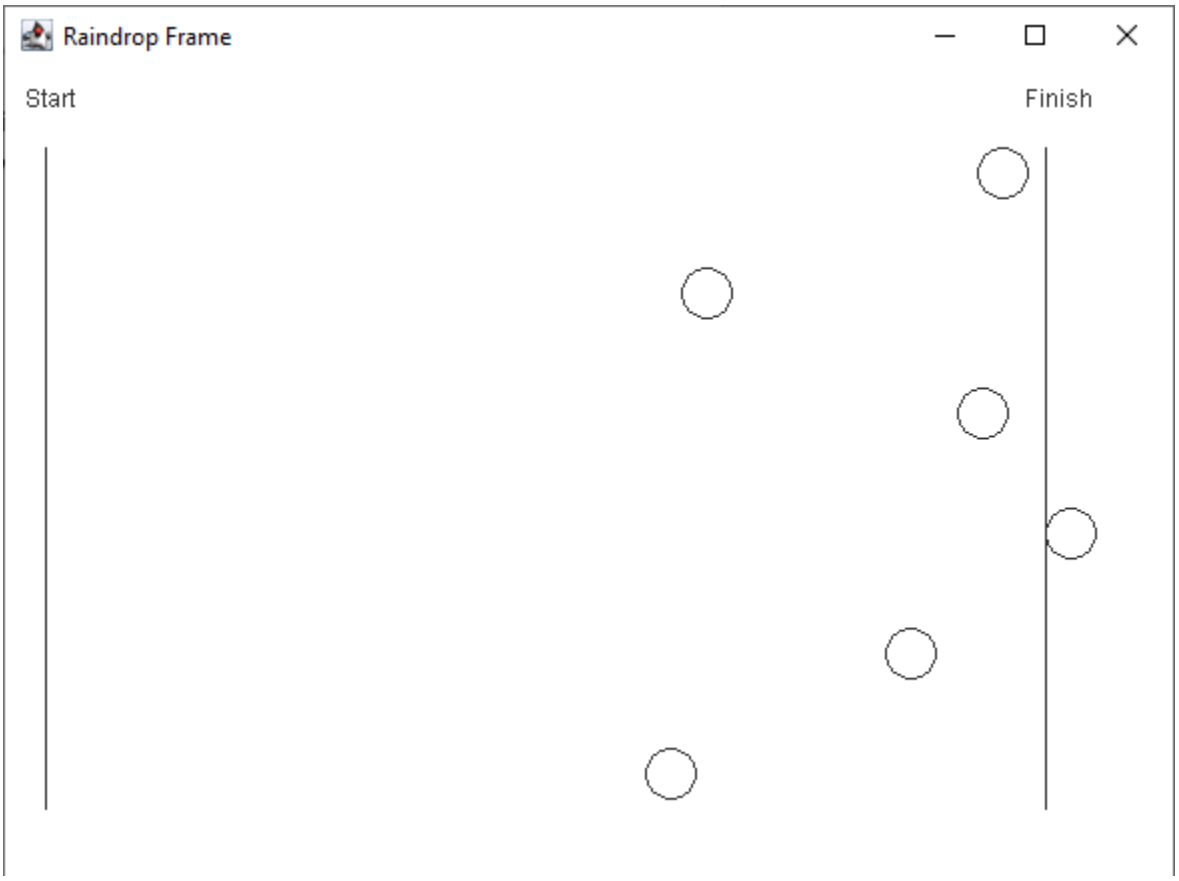
Noah Streveler

Assignment 6




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Simulate {
	public static void main(String[] args) {
		RainSimuFrame fr= new RainSimuFrame("Raindrop Frame");
		fr.setVisible(true);;
	}
}
```

```
abstract class AnimationFrame extends JFrame{
	protected Thread animator;
	protected JPanel pnl;

	public AnimationFrame(String title){
		super(title);
		this.setSize(600, 600);
		this.setLocation(300, 100);
		pnl = getAnimationPanel();
		this.add(pnl);
		this.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				animator.interrupt();
				e.getWindow().dispose();
				System.exit(0);
			}
		});
		init();
		animator = new Thread(getTask());
		animator.start();
	}

	public void repaint(){
		super.repaint();
		pnl.repaint();
	}
```

```
	abstract void init();
	abstract Runnable getTask();
	abstract JPanel getAnimationPanel();
}

class RainSimuFrame extends AnimationFrame{
	private Race[] racers;

	public RainSimuFrame(String title){
		super(title);
	}

	@Override
	public void init() {
		racers = new Race[6];
		int x = 20, y = 40;
		for(int i = 0; i < racers.length; i++){
			racers[i] = new Race();
			racers[i].setPosition((int) (x), (int) (y));
			y += 60;
		}
	}

	@Override
	Runnable getTask() {
		return new RacerTask();
	}
```

```
@Override
JPanel getAnimationPanel() {
	class RainPanel extends JPanel{
		public RainPanel(){
			this.setBackground(Color.white);
		}
		public void paintComponent(Graphics g){
			super.paintComponent(g);
			Graphics2D g2 = (Graphics2D) g;
			g2.drawLine(20, 40, 20, 370);
			g2.drawString("Start", 10, 20);
			g2.drawString("Finish", 510, 20);
			g2.drawLine(520, 40, 520, 370);
			for(int i = 0; i < racers.length; i++){
				g2.draw(racers[i].getRacer());
			}
		}
	}
	return new RainPanel();
}

class RacerTask implements Runnable{
	@Override
	public void run() {
		while(isWinner() == -1){
			for(int i = 0; i < racers.length; i++){
```

```
                    Race player = racers[i];
                    player.move(findFirst(), findLast());
                    player.setPosition(player.getX(), player.getY());
                }
                try{
                    Thread.sleep(30);
                }catch(InterruptedException e){
                    System.out.println(e.getStackTrace().toString());
                }
                pnl.repaint();
            }
        }

        public int isWinner(){
            for(int i = 0; i < racers.length; i++){
                if (racers[i].getX() >= 520){
                    racers[i].setX(520);
                    return 1;
                }
            }
            return -1;
        }
        public int findFirst() {
            int max = Integer.MIN_VALUE;
            for(int i = 0; i < racers.length; i++){
                if(racers[i].getX() > max)
                    max = racers[i].getX();
```

```
                }
                return max;
        }

        public int findLast() {
                int min = Integer.MAX_VALUE;
                for(int i = 0; i < racers.length; i++){
                        if(racers[i].getX() < min)
                                min = racers[i].getX();
                }
                return min;
        }
    }
}
```

```
import java.awt.*;
import java.util.*;
import java.awt.geom.Ellipse2D;
```

```
public class Race {
	private int x, y;
	Mover racer;

	public Race(){
		x = 40;
		y = 20;
		racer = new Mover();
	}

	public int getX() {
		return x;
	}

	public void setX(int x) {
		this.x = x;
	}

	public int getY() {
		return y;
	}

	public void setY(int y) {
		this.y = y;
	}
```

```
	public void setPosition(int xPos, int yPos){
		x = xPos;
		y = yPos;
	}

	public Shape getRacer(){
		return new Ellipse2D.Double(x, y, 25, 25);
	}

	public void move(int first, int last) {
		racer.play(first, last, x);
		x = racer.position;
	}
}
```

```
public class Mover {
	int first, last, position, duration = 0, move = 0;

	public Mover() {
	}

	public void play(int f, int l, int p) {
		first = f;
		last = l;
```

```
        position = p;
        getMove();
        position += moves(move);
        isNegative();
}

public void getMove() {
         if(duration == 0) {
               move = moveType();
               duration = duration();
         }
         else {
               duration--;
         }
}

public int moves(int number) {
        int newPos = 0;
        int dice = diceRoll();
        switch (number){
        case 1:
               newPos = dice + (first - position)/2;
               if(dice == 1 || dice == 2)
                     newPos = newPos - (2 * newPos);
               break;
        case 2:
               newPos = dice;
               if((dice % 2) == 0)
                     newPos = 3 * newPos;
               break;
        case 3:
               newPos = dice + (position - last)/2;
               if(dice >= 3 && dice <= 6)
                     newPos = newPos - (2 * newPos);
               break;
        }
        return newPos;
}

public int diceRoll() {
        int dice = ((int)(Math.random() * 6 + 1));
        return dice;
}

public int moveType() {
        int random = ((int)(Math.random() * 3 + 1));
        return random;
}

public int duration() {
        int random = ((int)(Math.random() * 5 + 1));
        return random;
}

public void isNegative() {
```

```
                if(position < 20)
                        position = 20;
        }
}
```