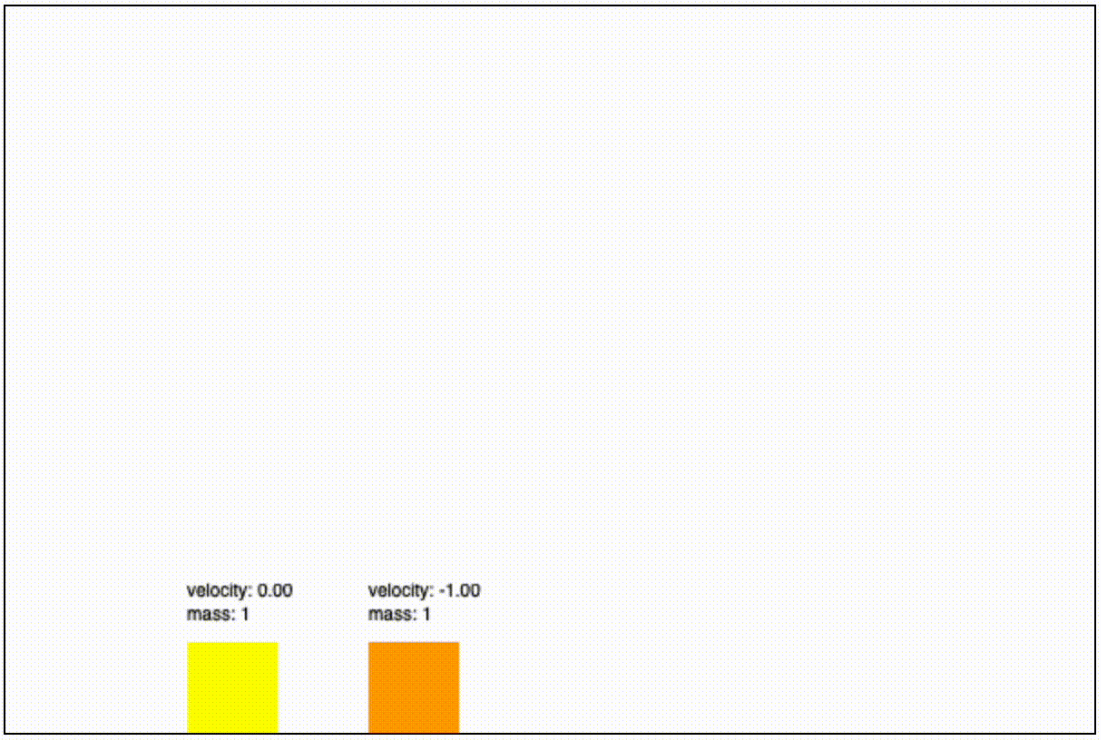
Моделирование

↗ Subject	⚛ Physics



Задание

Выполнить численное моделирование абсолютно упругого взаимодействия двух тел разной массы.

Тело массой m_2, движущееся со скоростью v_0, сталкивается с неподвижным телом массой m_1. Масса $m_1 < m_2$. Сколько попаданий N со стеной слева и телом m_2 совершит тело m_1 до полной остановки. Столкновения со стеной и телами считаются абсолютно упругими.

Результаты

m_1	m_2	N
1	1	3
1	10	31
1	100	314
1	1000	3141
1	10000	31415

Вывод

Количество столкновений между блоками и стеной (если учитывать определенные пропорции масс блоков) стремится дать нам приближенное значение числа π.

Это невероятно неоптимальный способ вычислить число π.

HTML-код:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Моделирование</title>
  <style>
    #canvas {
        border: 1px solid black;
    }

    #content {
      display: grid;
    }

    #buttons {
      padding-top: 2vh;
    }

    body {
      display: grid;
      align-items: center;
      justify-items: center;
      height: 80vh;
      width: 100vw;
    }
  </style>
</head>
```

```
<body>
  <div id="content">
    <canvas id="canvas" width="600" height="400"></canvas>
    <div id="buttons">
      <button onclick="play()">Воспроизвести</button>
      <button onclick="pause()">Пауза</button>
      <button onclick="reset()">Сбросить</button>
    </div>
  </div>
  <script src="index.js"></script>
</body>
</html>
```

Пояснение:

- HTML-код устанавливает структуру страницы.
- Элемент `canvas` используется для отображения анимации.
- В блоке `buttons` содержатся три кнопки: "Воспроизвести", "Пауза" и "Сбросить". Эти кнопки позволяют пользователю управлять анимацией.
- CSS-код определяет стили для элементов на странице, таких как canvas, content div, buttons div и body страницы.

Код JavaScript:

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

Пояснение:

- Элемент `canvas` выбирается из HTML-документа по его идентификатору.
- Метод `getContext` используется для получения 2D-контекста рендеринга canvas. Этот контекст сохраняется в переменной `ctx` и будет использоваться для рисования на canvas.

```
class Ball {
  constructor(x, y, radius, mass, velocityX, color = "") {
    this.x = x;
    this.y = y;
    this.radius = radius;
    this.mass = mass;
    this.velocityX = velocityX;
    this.color = "black";
```

```
    if (color != "") {
      this.color = color;
    }
    this.wall_bounce = 0;
    this.object_bounce = 0;
  }
```

Пояснение:

- Определяется класс `Ball` для представления объекта шара в анимации.
- Класс имеет свойства, такие как координаты `x` и `y`, радиус `radius`, массу `mass`, скорость `velocityX`, цвет `color`, `wall_bounce` и `object_bounce`.
- Метод `constructor` вызывается при создании нового экземпляра класса `Ball`. Он инициализирует свойства объекта шара на основе предоставленных аргументов.
- Если цвет не указан, устанавливается значение по умолчанию - черный.
- Свойства `wall_bounce` и `object_bounce` изначально устанавливаются в 0.

```
  update() {
    this.x += this.velocityX;

    // Проверка столкновения со стенами и изменение направления при обнаружении столкн
овения
    if (this.x - this.radius <= 0 || this.x + this.radius >= canvas.width) {
      this.velocityX = -this.velocityX;
      this.wall_bounce += 1;
    }
  }

  draw() {
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2);
    ctx.closePath();
    let old = ctx.fillStyle;
    ctx.fillStyle = this.color;
    ctx.fill();
    ctx.fillStyle = old;

    ctx.font = "10px helvetica";
    ctx.fillText(`Масса: ${this.mass}`, this.x, this.y + this.radius * 1.25)
    ctx.fillText(`Скорость: ${this.velocityX.toFixed(2)}`, this.x, this.y + this.radiu
s * 1.5)
    ctx.fillText(`x, y: (${this.x.toFixed(2)}, ${this.y.toFixed(2)})`, this.x, this.y
+ this.radius * 1.75);
  }
```

```
}
```

Пояснение:

- Метод `update` используется для обновления позиции шара.
- Он увеличивает значение координаты `x` шара на значение его скорости `velocityX`.
- Также происходит проверка столкновения с боковыми стенками canvas. Если столкновение обнаружено (когда шар касается левой или правой стенки), скорость `velocityX` меняется на противоположную, и счетчик `wall_bounce` увеличивается.
- Метод `draw` используется для рисования шара на canvas.
- Он использует метод `arc` контекста canvas для рисования окружности с указанными координатами `x`, `y`, радиусом и цветом.
- Сохраняется текущий стиль заливки в переменной `old`, и устанавливается стиль заливки цветом шара.
- Заливается окружность указанным цветом.
- Восстанавливается предыдущий стиль заливки.
- Используется метод `fillText` для отображения дополнительной информации о шаре, такой как его масса, скорость и позиция.

```
function checkCollision(ball1, ball2) {
  const dist = Math.hypot(ball1.x - ball2.x, ball1.y - ball2.y);

  if (dist <= ball1.radius + ball2.radius) {
    const v1Final = ((ball1.mass - ball2.mass) / (ball1.mass + ball2.mass)) * ball1.ve
locityX + (2 * ball2.mass / (ball1.mass + ball2.mass)) * ball2.velocityX;
    const v2Final = (2 * ball1.mass / (ball1.mass + ball2.mass)) * ball1.velocityX +
((ball2.mass - ball1.mass) / (ball1.mass + ball2.mass)) * ball2.velocityX;

    ball1.velocityX = v1Final;
    ball2.velocityX = v2Final;

    ball1.object_bounce += 1;
    ball2.object_bounce += 1;

    // Разрешение перекрытия путем перемещения шаров друг от друга
    const overlap = ball1.radius + ball2.radius - dist;
    ball1.x += (overlap / 2) * (ball1.x - ball2.x) / dist;
    ball2.x -= (overlap / 2) * (ball1.x - ball2.x) / dist;
  }
```

```
}
```

Пояснение:

- Функция `checkCollision` используется для проверки столкновений между двумя объектами-шарами.
- Она вычисляет расстояние между центрами двух шаров с помощью метода `Math.hypot`.
- Если расстояние меньше или равно сумме радиусов шаров, обнаруживается столкновение.
- Скорости шаров обновляются на основе принципов упругого столкновения.
- Счетчики `object_bounce` увеличиваются.
- Шары двигаются друг от друга, чтобы разрешить перекрытие.

```
let n = 1;
let v = -2;
let ball1 = new Ball(100, 200, 40, 1, 0, "yellow");
let ball2 = new Ball(500, 200, 50, Math.pow(10,n), v, "red");
let objects = [ball1, ball2];
let stop_animation = true;
let frame = 0;
```

Пояснение:

- Переменные `n` и `v` устанавливаются в начальные значения.
- Создаются два объекта шара `ball1` и `ball2` с использованием класса `Ball` с соответствующими начальными позициями, радиусами, массами, скоростями и цветами.
- Объекты шаров добавляются в массив `objects`.
- Переменная `stop_animation` изначально устанавливается в `true`, чтобы приостановить анимацию.
- Переменная `frame` устанавливается в `0`, чтобы отслеживать количество кадров.

```
function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
  frame++;

  objects.forEach( (object) => {
    object.update();
    object.draw();

    if (object.velocityX.toFixed(2) == 0 && frame > 200 ) stop_animation = true;
  })

  checkCollision(ball1, ball2);

  if (!stop_animation) requestAnimationFrame(animate);
}
```

Пояснение:

- Функция `animate` является основной функцией, которая обрабатывает анимацию.
- Она начинается с очистки canvas с помощью метода `clearRect`.
- Увеличивается значение переменной `frame`.
- Происходит перебор каждого объекта в массиве `objects`, вызывается метод `update` для обновления позиции объекта и метод `draw` для рисования объекта на canvas.
- Если скорость объекта становится равной нулю и количество кадров превышает 200, переменная `stop_animation` устанавливается в `true` для приостановки анимации.
- Вызывается функция `checkCollision` для проверки столкновений между шарами.
- Если переменная `stop_animation` равна `false`, вызывается метод `requestAnimationFrame` для запроса следующего кадра анимации и рекурсивного вызова функции `animate`, создавая плавный цикл анимации.

```
function play() {
  if (stop_animation) {
    stop_animation = false;
    animate();
  }
}

function pause() {
  stop_animation = true;
}
```

```
function reset() {
  stop_animation = true;
  ball1 = new Ball(100, 200, 40, 1, 0, "yellow");
  ball2 = new Ball(500, 200, 50, Math.pow(10,n), v, "red");
  objects = [ball1, ball2];
  animate();
}

animate();
```

Пояснение:

- Функция `play` вызывается при нажатии кнопки "Воспроизвести".

После построение модели, можно увидеть, что объект массой м1 совершает ровно 9 столкновений со стеной