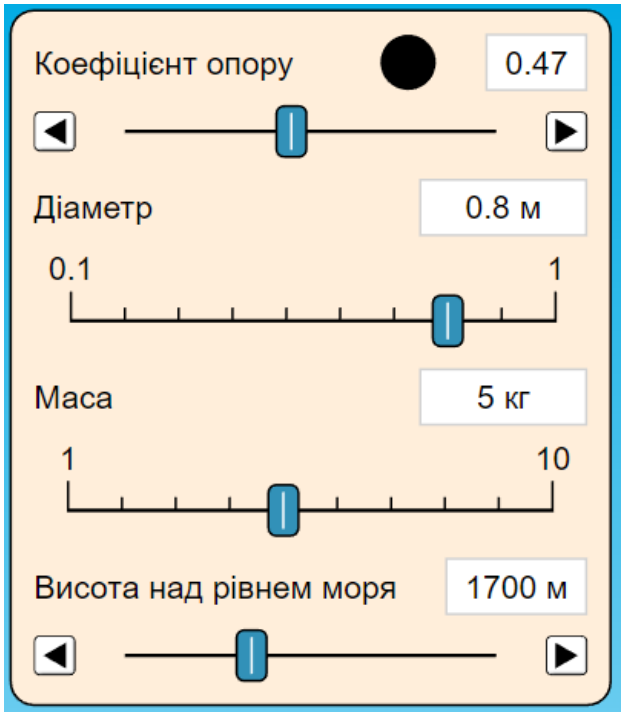


Рисунок 1.3 - Меню налаштування коефіцієнту опору, діаметру, маси і висоти над рівнем моря

Також розрахунок польоту снаряду є в багатьох іграх жанру шутер та воєнний симулятор.

1. **PUBG**

У цій грі, балістика відображає поведінку кулі під час польоту до місця призначення.

При пострілі зі зброї в PUBG, куля починає рухатися з високою швидкістю і залежно від того, який тип зброї використовується, має різну швидкість польоту. Наприклад, патрон 7,62 мм має меншу швидкість польоту, ніж 5,56 мм патрон.

Крім того, дальність до цілі також впливає на балістику в PUBG. Чим далі ціль знаходиться, тим більше часу потрібно кулі, щоб долетіти до неї.

Всі ці фактори разом враховуються в PUBG, щоб надати гравцям максимально реалістичне відчуття відстрілу з різних типів зброї в різних умовах. Це дозволяє гравцям точно визначити потрібний кут нахилу та висоту прицілу, щоб влучити у ціль на різних відстанях та в різних умовах.

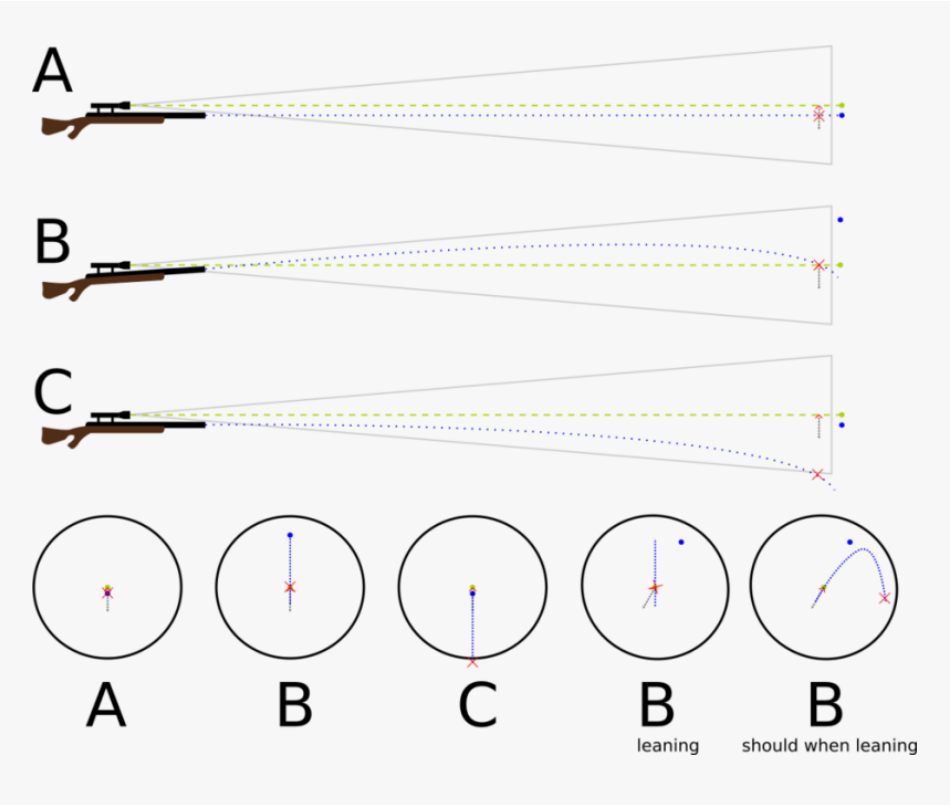


Рисунок 1.4 - Порівняння падіння кулі в прямому положені та під кутом

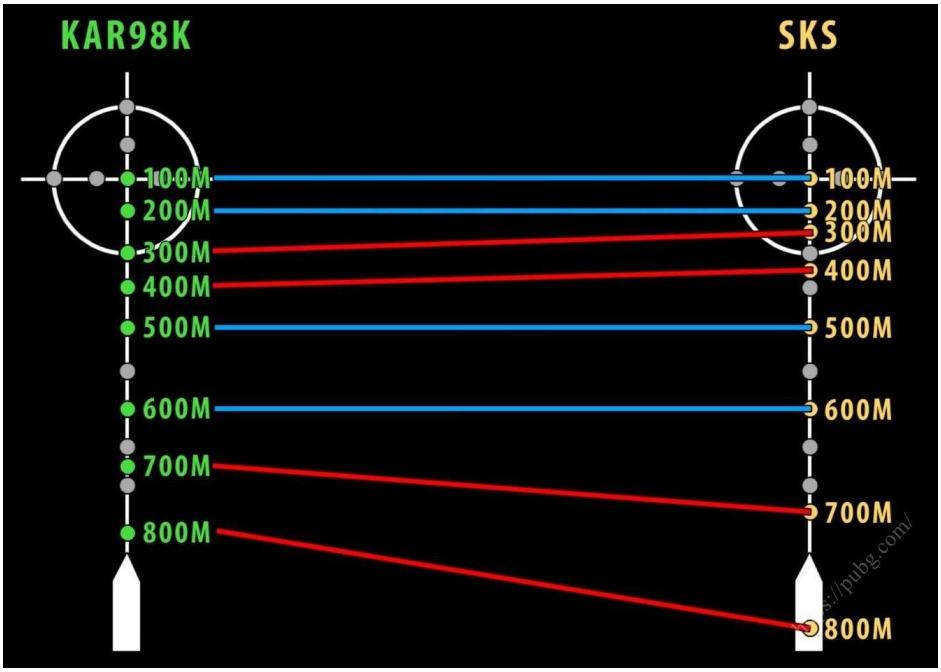


Рисунок 1.5 - Порівняння падіння пулі з різних відстаней з Kar98k та СКС

Рисунок 1.6 – Крива падіння пулі

2. War Thunder

У грі War Thunder, балістика описує поведінку снарядів та ракет під час польоту в повітрі та їх взаємодію з цілями. Кожен тип зброї має свою власну балістичну характеристику, яка визначає траєкторію польоту та поведінку снаряду в повітрі.

При пострілі зі зброї в War Thunder, снаряд вилітає зі ствола з певною початковою швидкістю, яка залежить від типу зброї та навантаження вогнепальної порохової суміші.

Також враховується вплив дальності до цілі на балістику в War Thunder. Чим далі ціль знаходиться, тим більше часу потрібно снаряду, щоб долетіти до неї, тому швидкість польоту снаряду буде меншою на більших відстанях.

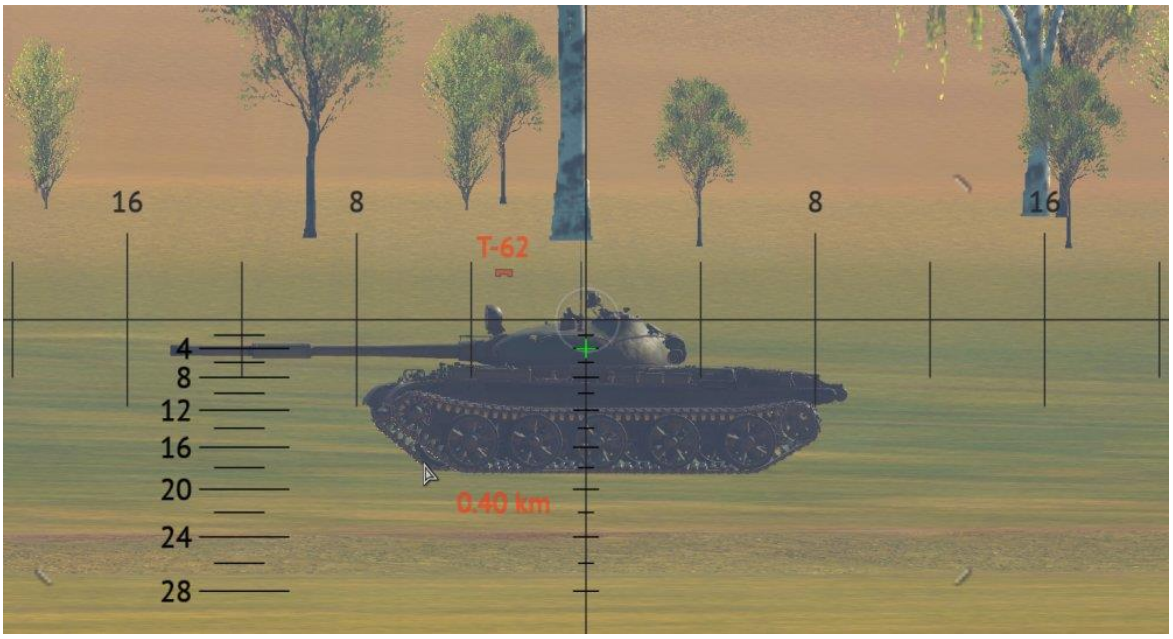


Рисунок 1.7 - Приціл танку KPz-70. Зелений хрестик вказує куди впаде снаряд на відстані 400 метрів.

У грі War Thunder також враховуються інші фактори, такі як тип польоту (наприклад, прямолінійний чи параболічний), маса та форма снаряду, а також властивості цілі (наприклад, броня та форма). Всі ці фактори взаємодіють та визначають траєкторію польоту та влучення снаряду у ціль.

Наприклад, при стрільбі з гармати, траєкторія польоту снаряду може бути параболічною, з висотою польоту, що залежить від кута нахилу гармати та початкової швидкості снаряду. Також враховується вплив гравітації на траєкторію польоту снаряду, яка змінюється залежно від кута нахилу гармати та відстані до цілі.

88mm KwK 43 PzGr 39/43					
Range	Q.E.	ToF	velocity	vertex	drop
m	mils	s	m/s	m	m
0	0	0	1000	0	0.00
100	0.50	0.101	989.2	0.01	0.05
200	1.01	0.202	978.5	0.05	0.20
300	1.53	0.305	968.0	0.11	0.46
400	2.06	0.409	957.5	0.21	0.82
500	2.59	0.514	947.2	0.33	1.30
600	3.13	0.620	937.0	0.47	1.88
700	3.68	0.727	926.9	0.65	2.59
800	4.24	0.836	916.9	0.86	3.43
900	4.80	0.945	907.0	1.10	4.38
1000	5.37	1.056	897.2	1.37	5.47
1100	5.95	1.168	887.5	1.68	6.69
1200	6.54	1.282	878.0	2.02	8.06
1300	7.14	1.396	868.5	2.39	9.56
1400	7.75	1.512	859.1	2.81	11.21
1500	8.36	1.629	849.8	3.26	13.01
1600	8.98	1.747	840.7	3.74	14.96
1700	9.62	1.867	831.6	4.27	17.09
1800	10.26	1.988	822.6	4.84	19.38
1900	10.91	2.110	813.8	5.45	21.83
2000	11.57	2.234	805.0	6.10	24.47
2100	12.24	2.358	796.3	6.80	27.26
2200	12.91	2.485	787.7	7.54	30.28
2300	13.60	2.612	779.2	8.33	33.45
2400	14.30	2.741	770.8	9.17	36.84
2500	15.01	2.872	762.5	10.05	40.44
2600	15.72	3.004	754.3	10.99	44.25
2700	16.45	3.137	746.1	11.97	48.25
2800	17.19	3.272	738.1	13.01	52.49
2900	17.93	3.408	730.1	14.10	56.95
3000	18.69	3.546	722.2	15.25	61.65
3100	19.46	3.685	714.4	16.46	66.58
3200	20.24	3.826	706.7	17.72	71.78
3300	21.03	3.968	699.1	19.04	77.20
3400	21.83	4.112	691.6	20.43	82.91
3500	22.64	4.257	684.1	21.87	88.86
3600	23.46	4.404	676.7	23.38	95.10
3700	24.30	4.553	669.4	24.96	101.64
3800	25.14	4.703	662.2	26.60	108.45
3900	26.00	4.855	655.1	28.31	115.58
4000	26.87	5.008	648.0	30.09	122.97

Рисунок 1.8 - таблиця балістики гармати 88mm KwK 43 PzGr 39/43. Range – дальність польоту снаряду, ToF – час польоту, velocity – швидкість польоту, drop – висота падіння снаряду під час польоту.

Крім того, при стрільбі по танку, враховуються його броня та форма. Наприклад, якщо снаряд попадає в танк з певного кута, він може не пройти через його броню, а просто відскочити від неї. Також враховується форма танка та його моделі в грі, що визначає зони слабкої та сильної броні.

Усі ці фактори дозволяють створити досить реалістичну балістику в грі War Thunder.

1.3. Формулювання актуальності та завдань роботи

Актуальність даної курсової роботи полягає в тому, що розробка класу-моделі польоту снарядів є важливою задачею для різних галузей техніки та промисловості, таких як військова техніка, авіація, космічна техніка, спортивні дисципліни тощо. Точність та ефективність снарядів є критичними факторами для успішного виконання завдань у цих галузях.

Основним завданням даної курсової роботи є розробка класу-моделі польоту снаряду. Конкретно, робота передбачає визначення аеродинамічних характеристик снаряду та їх впливу на поведінку снаряду в різних умовах, таких як швидкість вітру, висота польоту тощо.

Після розробки класу, будуть проведені різні тестові випробування з метою перевірки точності та ефективності розробленої моделі. Зокрема, будуть проведені випробування з використанням різних початкових умов та умов навколишнього середовища. Дані тестові випробування дозволять оцінити точність та ефективність розробленої моделі польоту снаряду.

Отже, головними завданнями даної курсової роботи є:

1. Розробка програми для обчислення траєкторії польоту снаряда;
2. Розробка блок-схеми роботи програми. Реалізувати наступні операції:
    2.1. Введення інформації про вагу снаряда, кут нахилу гармати, початкову швидкість;
    2.2. Обчислення траєкторії польоту снаряда (висота, дальність);
    2.3. Виведення результату на екран в консольному режимі.
3. Розробка класів;
4. Розробка структури меню для зручності використання програми;
5. Всі операції і алгоритми реалізувати за допомогою методів класів;
6. Створити звіт про роботу програми.

РОЗДІЛ 2. ТЕОРЕТИЧНИЙ БАЗИС ДЛЯ ВИРІШЕННЯ ЗАВДАНЬ РОБОТИ

2.1. Розробка математичних моделей

Один з підходів полягає у використанні чисельних методів, таких як метод Рунге-Кутти або метод Ейлера, для ітеративного розв'язання рівнянь руху з урахуванням маси, площі та коефіцієнта опору повітря. Ці методи дозволяють апроксимувати рух снаряду з кроком часу і оновлювати його координати та швидкість з кожним кроком.

Конкретний приклад розрахунку може виглядати наступним чином:

1. Визначення початкових умов:
    - Початкова швидкість (v0) снаряду
    - Кут відхилення (θ) снаряду від горизонту
    - Маса снаряду (m)
    - Площа снаряду (A)
    - Коефіцієнт опору повітря (C)
2. Встановлення початкових значень:
    - Час (t) = 0
    - Горизонтальна координата (x) = 0
    - Вертикальна координата (y) = 0
    - Горизонтальна складова швидкості (vx) = v0 * cos(θ) (Формула 1)
    - Вертикальна складова швидкості (vy) = v0 * sin(θ) (Формула 2)
3. Повторення наступних кроків до досягнення бажаного моменту часу або досягнення землі: a. Розрахунок прискорень у горизонтальному та вертикальному напрямках:
    - Горизонтальне прискорення (ax) = - (1/2) * ρ * A * C * vx^2 / m (Формула 3)

- Вертикальне прискорення (ay) = -g - (1/2) * ρ * A * C * vy^2 / m, де ρ - густина повітря
  (Формула 4)

4. Розрахунок нових значень координат та швидкостей:
   - x = x + vx * Δt (Формула 5)
   - y = y + vy * Δt (Формула 6)
   - vx = vx + ax * Δt (Формула 7)
   - vy = vy + ay * Δt (Формула 8)
   - Час (t) = t + Δt, де Δt - крок часу (Формула 9)
5. Повторення кроку 3 до досягнення бажаного моменту часу або поки снаряд не досягне землі (y <= 0).

2.2. Розробка структурної та функціональної (сценарій роботи) схем програмного комплексу

Функціональна та структурна схеми використовуються для кращого розуміння роботи та функціоналу програми.

Нижче представлена структурна схема, що показує які елементи будуть реалізовані в програмі.

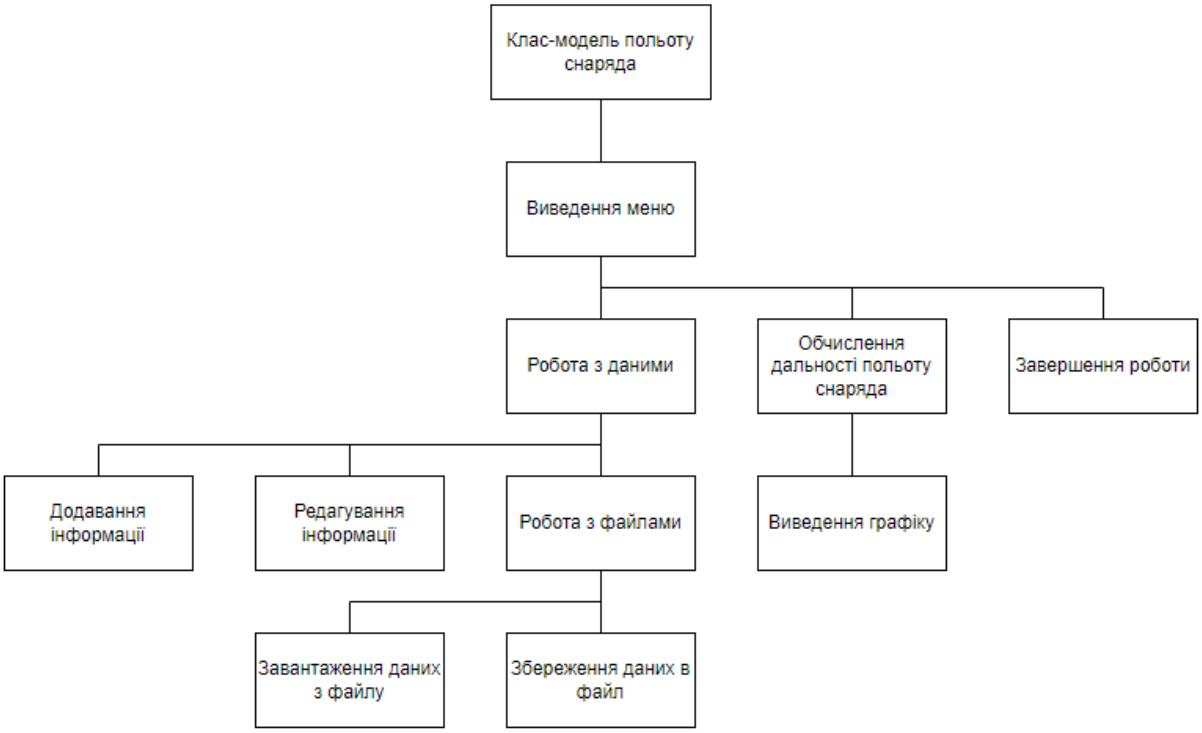


Рисунок 2.2 - Структурна схема програми

На рисунках 2.3 і 2.4 представлена функціональна схема, яка описує поведінку системи під час роботи програми.

Після введення користувачем даних про характеристики снаряду і даних про середовище польоту снаряда програма обчислює дальність польоту за допомогою формули і виводить результат в консольному режимі.

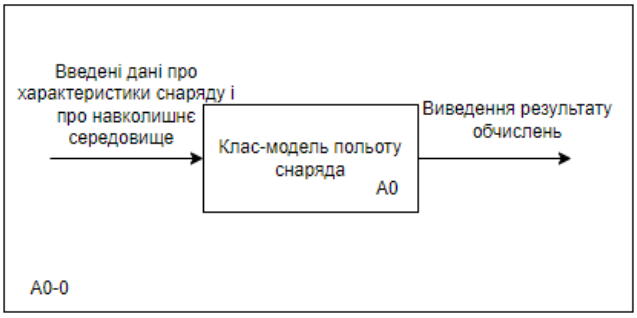


Рисунок 2.3 - Функціональна схема програми

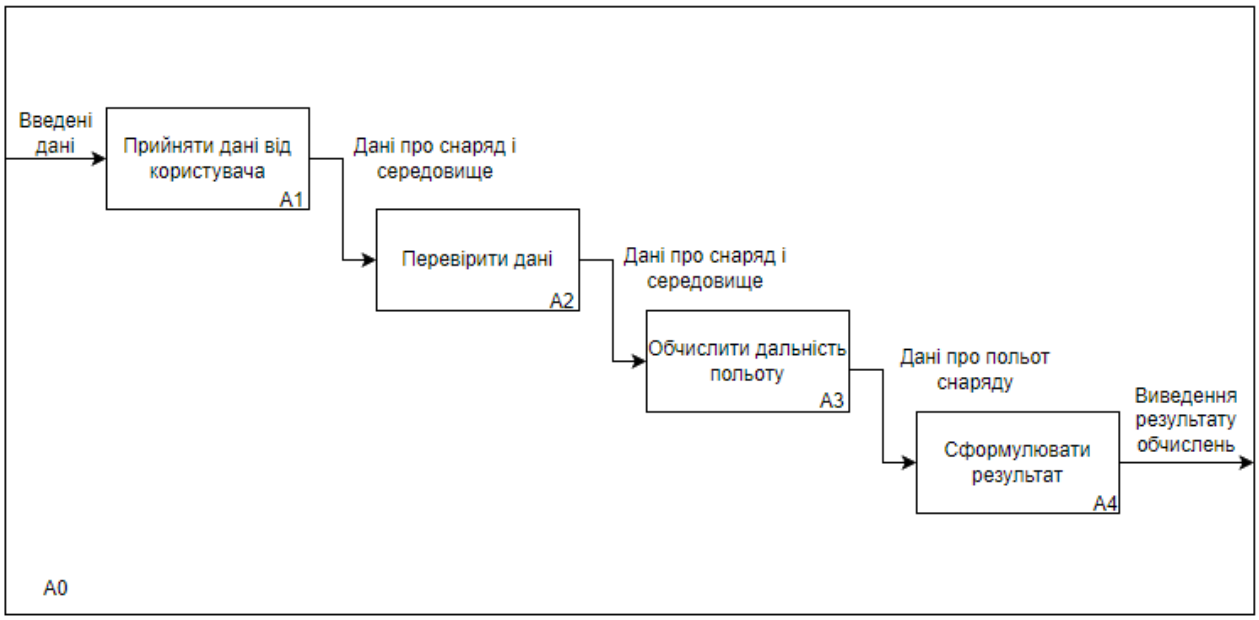


Рисунок 2.4 - Функціональна схема програми

2.3. Розробка діаграм структур даних

На рисунку 2.5 зображена структура одновимірного масиву, що містить швидкість вітру під час розрахунку дальності польоту снаряду.

На рисунку 2.6 зображена структура одновимірного масиву, що містить дальність польоту снаряду в попередніх обчисленнях.

Масив — впорядкований набір фіксованої кількості однотипних елементів, що зберігаються в послідовно розташованих комірках оперативної пам'яті, мають порядковий номер і спільне ім'я, що надає користувач.

1	2	3	4	5
734 м/с	687 м/с	620 м/с	655 м/с	785 м/с

Рисунок 2.5 - Структура одновимірного масиву, що містить початкову швидкість снаряду під час розрахунку дальності польоту снаряду

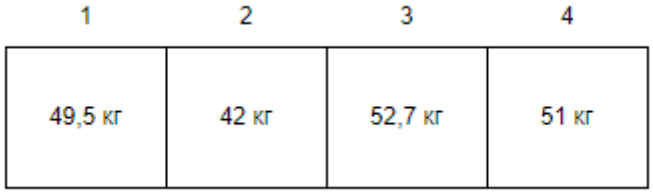


Рисунок 2.6 - Структура одновимірного масиву, що містить масу снаряду

2.4. Розробка структур файлів даних

Програма буде знаходитись в папці Projectile Flight Model і мати основний файл з назвою Form1, який буде містити клас-модель польоту снаряда. Також вона буде мати папку bin\Debug\Graphs, де будуть міститись графіки результатів тестування, і папку bin\Debug\ProjectileBase, де буде міститись початковий список снарядів. Також програма буде містити форму ProjectileBase, в якій буде знаходитись таблиця з даними про снаряди.

1. Projectile Flight Model
    - 1.1. bin/
        - 1.3.1 Debug/
            - Graphs/
                a. Graph1.png
                b. Graph2.png
            - ProjectilesBase/
                a. ProjectileList.txt
    - 1.2. Form1.cs
    - 1.3. ProjectilesBase.cs
    - 1.4. AboutBox1.cs
    - 1.5. user_instructions.docx

2.5. Розробка UML-діаграм класів

Клас Projectile містить властивості, які визначають фізичні характеристики снаряду, такі як маса, коефіцієнт опору повітря, поперечний переріз та початкова

швидкість. Також він містить властивості позиції, швидкості та прискорення, які визначають стан снаряду в даний момент часу.

1. Public поля:
    2.1 Name – назва снаряда;
    2.2 Mass – маса снаряда;
    2.3 CrossSectionalArea – площа поперечного перерізу;
    2.4 InitialVelocity – початкова швидкість.
    2.5 DragCoefficient – коефіцієнт опору повітря;
2. Public методи:
    3.1 Projectile(name: string,
    mass: float,
    crossSectionalArea: float,
    initialVelocity: float
    dragCoefficient: float) – встановити характеристики снаряда (масу, коефіцієнт опору повітря, площу поперечного перерізу, початкову швидкість), приймає числові дані;
    3.2 SaveToFile(fileName: string) – збереження даних до файлу, приймає текстові дані;
    3.3 LoadFromFile(fileName: string) – зчитування даних з файлу, приймає текстові дані;

Головний клас Form1 містить основні дані і функції проекту.

1. Public поля:
    1.1 projectiles – список об’єктів класу Projectile;
2. Public методи:
    2.1 Form1() – конструтор головного класу
    2.2 Projectile() – клас снаряда
    2.3 SaveGraph(bitmap: Bitmap,
    selectedProjectile: Projectile,
    angle: float) – збереження результуючого графіку в директорії проекту

2.4 UpdateProjectile(updatedProjectile: Projectile) – редагуваня об’єкту класа Projectile

3. Private методи:

3.1 ValidateFloatInput(input: string, result: out float) – перевірка даних на тип

3.2 CalculateProjectileFlight() – розрахунок і виведення дальності польоту снаряда і виведення результуючого графіку

Головний клас другої форми ProjectileBase містить методи для зберігання в таблиці та редагування об’єктів класу Projectile.

1. Private поля:

1.1 form1 – ініціалізація форми 1, для роботи з нею

2. Public методи:

2.1 ProjectileBase(form1: Form1) – конструктор класу ProjectileBase

2.2 LoadData(projectiles: List<Projectile>) – виведення даних в таблицю

2.3 UpdateProjectile(updatedProjectile: Projectile) – оновлення даних об’єкту Projectile

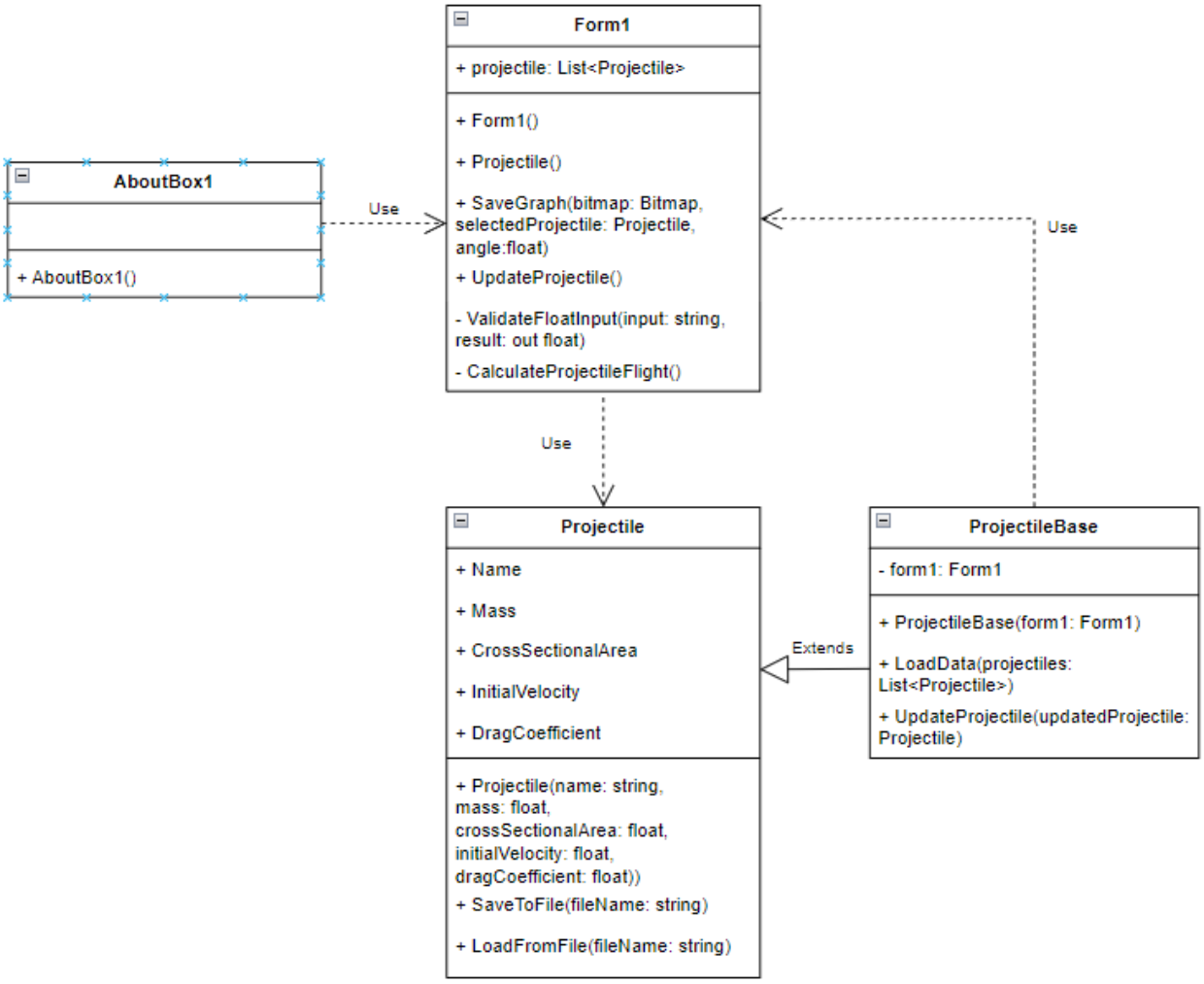
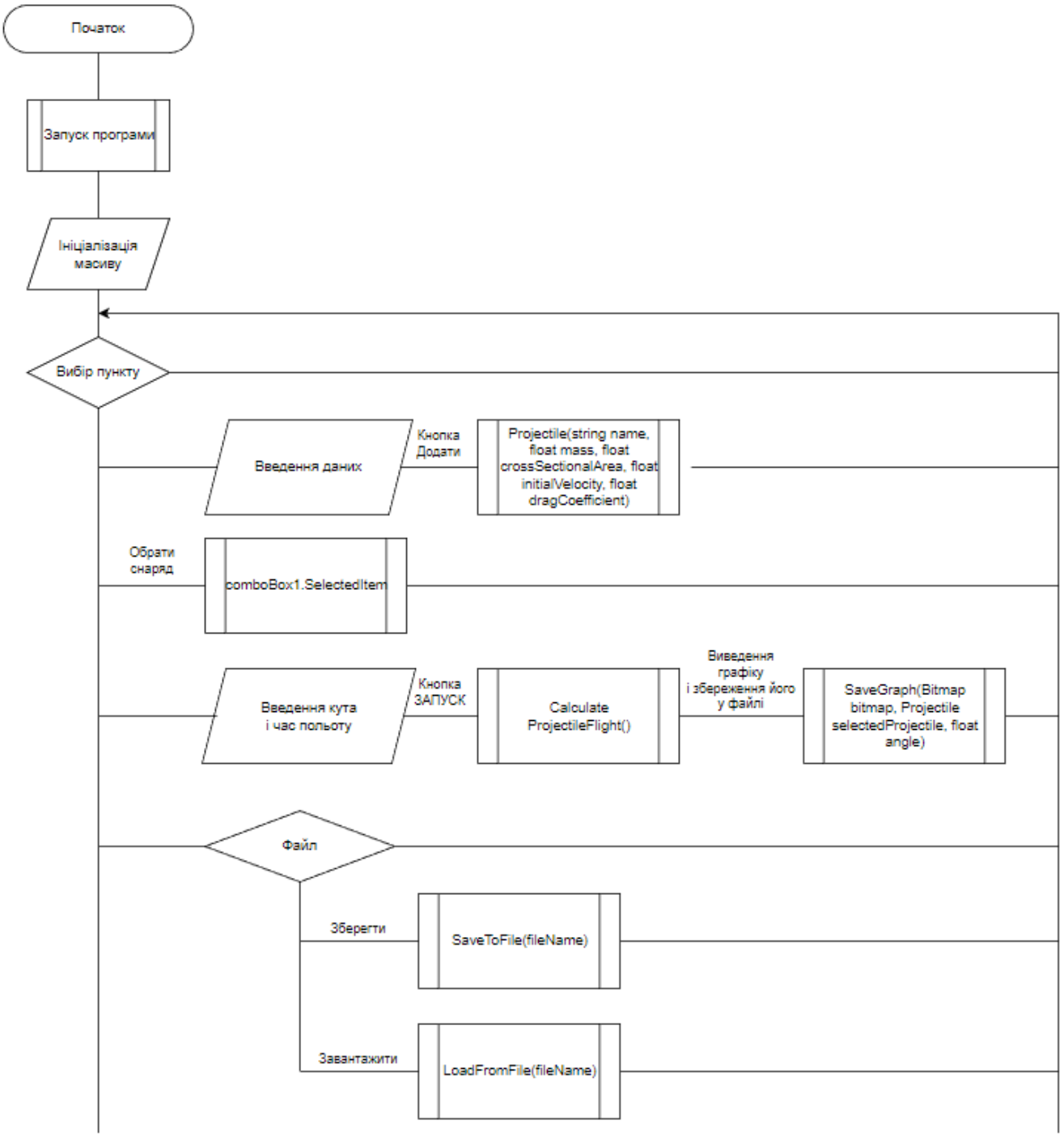


Рисунок 2.7 - UML-діаграма класів

2.6. Розробка блок-схем алгоритмів програмного комплексу

На рисунку 2.8 зображена блок-схема програми. При запуску програми відкривається меню з вибором дій. При натисканні клавіши від «1» до «6» застосовується одна з функцій програми. При натисканні іншої клавіши консоль очиститься і меню відкриється знову. При натисканні клавіши «Esc» програма завершує роботу.


Початок
Запуск програми
Ініціалізація масиву
Вибір пункту
Введення даних
Кнопка Додати
Projectile(string name, float mass, float crossSectionalArea, float initialVelocity, float dragCoefficient)
Обрати снаряд
comboBox1.SelectedItem
Введення кута і час польоту
Кнопка ЗАПУСК
Calculate ProjectileFlight()
Виведення графіку і збереження його у файлі
SaveGraph(Bitmap bitmap, Projectile selectedProjectile, float angle)
Файл
Зберегти
SaveToFile(fileName)
Завантажити
LoadFromFile(fileName)

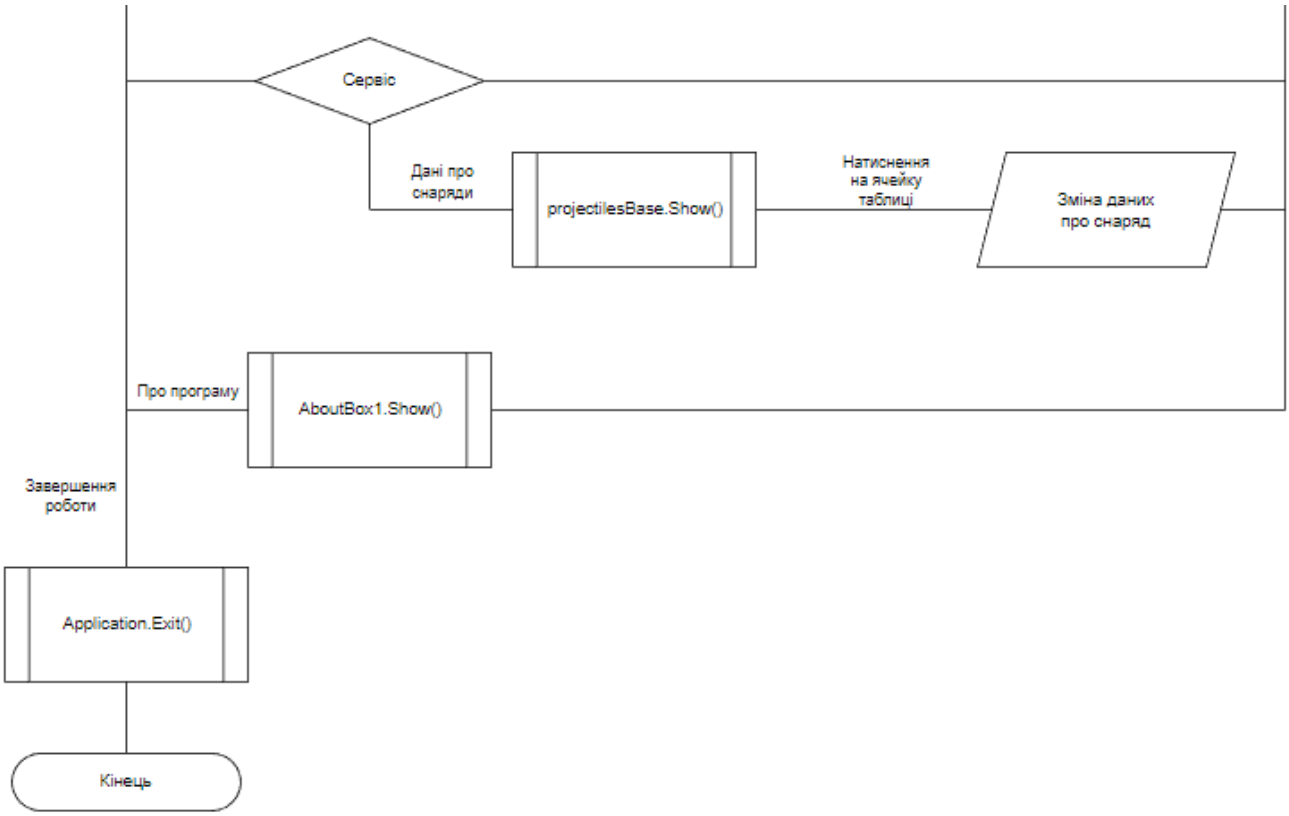


Рисунок 2.8 - Блок схема алгоритму програмного комплексу

2.7. Розробка моделі користувальницького інтерфейсу програмного комплексу

Структура меню:

1. Файл
    a. Зберегти
    b. Завантажити
2. Сервіс
    a. Дані про снаряди
3. Про програму
4. Завершення роботи
5. Дані про снаряд
    a. Назва
    b. Маса
    c. Калібр
    d. Початкова швидкість
    e. Коефіцієнт опору

f. Кнопка «Додати»

6. Снаряд
   a. Назва (Обрати снаряд)
7. Запуск снаряда
   a. Кут відхилення снаряду
   b. Час польоту
   c. Кнопка «ЗАПУСК»
8. Результат
   a. Дальність польоту

При натисненні користувачем кнопки консоль очищується і виводиться функція, відповідна нажатій кнопці. При натисненні неправильної кнопки консоль очищується і меню виводиться знову. При натисненні кнопки Esc програма завершує роботу.

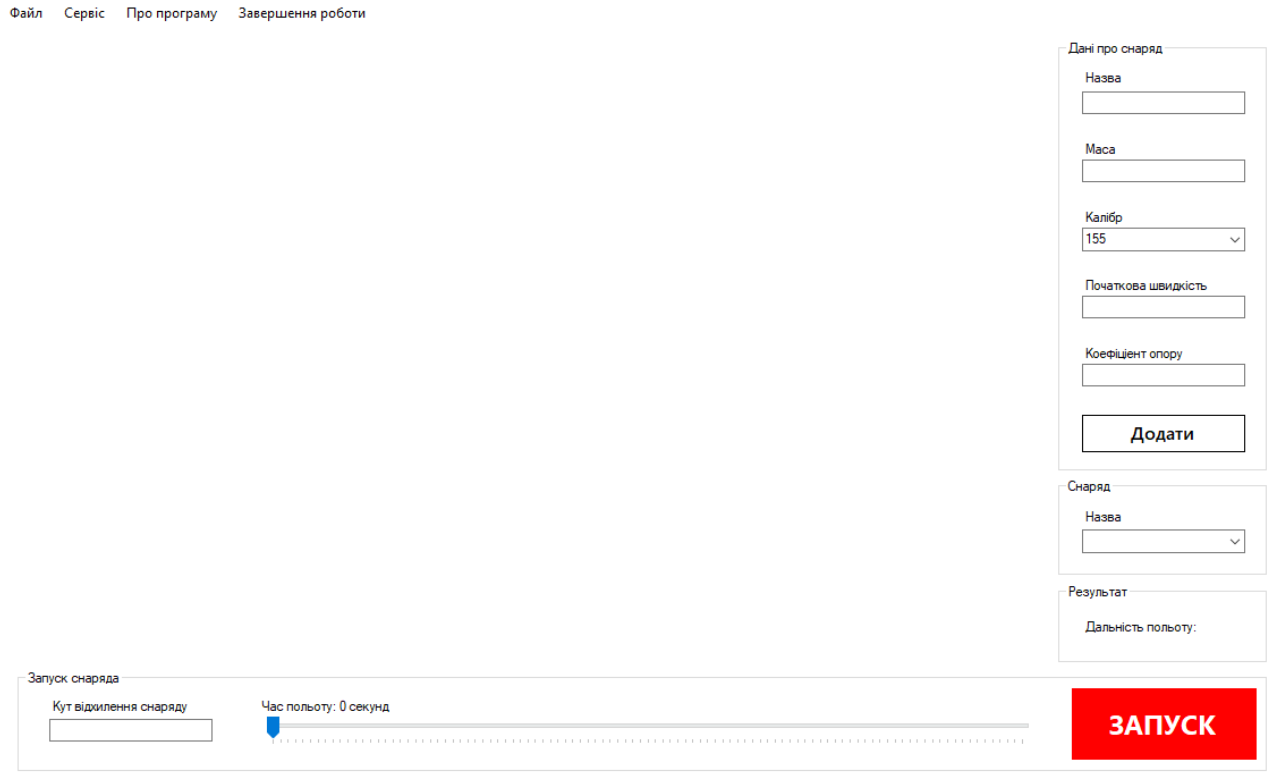


Рисунок 2.9 - Приклад меню програми

При додаванні даних про снаряд в консолі буде виводитись характеристика, яку користувач повинен вписати. Коли всі характеристики снаряда будуть вписані, консоль очиститься і меню виведеться знову.

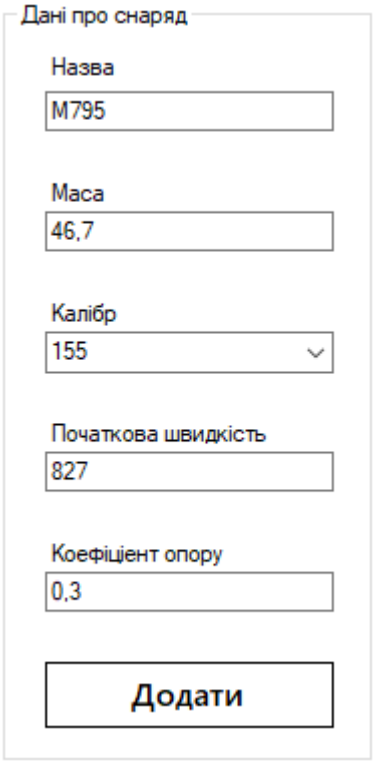
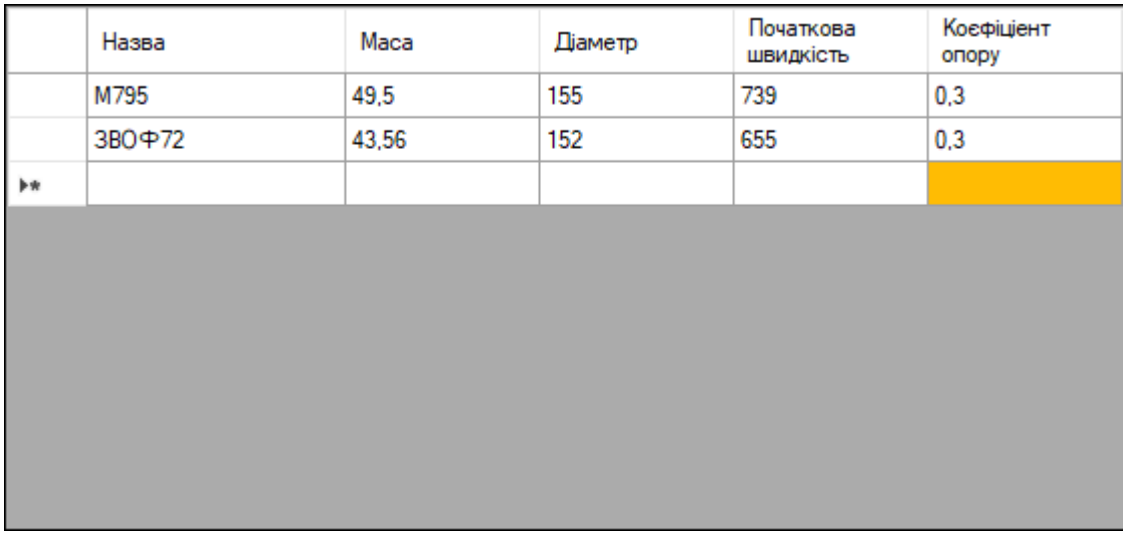


Рисунок 2.10 - Приклад додавання користувачем даних про снаряд.

Підкреслення – введений користувачем текст.

При натисненні користувачем клавіши від 2 до 5 в консолі виведеться інформація про снаряд.



	Назва	Маса	Діаметр	Початкова швидкість	Коефіцієнт опору
	M795	49,5	155	739	0,3
	ЗВОФ72	43,56	152	655	0,3
▶*					

Рисунок 2.11- Приклад виведення інформації про снаряд

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНИХ МОДУЛЕЙ ПРОЕКТА

3.1 Розробка основних класів програмних модулів

Нижче наведений опис класу, який використовується в даній програмі.

Клас Projectile містить такі публічні властивості, як Name, Mass, CrossSectionalArea, InitialVelocity та DragCoefficient, що мають вбудовані методи get, set. Для ініціалізації класу використовується конструктор Projectile(), який приймає дані та створює новий об’єкт. Також в класі реалізовані методи SaveToFile() для збереження файлу в обраній папці та LoadFromFile() для завантаження з файлу інформації про снаряди.

Лістинг 3.1 – Клас Projectile

```
public class Projectile
        {
            // Властивості снаряду
            public string Name { get; set; } // Назва снаряду
            public float Mass { get; set; } // Маса снаряду
            public float CrossSectionalArea { get; set; } // Площа розрізу снаряда
            public float InitialVelocity { get; set; } // Початкова швидкість
            public float DragCoefficient { get; set; } // Коефіцієнт опору


            // Конструктор класу
            public Projectile(string name, float mass, float crossSectionalArea,
float initialVelocity, float dragCoefficient)
            {
                Name = name;
                Mass = mass;
                CrossSectionalArea = crossSectionalArea;
                InitialVelocity = initialVelocity;
                DragCoefficient = dragCoefficient;
            }
            // Метод для збереження даних про снаряд у файл
            public void SaveToFile(string fileName)
            {
                try
                {
                    // Відкриваємо файл для запису
                    using (StreamWriter writer = new StreamWriter(fileName, true))
                    {
                        // Записуємо дані про снаряд
                        string line = $"{Name} {Mass} {CrossSectionalArea}
{InitialVelocity} {DragCoefficient}";
                        writer.WriteLine(line);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Під час збереження даних про снаряд сталася
помилка:\n" + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            // Метод для завантаження всіх снарядів з файлу
            public static List<Projectile> LoadFromFile(string fileName)
```

```
            {
                List<Projectile> projectiles = new List<Projectile>();

                try
                {
                    // Зчитуємо всі рядки з файлу
                    string[] lines = File.ReadAllLines(fileName);

                    foreach (string line in lines)
                    {
                        if (!string.IsNullOrEmpty(line))
                        {
                            // Розбиваємо рядок на компоненти
                            string[] values = line.Split(' ');

                            if (values.Length == 5)
                            {
                                // Перевіряємо типи даних та отримуємо значення
                                string name = values[0];
                                float mass, crossSectionalArea, initialVelocity,
dragCoefficient;

                                if (float.TryParse(values[1], out mass) &&
                                    float.TryParse(values[2], out
crossSectionalArea) &&
                                    float.TryParse(values[3], out initialVelocity)
&&
                                    float.TryParse(values[4], out dragCoefficient))
                                {
                                    // Створюємо об'єкт Projectile та додаємо його
до списку
                                    Projectile projectile = new Projectile(name,
mass, crossSectionalArea, initialVelocity, dragCoefficient);
                                    projectiles.Add(projectile);
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Помилка завантаження даних про снаряди: " +
ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }

                return projectiles;
            }


        }
```

Також є клас для зручного виведення даних про снаряд – ProjectilesBase. Він містить приватну властивість form1 для ініціалізації та роботи з формою Form1 та класом, який знаходиться в ній. Цей клас має такі методи, як завантаження даних про снаряди в таблицю – LoadData(), і метод UpdateProjectile() для оновлення даних про снаряд при зміні їх в ячейці таблиці.

Лістинг 3.2 – Клас ProjectilesBase

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static WindowsFormsApp1.Form1;

namespace WindowsFormsApp1
{
    public partial class ProjectilesBase : Form
    {
        private Form1 form1;
        public ProjectilesBase(Form1 form1)
        {
            InitializeComponent();

            this.form1 = form1;
        }

        public void LoadData(List<Projectile> projectiles)
        {
            foreach (Projectile projectile in projectiles)
            {
                dataGridView1.Rows.Add(projectile.Name, projectile.Mass,
projectile.CrossSectionalArea, projectile.InitialVelocity,
projectile.DragCoefficient);
            }
        }
        public void UpdateProjectile(Projectile updatedProjectile)
        {
            Form1 form1 = (Form1)Application.OpenForms["Form1"]; // Отримати
посилання на екземпляр Form1
            form1.UpdateProjectile(updatedProjectile);
        }

        private void dataGridView1_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
        {
            int rowIndex = e.RowIndex;
            DataGridViewRow row = dataGridView1.Rows[rowIndex];

            string projectileName = row.Cells[0].Value.ToString();
            Projectile modifiedProjectile = form1.projectiles.FirstOrDefault(p =>
p.Name == projectileName);

            if (modifiedProjectile != null)
            {
                // Оновіть поля об'єкта Projectile
                // на основі змінених значень у таблиці

                modifiedProjectile.Mass = Convert.ToSingle(row.Cells[1].Value);
                modifiedProjectile.CrossSectionalArea =
Convert.ToSingle(row.Cells[2].Value);
                modifiedProjectile.InitialVelocity =
Convert.ToSingle(row.Cells[3].Value);
                modifiedProjectile.DragCoefficient =
Convert.ToSingle(row.Cells[4].Value);
```

```
                UpdateProjectile(modifiedProjectile); // Виклик методу оновлення
об'єкта Projectile у Form1
            }
        }
    }
}
```

3.2 Розробка користувацького інтерфейсу програмного комплексу

Для створення інтерфейсу було вирішено використати Windows Forms. Всі функції інтерфейсу було реалізовано в класі Form1. Form1 – це головний клас програми, який запускається одразу при запуску програми.

Цей клас містить список List<Projectiles> projectiles для оголошення списку об’єктів класу Projectile. В цьому класі є конструктор Form1(), який містить ініціалізацію списку projectiles та метод InitilizeComponent() для ініціалізації компонентів програми та підтримки користувацького інтерфейсу. Також в цьому класі міститься клас Projectile, про який було зазначено вище. В цьому класі є багато приватних методів для працездатності елементів інтерфейсу:

- buttonAdd_Click() – метод для збереження даних про снаряд при натисканні на кнопку «Додати»;
- зберегтиToolStripMenuItem_Click() – метод для збереження даних у файл при натисканні на елемент меню «Зберегти»;
- завантажитиToolStripMenuItem_Click() – метод для зчитування даних з файлу при натисканні на елемент меню «Завантажити»;
- trackBar1_ValueChanged() – метод для вибору часу польоту снаряда при прокрутці елемента trackBar;
- buttonLaunch_Click() – метод для розрахунку та виведення дальності польоту снаряда та виведення результуючого графіку;
- даніПроСнарядиToolStripMenuItem_Click() – метод для виведення форми, в якій міститься таблиця з даними про снаряди при натисканні на елемент меню «Дані про снаряди»;
- проПрограмуToolStripMenuItem_Click() – метод для виведення форми «Про програму» при натисканні на елемент меню «Про програму»;

- завершенняРоботиToolStripMenuItem_Click() – метод для завершення роботи програми при натисканні на елемент «Завершення роботи».

Також в програмі міститься метод ValidateFloatInput() для перевірки даних на тип в текстових полях. Метод CalculateProjectileFlight() – це майже основний метод в програмі, який використовуючи формули розраховує дальність польоту снаряду, а також малює графік і виводить його в елементі pictureBox. Метод SaveGraph() зберігає графік в директорії проекту в папці Graphs з назвою снаряда та кутом відхилення снаряду.

Лістинг 3.3 – Головний клас програми Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using TrackBar = System.Windows.Forms.TrackBar;
using static WindowsFormsApp1.ProjectilesBase;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public List<Projectile> projectiles; // Список об'єктів Projectile

        public Form1()
        {
            InitializeComponent();

            projectiles = new List<Projectile>(); // Ініціалізація списку

            // Елемент TrackBar (з ім'ям trackBarFlightTime)
            TrackBar trackBar1 = new TrackBar();
            trackBar1.Minimum = 0; // Мінімальне значення - 1 секунда
            trackBar1.Maximum = 100; // Максимальне значення - 100 секунд
            trackBar1.TickFrequency = 1; // Інтервал між позначками трекбара
            trackBar1.ValueChanged += trackBar1_ValueChanged;
            this.Controls.Add(trackBar1);

            // Додати на форму елемент Label (з ім'ям labelFlightTime)
            Label labelFlightTime = new Label();
            labelFlightTime.Text = "Flight Time: " + trackBar1.Value + " seconds";
            this.Controls.Add(labelFlightTime);

            // Додати значення в crossSectionalAreaComboBox
            crossSectionalAreaComboBox.Items.Add("155");
```

```
            crossSectionalAreaComboBox.Items.Add("152");
            crossSectionalAreaComboBox.Items.Add("122");

            // Встановити початкове значення
            crossSectionalAreaComboBox.SelectedIndex = 0;
        }

        private void buttonAdd_Click(object sender, EventArgs e)
        {
            // Перевірка текстових полів на наявність даних
            if (nameTextBox.Text == "" || massTextBox.Text == "" ||
crossSectionalAreaComboBox.SelectedItem.ToString() == "" ||
initialVelocityTextBox.Text == "" || dragCoefficientTextBox.Text == "")
            {
                MessageBox.Show("Поля з даними порожні.", "Помилка!");
                return;
            }

            // Отримуємо значення з TextBox
            string name = nameTextBox.Text;
            float mass, crossSectionalArea, initialVelocity, dragCoefficient;

            string CSA = crossSectionalAreaComboBox.SelectedItem.ToString();
            crossSectionalArea = float.Parse(CSA);

            // Перевірка типу даних і конвертація значень
            if (!ValidateFloatInput(massTextBox.Text, out mass))
            {
                MessageBox.Show("Недійсне значення для маси. Введіть дійсне число з
плаваючою комою.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (!ValidateFloatInput(initialVelocityTextBox.Text, out
initialVelocity))
            {
                MessageBox.Show("Недійсне значення для початкової швидкості. Введіть
дійсне число з плаваючою комою.", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            if (!ValidateFloatInput(dragCoefficientTextBox.Text, out
dragCoefficient))
            {
                MessageBox.Show("Недійсне значення для коефіцієнта опору. Введіть
дійсне число з плаваючою комою.", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }

            // Створюємо об'єкт Projectile зі значеннями з TextBox
            Projectile projectile = new Projectile(name, mass, crossSectionalArea,
initialVelocity, dragCoefficient);

            // Додаємо об'єкт Projectile до списку
            projectiles.Add(projectile);

            // Очищаємо TextBox
            nameTextBox.Text = string.Empty;
            massTextBox.Text = string.Empty;
            initialVelocityTextBox.Text = string.Empty;
            dragCoefficientTextBox.Text = string.Empty;

            // Додаємо назву снаряду до ComboBox
            comboBox1.Items.Add(name);
```

```
        }
        // Метод для перевірки даних на тип в текстовому полі
        private bool ValidateFloatInput(string input, out float result)
        {
            return float.TryParse(input, out result);
        }

        private void зберегтиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Відкриваємо діалогове вікно для вибору файлу для збереження
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Text Files|*.txt";
            saveFileDialog.Title = "Save Projectile Data";

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                string fileName = saveFileDialog.FileName;

                // Зберігаємо дані про всі снаряди у файл
                foreach (Projectile projectile in projectiles)
                {
                    projectile.SaveToFile(fileName);
                }
            }
            MessageBox.Show("Дані про снаряд успішно збережено.", "Збереження
даних", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }


        private void завантажитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Відкриваємо діалогове вікно для вибору файлу для завантаження
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Text Files|*.txt";
            openFileDialog.Title = "Load Projectile Data";

            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                string fileName = openFileDialog.FileName;

                // Викликаємо метод LoadAllFromFile класу Projectile для
завантаження всіх снарядів з файлу
                List<Projectile> loadedProjectiles =
Projectile.LoadFromFile(fileName);

                if (loadedProjectiles.Count > 0)
                {
                    // Очищаємо список снарядів та ComboBox
                    projectiles.Clear();
                    comboBox1.Items.Clear();

                    // Додаємо завантажені снаряди до списку та ComboBox
                    projectiles.AddRange(loadedProjectiles);
                    foreach (Projectile projectile in loadedProjectiles)
                    {
                        comboBox1.Items.Add(projectile.Name);
                    }

                    MessageBox.Show("Усі дані про снаряди успішно завантажено.",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show("У файлі не знайдено дійсних даних про
снаряди.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                }
            }
        }

        private void trackBar1_ValueChanged(object sender, EventArgs e)
        {
            // Оновлення значення мітки з часом польоту
            Label labelFlightTime = this.Controls.Find("labelFlightTime",
true).FirstOrDefault() as Label;
            TrackBar trackBar1 = sender as TrackBar;

            if (labelFlightTime != null)
            {
                labelFlightTime.Text = "Час польоту: " + trackBar1.Value + "
секунд";
            }
        }
        private void CalculateProjectileFlight()
        {
            string selectedProjectileName = comboBox1.SelectedItem.ToString();

            // Знаходження обраного снаряда за назвою
            Projectile selectedProjectile = projectiles.FirstOrDefault(p => p.Name
== selectedProjectileName);

            if (selectedProjectile != null)
            {
                float mass = selectedProjectile.Mass;
                float dragCoefficient = selectedProjectile.DragCoefficient;
                float crossSectionalArea = selectedProjectile.CrossSectionalArea;
                float initialVelocity = selectedProjectile.InitialVelocity;
                float angle = float.Parse(angleTextBox.Text);
                float flightTime = trackBar1.Value;

                if (crossSectionalArea == 155)
                {
                    crossSectionalArea = 0.01875f;
                }
                else if (crossSectionalArea == 152)
                {
                    crossSectionalArea = 0.01811f;
                }
                else if (crossSectionalArea == 122)
                {
                    crossSectionalArea = 0.01173f;
                }

                // Приклад даних про політ снаряда
                List<PointF> flightData = new List<PointF>();
                float g = 9.8f; // Прискорення вільного падіння

                for (float t = 0; t <= flightTime; t += 0.1f)
                {
                    float x = initialVelocity * (float)Math.Cos(angle * Math.PI /
180) * t; // Обчислення координати X
                    float y = initialVelocity * (float)Math.Sin(angle * Math.PI /
180) * t - 0.5f * g * t * t; // Обчислення координати Y

                    // Застосування формули для врахування опору повітря
                    float velocity = (float)Math.Sqrt(Math.Pow(initialVelocity *
Math.Cos(angle * Math.PI / 180), 2) +
                                                      Math.Pow(initialVelocity *
Math.Sin(angle * Math.PI / 180) - g * t, 2));
```

```
                    float dragForce = 0.5f * dragCoefficient * crossSectionalArea *
velocity * velocity;
                    float acceleration = -dragForce / mass;
                    y += 0.5f * acceleration * t * t;

                    flightData.Add(new PointF(x, y));
                }


                // Отримання максимальних значень координат
                float maxX = flightData.Max(point => point.X);
                float maxY = flightData.Max(point => point.Y);

                // Створення нового графіку
                Bitmap bitmap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
                using (Graphics graphics = Graphics.FromImage(bitmap))
                {
                    // Очищення графіку
                    graphics.Clear(Color.White);

                    // Налаштування розмітки
                    Pen axisPen = new Pen(Color.Black);
                    Font axisFont = new Font("Arial", 10);
                    Brush axisBrush = new SolidBrush(Color.Black);

                    // Позначення осей координат
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, 20, 20); //
Вертикальна вісь X
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, bitmap.Width
- 20, bitmap.Height - 20); // Горизонтальна вісь Y

                    // Розмітка вертикальної вісі (X)
                    float stepX = maxX / 10;
                    for (float i = 0; i <= maxX; i += stepX)
                    {
                        int x = (int)(i + 20);
                        graphics.DrawLine(axisPen, x, bitmap.Height - 20, x,
bitmap.Height - 15); // Мітка на вісі X
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush, x
* bitmap.Width / maxX - 10, bitmap.Height - 15); // Підпис під міткою
                    }

                    // Розмітка горизонтальної вісі (Y)
                    float stepY = maxY / 50;
                    for (float i = 0; i <= maxY; i += stepY)
                    {
                        int y = (int)(bitmap.Height - 20 - i);
                        graphics.DrawLine(axisPen, 15, y, 20, y); // Мітка на вісі Y
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush,
0, y - 7); // Підпис біля мітки
                    }

                    // Налаштування лінії графіка
                    Pen linePen = new Pen(Color.Red);

                    // Малювання лінії графіка
                    for (int i = 1; i < flightData.Count; i++)
                    {
                        PointF startPoint = flightData[i - 1];
                        PointF endPoint = flightData[i];
                        graphics.DrawLine(linePen, startPoint.X * bitmap.Width /
maxX + 20, bitmap.Height - startPoint.Y * bitmap.Height / maxY - 20,
                            endPoint.X * bitmap.Width / maxX + 20, bitmap.Height - endPoint.Y
* bitmap.Height / maxY - 20);
```

```
                    }
                }

                // Визначення дальності польоту
                PointF arrivePoint = PointF.Empty;
                float range = 0;
                for (int i = 1; i < flightData.Count; i++)
                {
                    PointF previousPoint = flightData[i - 1];
                    PointF currentPoint = flightData[i];

                    if (previousPoint.Y >= 0 && currentPoint.Y <= 0)
                    {
                        float пропорція = (0 - previousPoint.Y) / (currentPoint.Y -
previousPoint.Y);
                        range = previousPoint.X + пропорція * (currentPoint.X -
previousPoint.X);
                        arrivePoint = new PointF(range, 0);
                        break;
                    }
                }
                labelRange.Text = $"Дальність: {range} м";

                // Збереження графіку в директорії проекта
                SaveGraph(bitmap, selectedProjectile, angle);

                // Відображення графіку
                pictureBox1.Image = bitmap;
            }
        }
        public void SaveGraph(Bitmap bitmap, Projectile selectedProjectile, float
angle)
        {
            // Генерування унікального імені для зображення
            string imageName = $"{selectedProjectile.Name}_{angle}degrees.png";

            // Отримання шляху до папки "Графіки" у вашому проекті
            string folderPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Graphs");

            // Переконайтеся, що папка існує, інакше створіть її
            if (!Directory.Exists(folderPath))
            {
                Directory.CreateDirectory(folderPath);
            }

            // Збереження зображення у папку "Графіки" з унікальним іменем
            string imagePath = Path.Combine(folderPath, imageName);
            bitmap.Save(imagePath);
        }

        public void UpdateProjectile(Projectile updatedProjectile)
        {
            int index = projectiles.FindIndex(p => p.Name ==
updatedProjectile.Name);
            if (index >= 0)
            {
                projectiles[index] = updatedProjectile;
            }
        }

        private void buttonLaunch_Click(object sender, EventArgs e)
        {
            if (comboBox1.SelectedItem == null)
```

```
            {
                MessageBox.Show("Оберіт снаряд зі списку.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (angleTextBox.Text == "")
            {
                MessageBox.Show("Введіть кут відхилення снаряду.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            CalculateProjectileFlight();
        }

        private void даніПроСнарядиToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            ProjectilesBase projectilesBase = new ProjectilesBase(this);
            projectilesBase.LoadData(projectiles); // Передача списку projectiles до
Form2
            projectilesBase.Show();
        }
        private void проПрограмуToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutBox1 aboutBox = new AboutBox1();
            aboutBox.Show();
        }

        private void завершенняРоботиToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

3.3 Розробка модуля введення-виведення даних

Введення даних реалізовано за допомогою елементів textBox. Користувач вводить дані в textBox і при натисненні кнопки вони додаються до об’єкту. Змінювати дані можна за допомогою редагування таблиці в формі ProjectilesBase. Для цього використовується метод UpdateProjectile і подія CellEndEdit елемента dataGridView. Також дані можна ввести за допомогою зчитування інформації з файлу. Для цього використовується метод LoadFromFile класу Projectile.

Лістинг 3.4 – Метод для редагування даних в таблиці

```
        public void UpdateProjectile(Projectile updatedProjectile)
        {
            Form1 form1 = (Form1)Application.OpenForms["Form1"]; // Отримати
посилання на екземпляр Form1
```

```
            form1.UpdateProjectile(updatedProjectile);
        }

        private void dataGridView1_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
        {
            int rowIndex = e.RowIndex;
            DataGridViewRow row = dataGridView1.Rows[rowIndex];

            string projectileName = row.Cells[0].Value.ToString();
            Projectile modifiedProjectile = form1.projectiles.FirstOrDefault(p =>
p.Name == projectileName);

            if (modifiedProjectile != null)
            {
                // Оновлення поля об'єкта Projectile
                // на основі змінених значень у таблиці

                modifiedProjectile.Mass = Convert.ToSingle(row.Cells[1].Value);
                modifiedProjectile.CrossSectionalArea =
Convert.ToSingle(row.Cells[2].Value);
                modifiedProjectile.InitialVelocity =
Convert.ToSingle(row.Cells[3].Value);
                modifiedProjectile.DragCoefficient =
Convert.ToSingle(row.Cells[4].Value);

                UpdateProjectile(modifiedProjectile); // Виклик методу оновлення
об'єкта Projectile у Form1
            }
        }
```

Лістинг 3.5 – Метод для збереження даних у файлі

```
            // Метод для завантаження всіх снарядів з файлу
            public static List<Projectile> LoadFromFile(string fileName)
            {
                List<Projectile> projectiles = new List<Projectile>();

                try
                {
                    // Зчитуємо всі рядки з файлу
                    string[] lines = File.ReadAllLines(fileName);

                    foreach (string line in lines)
                    {
                        if (!string.IsNullOrEmpty(line))
                        {
                            // Розбиваємо рядок на компоненти
                            string[] values = line.Split(' ');

                            if (values.Length == 5)
                            {
                                // Перевіряємо типи даних та отримуємо значення
                                string name = values[0];
                                float mass, crossSectionalArea, initialVelocity,
dragCoefficient;

                                if (float.TryParse(values[1], out mass) &&
                                    float.TryParse(values[2], out
crossSectionalArea) &&
                                    float.TryParse(values[3], out initialVelocity)
&&
```

```
                                    float.TryParse(values[4], out dragCoefficient))
                                {
                                    // Створюємо об'єкт Projectile та додаємо його
до списку
                                    Projectile projectile = new Projectile(name,
mass, crossSectionalArea, initialVelocity, dragCoefficient);
                                    projectiles.Add(projectile);
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Помилка завантаження даних про снаряди: " +
ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }

                return projectiles;
            }
```

Для виведення даних використовується елемент dataGridView, за допомогою якого дані виводяться у зручному форматі в вигляді таблиці. Для виведення результату польоту снаряду використовується елемент label для виведення дальності польоту, та елемент pictureBox для виведення графіка.

Лістинг 3.6 – Метод для виведення даних в таблиці

```
        public void LoadData(List<Projectile> projectiles)
        {
            foreach (Projectile projectile in projectiles)
            {
                dataGridView1.Rows.Add(projectile.Name, projectile.Mass,
projectile.CrossSectionalArea, projectile.InitialVelocity,
projectile.DragCoefficient);
            }
        }
```

Лістинг 3.7 – Метод для виведення дальності польоту та результучого графіку

```
        private void CalculateProjectileFlight()
        {
            string selectedProjectileName = comboBox1.SelectedItem.ToString();

            // Знаходження обраного снаряда за назвою
            Projectile selectedProjectile = projectiles.FirstOrDefault(p => p.Name
== selectedProjectileName);

            if (selectedProjectile != null)
            {
                float mass = selectedProjectile.Mass;
                float dragCoefficient = selectedProjectile.DragCoefficient;
                float crossSectionalArea = selectedProjectile.CrossSectionalArea;
                float initialVelocity = selectedProjectile.InitialVelocity;
                float angle = float.Parse(angleTextBox.Text);
```

```
                float flightTime = trackBar1.Value;

                if (crossSectionalArea == 155)
                {
                    crossSectionalArea = 0.01875f;
                }
                else if (crossSectionalArea == 152)
                {
                    crossSectionalArea = 0.01811f;
                }
                else if (crossSectionalArea == 122)
                {
                    crossSectionalArea = 0.01173f;
                }

                // Приклад даних про політ снаряда
                List<PointF> flightData = new List<PointF>();
                float g = 9.8f; // Прискорення вільного падіння

                for (float t = 0; t <= flightTime; t += 0.1f)
                {
                    float x = initialVelocity * (float)Math.Cos(angle * Math.PI /
180) * t; // Обчислення координати X
                    float y = initialVelocity * (float)Math.Sin(angle * Math.PI /
180) * t - 0.5f * g * t * t; // Обчислення координати Y

                    // Застосування формули для врахування опору повітря
                    float velocity = (float)Math.Sqrt(Math.Pow(initialVelocity *
Math.Cos(angle * Math.PI / 180), 2) +
                                                      Math.Pow(initialVelocity *
Math.Sin(angle * Math.PI / 180) - g * t, 2));
                    float dragForce = 0.5f * dragCoefficient * crossSectionalArea *
velocity * velocity;
                    float acceleration = -dragForce / mass;
                    y += 0.5f * acceleration * t * t;

                    flightData.Add(new PointF(x, y));
                }

                // Отримання максимальних значень координат
                float maxX = flightData.Max(point => point.X);
                float maxY = flightData.Max(point => point.Y);

                // Створення нового графіку
                Bitmap bitmap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
                using (Graphics graphics = Graphics.FromImage(bitmap))
                {
                    // Очищення графіку
                    graphics.Clear(Color.White);

                    // Налаштування розмітки
                    Pen axisPen = new Pen(Color.Black);
                    Font axisFont = new Font("Arial", 10);
                    Brush axisBrush = new SolidBrush(Color.Black);

                    // Позначення осей координат
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, 20, 20); //
Вертикальна вісь X
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, bitmap.Width
- 20, bitmap.Height - 20); // Горизонтальна вісь Y

                    // Розмітка вертикальної вісі (X)
                    float stepX = maxX / 10;
                    for (float i = 0; i <= maxX; i += stepX)
```

```
                    {
                        int x = (int)(i + 20);
                        graphics.DrawLine(axisPen, x, bitmap.Height - 20, x,
bitmap.Height - 15); // Мітка на вісі X
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush, x
* bitmap.Width / maxX - 10, bitmap.Height - 15); // Підпис під міткою
                    }

                    // Розмітка горизонтальної вісі (Y)
                    float stepY = maxY / 50;
                    for (float i = 0; i <= maxY; i += stepY)
                    {
                        int y = (int)(bitmap.Height - 20 - i);
                        graphics.DrawLine(axisPen, 15, y, 20, y); // Мітка на вісі Y
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush,
0, y - 7); // Підпис біля мітки
                    }

                    // Налаштування лінії графіка
                    Pen linePen = new Pen(Color.Red);

                    // Малювання лінії графіка
                    for (int i = 1; i < flightData.Count; i++)
                    {
                        PointF startPoint = flightData[i - 1];
                        PointF endPoint = flightData[i];
                        graphics.DrawLine(linePen, startPoint.X * bitmap.Width /
maxX + 20, bitmap.Height - startPoint.Y * bitmap.Height / maxY - 20,
                    endPoint.X * bitmap.Width / maxX + 20, bitmap.Height - endPoint.Y
* bitmap.Height / maxY - 20);

                    }
                }

                // Визначення дальності польоту
                PointF arrivePoint = PointF.Empty;
                float range = 0;
                for (int i = 1; i < flightData.Count; i++)
                {
                    PointF previousPoint = flightData[i - 1];
                    PointF currentPoint = flightData[i];

                    if (previousPoint.Y >= 0 && currentPoint.Y <= 0)
                    {
                        float пропорція = (0 - previousPoint.Y) / (currentPoint.Y -
previousPoint.Y);
                        range = previousPoint.X + пропорція * (currentPoint.X -
previousPoint.X);
                        arrivePoint = new PointF(range, 0);
                        break;
                    }
                }
                labelRange.Text = $"Дальність: {range} м";

                // Збереження графіку в директорії проекта
                SaveGraph(bitmap, selectedProjectile, angle);

                // Відображення графіку
                pictureBox1.Image = bitmap;
            }
        }
```

3.4 Розробка модуля публікації даних проекту (звітів)

Після введення даних про снаряди їх можна зберегти в файл за допомогою функції SaveToFile(). У разі помилки збереження файлу користувачу виведеться відповідне повідомлення.

Також за формулами малюється графік і зберігається в директорії проекту в папці Graphs за допомогою методу SaveGraph().

Лістинг 3.8 – Метод для збереження графіка в папці

```
        public void SaveGraph(Bitmap bitmap, Projectile selectedProjectile, float
angle)
        {
            // Генерування унікального імені для зображення
            string imageName = $"{selectedProjectile.Name}_{angle}degrees.png";

            // Отримання шляху до папки "Графіки" у вашому проекті
            string folderPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Graphs");

            // Переконайтеся, що папка існує, інакше створіть її
            if (!Directory.Exists(folderPath))
            {
                Directory.CreateDirectory(folderPath);
            }

            // Збереження зображення у папку "Графіки" з унікальним іменем
            string imagePath = Path.Combine(folderPath, imageName);
            bitmap.Save(imagePath);
        }
```

Лістинг 3.9 – Метод для збереження даних про снаряди в файлі

```
            // Метод для збереження даних про снаряд у файл
            public void SaveToFile(string fileName)
            {
                try
                {
                    // Відкриваємо файл для запису
                    using (StreamWriter writer = new StreamWriter(fileName, true))
                    {
                        // Записуємо дані про снаряд
                        string line = $"{Name} {Mass} {CrossSectionalArea}
{InitialVelocity} {DragCoefficient}";
                        writer.WriteLine(line);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Під час збереження даних про снаряд сталася
помилка:\n" + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
```

3.5 Розробка інструкцій для користувача.

Посібник для користування програмою записаний в файлі user_instructions.txt в кореневій папці програми. Посібник складається з коротких інструкцій та скріншотів щодо користування окремих пунктів меню та кнопок програми.

Початок роботи

Програма створена для розрахунку дальності польоту снаряда з різними характеристиками та за різних умов.

При запуску програми відкривається просте меню з усім функціоналом системи. Користувачу надається змога ввести дані про снаряд (маса в кілограмах, початкова швидкість в м/с, стандартний коефіцієнт опору – 0,3), або завантажити текстовий документ із вже готовим списком снарядів.

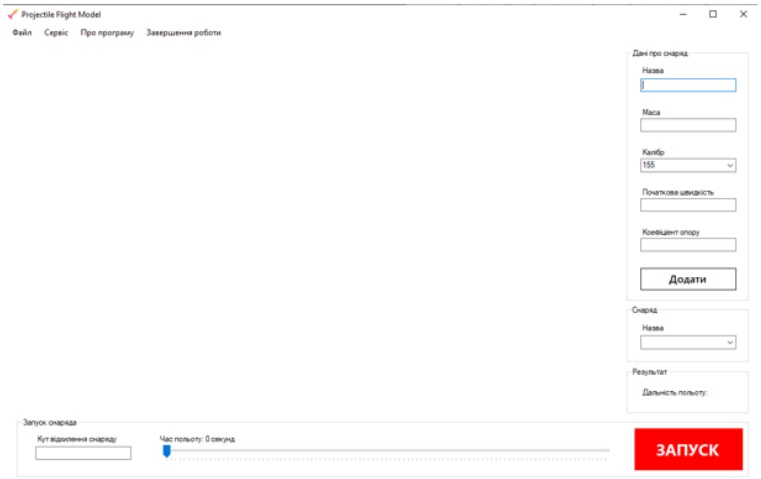


Рисунок 1 - Головне меню програми

Рисунок 3.1 - Інструкція користувача. Початок роботи

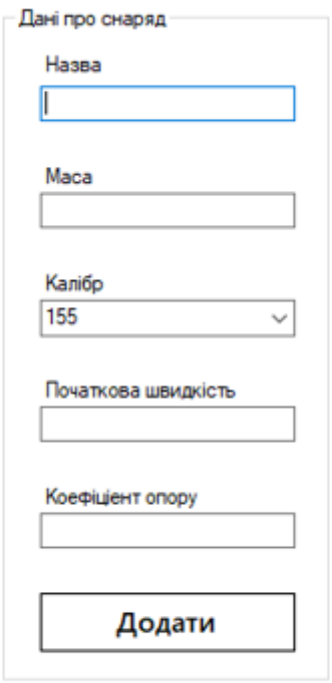


Рисунок 2 - Меню введення інформації про снаряд

Рисунок 3.2 - Інструкція користувача. Введення даних про снаряд

В полі «Снаряд» надається вибір наявних в системі снарядів. Щоб загрузити вже готовий список натисніть в пункті меню «Файл» кнопку «Завантажити» і виберіть файл ProjectileList.txt в папці ProjectileBase в директорії проекту.

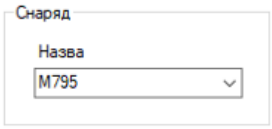


Рисунок 3 - Список наявних в системі снарядів

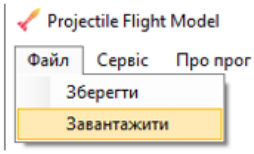


Рисунок 4 - Пункт меню "Завантажити"

Рисунок 5 - Шлях до файлу зі списком снарядів

Рисунок 3.3 - Інструкція користувача. Завантаження готових даних про снаряди

В полі «Запуск снаряда» користувач вводить кут відхилення снаряду (нормальний кут відхилення – від 30 до 60 градусів) та обрати час польоту снаряда в секундах за допомогою повзунка.

Рисунок 6 - Меню "Запуск снаряда"

Рисунок 3.4 - Інструкція користувача. Меню "Запуск снаряда"

При натисненні кнопки «ЗАПУСК» малюється графік польоту снаряда, де вісь Y – висота польоту, а вісь X – дальність польоту. Також в полі результат виводиться дальність польоту снаряда в метрах.

Графіки з результатами польоту снарядів можна знайти в папці Graphs в директорії програми.

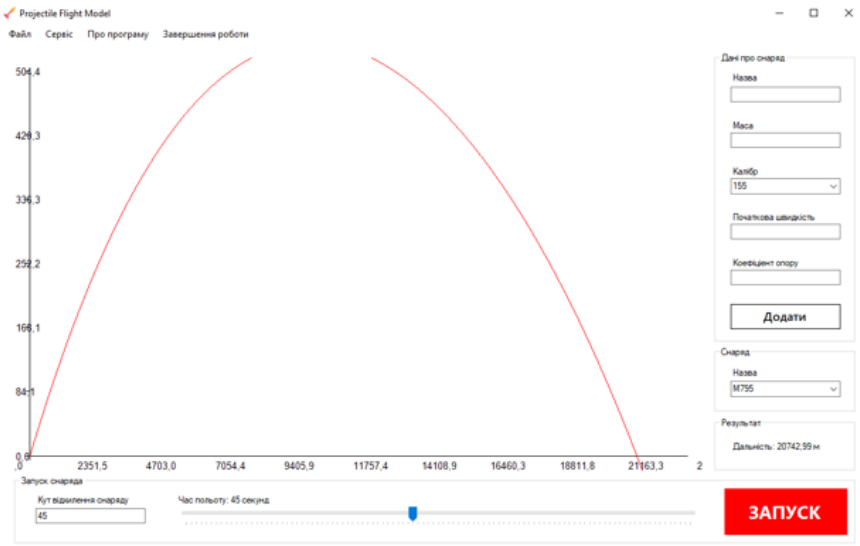


Рисунок 7 - Результат розрахунку польоту снаряда

Рисунок 3.5 - Інструкція користувача. Запуск снаряда

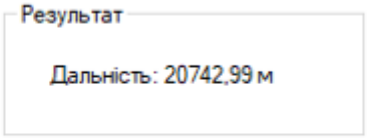


Рисунок 8 – Поле з дальністю польоту

Рисунок 3.6 - Інструкція користувача. Дальність польоту

Для редагування характеристики снарядів в пункті меню «Сервіс» треба натиснути на кнопку «Дані про снаряди». Користувачу виведеться таблиця з інформацією про наявні в системі снаряди. При натисненні на клітинку в таблиці користувач може змінити її значення.

УВАГА! Значення назви снаряду змінити не можна!

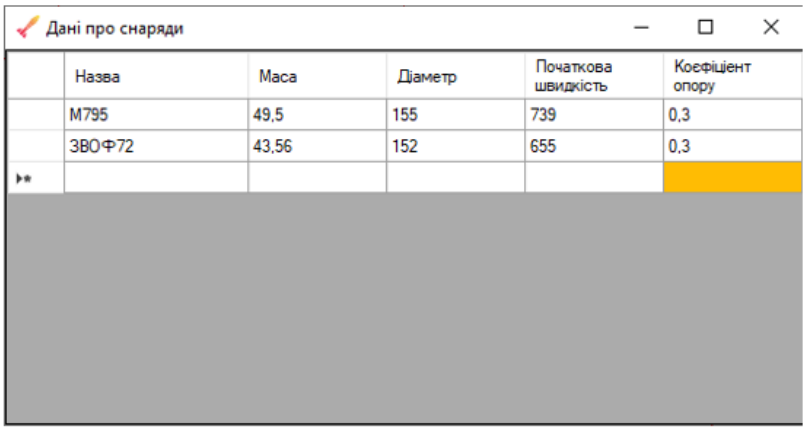


Рисунок 9 - Таблиця з даними про снаряди

Рисунок 3.7 - Інструкція користувача. Таблиця з даними

Щоб зберегти в файл інформацію про снаряди треба зайти в пункт меню «Файл» і натиснути кнопку «Зберегти». Користувач повинен буде обрати папку і дати назву файлу, щоб зберегти його.

При натисненні на пункт «Про програму» виведеться коротка інформація про програму.

При натисненні на пункт «Завершення роботи» програма завершить роботу.

Рисунок 3.8 - Інструкція користувача. Останні пункти

ВИСНОВОК

В даній курсовій роботі була розроблена клас-модель польоту снаряду, яка наразі є дуже актуальною. Вона має важливе значення для різних галузей, таких як військова техніка та авіація. Вона допомогає дослідити і розрахувати точність та ефективність снарядів, враховуючи різні аеродинамічні характеристики та вплив навколишнього середовища, та забезпечити оптимальне використання їх потенціалу.

Подальші дослідження в галузі розробки класу-моделі польоту снаряду можуть включати розширені моделі для врахування додаткових факторів, вдосконалення чисельних методів розв’язання диференційних рівнянь, а також валідацію моделі на реальних експериментах.

У цілому, розробка класу-моделі польоту снаряду є важливим кроком у напрямку покращення точності та ефективності снарядів у різних галузях техніки та промисловості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зовнішня балістика наземної техніки War Thunder. [Електронний ресурс] / Режим доступу: https://forum.warthunder.com/index.php?/topic/397647-unrealistic-external-ballistics-for-ground-vehicles/
2. Балістика в грі PUBG. [Електронний ресурс] / Режим доступу: https://pubg.ru/news/17169-kak-vedut-sebya-kar98k-sks-s-pricelom-8x-v-pubg-1-0
3. Інформація про програму Projectile Motion від PhET. [Електронний ресурс] / Режим доступу: https://phet.colorado.edu/en/simulations/projectile-motion/about
4. Програма Projectile Motion від PhET. [Електронний ресурс] / Режим доступу: https://phet.colorado.edu/sims/html/projectile-motion/latest/projectile-motion_en.html
5. Технічні характеристики гаубиці FH70. [Електронний ресурс] / Режим доступу: https://tsn.ua/ru/ato/155-mm-samohodnye-gaubicy-fh70-osobennosti-harakteristiki-artilleriyskoy-sistemy-s-dvigatelem-2074084.html
6. Снаряд M795. [Електронний ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/M795
7. Калібр 155 мм. [Електронний ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BB%D1%96%D0%B1%D1%80_155_%D0%BC%D0%BC
8. Аєродинамічний опір. [Електронний ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/%D0%90%D0%B5%D1%80%D0%BE%D0%B3%D1%96%D0%B4%D1%80%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%BE%D0%BF%D1%96%D1%80
9. Масив. [Електронний ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%B8

%D0%B2_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)

ДОДАТКИ

Додаток 1 – Головний клас програми Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using TrackBar = System.Windows.Forms.TrackBar;
using static WindowsFormsApp1.ProjectilesBase;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public List<Projectile> projectiles; // Список об'єктів Projectile

        public Form1()
        {
            InitializeComponent();

            projectiles = new List<Projectile>(); // Ініціалізація списку

            // Елемент TrackBar (з ім'ям trackBarFlightTime)
            TrackBar trackBar1 = new TrackBar();
            trackBar1.Minimum = 0; // Мінімальне значення - 1 секунда
            trackBar1.Maximum = 100; // Максимальне значення - 100 секунд
            trackBar1.TickFrequency = 1; // Інтервал між позначками трекбара
            trackBar1.ValueChanged += trackBar1_ValueChanged;
            this.Controls.Add(trackBar1);

            // Додати на форму елемент Label (з ім'ям labelFlightTime)
            Label labelFlightTime = new Label();
            labelFlightTime.Text = "Flight Time: " + trackBar1.Value + " seconds";
            this.Controls.Add(labelFlightTime);

            // Додати значення в crossSectionalAreaComboBox
            crossSectionalAreaComboBox.Items.Add("155");
            crossSectionalAreaComboBox.Items.Add("152");
            crossSectionalAreaComboBox.Items.Add("122");

            // Встановити початкове значення
            crossSectionalAreaComboBox.SelectedIndex = 0;
        }

        public class Projectile
        {
            // Властивості снаряду
            public string Name { get; set; } // Назва снаряду
            public float Mass { get; set; } // Маса снаряду
            public float CrossSectionalArea { get; set; } // Площа розрізу снаряда
            public float InitialVelocity { get; set; } // Початкова швидкість
            public float DragCoefficient { get; set; } // Коефіцієнт опору


            // Конструктор класу
```

```
            public Projectile(string name, float mass, float crossSectionalArea, float
initialVelocity, float dragCoefficient)
            {
                Name = name;
                Mass = mass;
                CrossSectionalArea = crossSectionalArea;
                InitialVelocity = initialVelocity;
                DragCoefficient = dragCoefficient;
            }
            // Метод для збереження даних про снаряд у файл
            public void SaveToFile(string fileName)
            {
                try
                {
                    // Відкриваємо файл для запису
                    using (StreamWriter writer = new StreamWriter(fileName, true))
                    {
                        // Записуємо дані про снаряд
                        string line = $"{Name} {Mass} {CrossSectionalArea}
{InitialVelocity} {DragCoefficient}";
                        writer.WriteLine(line);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Під час збереження даних про снаряд сталася
помилка:\n" + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            // Метод для завантаження всіх снарядів з файлу
            public static List<Projectile> LoadFromFile(string fileName)
            {
                List<Projectile> projectiles = new List<Projectile>();

                try
                {
                    // Зчитуємо всі рядки з файлу
                    string[] lines = File.ReadAllLines(fileName);

                    foreach (string line in lines)
                    {
                        if (!string.IsNullOrEmpty(line))
                        {
                            // Розбиваємо рядок на компоненти
                            string[] values = line.Split(' ');

                            if (values.Length == 5)
                            {
                                // Перевіряємо типи даних та отримуємо значення
                                string name = values[0];
                                float mass, crossSectionalArea, initialVelocity,
dragCoefficient;

                                if (float.TryParse(values[1], out mass) &&
                                    float.TryParse(values[2], out crossSectionalArea)
&&
                                    float.TryParse(values[3], out initialVelocity) &&
                                    float.TryParse(values[4], out dragCoefficient))
                                {
                                    // Створюємо об'єкт Projectile та додаємо його до
списку
                                    Projectile projectile = new Projectile(name, mass,
crossSectionalArea, initialVelocity, dragCoefficient);
```

```
                                projectiles.Add(projectile);
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка завантаження даних про снаряди: " +
ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            return projectiles;
        }


    }

    private void buttonAdd_Click(object sender, EventArgs e)
    {
        // Перевірка текстових полів на наявність даних
        if (nameTextBox.Text == "" || massTextBox.Text == "" ||
crossSectionalAreaComboBox.SelectedItem.ToString() == "" ||
initialVelocityTextBox.Text == "" || dragCoefficientTextBox.Text == "")
        {
            MessageBox.Show("Поля з даними порожні.", "Помилка!");
            return;
        }

        // Отримуємо значення з TextBox
        string name = nameTextBox.Text;
        float mass, crossSectionalArea, initialVelocity, dragCoefficient;

        string CSA = crossSectionalAreaComboBox.SelectedItem.ToString();
        crossSectionalArea = float.Parse(CSA);

        // Перевірка типу даних і конвертація значень
        if (!ValidateFloatInput(massTextBox.Text, out mass))
        {
            MessageBox.Show("Недійсне значення для маси. Введіть дійсне число з
плаваючою комою.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        if (!ValidateFloatInput(initialVelocityTextBox.Text, out initialVelocity))
        {
            MessageBox.Show("Недійсне значення для початкової швидкості. Введіть
дійсне число з плаваючою комою.", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }
        if (!ValidateFloatInput(dragCoefficientTextBox.Text, out dragCoefficient))
        {
            MessageBox.Show("Недійсне значення для коефіцієнта опору. Введіть
дійсне число з плаваючою комою.", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // Створюємо об'єкт Projectile зі значеннями з TextBox
        Projectile projectile = new Projectile(name, mass, crossSectionalArea,
initialVelocity, dragCoefficient);

        // Додаємо об'єкт Projectile до списку
        projectiles.Add(projectile);
```

```
            // Очищаємо TextBox
            nameTextBox.Text = string.Empty;
            massTextBox.Text = string.Empty;
            initialVelocityTextBox.Text = string.Empty;
            dragCoefficientTextBox.Text = string.Empty;

            // Додаємо назву снаряду до ComboBox
            comboBox1.Items.Add(name);
        }
        // Метод для перевірки даних на тип в текстовому полі
        private bool ValidateFloatInput(string input, out float result)
        {
            return float.TryParse(input, out result);
        }

        private void зберегтиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Відкриваємо діалогове вікно для вибору файлу для збереження
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Text Files|*.txt";
            saveFileDialog.Title = "Save Projectile Data";

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                string fileName = saveFileDialog.FileName;

                // Зберігаємо дані про всі снаряди у файл
                foreach (Projectile projectile in projectiles)
                {
                    projectile.SaveToFile(fileName);
                }
            }
            MessageBox.Show("Дані про снаряд успішно збережено.", "Збереження даних",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }


       private void завантажитиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Відкриваємо діалогове вікно для вибору файлу для завантаження
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Text Files|*.txt";
            openFileDialog.Title = "Load Projectile Data";

            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                string fileName = openFileDialog.FileName;

                // Викликаємо метод LoadAllFromFile класу Projectile для завантаження
всіх снарядів з файлу
                List<Projectile> loadedProjectiles =
Projectile.LoadFromFile(fileName);

                if (loadedProjectiles.Count > 0)
                {
                    // Очищаємо список снарядів та ComboBox
                    projectiles.Clear();
                    comboBox1.Items.Clear();

                    // Додаємо завантажені снаряди до списку та ComboBox
                    projectiles.AddRange(loadedProjectiles);
                    foreach (Projectile projectile in loadedProjectiles)
                    {
```

```
                    comboBox1.Items.Add(projectile.Name);
                }

                MessageBox.Show("Усі дані про снаряди успішно завантажено.",
"Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("У файлі не знайдено дійсних даних про снаряди.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }

    private void trackBar1_ValueChanged(object sender, EventArgs e)
    {
        // Оновлення значення мітки з часом польоту
        Label labelFlightTime = this.Controls.Find("labelFlightTime",
true).FirstOrDefault() as Label;
        TrackBar trackBar1 = sender as TrackBar;

        if (labelFlightTime != null)
        {
            labelFlightTime.Text = "Час польоту: " + trackBar1.Value + " секунд";
        }
    }
    private void CalculateProjectileFlight()
    {
        string selectedProjectileName = comboBox1.SelectedItem.ToString();

        // Знаходження обраного снаряда за назвою
        Projectile selectedProjectile = projectiles.FirstOrDefault(p => p.Name ==
selectedProjectileName);

        if (selectedProjectile != null)
        {
            float mass = selectedProjectile.Mass;
            float dragCoefficient = selectedProjectile.DragCoefficient;
            float crossSectionalArea = selectedProjectile.CrossSectionalArea;
            float initialVelocity = selectedProjectile.InitialVelocity;
            float angle = float.Parse(angleTextBox.Text);
            float flightTime = trackBar1.Value;

            if (crossSectionalArea == 155)
            {
                crossSectionalArea = 0.01875f;
            }
            else if (crossSectionalArea == 152)
            {
                crossSectionalArea = 0.01811f;
            }
            else if (crossSectionalArea == 122)
            {
                crossSectionalArea = 0.01173f;
            }

            // Приклад даних про політ снаряда
            List<PointF> flightData = new List<PointF>();
            float g = 9.8f; // Прискорення вільного падіння

            for (float t = 0; t <= flightTime; t += 0.1f)
            {
                float x = initialVelocity * (float)Math.Cos(angle * Math.PI / 180)
* t; // Обчислення координати X
```

```
                    float y = initialVelocity * (float)Math.Sin(angle * Math.PI / 180)
* t - 0.5f * g * t * t; // Обчислення координати Y

                    // Застосування формули для врахування опору повітря
                    float velocity = (float)Math.Sqrt(Math.Pow(initialVelocity *
Math.Cos(angle * Math.PI / 180), 2) +
                                                      Math.Pow(initialVelocity *
Math.Sin(angle * Math.PI / 180) - g * t, 2));
                    float dragForce = 0.5f * dragCoefficient * crossSectionalArea *
velocity * velocity;
                    float acceleration = -dragForce / mass;
                    y += 0.5f * acceleration * t * t;

                    flightData.Add(new PointF(x, y));
                }

                // Отримання максимальних значень координат
                float maxX = flightData.Max(point => point.X);
                float maxY = flightData.Max(point => point.Y);

                // Створення нового графіку
                Bitmap bitmap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
                using (Graphics graphics = Graphics.FromImage(bitmap))
                {
                    // Очищення графіку
                    graphics.Clear(Color.White);

                    // Налаштування розмітки
                    Pen axisPen = new Pen(Color.Black);
                    Font axisFont = new Font("Arial", 10);
                    Brush axisBrush = new SolidBrush(Color.Black);

                    // Позначення осей координат
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, 20, 20); //
Вертикальна вісь X
                    graphics.DrawLine(axisPen, 20, bitmap.Height - 20, bitmap.Width -
20, bitmap.Height - 20); // Горизонтальна вісь Y

                    // Розмітка вертикальної вісі (X)
                    float stepX = maxX / 10;
                    for (float i = 0; i <= maxX; i += stepX)
                    {
                        int x = (int)(i + 20);
                        graphics.DrawLine(axisPen, x, bitmap.Height - 20, x,
bitmap.Height - 15); // Мітка на вісі X
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush, x *
bitmap.Width / maxX - 10, bitmap.Height - 15); // Підпис під міткою
                    }

                    // Розмітка горизонтальної вісі (Y)
                    float stepY = maxY / 50;
                    for (float i = 0; i <= maxY; i += stepY)
                    {
                        int y = (int)(bitmap.Height - 20 - i);
                        graphics.DrawLine(axisPen, 15, y, 20, y); // Мітка на вісі Y
                        graphics.DrawString(i.ToString("F1"), axisFont, axisBrush, 0,
y - 7); // Підпис біля мітки
                    }

                    // Налаштування лінії графіка
                    Pen linePen = new Pen(Color.Red);

                    // Малювання лінії графіка
                    for (int i = 1; i < flightData.Count; i++)
```

```
                    {
                        PointF startPoint = flightData[i - 1];
                        PointF endPoint = flightData[i];
                        graphics.DrawLine(linePen, startPoint.X * bitmap.Width / maxX
+ 20, bitmap.Height - startPoint.Y * bitmap.Height / maxY - 20,
                    endPoint.X * bitmap.Width / maxX + 20, bitmap.Height - endPoint.Y *
bitmap.Height / maxY - 20);

                    }
                }


                // Визначення дальності польоту
                PointF arrivePoint = PointF.Empty;
                float range = 0;
                for (int i = 1; i < flightData.Count; i++)
                {
                    PointF previousPoint = flightData[i - 1];
                    PointF currentPoint = flightData[i];

                    if (previousPoint.Y >= 0 && currentPoint.Y <= 0)
                    {
                        float пропорція = (0 - previousPoint.Y) / (currentPoint.Y -
previousPoint.Y);
                        range = previousPoint.X + пропорція * (currentPoint.X -
previousPoint.X);
                        arrivePoint = new PointF(range, 0);
                        break;
                    }
                }
                labelRange.Text = $"Дальність: {range} м";

                // Збереження графіку в директорії проекта
                SaveGraph(bitmap, selectedProjectile, angle);

                // Відображення графіку
                pictureBox1.Image = bitmap;
            }
        }
        public void SaveGraph(Bitmap bitmap, Projectile selectedProjectile, float
angle)
        {
            // Генерування унікального імені для зображення
            string imageName = $"{selectedProjectile.Name}_{angle}degrees.png";

            // Отримання шляху до папки "Графіки" у вашому проекті
            string folderPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Graphs");

            // Переконайтеся, що папка існує, інакше створіть її
            if (!Directory.Exists(folderPath))
            {
                Directory.CreateDirectory(folderPath);
            }

            // Збереження зображення у папку "Графіки" з унікальним іменем
            string imagePath = Path.Combine(folderPath, imageName);
            bitmap.Save(imagePath);
        }

        public void UpdateProjectile(Projectile updatedProjectile)
        {
            int index = projectiles.FindIndex(p => p.Name == updatedProjectile.Name);
            if (index >= 0)
```

```
            {
                projectiles[index] = updatedProjectile;
            }
        }

        private void buttonLaunch_Click(object sender, EventArgs e)
        {
            if (comboBox1.SelectedItem == null)
            {
                MessageBox.Show("Оберіт снаряд зі списку.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (angleTextBox.Text == "")
            {
                MessageBox.Show("Введіть кут відхилення снаряду.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            CalculateProjectileFlight();
        }

        private void даніПроСнарядиToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ProjectilesBase projectilesBase = new ProjectilesBase(this);
            projectilesBase.LoadData(projectiles); // Передача списку projectiles до
Form2
            projectilesBase.Show();
        }
        private void проПрограмуToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutBox1 aboutBox = new AboutBox1();
            aboutBox.Show();
        }

        private void завершенняРоботиToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            Application.Exit();
        }
    }
}
```

Додаток 2 – Клас ProjectileBase

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static WindowsFormsApp1.Form1;

namespace WindowsFormsApp1
{
    public partial class ProjectilesBase : Form
    {
        private Form1 form1;
        public ProjectilesBase(Form1 form1)
        {
```

```
            InitializeComponent();

            this.form1 = form1;
        }

        public void LoadData(List<Projectile> projectiles)
        {
            foreach (Projectile projectile in projectiles)
            {
                dataGridView1.Rows.Add(projectile.Name, projectile.Mass,
projectile.CrossSectionalArea, projectile.InitialVelocity,
projectile.DragCoefficient);
            }
        }
        public void UpdateProjectile(Projectile updatedProjectile)
        {
            Form1 form1 = (Form1)Application.OpenForms["Form1"]; // Отримати посилання
на екземпляр Form1
            form1.UpdateProjectile(updatedProjectile);
        }

        private void dataGridView1_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
        {
            int rowIndex = e.RowIndex;
            DataGridViewRow row = dataGridView1.Rows[rowIndex];

            string projectileName = row.Cells[0].Value.ToString();
            Projectile modifiedProjectile = form1.projectiles.FirstOrDefault(p =>
p.Name == projectileName);

            if (modifiedProjectile != null)
            {
                // Оновіть поля об'єкта Projectile
                // на основі змінених значень у таблиці

                modifiedProjectile.Mass = Convert.ToSingle(row.Cells[1].Value);
                modifiedProjectile.CrossSectionalArea =
Convert.ToSingle(row.Cells[2].Value);
                modifiedProjectile.InitialVelocity =
Convert.ToSingle(row.Cells[3].Value);
                modifiedProjectile.DragCoefficient =
Convert.ToSingle(row.Cells[4].Value);

                UpdateProjectile(modifiedProjectile); // Виклик методу оновлення
об'єкта Projectile у Form1
            }
        }
    }
}
```

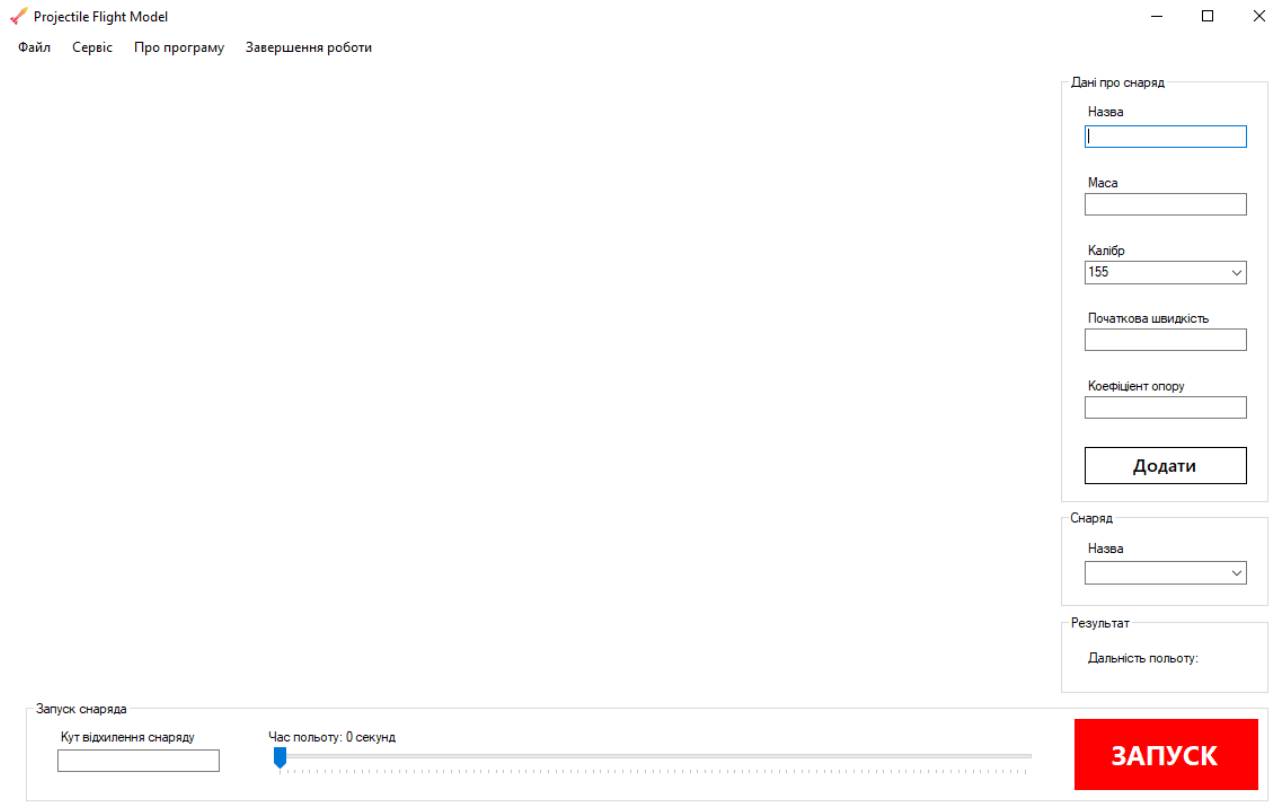
Додаток 3 – Клас AboutBox1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Reflection;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
```

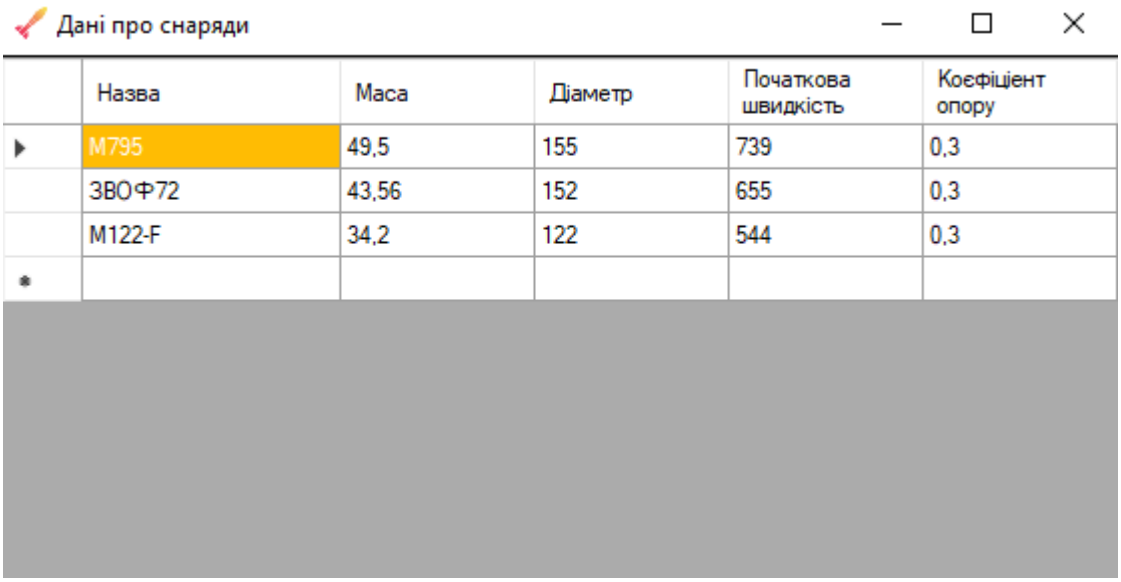
```
{
    partial class AboutBox1 : Form
    {
        public AboutBox1()
        {
            InitializeComponent();
            this.Text = "Про програму";
            this.labelProductName.Text = "Projectile Flight Model"; ;
            this.labelVersion.Text = String.Format("Version {0}", AssemblyVersion);
            this.labelCompanyName.Text = "Rostik Hubariev";
            this.textBoxDescription.Text = "Програма створена для розрахунку дальності
польоту снаряда з різними характеристиками та за різних умов.";
        }

        #region Assembly Attribute Accessors

        public string AssemblyTitle
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttribute),
false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute titleAttribute =
(AssemblyTitleAttribute)attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
                return
System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeBase);
            }
        }

        public string AssemblyVersion
        {
            get
            {
                return Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAttribut
e), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyDescriptionAttribute)attributes[0]).Description;
            }
        }

        public string AssemblyProduct
        {
            get
            {
```

```
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttribute),
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyProductAttribute)attributes[0]).Product;
            }
        }

        public string AssemblyCopyright
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAttribute)
, false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
            }
        }

        public string AssemblyCompany
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttribute),
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion

        private void okButton_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Додаток 4 – Скріншот головного меню програми

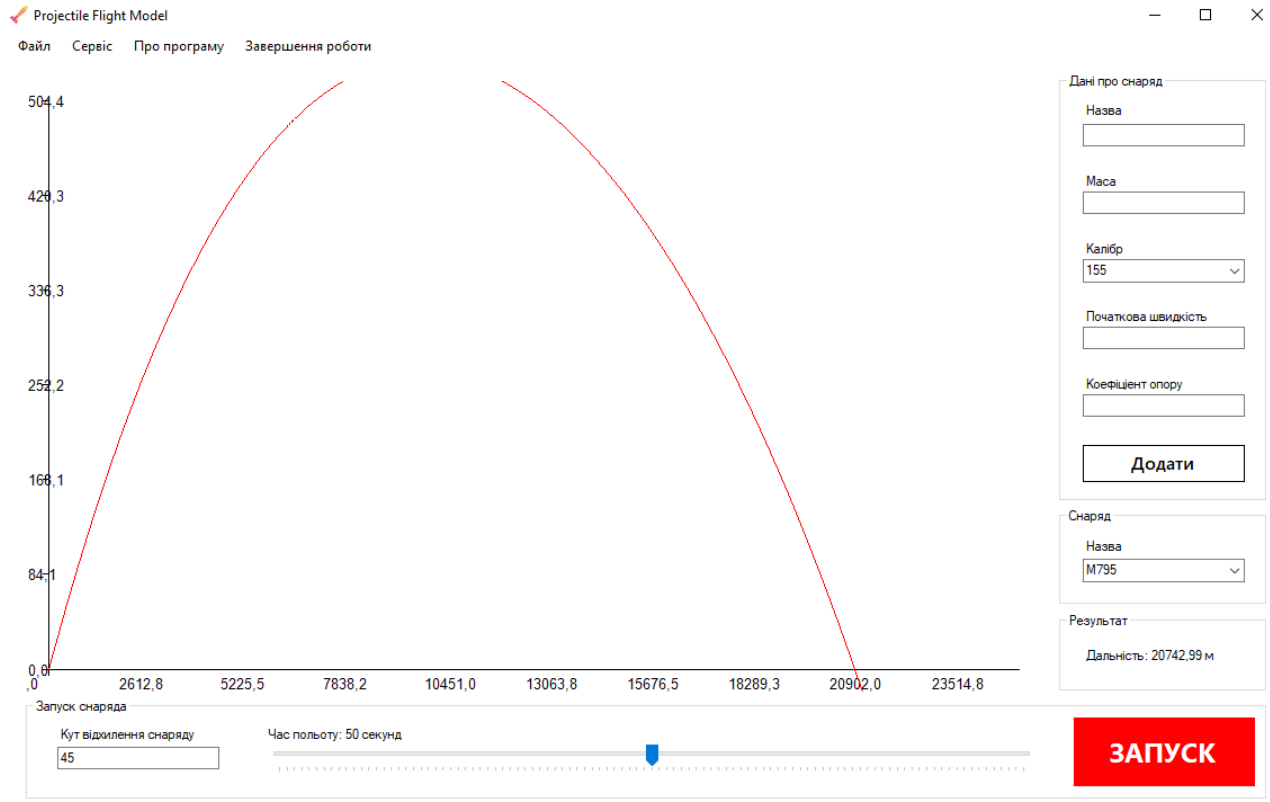


Додаток 5 – Скріншот таблиці з даними про снаряди

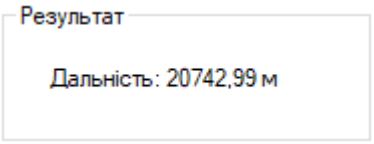


	Назва	Маса	Діаметр	Початкова швидкість	Коефіцієнт опору
▶	M795	49,5	155	739	0,3
	ЗВОФ72	43,56	152	655	0,3
	M122-F	34,2	122	544	0,3
*					

Додаток 6 – Скріншот роботи програми



Додаток 7 – Скріншот результату роботи програми



Додаток 8 – Скріншот вікна «Про програму»