Bellodi Pietro, Faggioli Filippo, Sgueglia della Marra Elena

Pandemic Studio

Documentazione

1.0 Sketch Principale

Le schermate del programma sono suddivise in base a delle **costanti** numeriche

```
private final int PAGE_HOME_OPENING = 11;
private final int PAGE_MAP_OPENING = 12;
private final int PAGE_GRAPH_OPENING = 13;
private final int PAGE_HOME = 1;
private final int PAGE_MAP = 2;
private final int PAGE_GRAPH = 3;
```

che vengono **aggiornate** e **inizializzate** nel draw grazie ad uno **switch**.

```
switch (page) {
    // Controlla se ci sono pagine che attendono di essere inizializzate
    case PAGE_HOME_OPENING:
        homeInit();
        break;
    case PAGE_MAP_OPENING:
        mapInit();
        break;
    case PAGE_GRAPH_OPENING:
        graphInit();
        break;

    // Aggiorna la finestra attualmente aperta
    case PAGE_HOME:
        home();
        break;
    case PAGE_MAP:
        map();
        break;
    case PAGE_GRAPH:
        graph();
        break;
    }
```

2.0 Interfaccia grafica

Ogni schermata utilizza **4 classi grafiche** principali

- *Button*
- *TextView*
- *EditText*
- *Legend*

2.1 Button

La classe *Button* genera un **pulsante** e viene utilizzata mediante il costruttore

```
public Button(float x, float y, float w, float h, String text, float textDim)
```

e i due metodi ***show()*** e ***isClicked()***

```
public void show() {
        if (isClicked()) {
            stroke(#ffffff);
            strokeWeight(2);
        } else {
            noStroke();
            strokeWeight(1);
        }

        fill(bgColor);
        rectMode(CENTER);
        rect(x, y, w, h, cornerRadius);

        fill(textColor);
        textSize(textDim);
        text(text, x - textWidth(text)/2, y + textAscent()/2);
}
```

```
public boolean isClicked() {
        float cx = mouseX;
        float cy = mouseY;
        return (cx > x - w/2 && cx < x + w/2) && (cy > y - h/2 && cy < y + h/2);
}
```

2.2 TextView

La classe *TextView* **mostra** un semplice **testo** e si utilizza con il costruttore

```
public TextView(float x, float y, float w, float h, String text, float textDim)
```

e il metodo ***update()*** analogamente alle altre classi.

2.3 EditText

La classe *EditText* gestisce delle caselle di testo interattive all'interno delle quali l'utente può scrivere e si utilizzano con il costruttore

```
1 public EditText(float x, float y, float w, float h, String text, float textDim)
```

e le funzioni ***show()*** (analogamente alle altre classi) e ***setFocused()*** per specificare quale *EditText* debba essere attivata.

```
1 public void setFocused(boolean focused) {
2       this.focused = focused;
3 }
```

2.4 Legend

La classe *Legend* crea una **legenda** consultabile dall'utente che permette di distinguere il **significato dei colori** visualizzati.
Si inizializza con il costruttore rispettivo

```
1 public Legend(float x, float y)
```

e si aggiorna con il metodo ***draw()***

3.0 Graph

Il grafico è uno strumento che è in grado (**a partire** dai **dati inseriti** dall'utente nella schermata home) di **tracciare il grafico** delle equazioni differenziali ordinarie (EDO) derivate di dati dell'epidemia.
La classe relativa è la classe ***Graph***.

3.1 Classe Graph

La classe Graph viene inizializzata con il costruttore dove

- **S** Numero di suscettibili iniziale

- **I** Numero di infetti iniziale
- **beta** coefficiente di infezione
- **gamma** coefficiente di guarigione

```
public Graph(float S, float I, float beta, float gamma) {
        float N = S + I;
        this.S = S/N;
        this.I = I/N;
        this.R = 0.0;
        this.colS = color(#00aa00);
        this.colI = color(#ff0000);
        this.colR = color(#0000ff);
        this.beta = beta;
        this.gamma = gamma;
}
```

3.2 Calcolo dei delta delle EDO

I **delta** vengono calcolati nella funzione ***update()*** in base ai coefficienti beta e gamma passati al costruttore.

```
private void update() {
    float dS, dI, dR;

    // N = S + I + R; sempre 1.0
    dS = -beta*(S/1)*I;
    dI = beta*(S/1)*I - gamma*I;
    dR = gamma*I;

    dS *= RESOLUTION;
    dI *= RESOLUTION;
    dR *= RESOLUTION;

    S += dS;
    I += dI;
    R += dR;
}
```

3.3 Tracciare il grafico

Il grafico viene tracciato con un'approssimazione lineare da parte della funzione ***plot()***

```java
public void plot(int days, float day_duration) {
    strokeWeight(LINE_WEIGHT);
    float x = 0.0;
    for (int day = 0; day < days; day++) {

        // Calcola punti iniziali
        float x1 = x;
        float yS1 = height - BOTTOM_OFFSET - S * Y_AXIS_AMPLIFIER;
        float yI1 = height - BOTTOM_OFFSET - I * Y_AXIS_AMPLIFIER;
        float yR1 = height - BOTTOM_OFFSET - R * Y_AXIS_AMPLIFIER;

        // Calcola punti finali
        x += RESOLUTION * day_duration;
        update();
        float x2 = x;
        float yS2 = height - BOTTOM_OFFSET - S * Y_AXIS_AMPLIFIER;
        float yI2 = height - BOTTOM_OFFSET - I * Y_AXIS_AMPLIFIER;
        float yR2 = height - BOTTOM_OFFSET - R * Y_AXIS_AMPLIFIER;

        // Disegna linee di approssimazione tra i punti
        stroke(colS);
        line(x1, yS1, x2, yS2);
        stroke(colI);
        line(x1, yI1, x2, yI2);
        stroke(colR);
        line(x1, yR1, x2, yR2);
    }
}
```

4.0 Map

Lo strumento mappa permette di **simulare** una **pandemia** che, a seconda dei parametri impostati dall'utente, potrebbe diventare tale, oppure fermarsi poco dopo lo scoppio dell'epidemia iniziale.
Il tutto avviene su una **mappa mondiale** che permette la **visualizzazione in tempo reale** dello stato delle singole epidemie nelle **città** grazie a cerchi concentrici di colore diverso, ma anche nel **resto del mondo** grazie a punti di colore diverso in base allo stato epidemiologico di quella determinata zona geografica.

4.1 Città

Le città sono gestite dalla **classe Cluster** che utilizza il **modello** compartimentale **SIR** per controllare l'evolversi dell'epidemia all'interno della città.

L’inizializzazione avviene con il rispettivo costruttore dove **x_ratio** e **y_ratio** sono le **posizioni relative** della città per evitare un posizionamento errato al variare della dimensione del monitor.

```
1 public Cluster(String name, float x_ratio, float y_ratio, float population)
```

La classe Cluster mette a disposizione svariati strumenti per la gestione della città:

- ***setEpidemiologicalParameters()***
  - Aggiorna i coefficienti epidemiologici della città
- ***infect()***
  - Infetta un singolo individuo all’interno della città
- ***update()***
  - Aggiorna di un istante di tempo le EDO che controllano la città
- ***draw()***
  - Disegna la città sullo schermo
- ***showName()***
  - Mostra il nome della città al passaggio del mouse
- ***getNearestMiniCluster()***
  - Trova il mini-cluster (gruppo abitativo) più vicino

```java
public void setEpidemiologicalParameters(float beta, float gamma) {
    this.beta = beta;
    this.gamma = gamma;
}

public void infect() {
    if (I == 0) I = 1;
}

public void update() {
    float dS, dI, dR;

    dS = -beta*(S/population)*I;
    dI = beta*(S/population)*I - gamma*I;
    dR = gamma*I;

    S += dS/SLOWDOWN;
    I += dI/SLOWDOWN;
    R += dR/SLOWDOWN;
}

public void draw() {
    float rs = getRadius(S);
    float ri = getRadius(I);
    float rr = getRadius(R);
    if (I == 0 && rs < 5) rs = 5;
    if (I < 0.9) ri = 0;
    if (R < 0.9) rr = 0;
    fill(0, 200, 0, 100);
    circle(x, y, rs);
    fill(0, 0, 200, 255);
    circle(x, y, rr);
    fill(200, 0, 0, 255);
    circle(x, y, ri);
    showName();
}

public boolean isClicked() {
    return (Math.sqrt((mouseX - x)*(mouseX - x) + (mouseY - y)*(mouseY - y)) < r/2) && mousePressed;
}

private void showName() {
    if (Math.sqrt((mouseX - x)*(mouseX - x) + (mouseY - y)*(mouseY - y)) < r/2) {
        nameTv.show();
    }
}

private float getRadius(float population) {
    return (25 * population)/37f;
}

public int getNearestMiniCluster(int[][] populationPositions, int population) {
    double minDist = 1000000;
    int index = 0;
    for (int i = 0; i < population; i++) {
        int cx = populationPositions[i][0];
        int cy = populationPositions[i][1];
        double dist = Math.sqrt((x - cx)*(x - cx) + (y - cy)*(y - cy));

        if (dist < minDist) {
            minDist = dist;
            index = i;
        }
    }
    return index;
}
```

4.2 Popolazione fuori dalle città

Il resto della popolazione mondiale è **distribuito** equamente dalla classe ***PopulationDistributer*** e l'evolversi della pandemia tra la popolazione è **gestita** da ***PopulationManager***.

4.2.1 PopulationDistributer

PopulationDistributer **distribuisce** la **popolazione** in base a dei quadrati virtuali all'interno dei quali posiziona un certo numero di mini-cluster in base alla densità abitativa specificata nella costante AVG_WORLD_POP_DENSITY.
Alcune zone della mappa sono di un colore leggermente più chiaro o scuro per indicare le zone a maggiore o minore densità abitativa.
Ogni mini-cluster prima di essere posizionato verifica di essere in un punto nel quale il colore del pixel è esattamente quello della mappa per evitare di creare mini-cluster in acqua.

La distribuzione viene eseguita dal metodo ***distribute()***

```java
public void distribute() {
    loadPixels();
    for (int y = 0; y < height; y += RESOLUTION_SQUARE) {
        for (int x = 0; x < width; x += RESOLUTION_SQUARE) {
            for (int p = 0; p < AVG_WORLD_POP_DENSITY; p++) {
                int p_x = x + (int) random(-RESOLUTION_NOISE_OFFSET, RESOLUTION_NOISE_OFFSET);
                int p_y = y + (int) random(-RESOLUTION_NOISE_OFFSET, RESOLUTION_NOISE_OFFSET);
                int pos = y*width + x;
                if (pixels[pos] == MAP_COLOR) {
                    populationPositions[population] = new int[]{p_x, p_y};
                    population++;
                } else if (pixels[pos] == MAP_LOW_DENSITY_COLOR) {
                    if (random(0, 1) < 0.65) {
                        populationPositions[population] = new int[]{p_x, p_y};
                        population++;
                    }
                } else if (pixels[pos] == MAP_HIGH_DENSITY_COLOR) {
                    populationPositions[population] = new int[]{p_x, p_y};
                    population++;
                    int np_x = x + (int) random(-RESOLUTION_NOISE_OFFSET, RESOLUTION_NOISE_OFFSET)
                    int np_y = y + (int) random(-RESOLUTION_NOISE_OFFSET, RESOLUTION_NOISE_OFFSET)
                    populationPositions[population] = new int[]{np_x, np_y};
                    population++;
                }
            }
        }
    }
}
```

4.2.2 PopulationManager

PopulationManager gestisce le nuove **infezioni** e **guarigioni** all'interno della popolazione mondiale.
Viene inizializzato con il costruttore relativo, dove *populationPositions* è un array bidimensionale di interi contenente le **x** e le **y** di tutti i **mini-cluster**. Tale array solitamente è creato dalla classe *PopulationDistributer*.

```java
public PopulationManager(int[][] populationPositions, int population, float beta, float gamma)
```

Per effettuare tale gestione necessita di array molto grandi (uno dei quali arriva ad una lunghezza di oltre tre milioni di variabili) che contengono:

- ***populationStatuses*** contiene lo stato (suscettibile, infetto o guarito) di ognuno dei mini-cluster.
- ***populationDistances*** contiene le distanze mutuali di tutti i mini-cluster
- ***clusterDistances*** contiene le distanze tra cluster e ognuno dei mini-cluster

```java
public void updatePopulation(int[][] populationPositions, int population) {
    this.population = population;
    this.S = population;
    this.I = 0;
    this.R = 0;
    this.populationStatuses = new int[population];
    this.populationDistances = new double[population * population];
    this.clusterDistances = new double[population * clusters.size()];
    this.populationPositions = populationPositions;
}
```

Il metodo ***update()*** gestisce le nuove infezioni e guarigioni

```java
public void update() {
    int index = 0;
    int index2 = 0;
    for (int i = 0; i < population; i++) {
        for (int j = 0; j < population; j++) {
            if (j != i) {
                double distance = populationDistances[index];
                if (distance < PopulationDistributer.RESOLUTION_SQUARE * 1.2) {
                    evaluateInfection(i, j);
                }
            }
            index++;
        }
        for (Cluster city : clusters) {
            double distance = clusterDistances[index2];
            if (distance < PopulationDistributer.RESOLUTION_SQUARE * 2 && city.I == 0) {
                evaluateClusterInfection(city, i);
            }
            index2++;
        }
        evaluateRecovery(i);
    }
}
```

All'interno di update sono importanti i tre metodi:

- ***evaluateInfection()*** valuta se due cluster possono infettarsi tra loro
- ***evaluateRecovery()*** valuta se un cluster può guarire

- ***evaluateClusterInfection()*** valuta se una città può infettarsi per vicinanza a un mini-cluster infetto.

```
public void evaluateClusterInfection(Cluster city, int miniCluster) {
    if (populationStatuses[miniCluster] == INFECTED) {
        city.infect();
        println("Infezione di: " + city.name);
    }
}

private void evaluateInfection(int cluster1, int cluster2) {
    if (populationStatuses[cluster1] == INFECTED && populationStatuses[cluster2] == SUSCEPTIBLE) {
        if (random(0, 1) < beta / 10)
            populationStatuses[cluster2] = INFECTED;
    } else if (populationStatuses[cluster2] == INFECTED && populationStatuses[cluster1] == SUSCEPTIBLE) {
        if (random(0, 1) < beta / 10)
            populationStatuses[cluster1] = INFECTED;
    }
}

private void evaluateRecovery(int cluster) {
    if (populationStatuses[cluster] == INFECTED) {
        if (random(0, 1) < gamma / 60)
            populationStatuses[cluster] = RECOVERED;
    }
}
```

Il metodo ***generateDistances()*** genera le **distanze** mutuali tra **mini-cluster** e quelle dai **cluster** mettendole rispettivamente in *populationDistances* e *clusterDistances*.

```
public void generateDistances() {
    int index = 0;
    for (int i = 0; i < population; i++) {
        for (int j = 0; j < population; j++) {
            if (j != i) {
                int p1x = populationPositions[i][0];
                int p2x = populationPositions[j][0];
                int p1y = populationPositions[i][1];
                int p2y = populationPositions[j][1];
                populationDistances[index] = Math.sqrt((p1x - p2x)*(p1x - p2x) + (p1y - p2y)*(p1y - p2y));
            }
            index++;
        }
    }
    println("Distanze mini-cluster generate: " + String.valueOf(index));
    index = 0;
    for (int i = 0; i < population; i++) {
        for (Cluster city : clusters) {
            int p1x = populationPositions[i][0];
            int p2x = (int) city.x;
            int p1y = populationPositions[i][1];
            int p2y = (int) city.y;
            clusterDistances[index] = Math.sqrt((p1x - p2x)*(p1x - p2x) + (p1y - p2y)*(p1y - p2y));
            index++;
        }
    }
    println("Distanze cluster generate: " + String.valueOf(index));
}
```

Il metodo ***draw()*** **disegna** tutti i **mini-cluster** con i rispettivi colori.

```java
public void draw() {
    for (int i = 0; i < population; i++) {
        switch (populationStatuses[i]) {
            case SUSCEPTIBLE:
                fill(0, 200, 0, 100);
                continue;

            case INFECTED:
                fill(200, 0, 0, 100);
                break;

            case RECOVERED:
                fill(0, 0, 200, 100);
                break;
            default:
                println("Tipologia cluster sconosciuta");
                fill(#ffff00);
                break;
        }
        circle(populationPositions[i][0], populationPositions[i][1], 5);
    }
}
```

4.3 Rotte aeree

Per simulare con maggiore precisione una pandemia moderna, la classe ***FlightRoute*** gestisce molteplici linee aeree (ogni istanza ne rappresenta una sola) che collegano cluster di grandi città.
Per inizializzare una rotta aerea è sufficiente utilizzare il costruttore, dove ***c1*** e ***c2*** sono le città di partenza e arrivo.

```java
public FlightRoute(Cluster c1, Cluster c2)
```

Il metodo ***draw()*** gestisce il rendering e aggiornamento della rotta aerea controllando se è giunto a destinazione e se i passeggeri sono infetti o meno.

```
public void draw() {
    x += vx;
    y += vy;
    if (route == C1_TO_C2) {
        if (coordsSimilar(x, c2.x, y, c2.y, Math.max(vx, vy) * 7)) {
            route = C2_TO_C1;
            if (infected) infectCluster(c2);
            else if (c2.I > 1) infected = true;
            vx = - vx;
            vy = - vy;
        }
        stroke(100, 100, 100, 60);
        strokeWeight(1);
        line(x + 6, y + 6, c2.x, c2.y);
        noStroke();
    } else if (route == C2_TO_C1) {
        if (coordsSimilar(x, c1.x, y, c1.y, Math.max(vx, vy) * 7)) {
            route = C1_TO_C2;
            if (infected) infectCluster(c1);
            else if (c1.I > 1) infected = true;
            vx = - vx;
            vy = - vy;
        }
        stroke(100, 100, 100, 60);
        strokeWeight(1);
        line(x + 6, y + 6, c1.x, c1.y);
        noStroke();
    }
    float rotation = getRotation();
    planeImg.rotate(rotation);
    if (infected) shape(planeInfectedImg, x, y, 12, 12);
    else shape(planeImg, x, y, 12, 12);
    planeImg.rotate(-rotation);
}
```