ФИО: Товстик Мария

Номер студенческого билета: 244076

Название предмета: Практикум по программированию

Номер лабораторной работы: 3

Номер группы: ИД24-3

Дата сдачи: 25.11.2025

Название назначенной задачи: 16-Силы сопротивления

Задание:

Реализуйте систему разноразмерных частиц, имеющих одинаковую вертикальную скорость. При прохождении границы сред частицами должны испытывать сопротивление.

1. Реализуйте более физически достоверную модель сопротивления, зависящего от размера частицы.
2. Добавьте учёт инерции частиц в зависимости от их массы.

Ключевые слова:

Частицы, скорость, среда, граница, сопротивление, размер, масса

Достигнутая сложность и реализация:

- реализована физически достоверная модель падения частиц в воду

- добавлен учет инерции частиц в зависимости от массы, объема и плотности воды

- добавлена остановка и выход из анимации с помощью escape

-использованы различные дизайнерские решения (цветовая палитра, выделение шара при наведении курсора)

- возможность изменения плотности воды

1. Архитектура программы

1. Архитектура MVC-подобная:

- Model: `Simulation`, `Ball` - логика и данные

- View: `Graphics` - отображение

- Controller: `Manager` - управление процессом

2. Структура ООП

- Ball: данные шара (позиция, масса, скорость, состояние)

- Simulation: физическая модель, управление шарами

- Graphics: отрисовка на tk.Canvas

- Manager: главный цикл, связь компонентов

3. Организация кода

Разделение ответственности:

- Физика в `Simulation`

- Графика в `Graphics`

- Конфигурация в JSON

- Главный цикл в `Manager`

4. Математическое исследование

- Свободное падение: `y += v + a*t`

- Сопротивление воды: `v -= resistance`

- Отскоки: `v = -v * coefficient`

- Выталкивающая сила: зависимость от объема

5. Управление настройками

Config.json

```
{
  "width": 400,
  "height": 400,
  "colors": {
    "bg": "seashell1",
    "water": "DodgerBlue4",
    "water_activefill": "DodgerBlue4",
    "ball_outline": "pink",
    "oval_outline": "pink",
    "oval_activeclick": "DeepPink4"
  },
  "speeds": {
    "animation": 60,
    "initial_velocity": 4,
    "gravity": 0.5,
    "time_step": 0.1
  },
  "behavior": {
    "bounce_height": 30,
    "archimed_coeff" : 0.01,
    "water_resistance": 0.8,
    "bounce_coef": 0.7
  }
}
```

Использование: централизованное хранение параметров, легкая настройка без изменения кода.

Основная часть

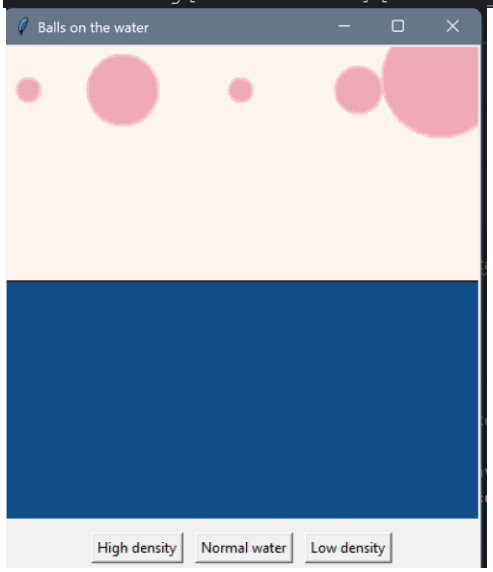
Требование: Система разноразмерных частиц с одинаковой начальной скоростью, испытывающих сопротивление при переходе сред

Решение:

- Создание класса Ball с параметрами размера и массы
- Фазы движения: падение, вода, отскоки
- Реализация сопротивление воды через коэффициент сопротивления воды

Трудности: Шары вели себя нереалистично - слишком быстро/медленно тонули, слишком сильное сопротивление воды выталкивало шары за пределы окна

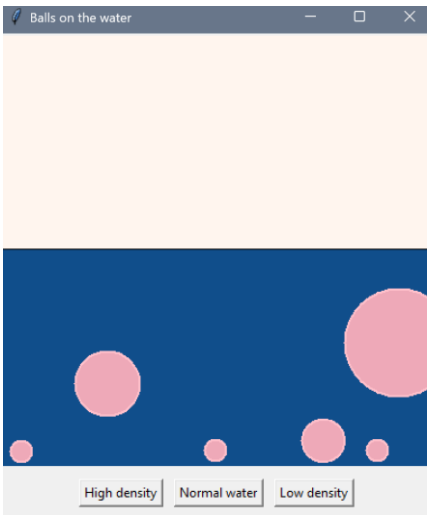
```
class Ball:
    def __init__(self, simulation, x, y, r, m, color='pink2'):
        self.config = simulation.config
        self.x = x
        self.y = y
        self.r = r #радиус шара
        self.m = m #масса
        self.v = 0  # Начинаем с нулевой скорости
        self.a = 0  # Ускорение
        self.g = self.config['speeds']['gravity'] #ускорение падения
        self.water_resistance_coef =
self.config['behavior']['water_resistance']
```



Требование: Более физически достоверная модель сопротивления от размера

Решение:

- Разные формулы сопротивления для маленьких и больших шаров
- Большие шары получают меньший коэффициент сопротивления (0.5x)



Трудности: Подбор коэффициентов для реалистичного поведения

```
class Ball:
    def __init__(self, simulation, x, y, r, m, color='pink2'):
        self.simulation = simulation# ссылка на родительскую симуляцию
        self.config = simulation.config
        self.x = x
        self.y = y
        self.r = r  #радиус шара
        self.m = m  #масса
self.g = self.config['speeds']['gravity']  #ускорение свободного падения
self.water_resistance_coef = self.config['behavior']['water_resistance']
#коэффициент сопротивления воды

…

def update_water(self, ball, current_y):
    if current_y < self.ground_level:
        ball.v += ball.g * 0.1
        if ball.r < 20:
            water_resistance = ball.water_resistance_coef * ball.m * ball.r *
self.water_density  #для маленьких шаров-полное сопротивление
        else:
            water_resistance = ball.water_resistance_coef * ball.m * ball.r *
self.water_density * 0.5  #для больших шаров-уменьшенное сопротивление
(коэффициент 0.5)
```

Требование: Учет инерции частиц в зависимости от их массы

Решение:

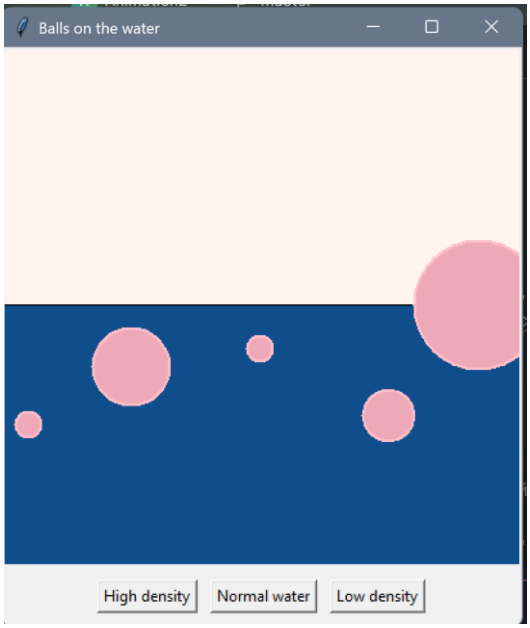
- Масса влияет на высоту отскока (тяжелые шары отскакивают выше)
- Масса учитывается в силе сопротивления воды
- Разные массы при одинаковом размере ведут себя по-разному

Трудности:

- Создание заметной разницы в поведении шаров с разной массой
- Избежание доминирования массы над другими факторами

```
def update_bounce1(self, ball):
    bounce_height = ball.m * 0.003 *
self.config['behavior']['bounce_height']#вычисляем высоту отскока на основе
```

```
массы шара
    if ball.y > self.ground_level - ball.r - bounce_height:#проверяем,
находится ли шар в фазе подъема после отскока
        ball.v = -ball.v * self.config['behavior']['bounce_coef'] * 0.5  #
Коэффициент отскока
        ball.y += ball.v#обновляем позицию шара с новой скоростью
    else:
        ball.phase = 'bounce2'#шар достиг максимальной высоты отскока -
переходим ко второй фазе
```



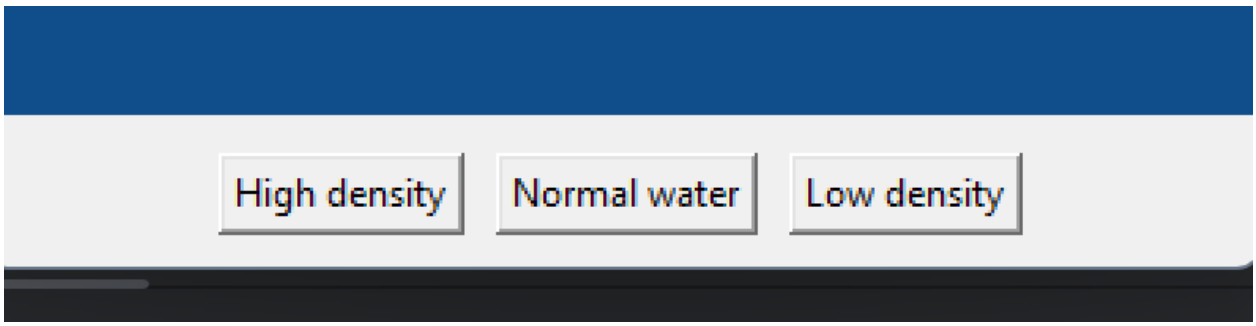
Требование: Возможность изменения параметров среды во время выполнения

Решение:

- Добавлены кнопки для изменения плотности воды
- Три варианта плотности: низкая, нормальная, высокая

```
dense_btn = tk.Button(self.button_frame, text='High density',
                      command=lambda: setattr(self.simulation,
'water_density', 3.0))
dense_btn.pack(side=tk.LEFT, padx=5)

normal_btn = tk.Button(self.button_frame, text='Normal water',
                       command=lambda: setattr(self.simulation,
'water_density',
                                               1.0))  # Кнопка сброса к
обычной воде
normal_btn.pack(side=tk.LEFT, padx=5)
light_btn = tk.Button(self.button_frame, text='Low density',
                      command=lambda: setattr(self.simulation,
'water_density', 0.1))
light_btn.pack(side=tk.LEFT, padx=5)
```



Весь код:

```
import tkinter as tk
import json


class Ball:
    def __init__(self, simulation, x, y, r, m, color='pink2'):
        self.simulation = simulation# ссылка на родительскую симуляцию
        self.config = simulation.config
        self.x = x
        self.y = y
        self.r = r  #радиус шара
        self.m = m  #масса
        self.v = 0  # Начинаем с нулевой скорости
        self.a = 0  # Ускорение
        self.g = self.config['speeds']['gravity']  #ускорение свободного
падения
        self.water_resistance_coef =
self.config['behavior']['water_resistance']  #коэффициент сопротивления воды
        self.color = color
        self.is_moving = True
        self.phase = 'falling'
        self.graphic_id = None


class Simulation:
    def __init__(self, config):
        self.config = config
        self.width = config['width']
        self.height = config['height']
        self.water_level = self.height // 2
        self.ground_level = self.height - 2
        self.water_density = 1.0
        self.balls = []

    def add_ball(self, x, y, r, m, color='pink2'):
        ball = Ball(self, x, y, r, m, color)
        self.balls.append(ball)
        return ball

    def update(self):
        for ball in self.balls:
            if not ball.is_moving:
                continue
            current_bottom = ball.y + ball.r
            if ball.phase == 'falling':#фаза падения в воздухе
                self.update_falling(ball, current_bottom)
```

```
            elif ball.phase == 'water':#фаза падения в воде
                self.update_water(ball, current_bottom)
            elif ball.phase == 'bounce1':#фаза отскока 1
                self.update_bounce1(ball)
            elif ball.phase == 'bounce2':#фаза отскока 2
                self.update_bounce2(ball, current_bottom)
            elif ball.phase == 'stopping':#остановка
                self.update_stopping(ball)

    def update_falling(self, ball, current_bottom):
        if current_bottom < self.water_level:
            #F = ma, ускорение постоянно
            ball.a = ball.g
            ball.v += ball.a * self.config['speeds']['time_step']
            ball.y += ball.v  #оновляем позицию шара с учетом новой скорости
        else:
            ball.phase = 'water'

    def update_water(self, ball, current_y):
        if current_y < self.ground_level:
            ball.v += ball.g * 0.1
            if ball.r < 20:
                water_resistance = ball.water_resistance_coef * ball.m *
ball.r * self.water_density  #для маленьких шаров-полное сопротивление
            else:
                water_resistance = ball.water_resistance_coef * ball.m *
ball.r * self.water_density * 0.5  #для больших шаров-уменьшенное
сопротивление (коэффициент 0.5)
            if ball.v > 0:
                ball.v -= water_resistance * 0.001#обновление скорости с
учетом сопротивления воды
            else:
                ball.v += water_resistance * 0.005
            ball.y += ball.v
        else:
            ball.y = self.ground_level - ball.r  #шар достиг дна-фиксируем
позицию и переходим к фазе отскока
            ball.phase = 'bounce1'

    def update_bounce1(self, ball):
        bounce_height = ball.m * 0.003 *
self.config['behavior']['bounce_height']#вычисляем высоту отскока на основе
массы шара
        if ball.y > self.ground_level - ball.r - bounce_height:#проверяем,
находится ли шар в фазе подъема после отскока
            ball.v = -ball.v * self.config['behavior']['bounce_coef'] * 0.5
# Коэффициент отскока
            ball.y += ball.v#обновляем позицию шара с новой скоростью
        else:
            ball.phase = 'bounce2'#шар достиг максимальной высоты отскока -
переходим ко второй фазе

    def update_bounce2(self, ball, current_bottom):
        if current_bottom < self.ground_level:#проверяем, не достиг ли шар
дна после отскока
            ball.v += ball.g * 0.05#добавляем гравитацию для ускорения
падения
            ball.y += ball.v
        else: #шар снова достиг дна - фиксируем позицию и переходим к
остановке
            ball.y = self.ground_level - ball.r
            ball.phase = 'stopping'

    def update_stopping(self, ball):#если шар еще не полностью опустился на
```

```
дно
        if ball.y < self.ground_level - ball.r:
            ball.y += 0.1
        else:
            ball.y = self.ground_level - ball.r#фиксируем шар точно на уровне
дна
            ball.is_moving = False

    def all_balls_stopped(self):
        '''Проверяет, все ли шары остановили движение'''
        return all(not ball.is_moving for ball in self.balls)#используем
функцию all() для проверки, что у всех шаров


class Graphics:
    def __init__(self, root, simulation):#сохраняем ссылки на главное окно и
симуляцию
        self.root = root
        self.simulation = simulation
        config = simulation.config

        self.canv = tk.Canvas(root, width=config['width'],
height=config['height'], bg=config['colors']['bg'])
        self.canv.pack()
        self.canv.create_rectangle(0, simulation.water_level, config['width']
+ 2, config['height'] + 2,
                                   fill=config['colors']['water'],
activefill=config['colors']['water_activefill'])#рисуем воду
        self.button_frame = tk.Frame(root)
        self.button_frame.pack(pady=10)#отступ сверху и снизу
        dense_btn = tk.Button(self.button_frame, text='High density',
                              command=lambda: setattr(self.simulation,
'water_density', 3.0))
        dense_btn.pack(side=tk.LEFT, padx=5)

        normal_btn = tk.Button(self.button_frame, text='Normal water',
                               command=lambda: setattr(self.simulation,
'water_density',
                                                       1.0))  # Кнопка сброса
к обычной воде
        normal_btn.pack(side=tk.LEFT, padx=5)
        light_btn = tk.Button(self.button_frame, text='Low density',
                              command=lambda: setattr(self.simulation,
'water_density', 0.1))
        light_btn.pack(side=tk.LEFT, padx=5)

        self.ball_ids = []#список для хранения идентификаторов графических
объектов шаров

    def draw(self):
        for ball_id in self.ball_ids:#удаляем предыдущие изображения шаров
            self.canv.delete(ball_id)
        self.ball_ids.clear()

        for ball in self.simulation.balls:#рисуем шар из симуляции
            x1 = ball.x - ball.r
            y1 = ball.y - ball.r
            x2 = ball.x + ball.r
            y2 = ball.y + ball.r

            ball_id = self.canv.create_oval(x1, y1, x2, y2, fill=ball.color,

outline=self.simulation.config['colors']['oval_outline'],
```

```
activefill=self.simulation.config['colors']['oval_activeclick'], width=2)
            self.ball_ids.append(ball_id)#сохраняем идентификатор для
последующего удаления


class Manager:
    def __init__(self):
        with open('config.json', 'r') as f:
            config = json.load(f)
        self.root = tk.Tk()#создаем главное окно приложения
        self.root.title('Balls on the water')
        self.simulation = Simulation(config) #создаем объект симуляции и
графики
        self.graphics = Graphics(self.root, self.simulation)

        self.simulation.add_ball(20, 20, 10, 0.1, color='pink2')
        self.simulation.add_ball(200, 20, 10, 120, color='pink2')
        self.simulation.add_ball(300, 20, 20, 3, color='pink2')
        self.simulation.add_ball(350, 20, 10, 100, color='pink2')
        self.simulation.add_ball(100, 20, 30, 10, color='pink2')
        self.simulation.add_ball(370, 10, 50, 10, color='pink2')
        self.root.bind('<Escape>', lambda e: self.escape())#привязываем
клавишу Escape для выхода
        self.running = True#флаг работы приложения
        self.game_loop()

    def game_loop(self):
        if self.running and not self.simulation.all_balls_stopped():
#продолжаем цикл пока приложение запущено и не все шары остановились
            self.simulation.update()
            self.graphics.draw()
            self.root.after(1000 //
self.simulation.config['speeds']['animation'], self.game_loop)#планируем
следующий кадр с учетом FPS из конфига
        elif not self.running:
            self.root.destroy()

    def escape(self):
        self.running = False
        self.root.destroy()#закрываем окно

    def run(self):
        self.root.mainloop()#запуск главного цикла приложения


if __name__ == '__main__':
    manager = Manager() #создаем и запускаем менеджер приложения
    manager.run()
```

Config.json:

```
{
  "width": 400,
  "height": 400,
  "colors": {
    "bg": "seashell1",
    "water": "DodgerBlue4",
    "water_activefill": "DodgerBlue4",
    "ball_outline": "pink",
    "oval_outline": "pink",
    "oval_activeclick": "DeepPink4"
```

```
  },
  "speeds": {
    "animation": 60,
    "initial_velocity": 4,
    "gravity": 0.5,
    "time_step": 0.1
  },
  "behavior": {
    "bounce_height": 30,
    "bounce_height_2": 20,
    "archimed_coeff" : 0.01,
    "water_resistance": 0.8,
    "buoyancy": 0.3,
    "bounce_coef": 0.7
  }
}
```