Michał Ducher 48410

Opis projektu zaliczeniowego

Program to prosta symulacja przedstawiająca działanie algorytmu genetycznego. Na planszy znajduje się meta(niebieska kropka), do której czarne kropki muszą nauczyć się dojść.

Kropki łączone są w populację o określonej wielkości i uczą się przez określoną na początku ilość epok.

Wykorzystane rozwiązania:

Blok Try-Catch w razie gdy użytkownik poda dane których nie da się rzutować na INT, w miejscu podawania takich parametrów jak wielkość populacji, ilość epok, czy liczba FPS

```
if(e.getSource() == startButton){
    try{
        epochs = Integer.parseInt(epochsTextField.getText());
        population = Integer.parseInt(populationSizeTextField.getText());
        fps = Integer.parseInt(fpsTextField.getText());
    new GameFrame(population, epochs, fps);
    }catch(NumberFormatException n){
        System.out.println("Niepoprawne dane! ");
    }
```

niestety komunikat jest wyświetlany w konsoli za co przepraszam :(

Dziedziczenie klas

```
public class GamePanel1 extends JPanel{
```

```
public class Population extends Dot {
```

"Słuchacze"

```
public class LaunchPage implements ActionListener {
```

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == startButton){
        try{
            epochs = Integer.parseInt(epochsTextField.getText());
            population = Integer.parseInt(populationSizeTextField.getText());
            fps = Integer.parseInt(fpsTextField.getText());
        new GameFrame(population, epochs, fps);
        }catch(NumberFormatException n){
            System.out.println("Niepoprawne dane! ");
        }
```

Klasa 'GamePanel1', tworząca obszar do symulacji - główne funkcje:

-Konstruktor (tylko fragment bo jest bardzo obszerny)

```
public GamePanel1(int p, int e, int f){
    epochsLabel.setBounds( x: 660, y: 0, width: 200, height: 60);
    resultLabel1.setBounds( x: 360, y: 250, width: 200, height: 200);
    resultLabel2.setBounds( x: 660, y: 15, width: 240, height: 60);

    epochsLabel.setFont(new Font( name: "Serif", Font.PLAIN,  size: 14));
    resultLabel1.setFont(new Font( name: "Serif", Font.PLAIN,  size: 50));
    resultLabel2.setFont(new Font( name: "Serif", Font.PLAIN,  size: 14));


    epochsLabel.setText("Epoka 0");
    resultLabel1.setText("");
    resultLabel2.setText("Skuteczność 0%");
```

-Funkcja rysująca stan początkowy symulacji(wywoływana tylko raz na początku)

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    test.draw(g);
    g.setColor(Color.blue);
    g.fillOval((int)finish.x, (int)finish.y,  width: 10,  height: 10);
}
```

-Główna pętla gry (przeniesiona do osobnego wątku). Zawiera algorytm genetyczny składający się z trzech funkcji:

1) fitnessFun()

2) selection()

3) mutate()

```
private void startGameLoop() {
    new Thread(() -> {
        while (true) {
            updateGameState();

            if(test.allDead()){
                test.fitnessFun(finish);
                test.selection();
                test.mutate();
                epochsLabel.setText("Epoka " + test.Generation);
                resultLabel2.setText("Skuetczność " + test.calculateResult() + "%");
                if(epochs > test.Generation)
                    Dot.help = 0;
            }
            repaint();
            try {
                Thread.sleep(DELAY);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if(test.Generation == epochs) {
                resultLabel1.setText(test.calculateResult() + "%");
                break;
            }


        }
    }).start();
}
```

Klasa 'Dot', tworząca obiekt kropki - główne funkcje:

-Konstruktor

```
Dot(){
    brain = new Brain( size: 1000);
    pos = new PVector( x: 400,  y: 750);
    vel = new PVector( x: 0,  y: 0);
    acc = new PVector( x: 0,  y: 0);
}
```

-Funkcja rysująca kropkę 'draw()'

```
public void draw(Graphics g) {

    if(!reachedGoal){
        g.setColor(Color.black);
        g.fillOval((int)pos.x, (int)pos.y,  width: 10,  height: 10);
    }else{
        g.setColor(Color.green);
        g.fillOval( x: (int)pos.x-5,  y: (int)pos.y-5,  width: 20,  height: 20);
    }
}
```

-Funkcja która zmienia położenie kropki 'move()'

```
void move() {

    if (brain.moves.length > brain.step) {
        acc = brain.moves[brain.step];
        brain.step++;
    } else
        dead = true;

    vel.add(acc);
    vel.limit( max: 5);
    pos.add(vel);
}
```

-Funkcja aktualizująca stan symulacji 'update()'

```
void update(){
    if(!dead && !reachedGoal){
        move();
        if(pos.x < 1 || pos.y < 1 || pos.x > 790 || pos.y > 790)
            dead = true;
        else if(this.pos.distance(new PVector( x: 400,  y: 20)) < 1){
            help++;
            reachedGoal = true;
        }
    }
}
```

-Funkcja dopasowania określająca progress danej kropki 'fitnessFun()' (bierze pod uwagę odległość od mety oraz liczbę ruchów wykonanych w trakcie 'życia')

```
void fitnessFun(PVector finish){
    float dist = this.pos.distance(finish);
    fitness = 1/(dist*dist) + 1/(this.brain.step);
}
```

-Funkcja klonująca najlepsze kropki do następnej populacji

```
Dot gimmieBaby(){
    Dot baby = new Dot();
    baby.brain = brain.clone();
    return baby;
}
```

Klasa 'PVector', tworząca wektory wg. których porusza się kropka - główne funkcje:

- Konstruktor

```
public PVector(float x, float y) {
    this.x = x;
    this.y = y;
    this.z = 0;
}
```

-Funkcje wykonujące podstawowe operacje na wektorach

```
public PVector add(PVector v) {
    x += v.x;
    y += v.y;
    z += v.z;
    return this;
}

no usages
public PVector sub(PVector v) {
    x -= v.x;
    y -= v.y;
    z -= v.z;
    return this;
}

1 usage
public PVector mult(float n) {
    x *= n;
    y *= n;
    z *= n;
    return this;
}

1 usage
public PVector div(float n) {
    x /= n;
    y /= n;
    z /= n;
    return this;
}
```

-Funckja licząca długość wektora 'distance()'

```
public float distance(PVector v) {
    double dx = this.x - v.x;
    double dy = this.y - v.y;

    return (float)Math.sqrt(dx * dx + dy * dy);
}
```

-Funkcja przekształcająca kąt na wektor 'fromAngle()'

```
public static PVector fromAngle(float angle) {
    float x = (float) Math.cos(angle);
    float y = (float) Math.sin(angle);
    return new PVector(x, y);
}
```

Klasa 'Brain', tworząca obiekt przechowujący ruchy danej kropki - główne funkcje:

-Kostruktor

```
Brain(int size){
    moves = new PVector[size];
    setRandomMoves();
}
```

oraz funkcja 'setRandomMoves()' wywoływana tylko raz na początku

```
void setRandomMoves(){
    for(int i = 0; i < moves.length; i++){
        float randomAngle = random.nextFloat() * 2 * (float)Math.PI;
        moves[i] = PVector.fromAngle(randomAngle);
    }
}
```

-Funkcja mutująca móżg najlepszych kropek do nowej populacji 'mutate()'

```
public void mutate(){
    float mutautionRate = (float)0.001;
    for(int i = 0; i < moves.length; i++){
        float rand = random.nextFloat( bound: 1);

        if(rand < mutautionRate){
            float randomAngle = random.nextFloat() * 2 * (float)Math.PI;
            moves[i] = PVector.fromAngle(randomAngle);
        }
    }
}
```

Klasa 'Population', tworząca populację kropek - główne funkcje:

-Konstruktor

```
Population(int size){
    Generation = 0;
    dots = new Dot[size];
    for(int i = 0; i < size; i++){
        dots[i] = new Dot();
    }
}
```

-Funkcja sprawdzająca stan kropek w całej populacji 'allDead()', zwraca true, gdy wszystkie są martwe lub dotarły do celu i można zacząc nową generację

```
boolean allDead(){
    for(int i = 0; i < dots.length; i++){
        if(!dots[i].dead && !dots[i].reachedGoal){
            return false;
        }
    }
    Generation++;
    return true;
}
```

-Funkcja selekcjonująca populacje na podstawie wyniku funkcji dopasowania 'selection()'

```
void selection(){
    Dot[] newGen = new Dot[dots.length];
    calculateFitenessSum();

    for(int i = 0; i < newGen.length; i++){
        Dot parent = selectParent();

        newGen[i] = parent.gimmieBaby();

    }
    dots = newGen.clone();
}
```

-Funkcja wybierająca kropki do mutacji

```
Dot selectParent(){
    Random random = new Random();
    float rand = random.nextFloat(fitnessSum);

    float runningSum = 0;

    for(int i = 0; i < dots.length; i++){
        runningSum += dots[i].fitness;
        if( runningSum > rand ){
            return dots[i];
        }
    }
     return null;
}
```
