**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**ЛАБОРАТОРНА РОБОТА № 2**

**Тема**: Моделювання руху тіла, кинутого під кутом до горизонту

Виконав студент ІІ курсу

спеціальності «Інженерія

програмного забезпечення»

Олексій Олексій

**Ужгород 2024**

**Мета:** Дослідити та візуалізувати рух тіла, кинутого під кутом до горизонту. Використовуючи математичні моделі аналізувати траєкторію руху, висоту максимального підйому, час польоту та іншіпараметри.

**Завдання до лабораторної роботи**

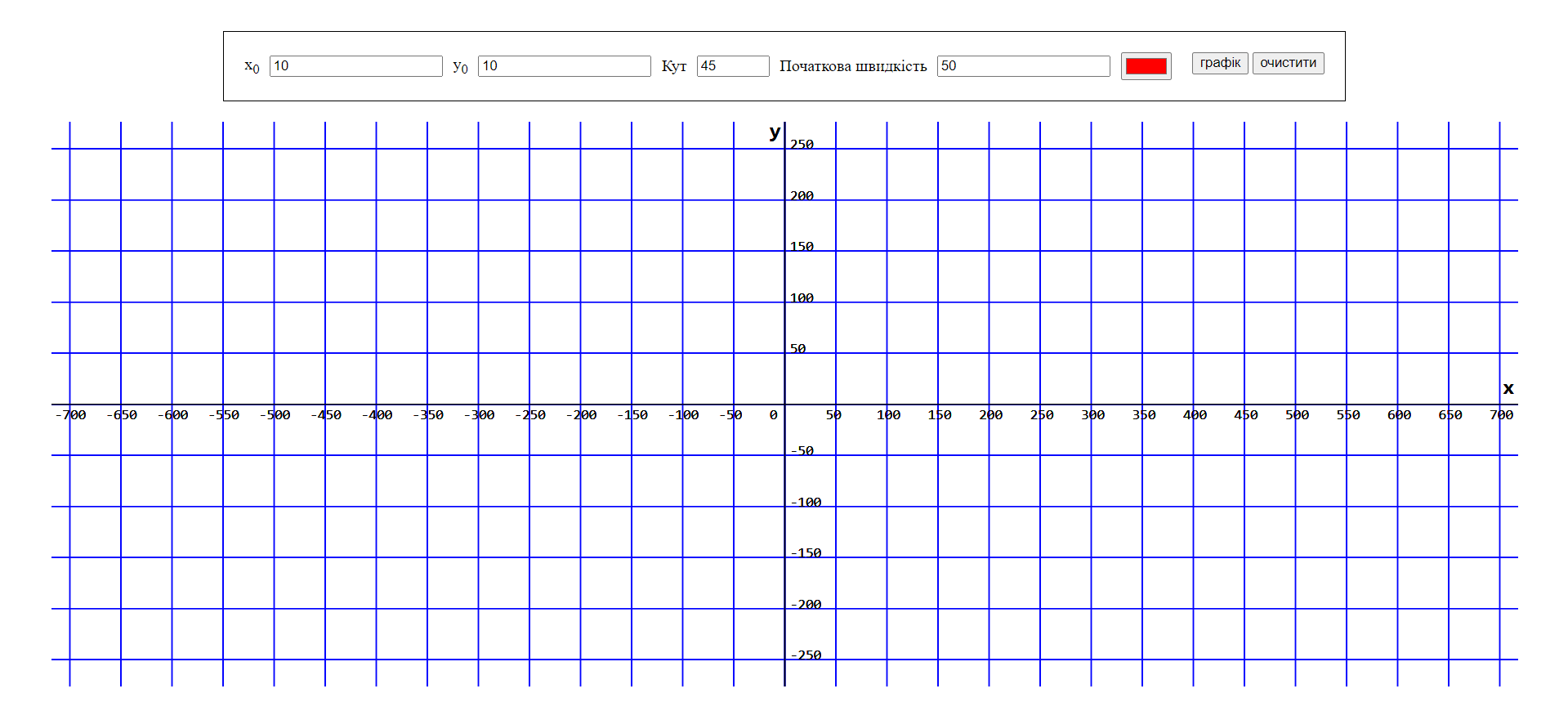
1. Визначте математичну модель руху тіла, кинутого під кутом к горизонті, без опору повітря. Використовуйте рівняння руху для горизонтального та вертикального напрямків
2. Використовуючи набуті в попередній роботі навички програмування на мові JavaScript, та за допомогою HTML розмітки і каскадних таблиць стилів CSS побудувати траєкторію руху тіла кинутого під кутом до горизонту

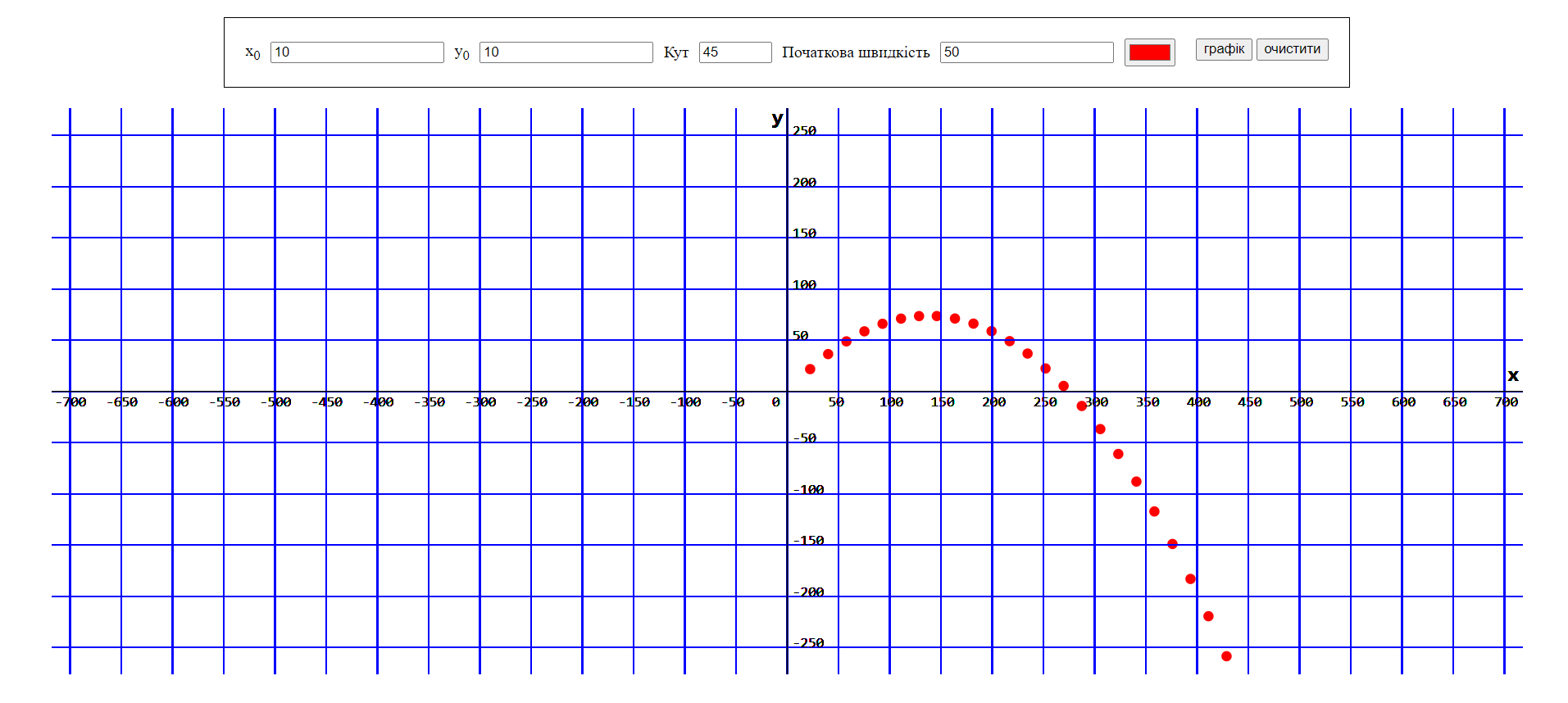
Для побудови графіка доцільно буде використати HTML таг Canvas

1. Використовуючи попередній результата зробити рефакторинг коду, та замість використання Canvas використати бібліотеку для побудови діаграм та графіків в JavaScript D3.JS
2. Оформити звіт виконаної роботи, згідно взірця
3. Отримані результати (лістинг коду завантажити на репозиторій, або у вигляді архіву, та звіт, завантажити у папку курсу, підпапку лабораторної роботи та підпапку із назвою, що відповідає вашому прізвищу). Якщо код завантажено на репозиторій, то у звіті потрібно вказати посилання на репозиторій.

**Хід роботи**

Користувач може задати початкові координати x та y, кут нахилу, початкову швидкість та колір, після натискання кнопки «графік» відбувається моделювання графіку по заданим параметрам, після натискання кнопки «очистити» очищується масив з графіками.





**Лістинг**

style.css:

body {

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

  gap: 20px;

}

.parameters {

  border: 1px solid black;

  padding: 20px;

  display: flex;

  gap: 10px;

}

.input-container {

  display: flex;

  gap: 10px;

  align-items: center;

  justify-content: center;

}

index.html:

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8">

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

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

    <title>Document</title>

    <script src="https://d3js.org/d3.v7.min.js"></script>

  </head>

  <body>

    <div id="container"></div>

    <div class="parameters">

      <div class="input-container">

        <label for="x-0">x<sub>0</sub></label>

        <input required type="number" name="x-0" id="x-0">

      </div>

      <div class="input-container">

        <label for="y-0">y<sub>0</sub></label>

        <input required type="number" name="y-0" id="y-0">

      </div>

      <div class="input-container">

        <label for="angle">Кут</label>

        <input required type="number" min="0" max="360" name="angle" id="angle">

      </div>

      <div class="input-container">

        <label for="v-0">Початкова швидкість</label>

        <input required type="number" name="v-0" id="v-0">

      </div>

      <!-- <div class="input-container">

        <label for="a">Прискорення</label>

        <input required type="number" name="a" id="a">

      </div> -->

      <input required type="color" name="color" id="color">

      <div id="chart-container"></div>

      <div class="btn-container">

        <button id="btn-chart" onclick="addParticle()">графік</button>

        <button id="btn-clear" onclick="clearParticles()">очистити</button>

      </div>

    </div>

    <script src="script.js">

    </script>

  </body>

</html>

Index.js

const width = window.innerWidth - 100

const height = window.innerHeight - 150

const svg = d3

  .select('body')

  .append('svg')

  .attr('width', width)

  .attr('height', height)

let particles = []

let v0 = document.getElementById('v-0')

let x0 = document.getElementById('x-0')

let y0 = document.getElementById('y-0')

// let a = document.getElementById('a')

let angle = document.getElementById('angle')

let color = document.getElementById('color')

const btnChart = document.getElementById('btn-chart')

const btnClear = document.getElementById('btn-clear')

function addParticle() {

  let validatedX =

    parseFloat(x0.value) > 0

      ? parseFloat(x0.value) + width / 2

      : width / 2 - Math.abs(parseFloat(x0.value))

  let validatedY =

    parseFloat(y0.value) > 0

      ? height / 2 - parseFloat(y0.value)

      : Math.abs(parseFloat(y0.value)) + height / 2

  let particle = new Particle(

    validatedX,

    validatedY,

    v0.value,

    angle.value,

    color.value

  )

  particles.push(particle)

}

function clearParticles() {

  particles = []

  svg.selectAll('\*').remove()

}

function Particle(x0, y0, v0, angle, color) {

  this.x0 = parseFloat(x0)

  this.y0 = parseFloat(y0)

  this.color = color

  this.v0 = parseFloat(v0)

  this.angle = (-parseFloat(angle) \* Math.PI) / 180

  this.startTime = Date.now()

  // Додамо гравітаційне прискорення (силу тяжіння)

  this.g = 9.8 // На цей момент це просто емпіричне значення. Ви можете налаштувати його відповідно до ваших потреб.

  this.draw = () => {

    let currentTime = Date.now()

    let elapsedTime = currentTime - this.startTime

    let t = elapsedTime / 1000

    // Обчислимо нові координати x та y, враховуючи силу тяжіння

    let x = this.x0 + this.v0 \* t \* Math.cos(this.angle)

    let y = this.y0 + this.v0 \* t \* Math.sin(this.angle) + 0.5 \* this.g \* t \*\* 2 // Додамо силу тяжіння

    svg

      .append('circle')

      .attr('cx', x)

      .attr('cy', y)

      .attr('r', 5)

      .style('fill', this.color)

  }

}

function Grid(scale = 50) {

  this.scale = scale

  this.cX = width / 2

  this.cY = height / 2

  this.draw = () => {

    for (let y = this.cY; y < height; y += this.scale) {

      svg

        .append('line')

        .attr('x1', 0)

        .attr('y1', y)

        .attr('x2', width)

        .attr('y2', y)

        .style('stroke', 'blue')

    }

    for (let y = this.cY; y > 0; y -= this.scale) {

      svg

        .append('line')

        .attr('x1', 0)

        .attr('y1', y)

        .attr('x2', width)

        .attr('y2', y)

        .style('stroke', 'blue')

    }

    for (let x = this.cX; x < width; x += this.scale) {

      svg

        .append('line')

        .attr('x1', x)

        .attr('y1', 0)

        .attr('x2', x)

        .attr('y2', height)

        .style('stroke', 'blue')

    }

    for (let x = this.cX; x > 0; x -= this.scale) {

      svg

        .append('line')

        .attr('x1', x)

        .attr('y1', 0)

        .attr('x2', x)

        .attr('y2', height)

        .style('stroke', 'blue')

    }

  }

}

function Axes(scale) {

  this.scale = scale

  this.cX = width / 2

  this.cY = height / 2

  this.draw = () => {

    svg

      .append('line')

      .attr('x1', this.cX)

      .attr('y1', 0)

      .attr('x2', this.cX)

      .attr('y2', height)

      .style('stroke', 'black')

    svg

      .append('line')

      .attr('x1', 0)

      .attr('y1', this.cY)

      .attr('x2', width)

      .attr('y2', this.cY)

      .style('stroke', 'black')

    let markup = new Markup(this.scale)

    markup.draw()

  }

}

function Markup(scale) {

  this.scale = scale

  this.cX = width / 2

  this.cY = height / 2

  this.draw = () => {

    for (let x = this.cX; x < width; x += scale) {

      let valX = this.getValidatedX(x)

      if (valX == 0) continue

      svg

        .append('text')

        .attr('x', x - 10)

        .attr('y', this.cY + 15)

        .text(valX)

        .attr('font-family', 'monospace')

        .attr('font-size', '14px')

    }

    for (let x = this.cX; x > 0; x -= scale) {

      let valX = this.getValidatedX(x)

      svg

        .append('text')

        .attr('x', x - 15)

        .attr('y', this.cY + 15)

        .text(valX)

        .attr('font-family', 'monospace')

        .attr('font-size', '14px')

    }

    for (let y = this.cY; y < height; y += scale) {

      let valY = this.getValidatedY(y)

      if (valY == 0) continue

      svg

        .append('text')

        .attr('x', this.cX + 5)

        .attr('y', y)

        .text(valY)

        .attr('font-family', 'monospace')

        .attr('font-size', '14px')

    }

    for (let y = this.cY; y > 0; y -= scale) {

      let valY = this.getValidatedY(y)

      if (valY == 0) continue

      svg

        .append('text')

        .attr('x', this.cX + 5)

        .attr('y', y)

        .text(valY)

        .attr('font-family', 'monospace')

        .attr('font-size', '14px')

    }

    this.drawAxesName()

  }

  this.getValidatedX = x => {

    return x > this.cX ? x - this.cX : (this.cX - x) \* -1

  }

  this.getValidatedY = y => {

    return y > this.cY ? this.cY - y : (y - this.cY) \* -1

  }

  this.drawAxesName = () => {

    svg

      .append('text')

      .attr('x', this.cX - 15)

      .attr('y', 15)

      .text('y')

      .attr('font-family', 'monospace')

      .attr('font-size', '20px')

      .attr('font-weight', '600')

    svg

      .append('text')

      .attr('x', width - 15)

      .attr('y', this.cY - 10)

      .text('x')

      .attr('font-family', 'monospace')

      .attr('font-size', '20px')

      .attr('font-weight', '600')

  }

}

function Board(scale) {

  this.scale = scale

  this.draw = () => {

    let grid = new Grid(this.scale)

    let axes = new Axes(this.scale)

    grid.draw()

    axes.draw()

  }

}

function animate() {

  setInterval(() => {

    particles.forEach(particle => {

      particle.draw()

    })

    let board = new Board(50)

    board.draw()

  }, 500)

}

animate()

**Висновки:** У ході лабораторної роботи було вивчено рух тіла, кинутого під кутом до горизонту, за допомогою математичних моделей та програмування на JavaScript. Використання D3.js бібліотеки спростило відображення графіків. Результатом стала успішна візуалізація траєкторії руху тіла та можливість користувача вводити початкові параметри для її моделювання.