HW9

1분반

2024년 12월 4일

32211792

박재홍

BoidState 인터페이스)

```
src > J BoidState.java > ...
1   import java.util.List;
2
3   public interface BoidState {
4       void applyBehavior(Boid boid, List<Boid> boids);
5   }
6
```

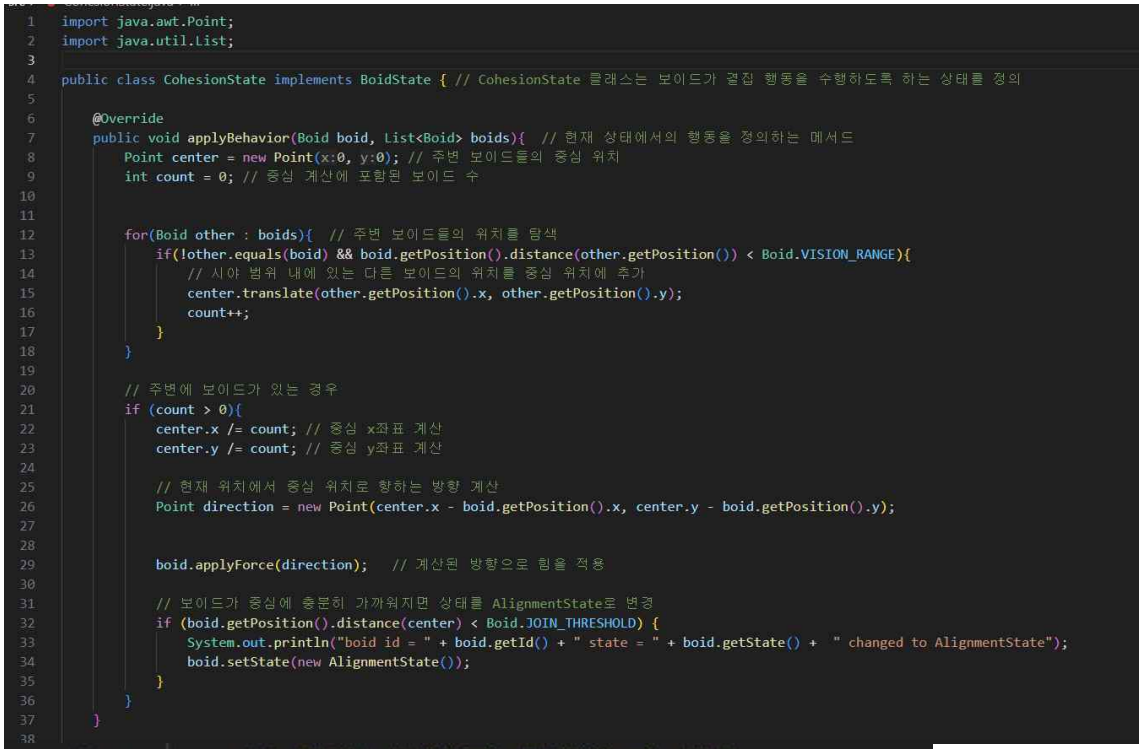
AlignmentState 클래스)

```
1   import java.awt.Point;
2   import java.util.List;
3
4   public class AlignmentState implements BoidState {
5
6
7       @Override
8       public void applyBehavior(Boid boid, List<Boid> boids) {  // applyBehavior 메서드는 Boid의 상태를 결정하는 주요 동작을 수행
9
10          Point avgVelocity = new Point(x:0, y:0); // avgVelocity는 주어진 영역 내의 다른 Boid들의 속도의 평균을 계산
11          int count = 0; // 다른 Boid들과의 거리가 VISION_RANGE 내에 있는 Boid들의 수를 셈
12
13          for (Boid other : boids) {  // 모든 Boid들을 순회하면서, 각 Boid와의 거리가 VISION_RANGE 내에 있는지 확인
14              // 자기 자신과 다른 Boid들만 고려 (다른 Boid와의 거리만 계산)
15              if (!other.equals(boid) && boid.getPosition().distance(other.getPosition()) < Boid.VISION_RANGE) {
16                  // 다른 Boid의 속도를 평균 속도에 추가
17                  avgVelocity.translate(other.getVelocity().x, other.getVelocity().y);
18                  count++;  // 평균을 계산하기 위한 Boid들의 수를 셈
19              }
20          }
21
22
23          if (count > 0) {    // 만약 count가 0보다 크면, 즉 다른 Boid들이 존재하고 그들의 속도를 고려한 평균을 계산했다면
24              avgVelocity.x /= count;  // 평균 속도 계산
25              avgVelocity.y /= count;
26
27              // 현재 Boid의 속도와 평균 속도 차이를 보정하는 힘을 적용
28              boid.applyForce(new Point(avgVelocity.x - boid.getVelocity().x, avgVelocity.y - boid.getVelocity().y));
29
30              // Boid가 다른 Boid와 너무 가까워져서 분리해야 하는 경우 SeparationState로 상태 전환
31              if (boid.getPosition().distance(boid.getPosition()) < Boid.SEPARATION_DISTANCE) {
32                  System.out.println("boid id = " + boid.getId() + " state = " + boid.getState() +  " changed to SeparationState");
33                  boid.setState(new SeparationState());  // SeparationState로 상태 변경
34              }
35          }
36      }
```

```
38      // 현재 상태의 이름을 반환하는 메서드, 디버깅 및 상태 추적에 사용
39      @Override
40      public String toString() {
41          return "AlignmentState";  // 상태를 "AlignmentState"로 표시
42      }
43  }
44
```

AligmentState 클래스에서 applyBehavior 메소드는 Boid의 상태를 결정하는 주요 동작을 수행한다. avgVelocity는 주어진 영역 내의 다른 boid들의 속도의 평균을 계산하고 count는 다른 boid들과의 거리가 VISION_RANGE 내에 있는 boid들의 수를 센다. 그 후 for-each문을 통해 모든 boid들을 순회하면서 각 boid들과의 거리가 VISION_RANGE 내에 있는지 확인한다. count가 0보다 크게되면 현재 boid의 속도와 평균속도차이를 보정하는 힘을 적용시킨다.boid가 다른 boid와 너무 가까워져서 분리해야한다면 separationstate전환

CohesionState 클래스)

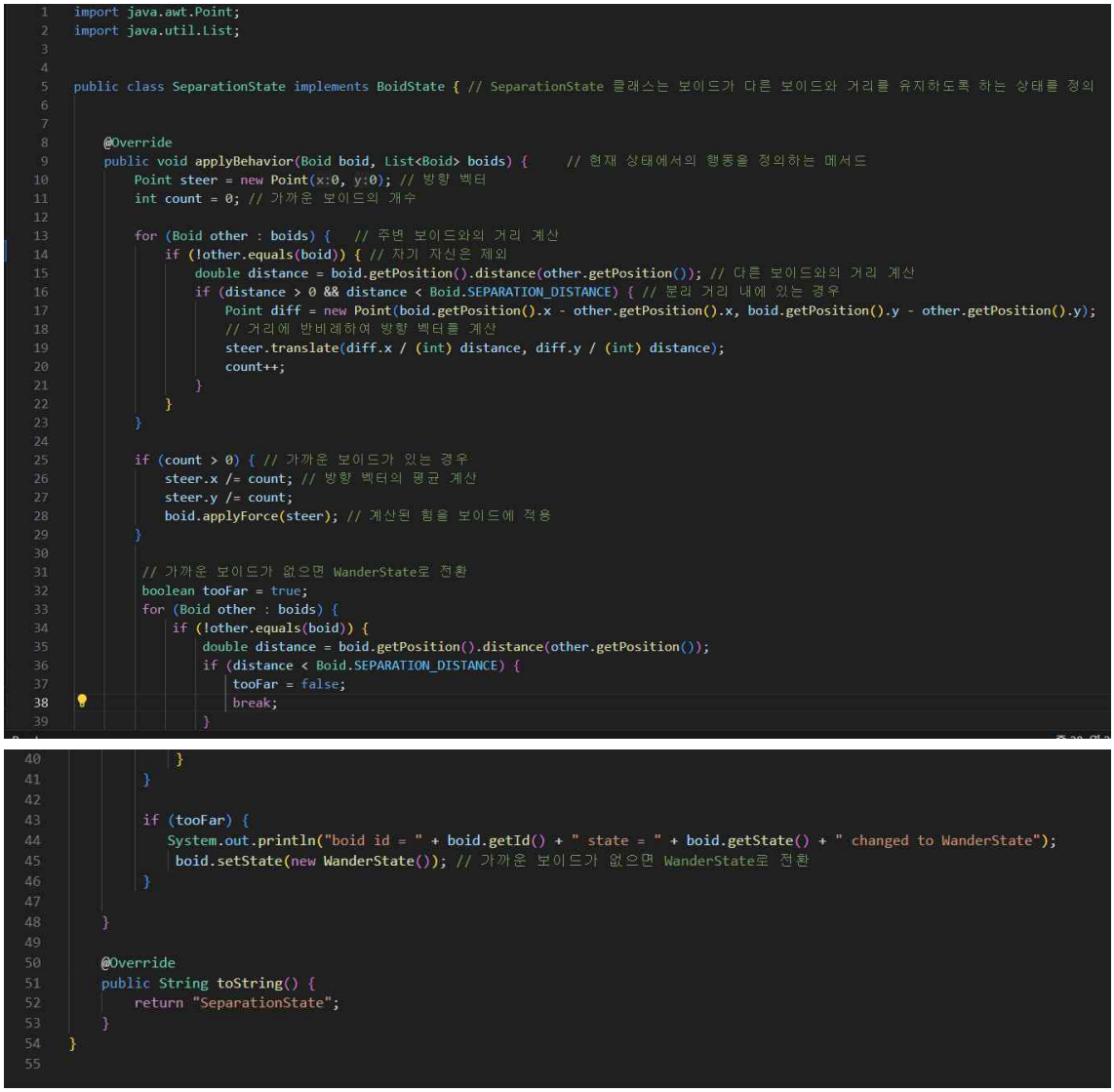


```java
import java.awt.Point;
import java.util.List;

public class CohesionState implements BoidState { // CohesionState 클래스는 보이드가 결집 행동을 수행하도록 하는 상태를 정의

    @Override
    public void applyBehavior(Boid boid, List<Boid> boids){  // 현재 상태에서의 행동을 정의하는 메서드
        Point center = new Point(x:0, y:0); // 주변 보이드들의 중심 위치
        int count = 0; // 중심 계산에 포함된 보이드 수


        for(Boid other : boids){  // 주변 보이드들의 위치를 탐색
            if(!other.equals(boid) && boid.getPosition().distance(other.getPosition()) < Boid.VISION_RANGE){
                // 시야 범위 내에 있는 다른 보이드의 위치를 중심 위치에 추가
                center.translate(other.getPosition().x, other.getPosition().y);
                count++;
            }
        }

        // 주변에 보이드가 있는 경우
        if (count > 0){
            center.x /= count; // 중심 x좌표 계산
            center.y /= count; // 중심 y좌표 계산

            // 현재 위치에서 중심 위치로 향하는 방향 계산
            Point direction = new Point(center.x - boid.getPosition().x, center.y - boid.getPosition().y);


            boid.applyForce(direction);   // 계산된 방향으로 힘을 적용

            // 보이드가 중심에 충분히 가까워지면 상태를 AlignmentState로 변경
            if (boid.getPosition().distance(center) < Boid.JOIN_THRESHOLD) {
                System.out.println("boid id = " + boid.getId() + " state = " + boid.getState() +  " changed to AlignmentState");
                boid.setState(new AlignmentState());
            }
        }
    }
```

```java
    // 현재 상태를 문자열로 반환
    @Override
    public String toString(){
        return "CohesionState";
    }
}
```

CohesionState 클래스는 보이드가 결집행동을 수행하도록 하는 상태를 정의하는 클래스이다. applyBehavior 메소드에서는 현재 상태에서의 행동을 정의한다. 주변 보이드들의 중심위치와 중심계산에 포함된 보이드 수를 정의 해주었고 for-each문을 통해 주변 보이드들의 위치를 탐색하고 시야 범위 내에 있는 다른 보이드들의 위치를 중심 위치에 추가시켰다. 그 후 if문을 통해 주변에 보이드가 있는 경우 중심 x좌표 계산, 중심 y좌표를 계산했고 현재 위치에서 중심 위치로 가는 방향을 계산해주었다. 그 후 applyforce를 통해 계산된 방향으로 힘을 적용시켰고, 보이드가 중심에 충분히 가까워지면 상태를 AlignmentState로 변경시켰다.

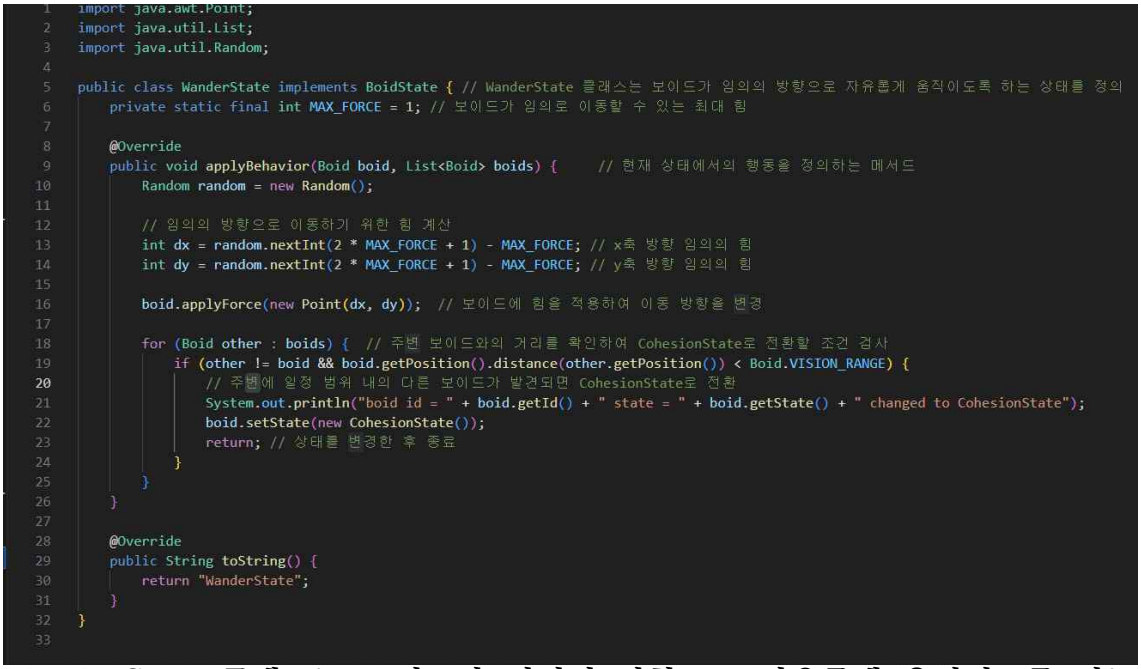
SeparationState 클래스)



```java
import java.awt.Point;
import java.util.List;


public class SeparationState implements BoidState { // SeparationState 클래스는 보이드가 다른 보이드와 거리를 유지하도록 하는 상태를 정의


    @Override
    public void applyBehavior(Boid boid, List<Boid> boids) {     // 현재 상태에서의 행동을 정의하는 메서드
        Point steer = new Point(x:0, y:0); // 방향 벡터
        int count = 0; // 가까운 보이드의 개수

        for (Boid other : boids) {   // 주변 보이드와의 거리 계산
            if (!other.equals(boid)) { // 자기 자신은 제외
                double distance = boid.getPosition().distance(other.getPosition()); // 다른 보이드와의 거리 계산
                if (distance > 0 && distance < Boid.SEPARATION_DISTANCE) { // 분리 거리 내에 있는 경우
                    Point diff = new Point(boid.getPosition().x - other.getPosition().x, boid.getPosition().y - other.getPosition().y);
                    // 거리에 반비례하여 방향 벡터를 계산
                    steer.translate(diff.x / (int) distance, diff.y / (int) distance);
                    count++;
                }
            }
        }

        if (count > 0) { // 가까운 보이드가 있는 경우
            steer.x /= count; // 방향 벡터의 평균 계산
            steer.y /= count;
            boid.applyForce(steer); // 계산된 힘을 보이드에 적용
        }

        // 가까운 보이드가 없으면 WanderState로 전환
        boolean tooFar = true;
        for (Boid other : boids) {
            if (!other.equals(boid)) {
                double distance = boid.getPosition().distance(other.getPosition());
                if (distance < Boid.SEPARATION_DISTANCE) {
                    tooFar = false;
                    break;
                }
            }
        }

        if (tooFar) {
            System.out.println("boid id = " + boid.getId() + " state = " + boid.getState() + " changed to WanderState");
            boid.setState(new WanderState()); // 가까운 보이드가 없으면 WanderState로 전환
        }

    }

    @Override
    public String toString() {
        return "SeparationState";
    }
}

```

separationState 클래스는 보이드가 다른 보이드와 거리를 유지하도록 하는 상태를 정의하는 클래스이다. applyBehavior 메소드는 현재 상태에서의 행동을 정의하는 메소드이며 방향벡터를 정의, 가까운 보이드의 개수를 정의했고 for-each 문을 통해 주변 보이드와의 거리를 계산했다. if문을 통해 가까운 보이드가 있는 경우 방향 벡터의 평균을 계산하고 계산된 힘을 보이드에 적용시켰다.

WanderState 클래스)



```java
import java.awt.Point;
import java.util.List;
import java.util.Random;

public class WanderState implements BoidState { // WanderState 클래스는 보이드가 임의의 방향으로 자유롭게 움직이도록 하는 상태를 정의
    private static final int MAX_FORCE = 1; // 보이드가 임의로 이동할 수 있는 최대 힘

    @Override
    public void applyBehavior(Boid boid, List<Boid> boids) {     // 현재 상태에서의 행동을 정의하는 메서드
        Random random = new Random();

        // 임의의 방향으로 이동하기 위한 힘 계산
        int dx = random.nextInt(2 * MAX_FORCE + 1) - MAX_FORCE; // x축 방향 임의의 힘
        int dy = random.nextInt(2 * MAX_FORCE + 1) - MAX_FORCE; // y축 방향 임의의 힘

        boid.applyForce(new Point(dx, dy));  // 보이드에 힘을 적용하여 이동 방향을 변경

        for (Boid other : boids) {  // 주변 보이드와의 거리를 확인하여 CohesionState로 전환할 조건 검사
            if (other != boid && boid.getPosition().distance(other.getPosition()) < Boid.VISION_RANGE) {
                // 주변에 일정 범위 내의 다른 보이드가 발견되면 CohesionState로 전환
                System.out.println("boid id = " + boid.getId() + " state = " + boid.getState() + " changed to CohesionState");
                boid.setState(new CohesionState());
                return; // 상태를 변경한 후 종료
            }
        }
    }

    @Override
    public String toString() {
        return "WanderState";
    }
}
```

WanderState 클래스는 보이드가 임의의 방향으로 자유롭게 움직이도록 하는 상태를 정의하는 클래스이다. 보이드가 임의로 이동 할 수 있는 최대 힘을 정의하고 applyBehavior 메소드를 통해 현재 상태에서의 행동을 정의해주었다. 임의의 방향으로 이동하기 위한 힘을 계산하고 보이드에 힘을 적용하여 이동 방향을 변경해주었고 for-each 문을 통해 주변 보이드와의 거리를 확인하여 CohesionState로 전환할 조건을 검사했다. 그 후 주변에 일정 범위 내의 다른 보이드가 발견되면 CohesionState 로 전환시켰다.

Boid 클래스)

```java
import java.util.List;
import java.util.Random;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;

// Boid 클래스는 보이드를 정의
public class Boid {
    public static final int SPEED = 3; // 보이드의 최대 속도
    public static final int SEPARATION_DISTANCE = 20; // 분리 거리
    public static final int VISION_RANGE = 50; // 주변 시야 범위
    public static final int JOIN_THRESHOLD = 15; // 합류 기준 거리

    private static int count = 0; // 보이드 ID를 부여하기 위한 static 카운터
    private Point position; // 현재 보이드의 위치
    private Point velocity; // 현재 보이드의 속도
    private int panelWidth; // 패널의 너비
    private int panelHeight; // 패널의 높이
    private BoidState state; // 보이드의 현재 상태
    private int id; // 보이드의 고유 ID
    private Random random; // 난수 생성기

    // 생성자: 보이드를 초기화하고 화면 크기에 따라 임의의 위치와 속도를 설정
    public Boid(int panelWidth, int panelHeight) {
        this.panelWidth = panelWidth;
        this.panelHeight = panelHeight;
        this.random = new Random();
        position = new Point(random.nextInt(panelWidth), random.nextInt(panelHeight));
        velocity = new Point(random.nextInt(2 * SPEED) - SPEED, random.nextInt(2 * SPEED) - SPEED);
        this.id = ++count;
        this.state = new WanderState(); // 초기 상태를 WanderState로 설정
    }

    // 군집 행동 메서드: 분리, 정렬, 결집 힘을 계산하고 속도에 적용
    public void flock(List<Boid> boids) {
        Point separation = separate(boids); // 분리 행동 계산
        Point alignment = align(boids); // 정렬 행동 계산
        Point cohesion = cohere(boids); // 결집 행동 계산

        // 힘을 가중치에 따라 합산
        velocity.translate((int)(separation.x * 1.5 + alignment.x * 1.0 + cohesion.x * 1.0),
                           (int)(separation.y * 1.5 + alignment.y * 1.0 + cohesion.y * 1.0));

        // 속도 제한
        limitSpeed();
    }

    // 분리 행동: 가까운 보이드와 거리 두기
    private Point separate(List<Boid> boids) {
        Point steer = new Point(x:0, y:0); // 방향 벡터
        int count = 0;

        for (Boid other : boids) {
            int distance = (int) position.distance(other.position); // 다른 보이드와의 거리 계산
            if (other != this && distance > 0 && distance < SEPARATION_DISTANCE) {
                Point diff = new Point(position.x - other.position.x, position.y - other.position.y);
                steer.translate(diff.x / distance, diff.y / distance);
                count++;
            }
        }

        if (count > 0) {
            steer.x /= count; // 평균화
            steer.y /= count;
        }
        return steer;
    }
```

```java
    // 정렬 행동: 주변 보이드의 평균 속도에 맞춤
    private Point align(List<Boid> boids) {
        Point avgVelocity = new Point(x:0, y:0); // 평균 속도
        int count = 0;

        for (Boid other : boids) {
            if (other != this && position.distance(other.position) < VISION_RANGE) {
                avgVelocity.translate(other.velocity.x, other.velocity.y);
                count++;
            }
        }

        if (count > 0) {
            avgVelocity.x /= count; // 평균화
            avgVelocity.y /= count;
            avgVelocity.translate(-velocity.x, -velocity.y); // 현재 속도와의 차이 계산
        }
        return avgVelocity;
    }
// 결집 행동: 주변 보이드의 중심으로 이동
private Point cohere(List<Boid> boids) {
    Point center = new Point(x:0, y:0); // 주변 보이드의 중심점
    int count = 0;

    for (Boid other : boids) {
        if (other != this && position.distance(other.position) < VISION_RANGE) {
            center.translate(other.position.x, other.position.y);
            count++;
        }
    }

    if (count > 0) {
        center.x /= count; // 평균화
        center.y /= count;
        center.translate(-position.x, -position.y); // 현재 위치와의 차이 계산
    }
    return center;
}

// 외부에서 가한 힘을 속도에 추가
public void applyForce(Point force) {
    velocity.translate(force.x, force.y);
}

// 속도 제한: 보이드의 속도를 SPEED로 제한
private void limitSpeed() {
    double magnitude = Math.sqrt(velocity.x * velocity.x + velocity.y * velocity.y); // 속도의 크기 계산
    if (magnitude > SPEED) {
        velocity.x = (int) ((velocity.x / magnitude) * SPEED);
        velocity.y = (int) ((velocity.y / magnitude) * SPEED);
    }
}
```

```
    // 보이드의 위치와 상태를 업데이트
    public void update(List<Boid> boids) {
        if(state != null){
            state.applyBehavior(this, boids); // 현재 상태에 따라 행동 수행
        }
        position.translate(velocity.x, velocity.y); // 위치 갱신
        // 화면 가장자리에서 반대쪽으로 이동
        if (position.x < 0) position.x = panelWidth;
        else if (position.x > panelWidth) position.x = 0;
        if (position.y < 0) position.y = panelHeight;
        else if (position.y > panelHeight) position.y = 0;
    }

    // 보이드를 삼각형 형태로 그리기
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;

        double angle = Math.atan2(velocity.y, velocity.x); // 속도의 방향 계산

        g2d.translate(position.x, position.y);
        g2d.rotate(angle);

        int[] xPoints = {0, -10, -10}; // 삼각형 x 좌표
        int[] yPoints = {0, -5, 5}; // 삼각형 y 좌표
        g2d.setColor(Color.BLUE); // 보이드 색상 설정
        g2d.fillPolygon(xPoints, yPoints, nPoints:3); // 삼각형 그리기

        g2d.rotate(-angle);
        g2d.translate(-position.x, -position.y);
    }

    // 목표 상태를 설정하는 메서드 (타겟 지점을 설정)
    public void moveToTarget(Point target){
        this.state = new TargetState(target);
    }
```

```
    // 보이드의 위치를 반환
    public Point getPosition() {
        return position;
    }

    // 속도를 설정
    public void setVelocity(Point velocity){
        this.velocity = velocity;
    }

    // 현재 속도를 반환
    public Point getVelocity() {
        return velocity ;
    }

    // 목표 지점에 도달했는지 체크
    public boolean hasReachedTarget(Point target) {
        double distance = position.distance(target);
        return distance < SEPARATION_DISTANCE; // 목표 지점과의 거리가 SEPARATION_DISTANCE보다 작으면 도달
    }

    // 현재 상태를 설정
    public void setState(BoidState state) {
        this.state = state ;
    }

    // 현재 상태를 반환
    public BoidState getState() {
        return state;
    }

    // 보이드 ID를 반환
    public int getId() {
        return id;
    }
```

```
        @Override
        public String toString() {
            return "{" +
                    " position='" + getPosition() + "'" +
                    ", velocity='" + getVelocity() + "'" +
                    ", state='" + getState() + "'" +
                    ", id='" + getId() + "'" +
                    "}";
        }
    }
```

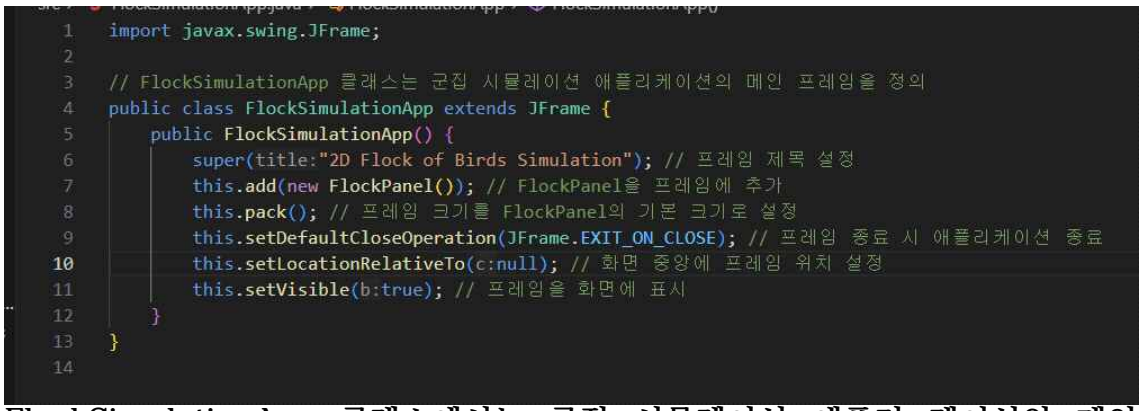
Boid 클래스에서는 보이드를 정의한다. 보이드의 최대속도, 분리 거리, 주변 시야 범위, 합류 기준 거리에 대해 값을 매겨줬고 보이드 ID를 부여하기 위한 Static 카운터, 현재 보이드의 위치, 현재 보이드의 속도, 패널의 너비, 패널의 높이, 보이드의 현재 상태, 보이드의 고유 ID, 난수생성기를 정의해주었고, 생성자를 생성해 보이드를 초기화하고 화면 크기에 따라 임의의 위치와 속도를 설정해주었다. 그리고 초기 상태를 WanderState로 설정해주었다. 그 후 Flock 메소드를 선언해주었는데 군집 행동 메소드로써, 분리, 정렬, 결집 힘을 계산하고 속도에 적용시켰다. 그 후 힘을 가중치에 따라 합산하고 속도 제한을 걸어주었다. 그 후 separate 메소드를 통해 가까운 보이드와 거리를 두도록 하게 했다. 그 다음 align 메소드를 통해 주변 보이드의 평균 속도에 맞추도록 하였다. 그 다음 cohere 메소드를 통해 주변 보이드의 중심으로 이동하도록 하였고 , applyforce 메소드를 통해 외부에서 가한 힘을 속도에 추가시켰다. limitSpeed메소드를 통해 보이드의 속도를 SPEED로 제한했고, update 메소드를 통해 보이드의 위치와 상태를 업데이트 시켰다. 현재 상태에 따라 행동을 수행하고 위치를 갱신해 화면 밖으로 나갈 경우 화면 가장자리에서 반대쪽으로 이동하도록 선언해주었다. draw 메소드를 통해 보이드를 삼각형 형태로 그려주었다. 그 다음 moveToTarget 메소드를 통해 타겟지점을 설정해주었다.

FlockPanel 클래스)

```java
import java.util.ArrayList;
import java.util.List;
import javax.swing.JPanel;
import javax.swing.Timer;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

// FlockPanel 클래스는 보이드들의 군집 행동을 시각화하고 조작할 수 있는 패널을 제공
public class FlockPanel extends JPanel {
    private static final int PANEL_WIDTH = 800; // 패널의 너비
    private static final int PANEL_HEIGHT = 600; // 패널의 높이
    private static final int NUM_BOIDS = 20; // 생성할 보이드의 수
    private static final int UPDATE_INTERVAL = 200; // 타이머 업데이트 간격

    private List<Boid> boids; // 보이드 리스트
    private Point targetPoint; // 마우스 클릭으로 설정되는 타겟 지점


    public FlockPanel() {  // 생성자: 패널 초기화 및 보이드 생성
        setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT)); // 패널 크기 설정
        setBackground(Color.WHITE); // 배경 색상 설정


        boids = new ArrayList<>();    // 보이드 생성 및 초기화
        for (int i = 0; i < NUM_BOIDS; i++) {
            Boid boid = new Boid(PANEL_WIDTH, PANEL_HEIGHT);
            boids.add(boid); // 리스트에 보이드 추가
        }
        addMouseListener(new MouseAdapter() {  // 마우스 클릭 이벤트 처리
            @Override
            public void mouseClicked(MouseEvent e){
                targetPoint = e.getPoint(); // 클릭한 위치를 타겟 지점으로 설정
                for (Boid boid : boids){
                    boid.moveToTarget(targetPoint); // 모든 보이드가 타겟 지점을 향하도록 설정
                }
            }
        });


        new Timer(UPDATE_INTERVAL, e -> {  // 타이머 생성 및 시작
            updateBoids(); // 보이드 상태와 위치 업데이트
            repaint(); // 패널 다시 그리기
        }).start();
    }


    private void updateBoids() {   // 보이드의 상태 및 위치를 업데이트
        for (Boid boid : boids) {
            boid.update(boids); // 현재 상태에 따른 행동 수행 및 위치 갱신
        }
    }


    @Override
    protected void paintComponent(Graphics g) {   // 패널에 보이드들을 그리는 메서드
        super.paintComponent(g); // 패널 초기화
        for (Boid boid : boids) {
            boid.draw(g); // 각 보이드를 그리기
            System.out.println(boid);
        }
    }
}

```

FlockPanel 클래스는 보이드들의 군집 행동을 시각화하고 조작할 수 있는 패널을 제공한다. 패널의 너비, 패널의 높이, 생성할 보이드의 수, 타이머 업데이트 간격을 선언하고, 보이드 리스트, 마우스 클릭으로 설정되는 타겟 지점을 선언해주고, 생성자를 생성해 패널 초기화 및 보이드를 생성해주었고 패널 크기 설정, 배경 색상 설정도 해주고, 보이드 생성 및 초기화를 하고 리스트에 보이드를 추가시켜주었고, 마우스 클릭 이벤트 처리 메소드를 통해 클릭한 위치를 타겟 지점으로 설정해주었다.

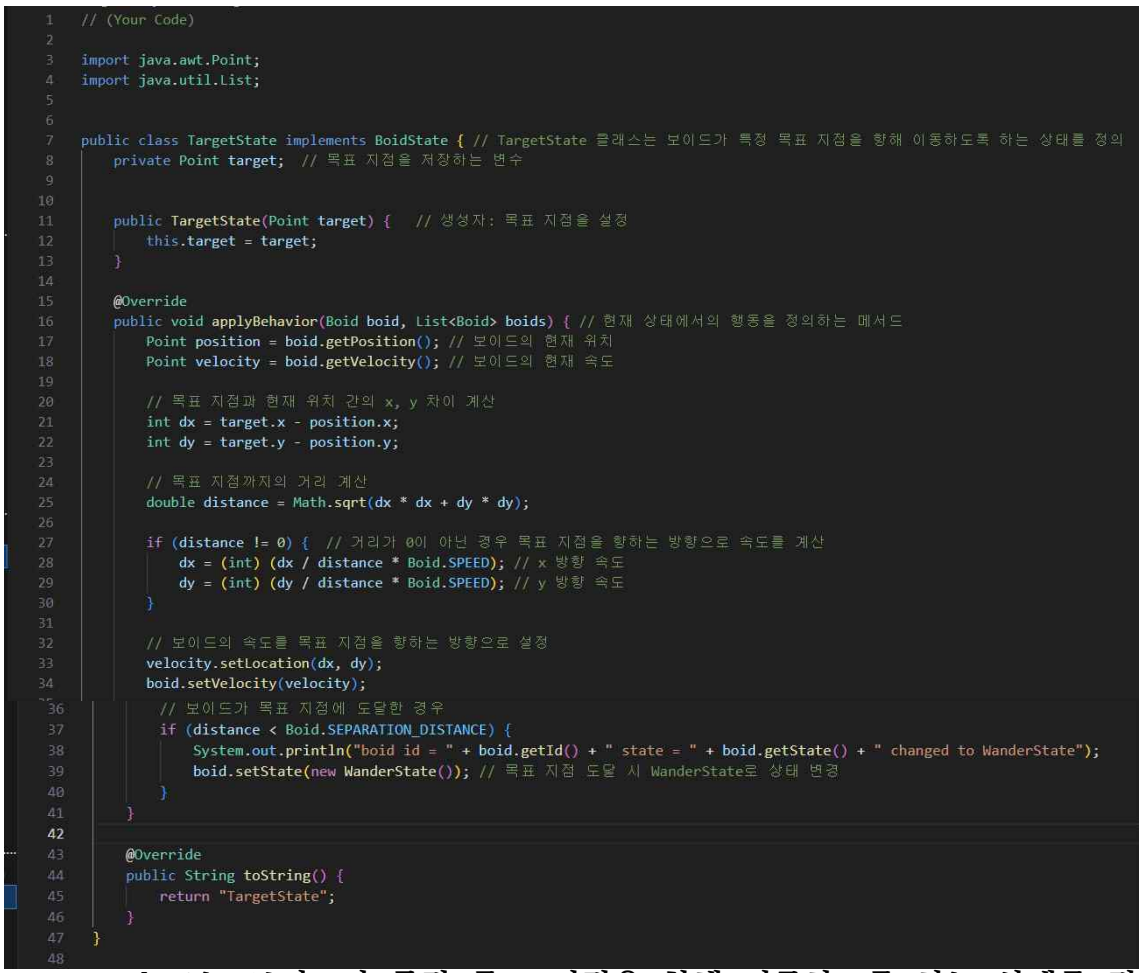
FlockSimulationApp 클래스)



```java
import javax.swing.JFrame;

// FlockSimulationApp 클래스는 군집 시뮬레이션 애플리케이션의 메인 프레임을 정의
public class FlockSimulationApp extends JFrame {
    public FlockSimulationApp() {
        super(title:"2D Flock of Birds Simulation"); // 프레임 제목 설정
        this.add(new FlockPanel()); // FlockPanel을 프레임에 추가
        this.pack(); // 프레임 크기를 FlockPanel의 기본 크기로 설정
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 프레임 종료 시 애플리케이션 종료
        this.setLocationRelativeTo(c:null); // 화면 중앙에 프레임 위치 설정
        this.setVisible(b:true); // 프레임을 화면에 표시
    }
}
```

FlockSimulationApp 클래스에서는 군집 시뮬레이션 애플리 케이션의 메인 프레임을 정의해주었다. 프레임 제목 설정을 해주었고, FlockPanel를 프레임에 추가, 프레임 크기를 FlockPanel의 기본 크기로 설정, 프레임 종료 시 애플리케이션 종료, 화면 중앙에 프레임 위치 설정, 프레임을 화면에 표시하도록 선언해주었다.

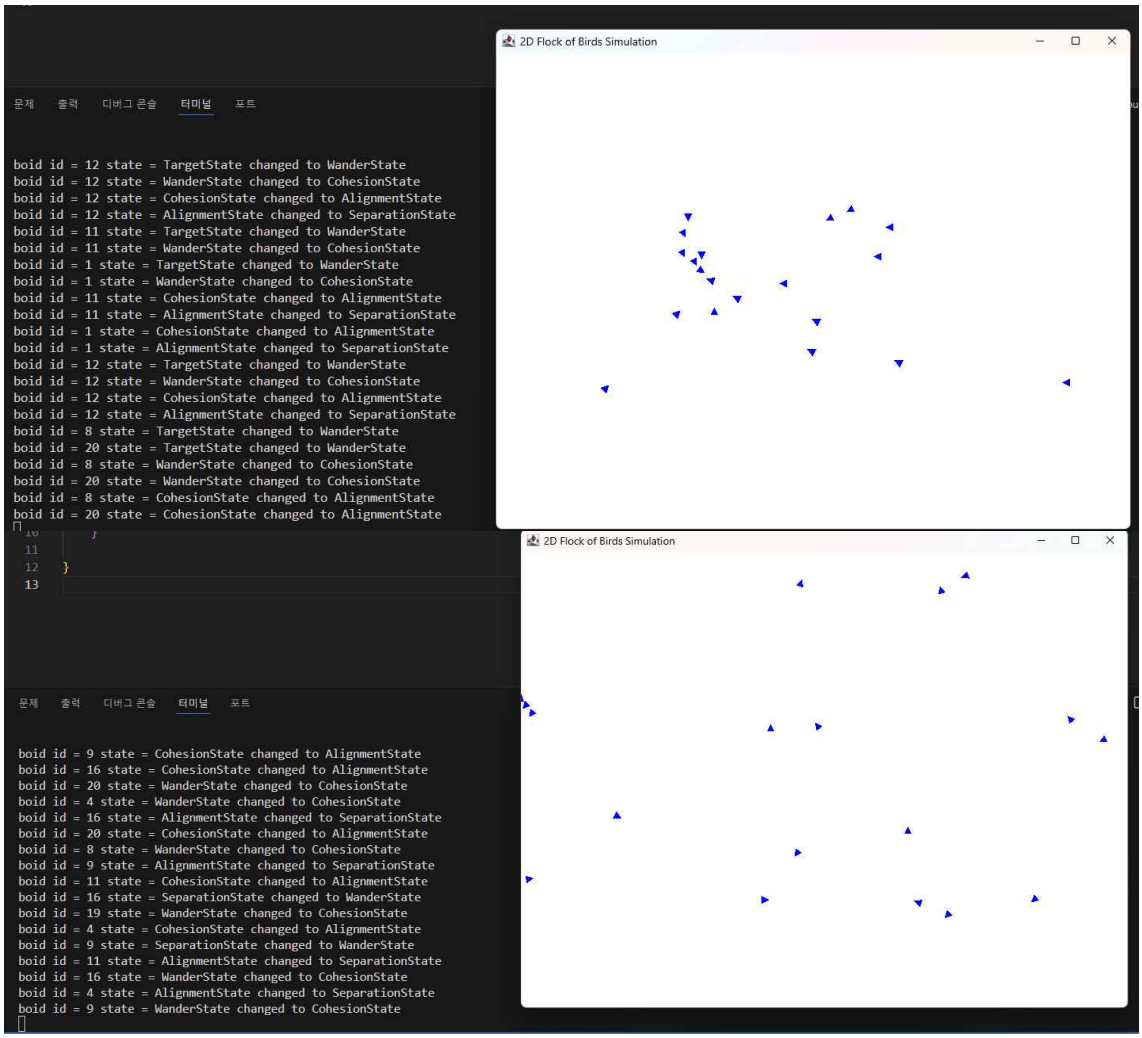
(Your Code)

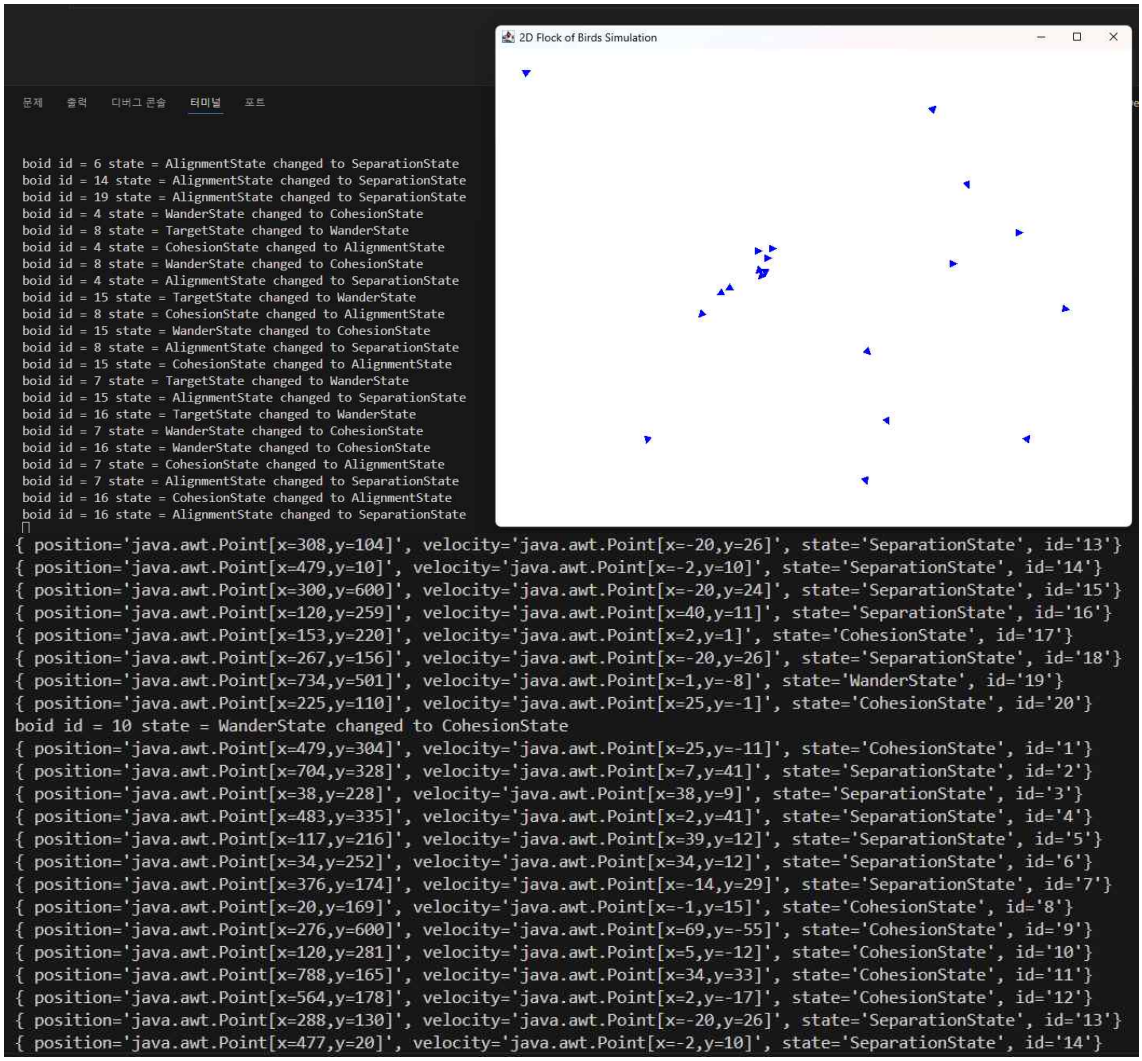


```java
// (Your Code)

import java.awt.Point;
import java.util.List;


public class TargetState implements BoidState { // TargetState 클래스는 보이드가 특정 목표 지점을 향해 이동하도록 하는 상태를 정의
    private Point target;  // 목표 지점을 저장하는 변수


    public TargetState(Point target) {   // 생성자: 목표 지점을 설정
        this.target = target;
    }

    @Override
    public void applyBehavior(Boid boid, List<Boid> boids) { // 현재 상태에서의 행동을 정의하는 메서드
        Point position = boid.getPosition(); // 보이드의 현재 위치
        Point velocity = boid.getVelocity(); // 보이드의 현재 속도

        // 목표 지점과 현재 위치 간의 x, y 차이 계산
        int dx = target.x - position.x;
        int dy = target.y - position.y;

        // 목표 지점까지의 거리 계산
        double distance = Math.sqrt(dx * dx + dy * dy);

        if (distance != 0) {  // 거리가 0이 아닌 경우 목표 지점을 향하는 방향으로 속도를 계산
            dx = (int) (dx / distance * Boid.SPEED); // x 방향 속도
            dy = (int) (dy / distance * Boid.SPEED); // y 방향 속도
        }

        // 보이드의 속도를 목표 지점을 향하는 방향으로 설정
        velocity.setLocation(dx, dy);
        boid.setVelocity(velocity);
          // 보이드가 목표 지점에 도달한 경우
          if (distance < Boid.SEPARATION_DISTANCE) {
              System.out.println("boid id = " + boid.getId() + " state = " + boid.getState() + " changed to WanderState");
              boid.setState(new WanderState()); // 목표 지점 도달 시 WanderState로 상태 변경
          }
      }

      @Override
      public String toString() {
          return "TargetState";
      }
}

```

your code로는 보이드가 특정 목표 지점을 향해 이동하도록 하는 상태를 정의 할 수 있도록 하였다. 목표 지점을 저장하는 변수를 선언해주었고 생성자를 생성해 목표 지점을 설정해주었고 applyBehavior 메소드를 통해 현재 상태에서의 행동을 정의하는 메소드를 선언해 보이드의 현재위치, 현재 속도를 선언하고, 목표지점과 현재 위치 간의 x,y 차이를 계산해주고 목표 지점까지의 거리를 계산해주었다. 거리가 0이 아닌 경우 목표 지점을 향하는 방향으로 속도를 계산하게 했고 보이드의 속도를 목표 지점을 향하는 방향으로 설정하고 보이드가 목표 지점에 도달했을시 WanderState로 다시 상태를 변경하도록 선언해주었다.

실행결과)


2D Flock of Birds Simulation
문제 출력 디버그 콘솔 터미널 포트
boid id = 12 state = TargetState changed to WanderState
boid id = 12 state = WanderState changed to CohesionState
boid id = 12 state = CohesionState changed to AlignmentState
boid id = 12 state = AlignmentState changed to SeparationState
boid id = 11 state = TargetState changed to WanderState
boid id = 11 state = WanderState changed to CohesionState
boid id = 1 state = TargetState changed to WanderState
boid id = 1 state = WanderState changed to CohesionState
boid id = 11 state = CohesionState changed to AlignmentState
boid id = 11 state = AlignmentState changed to SeparationState
boid id = 1 state = CohesionState changed to AlignmentState
boid id = 1 state = AlignmentState changed to SeparationState
boid id = 12 state = TargetState changed to WanderState
boid id = 12 state = WanderState changed to CohesionState
boid id = 12 state = CohesionState changed to AlignmentState
boid id = 12 state = AlignmentState changed to SeparationState
boid id = 8 state = TargetState changed to WanderState
boid id = 20 state = TargetState changed to WanderState
boid id = 8 state = WanderState changed to CohesionState
boid id = 20 state = WanderState changed to CohesionState
boid id = 8 state = CohesionState changed to AlignmentState
boid id = 20 state = CohesionState changed to AlignmentState
}
}
2D Flock of Birds Simulation
문제 출력 디버그 콘솔 터미널 포트
boid id = 9 state = CohesionState changed to AlignmentState
boid id = 16 state = CohesionState changed to AlignmentState
boid id = 20 state = WanderState changed to CohesionState
boid id = 4 state = WanderState changed to CohesionState
boid id = 16 state = AlignmentState changed to SeparationState
boid id = 20 state = CohesionState changed to AlignmentState
boid id = 8 state = WanderState changed to CohesionState
boid id = 9 state = AlignmentState changed to SeparationState
boid id = 11 state = CohesionState changed to AlignmentState
boid id = 16 state = SeparationState changed to WanderState
boid id = 19 state = WanderState changed to CohesionState
boid id = 4 state = CohesionState changed to AlignmentState
boid id = 9 state = SeparationState changed to WanderState
boid id = 11 state = AlignmentState changed to SeparationState
boid id = 16 state = WanderState changed to CohesionState
boid id = 4 state = AlignmentState changed to SeparationState
boid id = 9 state = WanderState changed to CohesionState



본래의 출력에서는 점들의 모든 좌표가 바뀌는걸 보여주며 어떤 id를 가진 보이드가 상태 변환을 했을 때 그걸 알려주는 출력문까지 나오게 했고 your code인 targetstate에서 한 점에 모였다가 다시 wanderstate로 돌아가며 실행되는 모습을 보여주었다.