Universidad Nacional Autónoma de México

Facultad de Contaduría y Administración

Desarrollo de Aplicaciones móviles

Profesor: Cristian Cardoso Arellano

Alumno: Ortega Maldonado Diego Damiel

Actividad M6 01.

Vista del aeropuerto (una cuadrícula)

```
public class VistaAeropuerto extends View {

    16 usages
    private int cellSize = 100;
    6 usages
    private List<Avion> aviones = new ArrayList<>();
    7 usages
    private int columnas = 10;
    7 usages
    private int filas = 10;

    public VistaAeropuerto(Context context, AttributeSet attrs) {
        super(context, attrs);
        initializeAirplanes();
    }

    1 usage
    private void initializeAirplanes() {
        aviones.clear();
        int airplaneCount = 20; // Más aviones
        java.util.Random random = new java.util.Random();
        // Usar una lista de posiciones posibles para evitar superposición y distribuir mejor
        List<int[]> posiciones = new ArrayList<>();
        for (int x = 0; x < columnas; x++) {
            for (int y = 0; y < filas; y++) {
                posiciones.add(new int[]{x, y});
            }
        }
```

```
        // Mezclar las posiciones
        java.util.Collections.shuffle(posiciones, random);
        for (int i = 0; i < airplaneCount && i < posiciones.size(); i++) {

            int[] pos = posiciones.get(i);
            Avion.Direction direction = Avion.Direction.values()[random.nextInt
                    (Avion.Direction.values().length)];
            aviones.add(new Avion(pos[0], pos[1], direction));

        }

    }
```

```java
public List<Avion> getAviones() {
    return aviones;
}

3 usages
public void setAviones(List<Avion> airplanes) {

    if (airplanes == null) {

        this.aviones = new ArrayList<>();

    } else {

        this.aviones = airplanes;

    }

}

no usages
public void setFilas(int filas) {
    this.filas = filas;
}

no usages
public void setColumnas(int columnas) {

    this.columnas = columnas;

}
```

```java
public int getColumnas() {

    return columnas;

}
```

3 usages

```java
public int getFilas() {

    return filas;

}

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);
    cellSize = Math.min(w / columnas, h / filas);
    columnas = w / cellSize;
    filas = h / cellSize;
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    drawGrid(canvas);
    drawAirplanes(canvas);
}
```

1 usage

```java
private void drawGrid(Canvas canvas) {
    Paint paint = new Paint();
    paint.setColor(Color.GRAY);

    for (int i = 0; i <= columnas; i++) {

        int x = i * cellSize;
        canvas.drawLine(x, startY: 0, x, stopY: filas * cellSize, paint);

    }

    for (int i = 0; i <= filas; i++) {

        int y = i * cellSize;
        canvas.drawLine( startX: 0, y, stopX: columnas * cellSize, y, paint);

    }

}
```

```java
private void drawAirplanes(Canvas canvas) {

    Paint paint = new Paint();
    paint.setStyle(Paint.Style.FILL_AND_STROKE);
    paint.setStrokeWidth(4);
    paint.setTextAlign(Paint.Align.CENTER);
    paint.setTextSize(cellSize * 0.7f);

    for (Avion avion : aviones) {

        int x = avion.getPosX() * cellSize + cellSize / 2;
        int y = avion.getPosY() * cellSize + cellSize / 2 + (int) (cellSize * 0.25f);

        if (avion.isAtDestination()) {
            paint.setColor(Color.GREEN);

        } else if (avion.isCollided()) {

            paint.setColor(Color.RED);

        } else {

            paint.setColor(Color.BLUE);

        }

        float angle = 0;
```

```
            switch (avion.getDirection()) {

                case NORTH: angle = 0; break;
                case EAST: angle = 90; break;
                case SOUTH: angle = 180; break;
                case WEST: angle = 270; break;

            }

            canvas.save();
            canvas.rotate(angle, x,  py: y - (int) (cellSize * 0.25f));

            if (avion.isCollided() || avion.isAtDestination()) {

                // Dibuja un círculo relleno
                float radio = cellSize * 0.3f;
                canvas.drawCircle(x,  cy: y - (int) (cellSize * 0.25f), radio, paint);

            } else {

                // Dibuja un vil asterisco.
                canvas.drawText( text: "*", x, y, paint);

            }

            canvas.restore();

        }

    }
```

```
}
```

Clase avión:

```java
public class Avion {

    12 usages
    private int posX;
    12 usages
    private int posY;
    9 usages
    private Direction direction;
    4 usages
    private boolean isCollided;
    2 usages
    private boolean isAtDestination = false;

    2 usages
    public Avion(int posX, int posY, Direction direction) {

        this.posX = posX;
        this.posY = posY;
        this.direction = direction;
        this.isCollided = false;

    }
```

```java
    public void mover(int columns, int rows) {

        if (!isCollided) {

            switch (direction) {
                case NORTH: if (posY > 0) posY--; break;
                case SOUTH: if (posY < rows - 1) posY++; break;
                case EAST: if (posX < columns - 1) posX++; break;
                case WEST: if (posX > 0) posX--; break;
            }

        }

    }

    3 usages
    public int getPosX() {
        return posX;
    }

    no usages
    public void setPosX(int posX) {
        this.posX = posX;
    }

    3 usages
    public int getPosY() {
        return posY;
    }
```

```java
public void setPosY(int posY) {
    this.posY = posY;
}

2 usages
public Direction getDirection() {
    return direction;
}

no usages
public void setDirection(Direction direction) {
    this.direction = direction;
}

no usages
public boolean isCollided() {
    return isCollided;
}

2 usages
public void setCollided(boolean collided) {
    isCollided = collided;
}

1 usage
public void setAtDestination(boolean atDestination) {
    isAtDestination = atDestination;
}
```

```java
public boolean isAtDestination() {
    return isAtDestination;
}

2 usages
public boolean isAtBorder(int columns, int rows) {

    switch (direction) {

        case NORTH:
            return posY == 0;
        case SOUTH:
            return posY == rows - 1;
        case EAST:
            return posX == columns - 1;
        case WEST:
            return posX == 0;
        default:
            return false;
    }
}
```

```java
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Avion avion = (Avion) obj;
        return posX == avion.posX && posY == avion.posY && direction == avion.direction;
    }

    @Override
    public int hashCode() {
        int result = posX;
        result = 31 * result + posY;
        result = 31 * result + (direction != null ? direction.hashCode() : 0);
        return result;
    }

    7 usages
    public enum Direction { NORTH, SOUTH, EAST, WEST }

}
```

MainActivity:

```java
public class MainActivity extends AppCompatActivity {
    18 usages
    private VistaAeropuerto vistaAeropuerto;
    2 usages
    private TextView pasosTxv, colisionesTxv;
    3 usages
    private int pasos = 0;
    2 usages
    private int choques = 0;
    6 usages
    private java.util.Stack<List<Avion>> historial;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        vistaAeropuerto = findViewById(R.id.vistaAeropuerto);
        pasosTxv = findViewById(R.id.txtPasos);
        colisionesTxv = findViewById(R.id.txtColisiones);

        findViewById(R.id.btnContinuar).setOnClickListener(v -> nextStep());
        findViewById(R.id.btnPrevio).setOnClickListener(v -> previousStep());
    }

    1 usage
```

```java
private void nextStep() {

    if (hasCollidedAirplanes() || hasBorderAirplanes()) {

        removeCollidedAndBorderAirplanes();

    } else {

        List<Avion> newAirplanes = calculateNewPositions();
        detectCollisions(newAirplanes);
        markBorderAirplanes(newAirplanes);
        if (historial == null) historial = new java.util.Stack<>();
        historial.push(new ArrayList<>(vistaAeropuerto.getAviones()));
        vistaAeropuerto.setAviones(newAirplanes);
        pasos++;
        updateCounters();
        vistaAeropuerto.invalidate();
    }
}

1 usage
private boolean hasCollidedAirplanes() {
    for (Avion avion : vistaAeropuerto.getAviones()) {
        if (avion.isCollided()) return true;
    }
    return false;
}
```

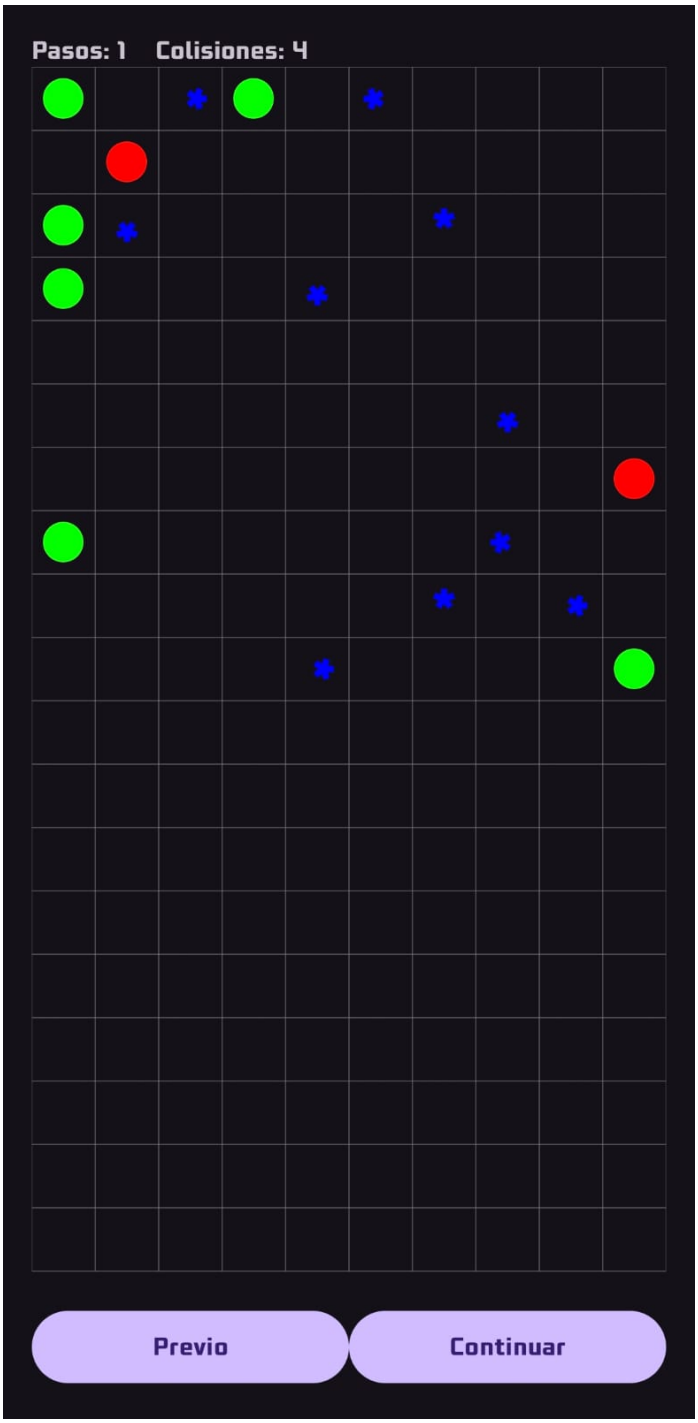
```java
private boolean hasBorderAirplanes() {
    for (Avion avion : vistaAeropuerto.getAviones()) {
        if (avion.isCollided() && avion.isAtBorder(
                vistaAeropuerto.getColumnas(), vistaAeropuerto.getFilas())) return true;
    }
    return false;
}

1 usage
private void removeCollidedAndBorderAirplanes() {
    List<Avion> airplanes = vistaAeropuerto.getAviones();
    airplanes.removeIf(Avion::isCollided);
    vistaAeropuerto.setAviones(airplanes);
    vistaAeropuerto.invalidate();
}

1 usage
private void markBorderAirplanes(List<Avion> airplanes) {
    for (Avion airplane : airplanes) {
        if (!airplane.isCollided() && airplane.isAtBorder(
                vistaAeropuerto.getColumnas(), vistaAeropuerto.getFilas())) {
            airplane.setCollided(true);
            airplane.setAtDestination(true);
        }
    }
}

1 usage
```

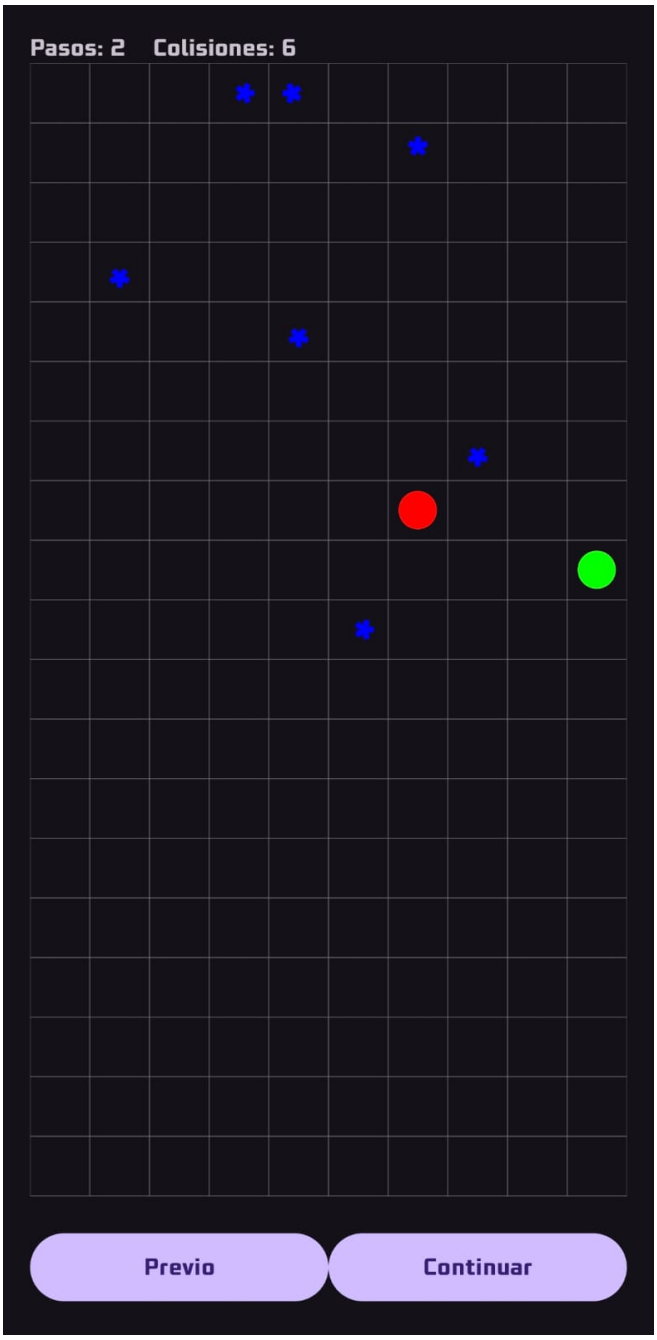
```java
private List<Avion> calculateNewPositions() {
    List<Avion> copy = new ArrayList<>();
    int columns = vistaAeropuerto.getColumnas();
    int rows = vistaAeropuerto.getFilas();

    for (Avion original : vistaAeropuerto.getAviones()) {

        Avion copia = new Avion(original.getPosX(), original.getPosY(), original.getDirection());
        copia.mover(columns, rows);
        copy.add(copia);

    }

    return copy;

}
```

```java
    private List<Avion> detectCollisions(List<Avion> aviones) {

        Map<String, List<Avion>> positions = new HashMap<>();
        List<Avion> collided = new ArrayList<>();

        for (Avion avion : aviones) {

            String key = avion.getPosX() + "-" + avion.getPosY();
            positions.computeIfAbsent(key, k -> new ArrayList<>()).add(avion);

        }

        for (List<Avion> group : positions.values()) {

            if (group.size() > 1) {

                choques += group.size();
                group.forEach(airplane -> {
                    airplane.setCollided(true);
                    collided.add(airplane);

                });

            }

        }
        return collided;
    }
```

```java
    private void updateCounters() {
        pasosTxv.setText(getString(R.string.pasos, pasos));
        colisionesTxv.setText(getString(R.string.colisiones, choques));
    }

    1 usage
    private void previousStep() {

        if (historial != null && !historial.isEmpty()) {

            vistaAeropuerto.setAviones(historial.pop());
            pasos--;
            updateCounters();
            vistaAeropuerto.invalidate();

        }

    }

}
```

Evidencias finales:


Pasos: 1 Colisiones: 4
Previo
Continuar



Conclusión: Esta actividad, sin duda alguna la más difícil de todas, tiene como objetivo dos cosas: la primera, reflejar lo que se aprendió en el semestre de esta asignatura, y la segunda, desarrollar un poco de lógica tanto de programación como de lógica en general, pues la resolución del problema de las colisiones es una que requiere pensar en la forma de solucionarlo. Un ejemplo de la lógica es que se usa en el código un switch para evaluar la dirección cardinal del avión, la cual se expresa mediante valores de un enum; dichos valores son evaluados para poder determinar si el avión ya chocó o si va en buen camino, junto con el uso de estructuras de repetición, en este caso un for para el dibujo de la cuadrícula del layout.