**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

До лабораторної роботи №1

**На тему: «**Побудова двовимірних фігур засобами мови програмування»

**З дисципліни: «**Комʼютерна графіка»

**Лектор:**

доц. каф. ПЗ

Левус Є.В.

**Виконав:**

ст. гр. ПЗ-24

Войтинський Д.О.

**Прийняв:**

доц. каф. ПЗ

Горечко О.М.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2025

**Тема роботи:** Побудова двовимірних зображень.

**Мета роботи:** Навчитись будувати двомірні зображення з допомогою графічних примітивів мови програмування.

**Теоретичні відомості**

Для реалізації цього коду використано технологію HTML Canvas у поєднанні з JavaScript, що є стандартним підходом для створення 2D-графіки у веб-додатках.

Вибір обумовлений такими перевагами:

* Простота інтеграції: Canvas легко вбудовується в HTML і керується через JavaScript.
* Гнучкість графіки: Підтримка малювання ліній, фігур і тексту через контекст 2d.
* Адаптивність: Обробка розмірів canvas з урахуванням devicePixelRatio для чіткості на різних екранах.
* Інтерактивність: Можливість реагувати на дії користувача (перетягування, введення даних).
* Продуктивність: Використання анімацій через requestAnimationFrame для плавного рендерингу.

**Базові функції JavaScript для роботи з графікою:**

* ctx.clearRect(x, y, width, height) — очищає задану прямокутну область на полотні.
* ctx.beginPath() — починає новий шлях для малювання.
* ctx.moveTo(x, y) — переміщує точку початку малювання без створення ліній.
* ctx.lineTo(x, y) — додає лінію від поточної точки до заданої.
* ctx.stroke() — обводить контур поточного шляху.
* ctx.fill() — заповнює замкнутий шлях кольором.
* ctx.fillRect(x, y, width, height) — малює заповнений прямокутник.
* ctx.arc(x, y, radius, startAngle, endAngle) — створює дугу або коло.
* ctx.fillText(text, x, y) — відображає текст у вказаній позиції.
* ctx.scale(x, y) — масштабує систему координат полотна.
* requestAnimationFrame(callback) — викликає функцію для плавної анімації.

**Індивідуальне завдання**

Написати програму згідно індивідуального варіанту вибраною мовою програмування з використанням її базових графічних примітивів. Програма має відповідати таким вимогам:

1. Відображення системи координат з початком у центрі області виведення з відповідними підписами та позначками (початок, одиничний відрізок, напрям, назва осей).

2. Задання фігур за введеними координати, що відповідають координатам відповідної побудованої декартової системи, а не координатам області виведення (Canvas).

3. Оптимальний ввід користувачем координат фігури з автоматичним обчисленням за можливості інших координат для уникнення зайвих обчислень користувачем.

4. Передбачити можливість некоректного введення даних.

5. Зручний інтерфейс користувача.

**Варіант 2**. Побудувати декілька рівносторонніх трикутників із заданими вручну координатами вершин однієї сторони та можливістю вибору кольору заливки та вигляду вершин трикутника (у вигляді квадратиків, кружечків).

**Код програми**

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Equilateral Triangles</title>

<link rel="stylesheet" href="./style.css">

</head>

<body>

<main>

<div class="container">

<div class="box">

<h1>Coordinate Plane with Triangles</h1>

<canvas id="canvas" class="canvas"></canvas>

</div>

<div class="controll">

<div class="triangle-form">

<h3>Add Equilateral Triangle</h3>

<div class="form-group">

<label>First Vertex (x1, y1):</label>

<input type="number" id="x1" placeholder="X1" step="0.5">

<input type="number" id="y1" placeholder="Y1" step="0.5">

</div>

<div class="form-group">

<label>Second Vertex (x2, y2):</label>

<input type="number" id="x2" placeholder="X2" step="0.5">

<input type="number" id="y2" placeholder="Y2" step="0.5">

</div>

<div class="form-group">

<label>Fill Color:</label>

<input type="color" id="fillColor" value="#3498db">

</div>

<div class="form-group">

<label>Vertex Style:</label>

<select id="vertexStyle">

<option value="circle">Circle</option>

<option value="square">Square</option>

</select>

</div>

<button id="addTriangle">Add Triangle</button>

</div>

<div class="triangle-list">

<h3>Triangles</h3>

<ul id="triangleList"></ul>

</div>

<div class="main-controls">

<button id="reset">Reset View</button>

<button id="clear">Clear All</button>

</div>

</div>

</div>

</main>

<img id="cat-image" src="cat.jpg" alt="Cat image" style="position: fixed; bottom: 20px; right: 20px; width: 300px; height: 300px; border-radius: 10px; box-shadow: 0 4px 8px rgba(0,0,0,0.2); z-index: 100;">

<script src="./scripts.js"></script>

</body>

</html>

**scripts.js**

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

// Set actual size in memory (scaled to account for extra pixel density)

const dpr = window.devicePixelRatio || 1;

canvas.width = canvas.offsetWidth \* dpr;

canvas.height = canvas.offsetHeight \* dpr;

// Normalize coordinate system to use CSS pixels

ctx.scale(dpr, dpr);

// Coordinate plane variables

let scale = 40; // pixels per unit

let offsetX = canvas.width / (2 \* dpr);

let offsetY = canvas.height / (2 \* dpr);

let animationId = null;

let isAnimating = false;

// Array to store all triangles

let triangles = [];

// Draw the coordinate plane

function drawGrid() {

ctx.clearRect(0, 0, canvas.width/dpr, canvas.height/dpr);

// Draw grid lines

ctx.strokeStyle = '#e0e0e0';

ctx.lineWidth = 1;

// Vertical lines

for (let x = offsetX % scale; x < canvas.width/dpr; x += scale) {

ctx.beginPath();

ctx.moveTo(x, 0);

ctx.lineTo(x, canvas.height/dpr);

ctx.stroke();

}

// Horizontal lines

for (let y = offsetY % scale; y < canvas.height/dpr; y += scale) {

ctx.beginPath();

ctx.moveTo(0, y);

ctx.lineTo(canvas.width/dpr, y);

ctx.stroke();

}

// Draw axes

ctx.strokeStyle = '#000';

ctx.lineWidth = 2;

// X-axis

ctx.beginPath();

ctx.moveTo(0, offsetY);

ctx.lineTo(canvas.width/dpr, offsetY);

ctx.stroke();

// Y-axis

ctx.beginPath();

ctx.moveTo(offsetX, 0);

ctx.lineTo(offsetX, canvas.height/dpr);

ctx.stroke();

// Draw numbers

ctx.fillStyle = '#000';

ctx.font = '12px Arial';

ctx.textAlign = 'center';

ctx.textBaseline = 'middle';

// X-axis numbers

for (let x = Math.ceil(-offsetX / scale) \* scale; x < (canvas.width/dpr - offsetX); x += scale) {

if (x === 0) continue;

const xPos = offsetX + x;

const value = x / scale;

ctx.fillText(value.toString(), xPos, offsetY + 20);

}

// Y-axis numbers

for (let y = Math.ceil(-offsetY / scale) \* scale; y < (canvas.height/dpr - offsetY); y += scale) {

if (y === 0) continue;

const yPos = offsetY + y;

const value = -y / scale;

ctx.fillText(value.toString(), offsetX - 20, yPos);

}

// Draw origin label

ctx.fillText('0', offsetX - 10, offsetY + 20);

// Draw arrow heads

drawArrow(canvas.width/dpr - 10, offsetY, 1, 0); // X-axis arrow

drawArrow(offsetX, 10, 0, -1); // Y-axis arrow

// X and Y labels

ctx.fillText('X', canvas.width/dpr - 10, offsetY - 20);

ctx.fillText('Y', offsetX + 20, 10);

// Draw all triangles

drawTriangles();

}

function drawArrow(x, y, dirX, dirY) {

const arrowSize = 15;

ctx.beginPath();

ctx.moveTo(x, y);

// Calculate the points for the arrow head based on direction

if (Math.abs(dirX) > Math.abs(dirY)) {

// Horizontal arrow (X-axis)

const xOffset = dirX \* arrowSize;

ctx.lineTo(x - xOffset, y - arrowSize/2);

ctx.lineTo(x - xOffset, y + arrowSize/2);

} else {

// Vertical arrow (Y-axis)

const yOffset = dirY \* arrowSize;

ctx.lineTo(x - arrowSize/2, y - yOffset);

ctx.lineTo(x + arrowSize/2, y - yOffset);

}

ctx.closePath();

ctx.fill();

}

// Convert graph coordinates to canvas coordinates

function graphToCanvas(x, y) {

return {

x: offsetX + x \* scale,

y: offsetY - y \* scale

};

}

// Convert canvas coordinates to graph coordinates

function canvasToGraph(x, y) {

return {

x: (x - offsetX) / scale,

y: (offsetY - y) / scale

};

}

// Calculate the third vertex of an equilateral triangle

function calculateThirdVertex(x1, y1, x2, y2) {

// Vector from point 1 to point 2

const dx = x2 - x1;

const dy = y2 - y1;

// Rotate this vector by 60 degrees (π/3 radians) to get the third point

// Rotation matrix for 60 degrees:

// [ cos(60°) -sin(60°) ] = [ 0.5 -0.866 ]

// [ sin(60°) cos(60°) ] [ 0.866 0.5 ]

const x3 = x1 + 0.5 \* dx - Math.sqrt(3) / 2 \* dy;

const y3 = y1 + Math.sqrt(3) / 2 \* dx + 0.5 \* dy;

return { x: x3, y: y3 };

}

// Draw all triangles

function drawTriangles() {

triangles.forEach(triangle => {

drawTriangle(triangle);

});

}

// Draw a single triangle

function drawTriangle(triangle) {

const { x1, y1, x2, y2, x3, y3, color, vertexStyle } = triangle;

// Convert to canvas coordinates

const p1 = graphToCanvas(x1, y1);

const p2 = graphToCanvas(x2, y2);

const p3 = graphToCanvas(x3, y3);

// Draw triangle

ctx.beginPath();

ctx.moveTo(p1.x, p1.y);

ctx.lineTo(p2.x, p2.y);

ctx.lineTo(p3.x, p3.y);

ctx.closePath();

ctx.fillStyle = color;

ctx.fill();

ctx.strokeStyle = '#000';

ctx.lineWidth = 1;

ctx.stroke();

// Draw vertices

drawVertex(p1.x, p1.y, vertexStyle);

drawVertex(p2.x, p2.y, vertexStyle);

drawVertex(p3.x, p3.y, vertexStyle);

}

// Draw a vertex as a square or circle

function drawVertex(x, y, style) {

const size = 6;

ctx.fillStyle = '#000';

if (style === 'square') {

ctx.fillRect(x - size/2, y - size/2, size, size);

} else { // circle

ctx.beginPath();

ctx.arc(x, y, size/2, 0, Math.PI \* 2);

ctx.fill();

}

}

// Handle dragging of the coordinate plane

function handleDrag(e) {

if (isAnimating) return;

const rect = canvas.getBoundingClientRect();

const startX = e.clientX - rect.left;

const startY = e.clientY - rect.top;

const startOffsetX = offsetX;

const startOffsetY = offsetY;

function move(e) {

const x = e.clientX - rect.left;

const y = e.clientY - rect.top;

offsetX = startOffsetX + (x - startX);

offsetY = startOffsetY + (y - startY);

drawGrid();

}

function stopDrag() {

document.removeEventListener('mousemove', move);

document.removeEventListener('mouseup', stopDrag);

}

document.addEventListener('mousemove', move);

document.addEventListener('mouseup', stopDrag);

}

// Add a new triangle

function addTriangle() {

const x1 = parseFloat(document.getElementById('x1').value);

const y1 = parseFloat(document.getElementById('y1').value);

const x2 = parseFloat(document.getElementById('x2').value);

const y2 = parseFloat(document.getElementById('y2').value);

const color = document.getElementById('fillColor').value;

const vertexStyle = document.getElementById('vertexStyle').value;

if (isNaN(x1) || isNaN(y1) || isNaN(x2) || isNaN(y2)) {

alert('Please enter valid coordinates');

return;

}

// Calculate third vertex

const thirdVertex = calculateThirdVertex(x1, y1, x2, y2);

// Add triangle to array

const triangle = {

x1, y1, x2, y2,

x3: thirdVertex.x, y3: thirdVertex.y,

color,

vertexStyle

};

triangles.push(triangle);

// Update the display

updateTriangleList();

drawGrid();

// Clear the input fields

document.getElementById('x1').value = '';

document.getElementById('y1').value = '';

document.getElementById('x2').value = '';

document.getElementById('y2').value = '';

}

// Update the triangle list display

function updateTriangleList() {

const list = document.getElementById('triangleList');

list.innerHTML = '';

triangles.forEach((triangle, index) => {

const li = document.createElement('li');

li.innerHTML = `

Triangle ${index + 1}

<div>

<button class="delete-triangle" data-index="${index}">Delete</button>

</div>

`;

list.appendChild(li);

});

// Add event listeners to delete buttons

document.querySelectorAll('.delete-triangle').forEach(button => {

button.addEventListener('click', function() {

const index = parseInt(this.getAttribute('data-index'));

triangles.splice(index, 1);

updateTriangleList();

drawGrid();

});

});

}

// Animate the coordinate plane (example animation)

function animate() {

// Example animation - rotate the coordinate system

offsetX += 1;

offsetY = canvas.height / (2 \* dpr) + Math.sin(Date.now() / 1000) \* 50;

drawGrid();

if (isAnimating) {

animationId = requestAnimationFrame(animate);

}

}

// Event listeners

canvas.addEventListener('mousedown', handleDrag);

document.getElementById('reset').addEventListener('click', function() {

scale = 40;

offsetX = canvas.width / (2 \* dpr);

offsetY = canvas.height / (2 \* dpr);

drawGrid();

});

document.getElementById('clear').addEventListener('click', function() {

scale = 40;

offsetX = canvas.width / (2 \* dpr);

offsetY = canvas.height / (2 \* dpr);

triangles = [];

updateTriangleList();

drawGrid();

});

document.getElementById('addTriangle').addEventListener('click', addTriangle);

// Initial draw when the page loads

window.addEventListener('load', function() {

drawGrid();

updateTriangleList();

});

**style.css**

.canvas {

box-sizing: border-box;

border: 2px solid #000;

display: block;

}

canvas {

width: 900px;

height: 700px;

}

.container {

display: flex;

justify-content: center;

align-items: flex-start;

min-height: 100vh;

padding: 20px;

}

.controll {

display: flex;

flex-direction: column;

row-gap: 20px;

padding: 20px;

width: 300px;

background-color: #f8f9fa;

border-radius: 8px;

margin-left: 20px;

}

button {

padding: 8px 16px;

cursor: pointer;

background-color: #f0f0f0;

border: 1px solid #ccc;

border-radius: 4px;

min-width: 100px;

}

button:hover {

background-color: #e0e0e0;

}

.box h1 {

text-align: center;

margin-bottom: 10px;

}

/\* New styles \*/

.form-group {

margin-bottom: 12px;

}

.form-group label {

display: block;

margin-bottom: 5px;

font-weight: bold;

}

.form-group input[type="number"] {

width: 70px;

padding: 5px;

margin-right: 5px;

}

.form-group input[type="color"] {

width: 50px;

height: 30px;

}

.form-group select {

width: 100px;

padding: 5px;

}

.triangle-form, .triangle-list, .main-controls {

padding: 15px;

border: 1px solid #ddd;

border-radius: 4px;

background-color: white;

}

.triangle-form h3, .triangle-list h3 {

margin-top: 0;

margin-bottom: 15px;

}

.triangle-list ul {

list-style: none;

padding: 0;

max-height: 150px;

overflow-y: auto;

}

.triangle-list li {

padding: 8px;

margin-bottom: 5px;

background-color: #f0f0f0;

border-radius: 4px;

display: flex;

justify-content: space-between;

align-items: center;

}

#addTriangle {

background-color: #28a745;

color: white;

border: none;

margin-top: 10px;

width: 100%;

}

#addTriangle:hover {

background-color: #218838;

}

.delete-triangle {

background-color: #dc3545;

color: white;

border: none;

padding: 3px 8px;

border-radius: 3px;

font-size: 0.8rem;

}

.delete-triangle:hover {

background-color: #c82333;

}

**Результат виконання**

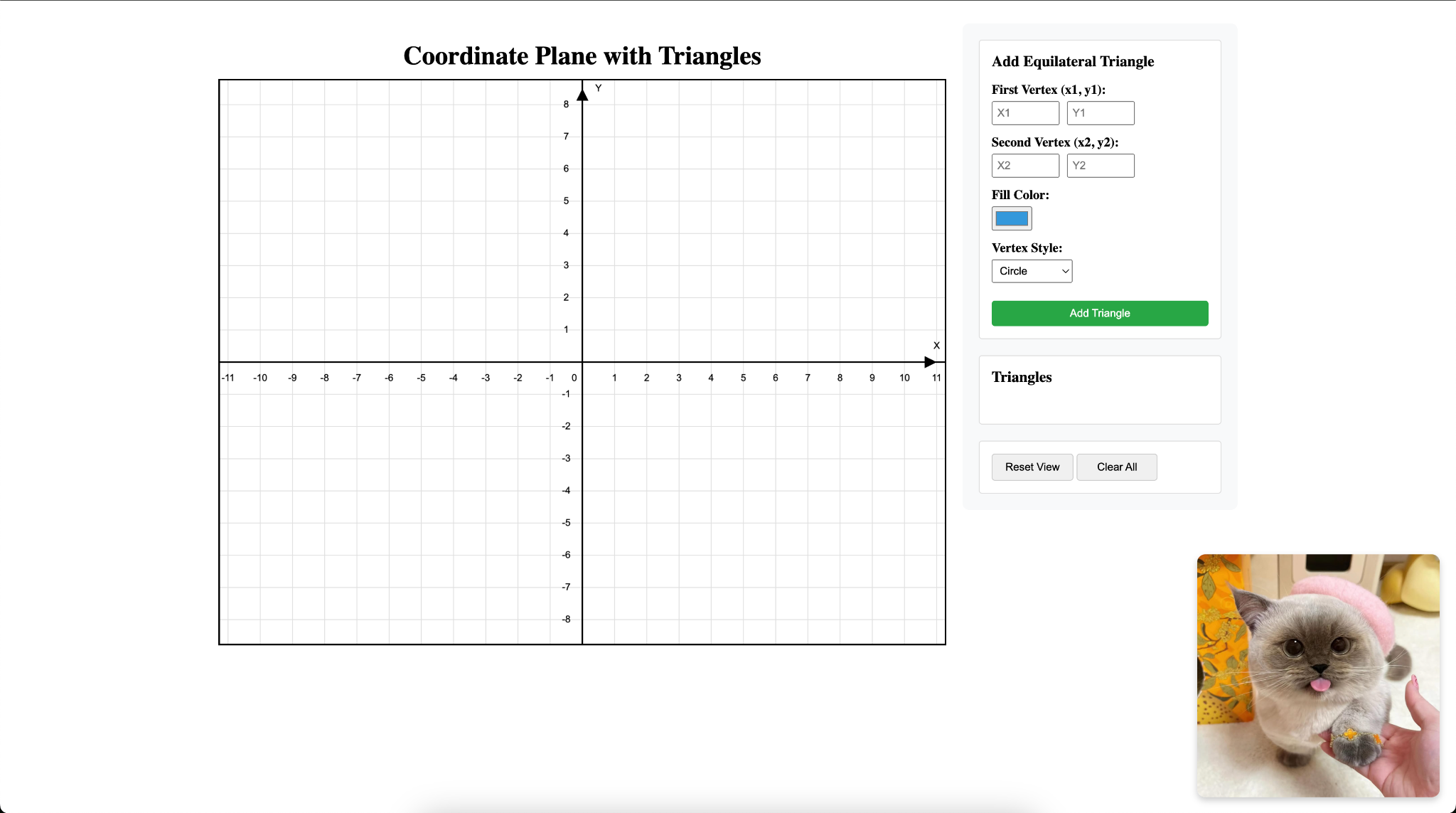
****

Рис.1. Загальний вигляд програми

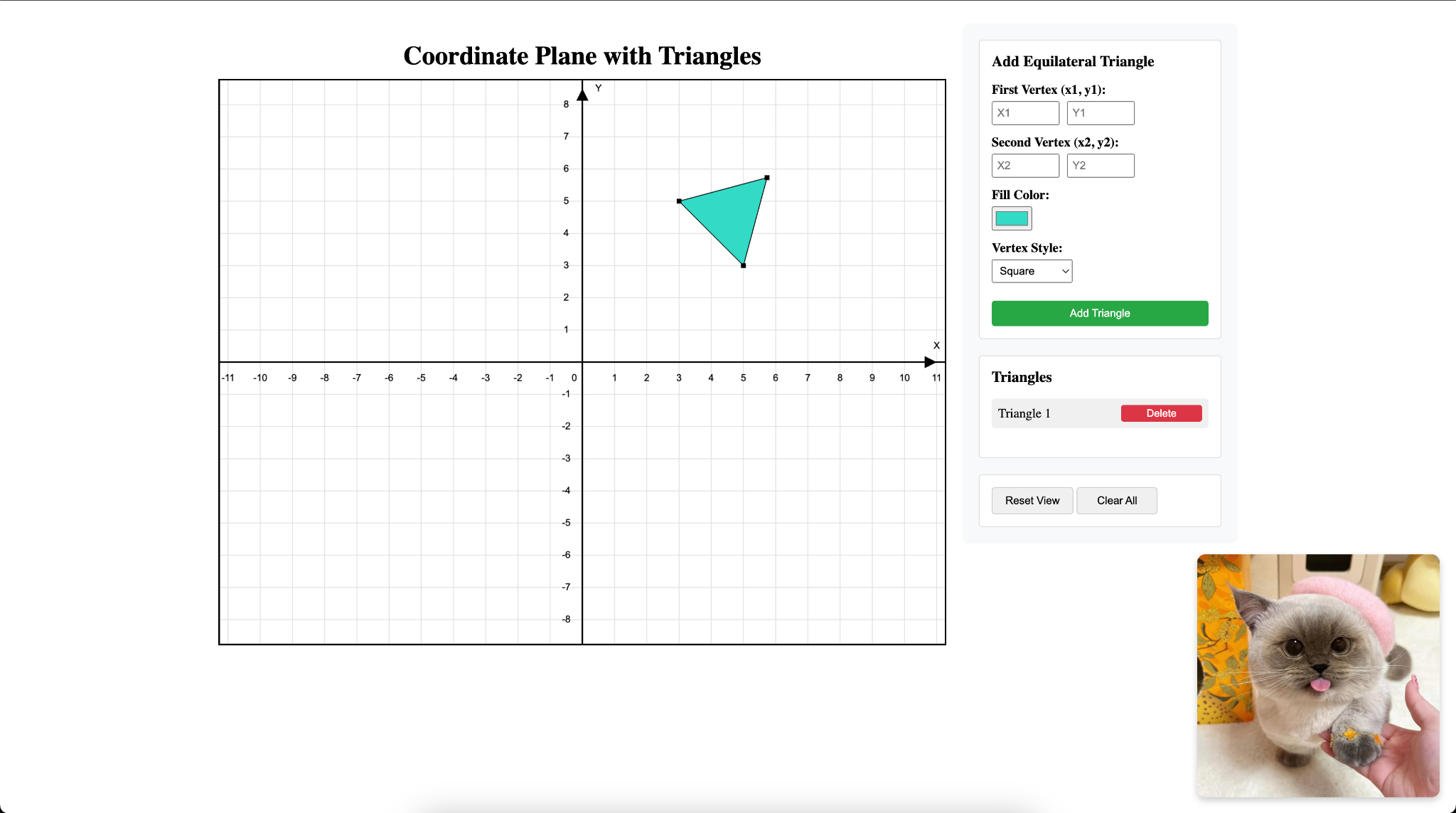


Рис.2. Відмалювання трикутника(Square)

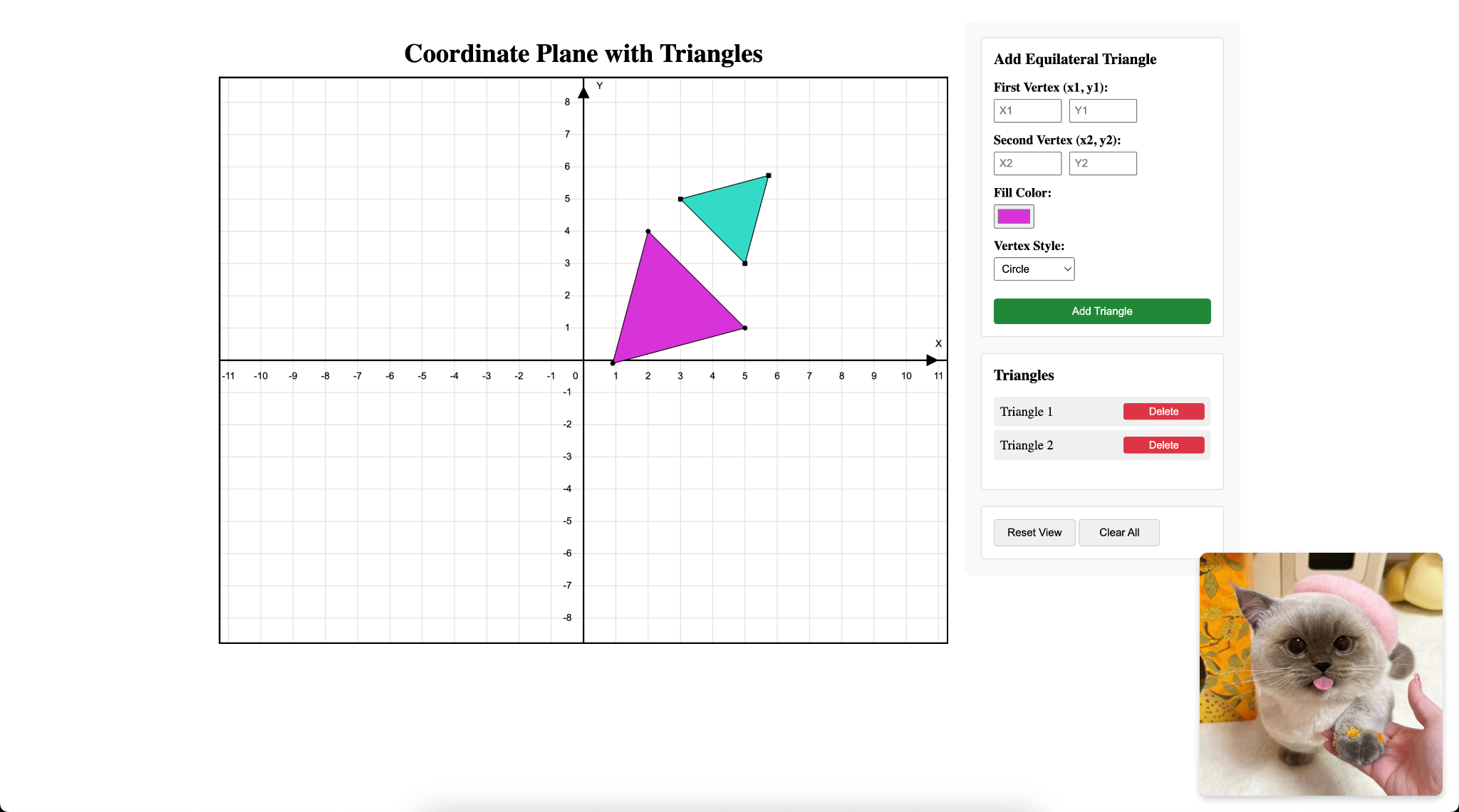


Рис.3. Відмалювання трикутника (Circle)

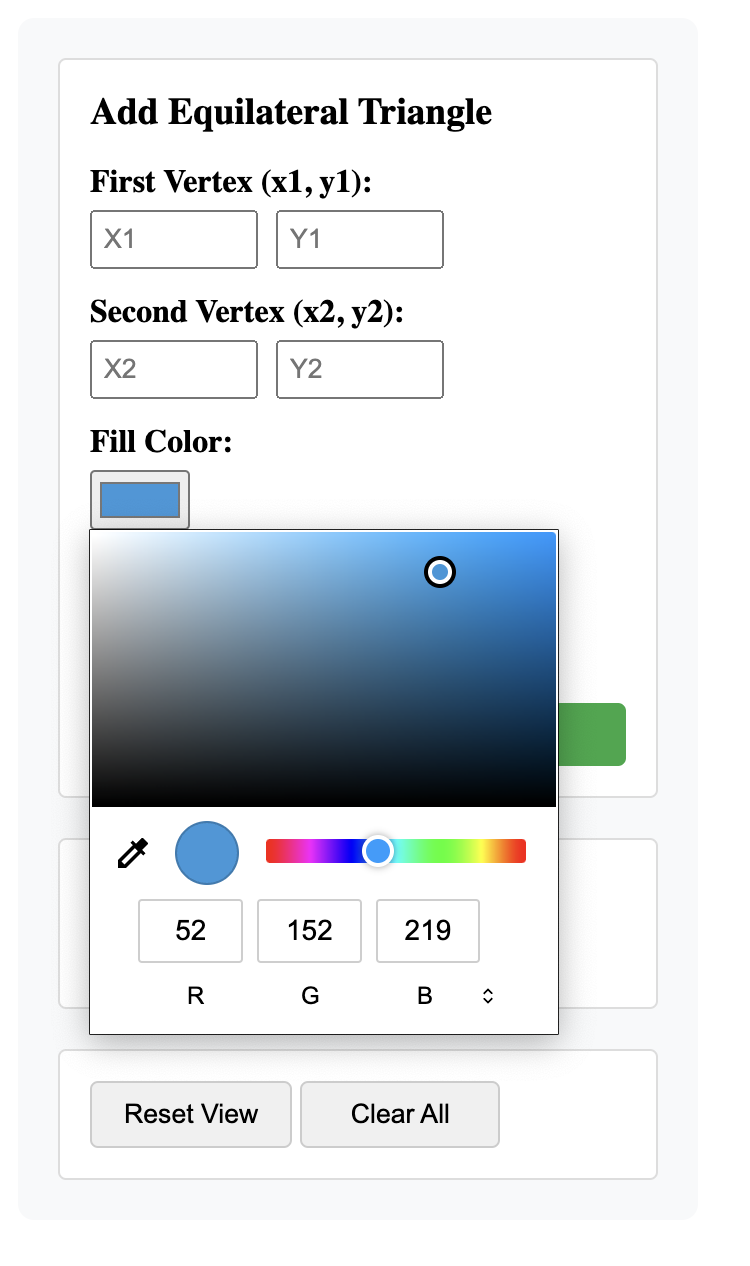


Рис.4. Вибір кольору трикутника

**Висновки**

На цій лабораторній роботі я навчився будувати двомірні зображення з допомогою графічних примітивів мови програмування. Для цього було використано інструмент Canvas у поєднанні з JavaScript, оскільки таке поєднання має багато можливостей для роботи з 2D графікою, і дозволяє створювати сучасні веб сторінки.