**Черкаський національний університет імені Богдана Хмельницького**

(повне найменування вищого навчального закладу)

**Кафедра програмного забезпечення автоматизованих систем**

(повна назва кафедри)

**КУРСОВА РОБОТА**

***З дисципліни «Об’єктно-орієнтоване програмування»***

**НА ТЕМУ «Ігровий застосунок «Tanks»»**

|  |  |
| --- | --- |
|  | **Студента (ки) 2 курсу, групи КС-231**  **Спеціальності 121 Інженерія програмного забезпечення**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (прізвище та ініціали)  Керівник    (посада, вчене звання, науковий ступінь, прізвище та ініціали)  Національна шкала: \_\_\_\_\_\_\_\_\_\_\_\_  Кількість балів: \_\_\_\_  Оцінка: ESTC **\_\_\_\_** |

**Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

**\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

**\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище та ініціали)

ЗМІСТ

Contents

[**ВСТУП** 2](#_Toc198913572)

[**Розділ 1. Проєктування ігрового застосунку «Tanks». Базові принципи та побудова** 3](#_Toc198913573)

[**1.1** **Алгоритм взаємодії об’єктів між собою** 3](#_Toc198913574)

[Список інформаційних джерел 21](#_Toc198913575)

[**Додаток А. Лістинг коду алгоритму «Pixel perfect collision detection»** 22](#_Toc198913576)

# **ВСТУП**

Курсова робота з ООП охоплює у собі використання всіх головних принципів ООП, таких як поліморфізм, інкапсуляція, успадковування та абстракція. Задля розуміння як використовуються вони у роботі, наведу їх тлумачення [1]:

1. Поліморфізм вміщує в собі використання одного інтерфейсу чи деякої структури та перевизначення цієї концепції під різні типи задач. Розглядають такі типи як спеціалізований, параметричний та поліформізм типів. Я у своїй програмі використовую спеціалізований поліморфізм, який перевизначає загально створені методи під конкретний тип задачі для класу.
2. Інкапсуляція – це поняття вміщує у себе приховування деталей реалізації полів, методів та інших об’єктів між різними класами. Тобто, деякі класи можуть мати доступ до деяких полів, використовувати їх, а вже інший клас цієї можливості не має. У моєму застосунку використовуються такі параметри інкапсуляції як *private* (поля та методи використовуються лише у класі, в якому вони створені; для доступу створюються метод доступу і(або) метод-мутатор), *public* (до об’єкта можна на пряму звертатись, нема жодної заборони) та *protected* ( поля та методи доступні лише всередині базового та дочірніхз класах).
3. Успадковування – концепція, яка дозволяє сторювати базовий клас загального призначення, що вміщує в собі загальні поняття, аби різні дочірні класи могли успадковувати їх. Як наслідок, нам не потрібно писати у дочірніх класах базові поняття, тому задля скорочення коду записують їх у батьківському класі.
4. Абстракція дозволяє нам спрощувати складні системи, зосереджуючись лише на їхніх характеристиках, приховуючи непотрібні деталі реалізації.Одним словом, ви вказуємо лише «що» об’єкт робить, але не «яким чином» це здійснює.

Метою моєї роботи є створення ігрового застосунку, з використанням принципів ООП, демонстрації та застосування прицнипів створення 2D-ігор з використанням допоміжньої бібліотеки SFML у мові програмування C#. Для цього я поставив перед собою завдання: максимально показати мої вміння використання вище перечислених концепцій, а також створення вигідних алгоритмів для побудови ігрового застосунку, руху об’єктів, взаємодії між ними, т.д.

# **Розділ 1. Проєктування ігрового застосунку «Tanks». Базові принципи та побудова**

## **Алгоритм взаємодії об’єктів між собою**

У моїй роботі задля того, аби об’єкти правильно між собою взаємоділяли, я скористався не простою перевіркою меж спрайтів, у вигляді прямокутників (Bounding Box Collision), а попіксельною перевіркою для спрайтів (Pixel Perfect Collision Detection) [2].

Загальна ідея алгоритму (повний лістинг алгоритму наведений у додатку А):

1. Створення маска прозорості для текстури спрайта: будується масив типу byte з розмірністю рівній площі текстури. Через вкладений цикл, здійснюється перевірка кожного пікселя зображення і зберігається значення альфа-каналу прозорості. Простими словами, ми перевіряємо чи піксель прозорий чи містить у собі якийсь колір (рисунок 1.1.1).

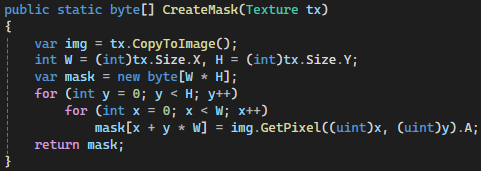


Рисунок 1.1.1. Перший етап алгоритму – створення маска

1. Перевірка обмежувальних прямокутників – аби уникнути зайвих ітерацій для перевірки перетину пікселів, виконується груба первірка на перетин прямокутників, якщо ні, то перевірка не потрібна.
2. У протилежному випадку, якщо перевірка обмежувальних прямокутників показала перетин, то береться його прямокутник, і для кожного пікселя в цій області обчислюються відповідні координати всередині кожної локальної системи спрайтів.
3. Використовуючи отримані локальні координати, перевіряємо в обох спрайтах, чи на місці отриманих координат піксель є непрозорим. Якщо так і є, то вважаємо колізія відбулась.

Цей алгоритм дає високу точність , адже враховує реальні форми об’єктів з усіма можливими кутами та формами, а не просто перевіряє прямокутник, що обмежує об’єкт. Через це ми не спостерігаємо порожніх колізій, коли два спрайти візуально не торкаються, проте прямокутники перетинаються.

Проте цей алгоритм є доволі громіздким, адже перевіряє кожен піксель на місці перетину спрайтів. Тому перш за все виконується груба перевірка на перетин самих прямокутників, а вже потім сам етап перевірки.

* 1. **Векторна математика руху**

Для обчислення напрямку руху, нам необхідно вирахувати відповідний кут для обчсилення вектора руху. Проте, перед цим ми маємо представити кути у радіанах, для цього використовується формула 1.2.1:

|  |  |  |
| --- | --- | --- |
|  |  | (1.2.1) |

– кут у радіанах;

– кут у градусах.

Після того як обчислено кут, ми можемо вирахувати уже сам вектор (формула 1.2.2):

|  |  |  |
| --- | --- | --- |
|  |  | (1.2.2) |

– вектор напрямку;

– X-складова вектору;

– Y-складова вектору.

Після цього ми множимо отриманий вектор на швидкість і наш об’єкт буде відповідно у цьому напрямку рухатись.

Тепер перейдемо до колізій. Коли об’єкт-танк колізує з якимсь об’єктом, ми маємо заборонити рух танкау цьому напрямку і відштовхнути їх звідти. Отже, тут потрібні три умови:

1. Обчислюєтьмя вектор різниці позицій (1.2.3):

|  |  |  |
| --- | --- | --- |
|  |  | (1.2.3) |

– вектор різниці;

– позиція першого об’єкта;

– позиція першого об’єкта.

1. Обчислення довжини вектора різниці (1.2.4):

|  |  |  |
| --- | --- | --- |
|  |  | (1.2.4) |

– довжина вектор різниці;

– Х-складова вектора різниці;

– Y-складова вектора різниці.

1. Коли ми отримали нашу довжину вектора різниці, перевіряємо на рівність нулю. Якщо так знаходимо вектор відштовхування (1.2.5):

|  |  |  |
| --- | --- | --- |
|  |  | (1.2.5) |

– новий вектор першого об’єкта;

– константа рівна *8f*.

# Список інформаційних джерел

1. Основні принципи ООП. [Електронний документ]. Режим доступу: <https://campus.epam.ua/ua/blog/275>. Перевірено 23.05.2025.
2. Nick Walton. Pixel Perfect Collision Detection in C. [Електронний докумени]. Режим доступу: <https://www.youtube.com/watch?v=9pnEBa4cy5w&t=130s>. Перевірено 23.05.2025.

**Додатки**

## **Додаток А. Лістинг коду алгоритму «Pixel perfect collision detection»**

public static class PixelPerfectCollision

{

public static byte[] CreateMask(Texture tx)

{

var img = tx.CopyToImage();

int W = (int)tx.Size.X, H = (int)tx.Size.Y;

var mask = new byte[W \* H];

for (int y = 0; y < H; y++)

for (int x = 0; x < W; x++)

mask[x + y \* W] = img.GetPixel((uint)x, (uint)y).A;

return mask;

}

public static bool Test(Sprite s1, byte[] mask1, Sprite s2, byte[] mask2, byte alphaLimit = 0)

{

var r1 = s1.GetGlobalBounds();

var r2 = s2.GetGlobalBounds();

if (!r1.Intersects(r2, out FloatRect inter))

return false;

for (int yi = 0; yi < inter.Height; yi++)

{

for (int xi = 0; xi < inter.Width; xi++)

{

float wx = inter.Left + xi;

float wy = inter.Top + yi;

var p1 = (Vector2f)s1.InverseTransform.TransformPoint(wx, wy);

var p2 = (Vector2f)s2.InverseTransform.TransformPoint(wx, wy);

int ix1 = (int)p1.X, iy1 = (int)p1.Y;

int ix2 = (int)p2.X, iy2 = (int)p2.Y;

if (ix1 >= 0 && iy1 >= 0 && ix2 >= 0 && iy2 >= 0 &&

ix1 < s1.TextureRect.Width && iy1 < s1.TextureRect.Height &&

ix2 < s2.TextureRect.Width && iy2 < s2.TextureRect.Height &&

mask1[ix1 + iy1 \* s1.TextureRect.Width] > alphaLimit &&

mask2[ix2 + iy2 \* s2.TextureRect.Width] > alphaLimit)

return true;

}

}

return false;

}

}