**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**ЛАБОРАТОРНА РОБОТА № 1**

**Тема**: Моделювання кінематики прямолінійного руху

Виконав студент ІІ курсу

спеціальності «Інженерія

програмного забезпечення»

**Олексій Олексій**

**Ужгород 2024**

**Мета**: отримати уявлення про принципи побудови траєкторії руху тіла. **Завдання до роботи**:

1. та за допомогою **HTML** розмітки і каскадних таблиць стилів **CSS** побудувати траєкторію руху матеріальної точки в залежності від вхідних параметрів, які будуть задані користувачем:

а ) х0

б) у0

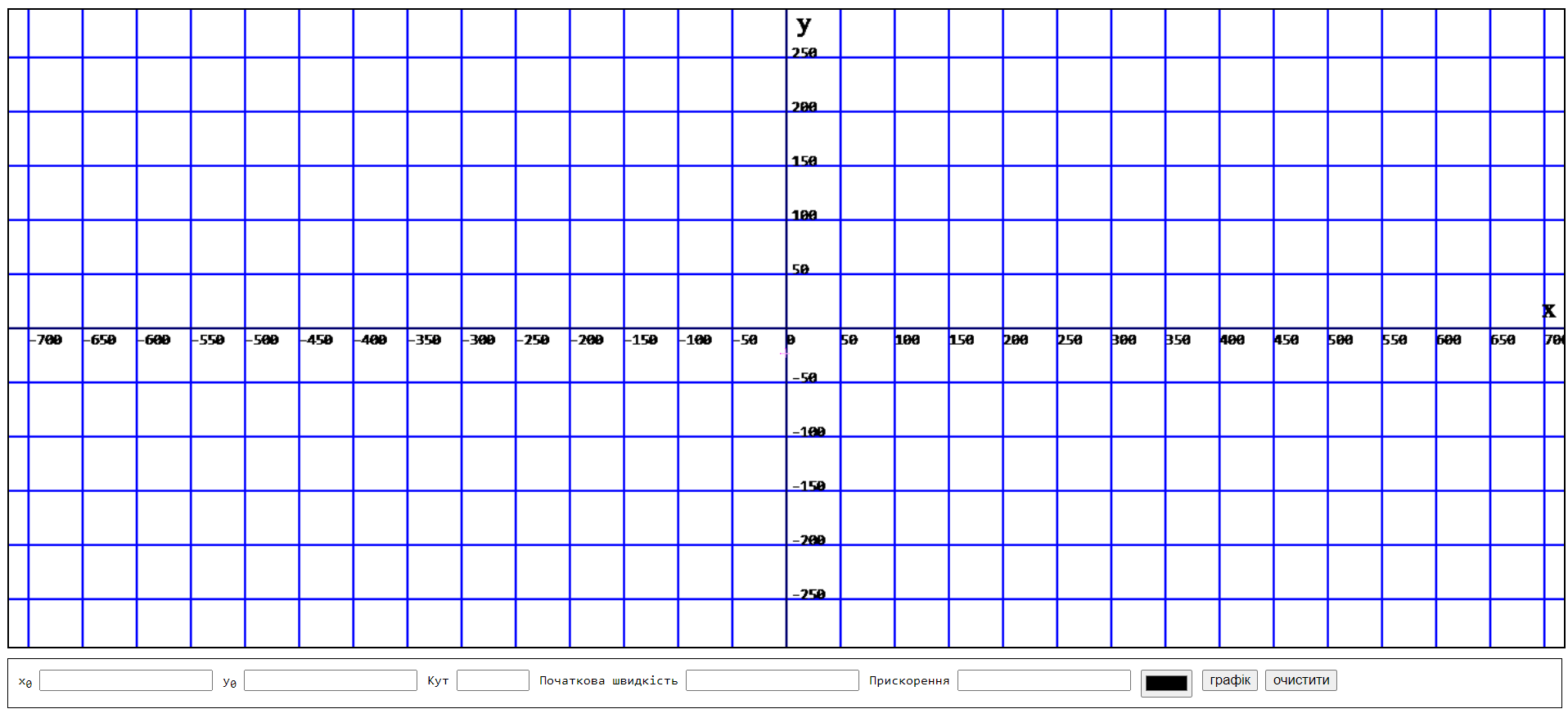
в) кут

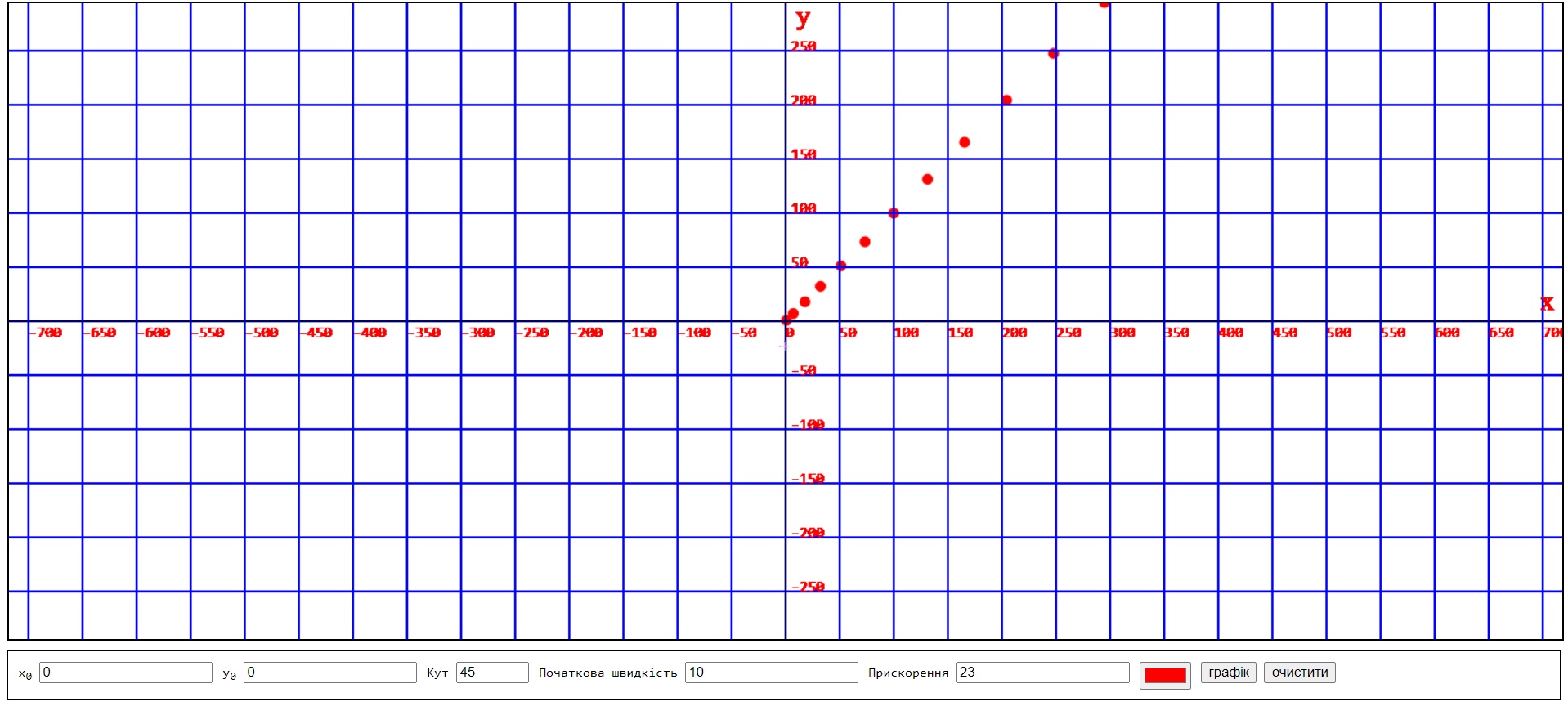
г) початкова швидкiсть

д) прискорення

1. Використовуючи попередній результата зробити рефакторинг коду, та замість використання Canvas використати бібліотеку для побудови діаграм та графіків в JavaScript [D3.JS](https://d3js.org)
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 *підпаку* із назвою, що відповідає вашому прізвищу). Якщо код завантажено на репозиторій, то у звіті потрібно вказати посилання на репозиторій.

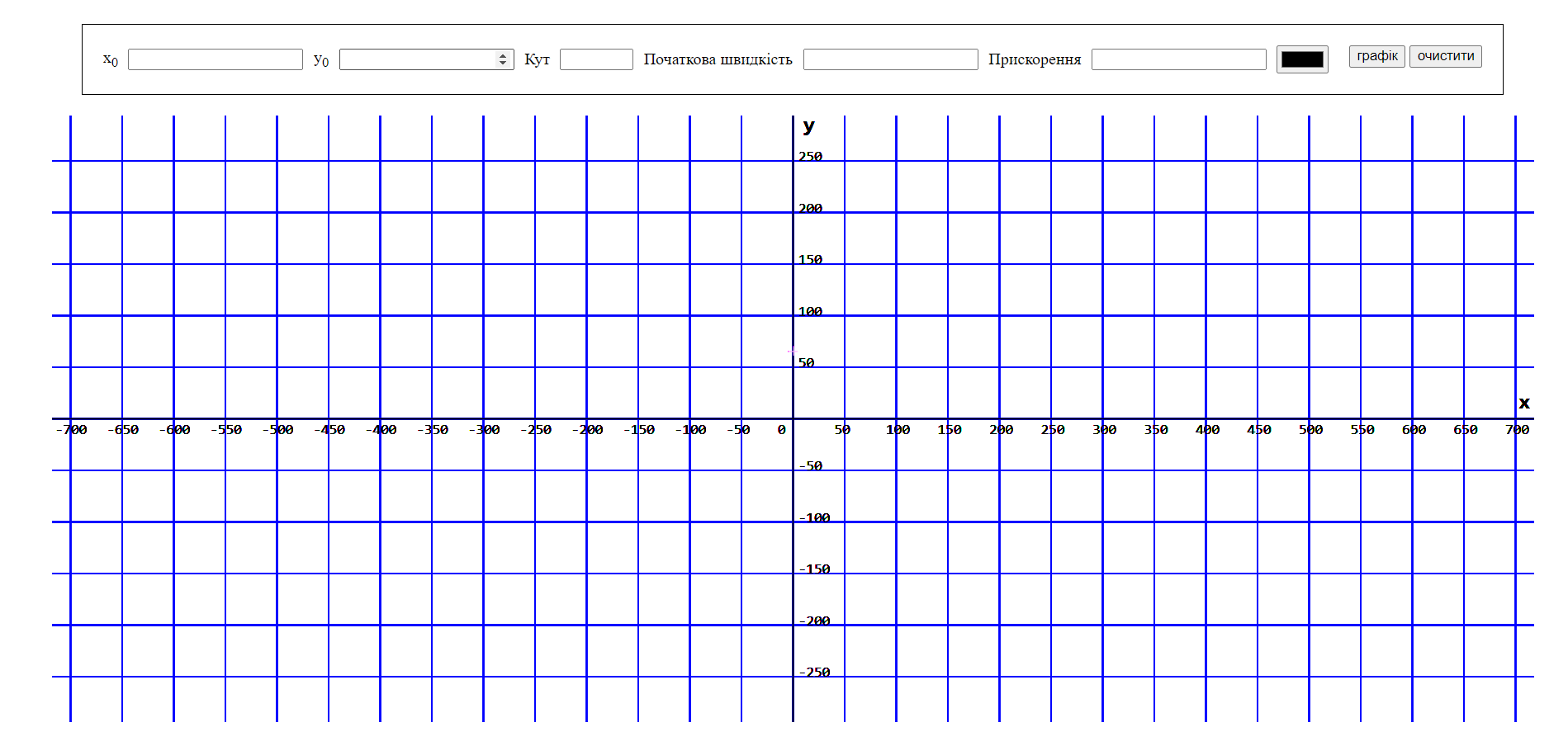
**Хід роботи**

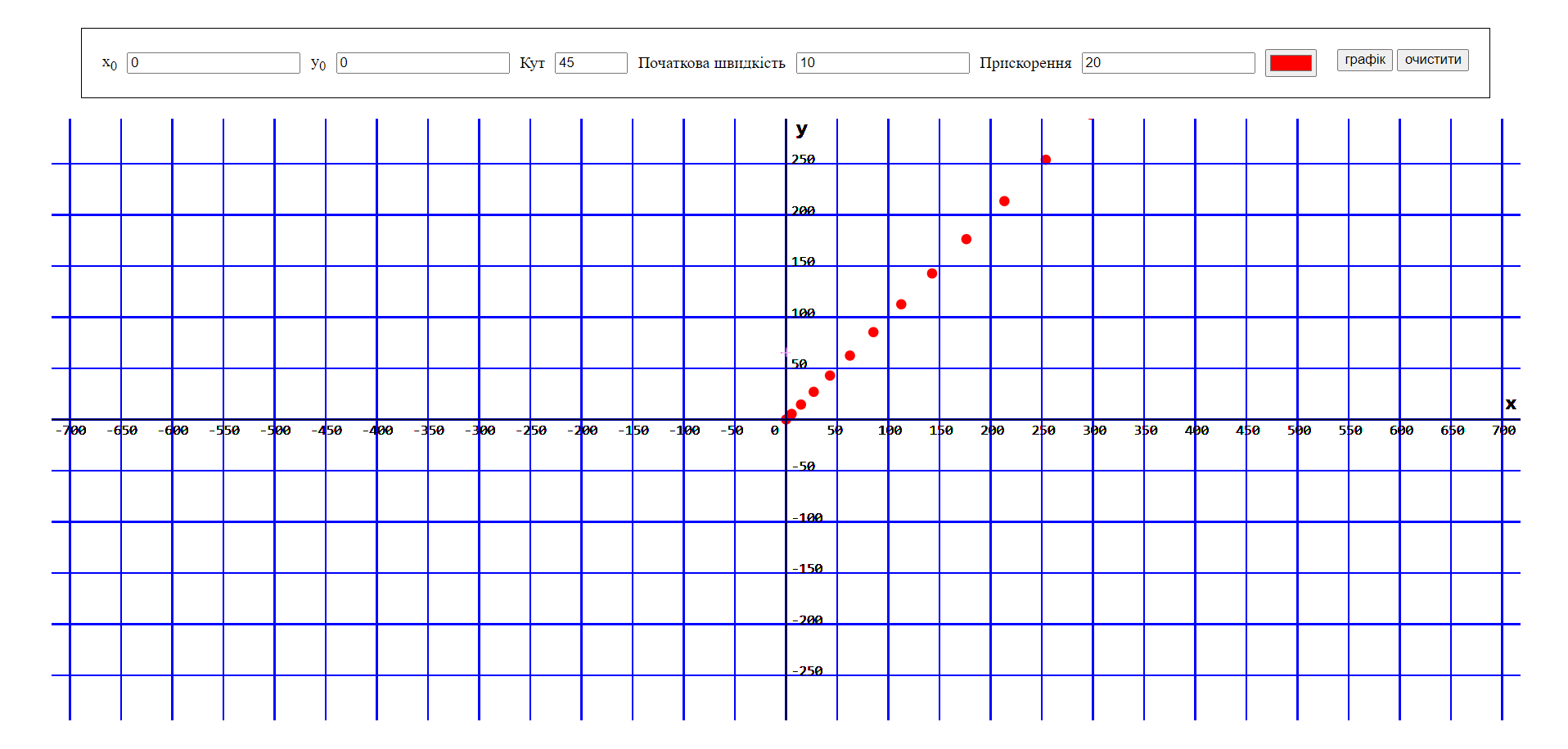




Користувач може задати початкові координати x та y, кут нахилу, початкову швидкість, прискорення та колір, після натискання кнопки «графік» відбувається моделювання графіку по заданим параметрам, після натискання кнопки «очистити» очищується масив з графіками.

Версія d3.js:





**Лістинг**

canvas:

style.css:

body {

  font-family: monospace;

  overflow: hidden;

  display: flex;

  flex-direction: column;

  gap: 10px;

}

#myCanvas {

  width: 100%;

  height: 100%;

  background-color: white;

  border: 2px solid black;

}

.parameters {

  display: flex;

  border: 1px solid black;

  padding: 10px;

  gap: 10px;

}

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>

  </head>

  <body>

    <canvas id="myCanvas"></canvas>

    <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 class="btn-container">

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

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

      </div>

    </div>

    <script src="script.js"></script>

  </body>

</html>

script.js:

const canvas = document.getElementById('myCanvas')

const c = canvas.getContext('2d')

canvas.width = window.innerWidth - 100

canvas.height = window.innerHeight - 150

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) + canvas.width / 2

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

  let validatedY =

    parseFloat(y0.value) > 0

      ? canvas.height / 2 - parseFloat(y0.value)

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

  let particle = new Particle(

    validatedX,

    validatedY,

    v0.value,

    a.value,

    angle.value,

    color.value

  )

  particles.push(particle)

}

function clearParticles() {

  particles = []

  c.clearRect(0, 0, canvas.width, canvas.height)

  c.fillStyle = 'black'

  c.fill()

}

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

  this.x0 = parseFloat(x0)

  this.y0 = parseFloat(y0)

  this.color = color

  this.v0 = parseFloat(v0)

  this.a = parseFloat(a)

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

  this.startTime = Date.now()

  this.draw = () => {

    let currentTime = Date.now()

    let elapsedTime = currentTime - this.startTime

    let t = elapsedTime / 1000

    let x =

      this.x0 +

      this.v0 \* t \* Math.cos(this.angle) +

      ((a \* t \*\* 2) / 2) \* Math.cos(this.angle)

    let y =

      this.y0 +

      this.v0 \* t \* Math.sin(this.angle) +

      ((a \* t \*\* 2) / 2) \* Math.sin(this.angle)

    c.beginPath()

    c.arc(x, y, 5, 0, Math.PI \* 2, false)

    c.fillStyle = this.color

    c.fill()

    c.closePath()

  }

}

function Grid(scale = 50) {

  this.scale = scale

  this.cX = canvas.width / 2

  this.cY = canvas.height / 2

  this.draw = () => {

    c.beginPath()

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

      c.moveTo(0, y)

      c.lineTo(canvas.width, y)

    }

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

      c.moveTo(0, y)

      c.lineTo(canvas.width, y)

    }

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

      c.moveTo(x, 0)

      c.lineTo(x, canvas.height)

    }

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

      c.moveTo(x, 0)

      c.lineTo(x, canvas.height)

    }

    c.strokeStyle = 'blue'

    c.stroke()

    c.closePath()

  }

}

function Axes(scale) {

  this.scale = scale

  this.cX = canvas.width / 2

  this.cY = canvas.height / 2

  this.draw = () => {

    c.beginPath()

    c.moveTo(this.cX, 0)

    c.lineTo(this.cX, canvas.height)

    c.moveTo(0, this.cY)

    c.lineTo(canvas.width, this.cY)

    c.strokeStyle = 'black'

    c.stroke()

    c.closePath()

    let markup = new Markup(this.scale)

    markup.draw()

  }

}

function Markup(scale) {

  this.scale = scale

  this.cX = canvas.width / 2

  this.cY = canvas.height / 2

  this.draw = () => {

    c.beginPath()

    c.font = '14px monospace'

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

      let valX = this.getValidatedX(x)

      c.fillText(valX, x - 10, this.cY + 15)

      if (valX == 0) continue

    }

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

      let valX = this.getValidatedX(x)

      if (valX == 0) continue

      c.fillText(valX, x - 15, this.cY + 15)

    }

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

      let valY = this.getValidatedY(y)

      if (valY == 0) continue

      c.fillText(valY, this.cX + 5, y - 5)

    }

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

      let valY = this.getValidatedY(y)

      if (valY == 0) continue

      c.fillText(valY, this.cX + 5, y - 5)

    }

    c.closePath()

    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 = () => {

    c.beginPath()

    c.font = '25px console'

    c.fillText('y', this.cX + 10, 20)

    c.fillText('x', canvas.width - 35, this.cY - 10)

    c.closePath()

  }

}

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()

d3:

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>

script.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,

    a.value,

    angle.value,

    color.value

  )

  particles.push(particle)

}

function clearParticles() {

  particles = []

  svg.selectAll('\*').remove()

}

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

  this.x0 = parseFloat(x0)

  this.y0 = parseFloat(y0)

  this.color = color

  this.v0 = parseFloat(v0)

  this.a = parseFloat(a)

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

  this.startTime = Date.now()

  this.draw = () => {

    let currentTime = Date.now()

    let elapsedTime = currentTime - this.startTime

    let t = elapsedTime / 1000

    let x =

      this.x0 +

      this.v0 \* t \* Math.cos(this.angle) +

      ((a \* t \*\* 2) / 2) \* Math.cos(this.angle)

    let y =

      this.y0 +

      this.v0 \* t \* Math.sin(this.angle) +

      ((a \* t \*\* 2) / 2) \* Math.sin(this.angle)

    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 + 10)

      .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 виконання лабораторної роботи було проведено моделювання кінематики прямолінійного руху за допомогою HTML, CSS, JavaScript та бібліотеки D3.js. Було створено веб-інтерфейс, який дозволяє користувачу вводити початкові параметри руху та відображати траєкторію матеріальної точки на екрані. При використанні Canvas бібліотеки було реалізовано візуалізацію за допомогою малювання в тезі «canvas», в той час як з використанням D3.js створено графічну візуалізацію з використанням SVG. Рефакторинг коду дозволив зробити його більш організованим та читабельним. В результаті отримано можливість ефективно моделювати рух тіла та візуалізувати його траєкторію за допомогою сучасних веб-технологій.