ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

| К.т.н. Доцент |  |  |  | В.А.Ушаков |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 |
| --- |
| **РАЗРАБОТКА ИЕРАРХИИ КЛАССОВ**  Вариант 5 |
|  |
| по курсу: КРОССПЛАТФОРМЕННОЕ ПРОГРАММИРОВАНИЕ |
|  |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР. № | 4128 |  |  |  | В. А. Воробьев |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**Цель работы:** Разработать модель линейно перемещающегося объекта. Объект должен выводиться на отдельном графическом элементе интерфейса. При нажатии левой кнопки мыши, объект должен постепенно перемещаться к точке, на которой находился указатель мыши в момент нажатия.

З**адание:**

Разработать программу на языке JAVA, использующую графический пользовательский интерфейс для управления движущимися объектами, которые представляют собой классы животных. На верхнем уровне иерархии должны располагаться абстрактные классы: "Животное", "Летающее животное", "Ходящее животное", "Водоплавающее животное", "Подземное животное". От одного из классов животных 2го уровня должны наследоваться определенные классы в соответствии с заданным вариантом.

**Предусмотреть следующие возможности:**

1. При задании координат назначения объекты не будут прерывать текущее движение и стремиться в направлении новых координат, а накапливать их и передвигаться последовательно по точкам нажатия кнопки мыши.
2. Предусмотреть возможность самостоятельного добавления пользователем заданных видов животных на графическую панель. В исходном состоянии объекты на панели отсутствуют. После нажатия кнопки (например, "+собака") на панель добавляется соответствующее животное. При добавлении, животному присваиваются характеристики (скорость, уровень относительно земли) генератором случайных чисел из допустимого интервала.

**Вариант:**

Номер варианта: 4128 + 5 = 4133

* Летающее животное - чайка
* Подземное животное - крыса
* Ходящее животное - лягушка

**Результаты работы программы:**

В ходе лабораторной работы была поставлена задача разработки программы на языке Java с графическим пользовательским интерфейсом для управления движущимися объектами, представляющими различные классы животных. Иерархия классов была организована с учетом абстрактных классов "Животное", "Летающее животное", "Ходящее животное", "Водоплавающее животное" и "Подземное животное", а также созданы конкретные классы животных на основе одного из второго уровня.

Основной акцент в работе был сделан на создании графического интерфейса с использованием библиотеки Swing. Добавлены элементы управления, позволяющие пользователю добавлять различные виды животных на графическую панель. Важной особенностью является сохранение координат нажатия кнопки мыши и последовательное движение объектов по этим точкам без прерывания текущего движения. Исходный код доступен в Приложении и на GitHub (URL - <https://github.com/vladcto/suai-labs/tree/355629db65b3539b04f89332b026364acbd0016d/5_semester/%D0%9A%D0%9F/src/lab4>).

Проведено тестирование программы, включая проверку корректности добавления объектов и их последующего движения, что подтвердило функциональность разработанной системы. Интерфейс программы представлен на рисунках 1-3.

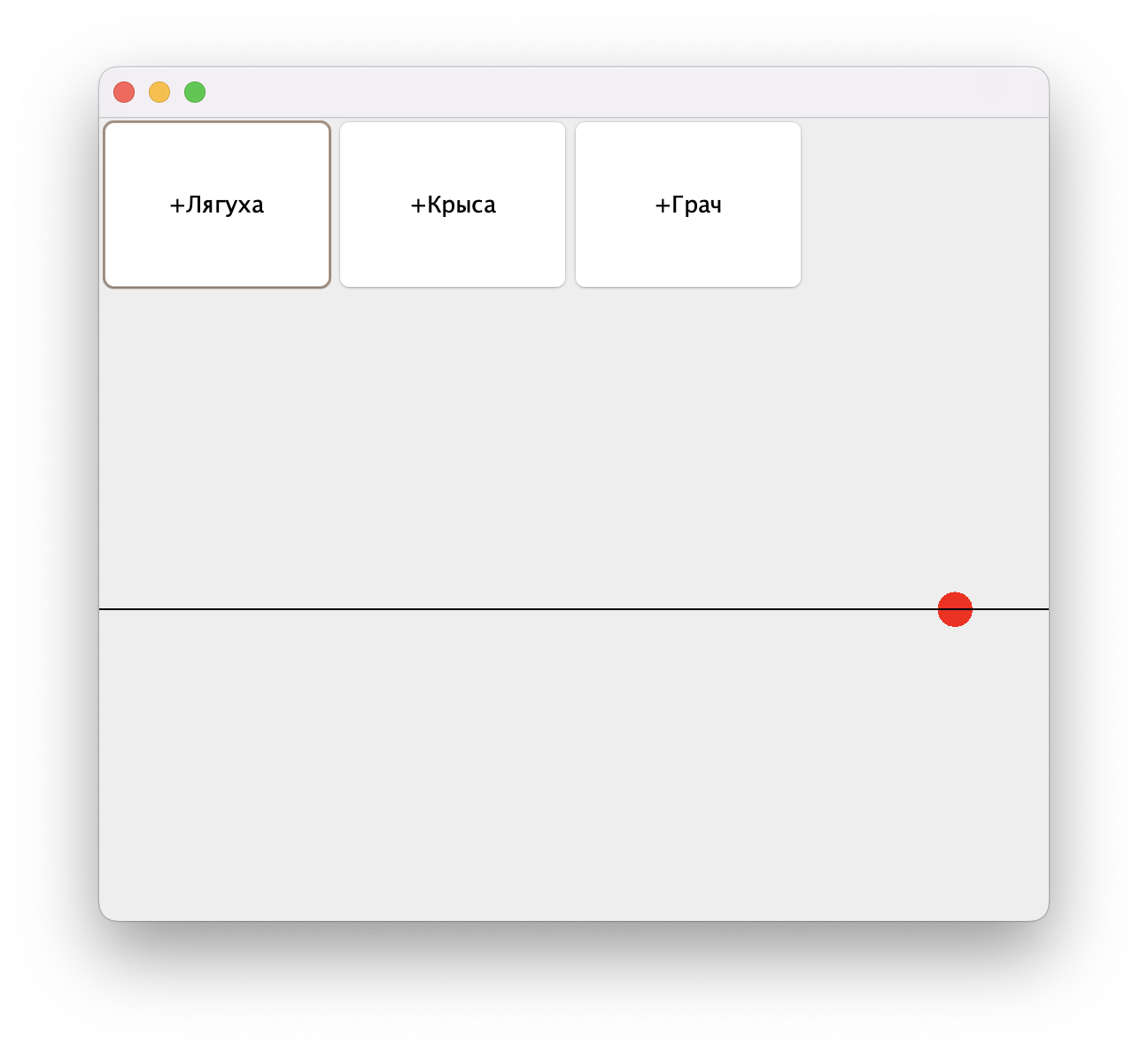


Рисунок 1 – Лягушка в программном интерфейсе

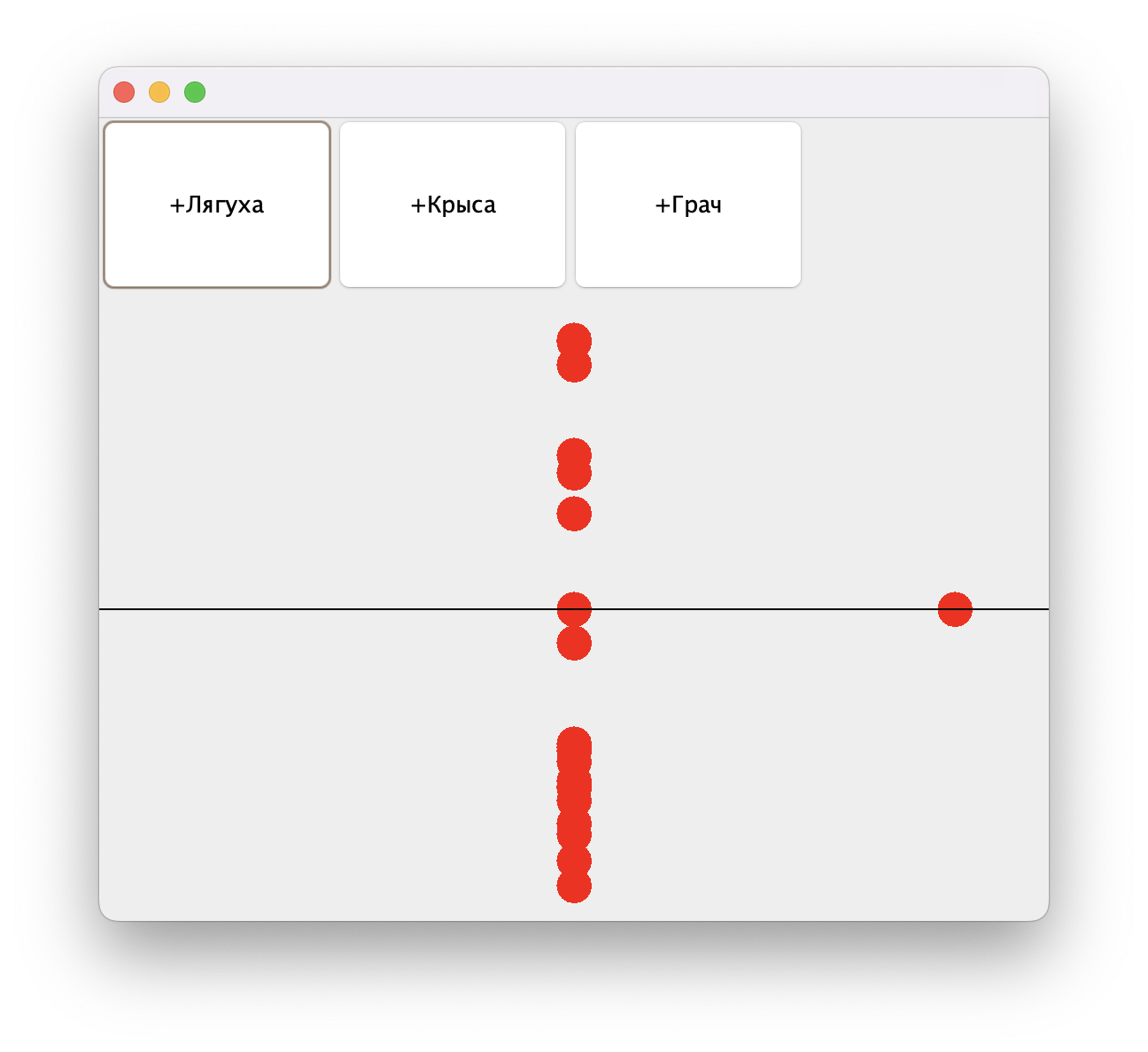


Рисунок 2 – Генерация новых животных

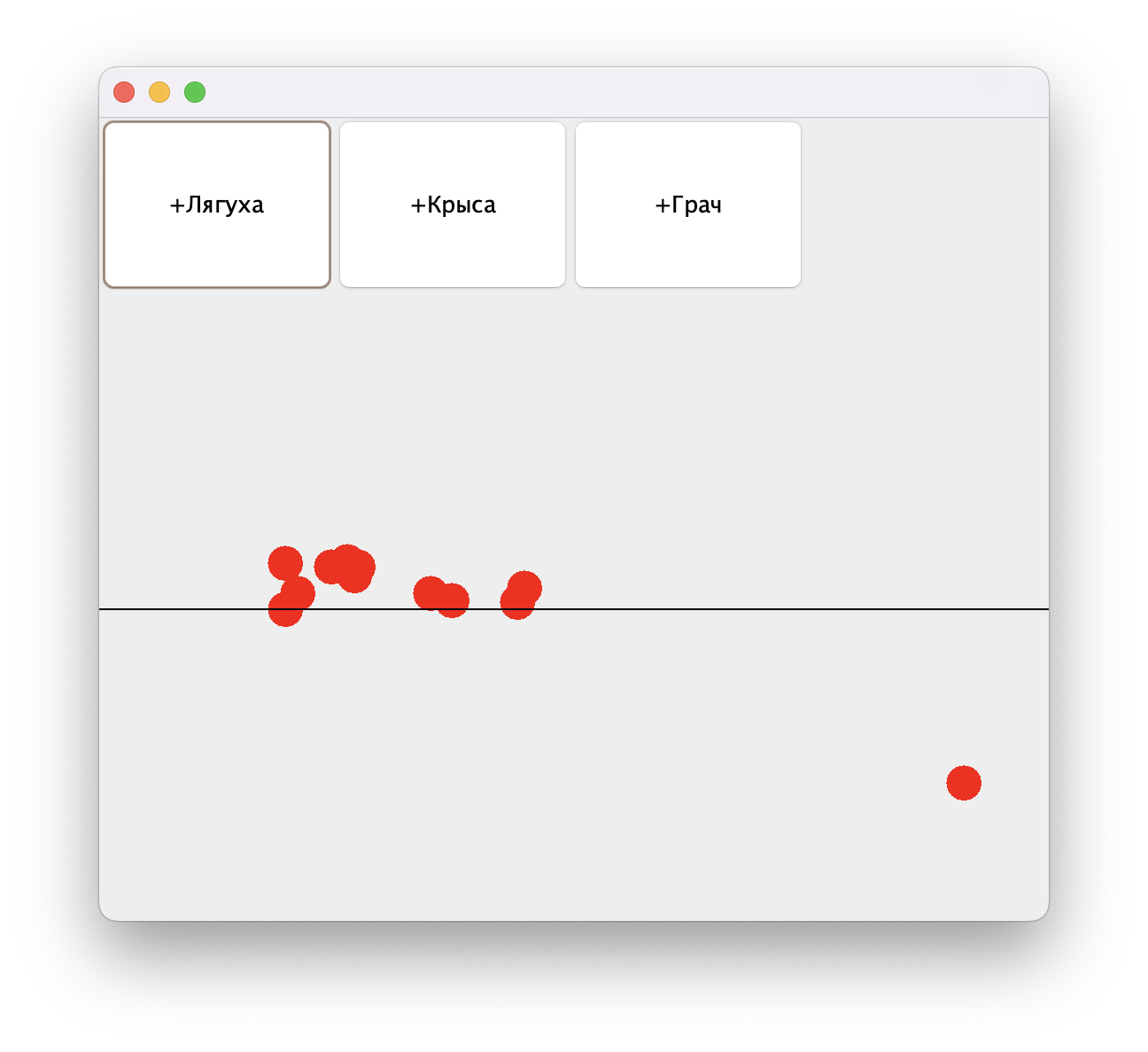


Рисунок 3 – Перемещение животных

**Вывод:**

В ходе лабораторной работы была разработана модель линейно перемещающегося объекта на языке программирования JAVA с использованием графического пользовательского интерфейса. Объект представляет собой абстрактные классы животных, включая "Летающее животное", "Подземное животное" и "Ходящее животное".

Мы реализовали функциональность перемещения объектов к точке, на которой находился указатель мыши в момент нажатия левой кнопки. Объекты накапливают координаты и последовательно передвигаются по ним, не прерывая текущего движения.

Также мы предоставили пользователю возможность добавлять на графическую панель различные виды животных, такие как чайка, крыса и лягушка. При добавлении каждому животному автоматически присваиваются характеристики, такие как скорость и уровень относительно земли, с использованием генератора случайных чисел из допустимого интервала.

Полученные навыки позволяет реализовывать адаптивные и анимированные интерфейсы.

**ПРИЛОЖЕНИЕ**

Point2.java

package lab4;

public class Point2 {

public final double x;

public final double y;

public Point2(double x, double y) {

this.x = x;

this.y = y;

}

public static Point2 one() {

return new Point2(1, 1);

}

public Point2 distance(Point2 end) {

return new Point2(end.x - x, end.y - y);

}

public float getLength() {

return (float) Math.sqrt(y \* y + x \* x);

}

public Point2 scale(double scale) {

return new Point2(x \* scale, y \* scale);

}

public Point2 add(Point2 a) {

return new Point2(x + a.x, y + a.y);

}

}

MovingAnimal.java

package lab4.animals;

import org.jetbrains.annotations.NotNull;

import org.jetbrains.annotations.Nullable;

import java.util.LinkedList;

import java.util.Queue;

import lab4.Point2;

public abstract class MovingAnimal {

private final Queue<Point2> targets = new LinkedList<>();

private long startTime = 0;

private Point2 lastPosition;

private final float speedMs;

protected MovingAnimal(Point2 startPosition, float speed) {

lastPosition = startPosition;

this.speedMs = speed;

}

public final void moveTo(Point2 globalTarget) {

final var newTarget = handleTarget(globalTarget, targets.peek());

if (newTarget != null) {

if (targets.isEmpty()) {

startTime = System.currentTimeMillis();

}

targets.add(newTarget);

}

}

public final Point2 getPosition() {

var target = targets.peek();

if (target == null) {

return lastPosition;

}

var distance = lastPosition.distance(target);

var moveTime = distance.getLength() / speedMs;

if (moveTime == 0) {

lastPosition = targets.remove();

return lastPosition;

}

var t = (System.currentTimeMillis() - startTime) / moveTime;

if (t >= 1) {

lastPosition = targets.remove();

startTime += (long) moveTime;

return getPosition();

}

return interpolatePoints(lastPosition, target, t);

}

@NotNull

protected Point2 interpolatePoints(Point2 startPoint, Point2 endPoint, double t) {

return startPoint.scale(1 - t).add(endPoint.scale(t));

}

@Nullable

protected abstract Point2 handleTarget(Point2 target, @Nullable Point2 currentTarget);

}

Rat.java

package lab4.animals;

import org.jetbrains.annotations.NotNull;

import lab4.Point2;

public class Rat extends MovingAnimal {

public Rat(Point2 startPosition, float speed) {

super(startPosition, speed);

}

@NotNull

@Override

protected Point2 interpolatePoints(Point2 startPoint, Point2 endPoint, double t) {

return super.interpolatePoints(startPoint, endPoint, t);

}

@Override

protected Point2 handleTarget(Point2 inputTarget, Point2 currentTarget) {

if (inputTarget.y > 0) {

return null;

}

return inputTarget;

}

}

Gull.java

package lab4.animals;

import org.jetbrains.annotations.NotNull;

import org.jetbrains.annotations.Nullable;

import lab4.Point2;

public class Gull extends MovingAnimal {

public Gull(Point2 startPosition, float speed) {

super(startPosition, speed);

}

@NotNull

@Override

protected Point2 interpolatePoints(Point2 startPoint, Point2 endPoint, double t) {

return super.interpolatePoints(startPoint, endPoint, Math.sqrt(t));

}

@Override

protected Point2 handleTarget(Point2 target, @Nullable Point2 currentTarget) {

if (target.y <= 0) {

return null;

}

return target;

}

}

Frog.java

package lab4.animals;

import org.jetbrains.annotations.NotNull;

import lab4.Point2;

public class Frog extends JumpingAnimal {

public Frog(Point2 startPosition, float speed, double jumpHeight) {

super(startPosition, speed, jumpHeight);

}

@Override

@NotNull

protected Point2 interpolatePoints(Point2 startPoint, Point2 endPoint, double t) {

final var tx = tx(t);

final var ty = Math.abs((Math.sin(3 \* t \* Math.PI))) \* jumpHeight;

return new Point2(super.interpolatePoints(startPoint, endPoint, tx).x, ty);

}

private double tx(double t) {

final var d3 = 1 / 3d;

final var start = ((int) (t / d3)) \* d3;

return start + Math.sqrt(t / d3 % 1) \* d3;

}

}

JumpingAnimal.java

package lab4.animals;

import org.jetbrains.annotations.Nullable;

import lab4.Point2;

abstract public class JumpingAnimal extends MovingAnimal {

protected final double jumpHeight;

protected JumpingAnimal(Point2 startPosition, float speed, double jumpHeight) {

super(startPosition, speed);

this.jumpHeight = jumpHeight;

}

@Override

protected final Point2 handleTarget(Point2 target, @Nullable Point2 currentTarget) {

if (target.y < 0 || currentTarget != null) {

return null;

}

return new Point2(target.x, 0);

}

}

Main.java

package lab4;

import javax.swing.\*;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import lab4.animals.Frog;

import lab4.animals.Gull;

import lab4.animals.MovingAnimal;

import lab4.animals.Rat;

class MovingAnimalPanel extends JPanel {

private List<MovingAnimal> animals = new ArrayList<>();

public MovingAnimalPanel() {

addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

int localX = e.getX();

int localY = e.getY();

float globalX = (float) localX / getWidth();

float globalY = (float) (1 - 2.0 \* localY / getHeight());

Point2 target = new Point2(globalX, globalY);

animals.forEach((x) -> {

x.moveTo(target);

});

repaint();

}

});

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

animals.forEach((movingAnimal -> {

Point2 currentPosition = movingAnimal.getPosition();

g.setColor(Color.RED);

int x = (int) (currentPosition.x \* getWidth());

int y = (int) ((1 - currentPosition.y) \* getHeight() / 2);

g.fillOval(x - 10, y - 10, 20, 20);

}));

g.setColor(Color.black);

g.drawLine(0, getHeight() / 2, getWidth(), getHeight() / 2);

}

public void addAnimal(MovingAnimal animal) {

animals.add(animal);

}

}

class MovingAnimalFrame extends JFrame {

MovingAnimalPanel panel;

JButton frogButton = new JButton("+Лягуха");

JButton ratButton = new JButton("+Крыса");

JButton gullButton = new JButton("+Грач");

public MovingAnimalFrame() {

setLayout(null);

setSize(400, 400);

panel = new MovingAnimalPanel();

panel.setBounds(0, 100, getWidth(), getHeight() - 125);

add(panel);

// Buttons

frogButton.setBounds(0, 0, getWidth() / 3, 100);

frogButton.addActionListener((x) -> panel.addAnimal(Generator.generateFrog()));

add(frogButton);

ratButton.setBounds(getWidth() / 3, 0, getWidth() / 3, 100);

ratButton.addActionListener((x) -> panel.addAnimal(Generator.generateRat()));

add(ratButton);

gullButton.setBounds(getWidth() / 3 \* 2, 0, getWidth() / 3, 100);

gullButton.addActionListener((x) -> panel.addAnimal(Generator.generateGull()));

add(gullButton);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

}

public void repaintPanel() {

panel.setBounds(0, 100, getWidth(), getHeight() - 125);

panel.repaint();

}

}

public class Main {

public static void main(String[] args) {

MovingAnimalFrame frame = new MovingAnimalFrame();

Timer timer = new Timer(16, (x) -> frame.repaintPanel());

timer.setRepeats(true);

timer.start();

}

}

class Generator {

static final float SCALE = 0.001F;

static final Random RANDOM = new Random();

static Frog generateFrog() {

return new Frog(

new Point2(0.5, 0),

(1.5F \* RANDOM.nextFloat() + 0.5F) \* SCALE,

(RANDOM.nextFloat() + 1F) \* SCALE \* 100

);

}

static Gull generateGull() {

return new Gull(

new Point2(0.5, 0.8 \* RANDOM.nextFloat() + 0.1),

(4F \* RANDOM.nextFloat() + 10F) \* SCALE

);

}

static Rat generateRat() {

return new Rat(

new Point2(0.5, -0.8 \* RANDOM.nextFloat() - 0.1),

(1.5F \* RANDOM.nextFloat() + .5F) \* SCALE

);

}

}