

## Ejercicio: Dedo (v.0.9)

Realice un juego con las siguientes características:

- el dedo se desplaza verticalmente por el borde izquierdo
- dispara un n.º de balas (ajustado en función de num vidas min y max de los enemigos)
- los enemigos se desplazan horizontalmente hacia la izquierda con distintos movimientos
- hay hiperespacios, visualizados en la barra de información de la parte inferior
- múltiples ajustes (ver región Ajustes en clase Mundo)
- //TODO pantallas con elección de balas,records,+enemigos,balas atraviesan,ganar hiperespacios,etc.

### Mundo.java

```
public class Mundo {
```

```
    //region Ajustes
```

```
    public static final float ANCHO=800;
    public static final float ALTO=640;
    public static final float ALTO_BARRA_INFO=80;
    public static final float ALTO_MENOS_DEDO=Mundo.ALTO-Mundo.DEDO_ALTO;
    public static final float ANCHO_MENOS_ENEMIGO=Mundo.ANCHO-Mundo.ENEMIGO_ANCHO;
    public static final int ENEMIGO_MIN VIDAS = 2;
    public static final int ENEMIGO_MAX VIDAS = 11;
    public static int NUM_BALAS=ENEMIGO_MAX VIDAS/2;
    public static final float TIEMPO_ENTRE_ENEMIGOS = 1.5f;
```

```
    public static final int HIPERESPACIOS = 3;
    public static final int HIPERESPACIOS_DURACION=3;
    public static final boolean DEDO_MOVIMIENTO_DURANTE_HIPERESPACIO = true;
    private static final boolean HIPERESPACIOS_PERMITIR_ENCADENAR = false;
```

```
    public static final float BALA_ANCHO=5;
    public static final float BALA_VELOCIDAD=300;
    private static final boolean BALA_DESAPARECE_CON_IMPACTO = true;
    //TODO con false hay que arreglar el problema de que la bala
    // sigue impactando con el enemigo si no ha desaparecido
```

```
    public static final float DEDO_ALTO=80;
    public static final float DEDO_ANCHO=80;
    public static final float DEDO_VELOCIDAD=300;
```

```
    public static final float ENEMIGO_ALTO=40;
    public static final float ENEMIGO_ANCHO=40;
    public static final float ENEMIGO_VELOCIDAD = 200;
    // endregion
```

```
    public static Dedo dedo=new Dedo();
    public static Array<Bala> balas = new Array();
    public static Array<Enemigo> enemigos = new Array();
```

```
    public static StringBuilder strHs=new StringBuilder();
```

```
    public static float stateTime,stateTimeAcabaHiperespacio;
    public static int numHiperespacios;
```

```
    public static void setHiperespacio() {
        if(!Mundo.HIPERESPACIOS_PERMITIR_ENCADENAR && enHiperEspacio()) return;
        if(numHiperespacios-->0)
            stateTimeAcabaHiperespacio=stateTime+Mundo.HIPERESPACIOS_DURACION;
        setStrHs();
    }
```

```
    private static void setStrHs() {
        strHs.setLength(0);
        for(int i=0;i<numHiperespacios;i++)
            strHs.append("H");
    }
```

```
    public static void quitarHiperespacio() { stateTimeAcabaHiperespacio--1; }
    public static boolean enHiperEspacio() { return stateTimeAcabaHiperespacio>0; }
```

```

public static void reset() {
    stateTime=0;
    numHiperespacios=Mundo.HIPERESPACIOS;
    setStrHs();
    quitarHiperespacio();
    dedo.reset();
    balas.clear();
    enemigos.clear();
}

// GESTIÓN DE BALAS (MEJORABLE CON POOL)
public static void nuevaBala() { if(!Mundo.enHiperEspacio() && balas.size<Mundo.NUM_BALAS) balas.add(new
Bala(Mundo.DEDO_ANCHO,dedo.getY()+Mundo.DEDO_ALTO*.6f)); }
public static void eliminaBala(Bala bala) { balas.removeValue(bala,true); }
public static void actualizaBalas(float delta) { for(Bala bala:balas) bala.actualiza(delta);}
public static void dibujaBalas(SpriteBatch sb, ShapeRenderer sr,BitmapFont fuente) { for(Bala bala:balas) bala.dibuja(sb,sr,fuente);}

// GESTIÓN DE ENEMIGOS
public static void nuevoEnemigo() {
    enemigos.add(random.nextBoolean()?new Circulo():new Rectangulo());
}
public static void eliminaEnemigo(Enemigo enemigo) { enemigos.removeValue(enemigo,true); }
public static void actualizaEnemigos(float delta) { for(Enemigo enemigo:enemigos) enemigo.actualiza(delta);}
public static void dibujaEnemigos(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) { for(Enemigo enemigo:enemigos)
enemigo.dibuja(sb,sr,fuente); }

public static float getYaparicionCirculo() {
    return getRandomInterval((int)ALTO_BARRA_INFO,(int)ALTO_MENOS_DEDO);
}

public static void fin() {
    Gdx.app.exit();
    System.exit(0);
}

public static void actualiza(float delta) {
    if(stateTime>stateTimeAcabaHiperespacio) quitarHiperespacio();
    dedo.actualiza(delta);
    actualizaBalas(delta);
    actualizaEnemigos(delta);
}

public static void dibujaBarralInfo(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
    sr.setColor(Color.BROWN);
    sr.rect(0,0,Mundo.ANCHO,Mundo.ALTO_BARRA_INFO);

    fuente.draw(sb,"Tpo: "+(int)stateTime + " "+strHs.toString(),10,Mundo.ALTO_BARRA_INFO*.5f);
}

public static void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
    dibujaBarralInfo(sb,sr,fuente);
    dedo.dibuja(sb,sr,fuente);
    dibujaBalas(sb,sr,fuente);
    dibujaEnemigos(sb,sr,fuente);
}

public static boolean colision() {
    for (Enemigo enemigo : enemigos)
        if (!enHiperEspacio() && Intersector.overlaps(enemigo.getHitBox(), dedo.getHitBox()))
            return true;
    else
        for (Bala bala : balas)
            if (Intersector.overlaps(enemigo.getHitBox(), bala.getHitBox())) {
                if(enemigo.impactoDefinitivo())
                    enemigos.removeValue(enemigo, true);
                if(Mundo.BALA_DESAPARECE_CON_IMPACTO)
                    balas.removeValue(bala,true);
            }
    return false;
}

// RANDOM UTILS
public static Random random=new Random();
public static int getRandomInterval(int min, int max) {
    return random.nextInt(max-min+1)+min;
}
}

```

## JuegoDedo.java

```

public class JuegoDedo extends InputAdapter implements ApplicationListener {
    float stateTimeProximoEnemigo;

    OrthographicCamera camara;
    BitmapFont fuente;
    SpriteBatch sb;
    ShapeRenderer sr;

    @Override public void create () {
        camara=new OrthographicCamera();
        sb=new SpriteBatch();
        sr = new ShapeRenderer();
        sr.setColor(Color.BLACK);
        fuente=new BitmapFont();
        fuente.getData().setScale(1.5f);
        stateTimeProximoEnemigo =0;
        Mundo.reset();
        Gdx.input.setInputProcessor(this);
    }

    @Override public void resize(int width, int height) {
        camara.setToOrtho(false,Mundo.ANCHO,Mundo.ALTO);
        sb.setProjectionMatrix(camara.combined);
        sr.setProjectionMatrix(camara.combined);
    }

    @Override public void render () {
        ScreenUtils.clear(1, 1, 1, 1);
        float delta= Gdx.graphics.getDeltaTime();
        Mundo.stateTime+=delta;

        if(Mundo.stateTime> stateTimeProximoEnemigo) {
            stateTimeProximoEnemigo +=Mundo.TIEMPO_ENTRE_ENEMIGOS;
            Mundo.nuevoEnemigo();
        }

        Mundo.actualiza(delta);
        sr.begin(ShapeRenderer.ShapeType.Line);
        sb.begin();
        Mundo.dibuja(sb,sr,fuente);
        sb.end();
        sr.end();

        if(Mundo.colision()) Mundo.fin();
    }

    @Override public void dispose () { sr.dispose(); }
    @Override public void pause() {}
    @Override public void resume() {}

    @Override public boolean keyDown(int keycode) {
        switch (keycode) { case Input.Keys.DOWN: Mundo.dedo.atras(); break;
                          case Input.Keys.UP: Mundo.dedo.adelante(); break;
                          case Input.Keys.SPACE: Mundo.nuevaBala(); break;
                          case Input.Keys.H: Mundo.setHiperespacio(); break;
                          case Input.Keys.ESCAPE: Mundo.fin(); break;
                        }
        return true;
    }

    @Override
    public boolean keyUp(int keycode) {
        switch (keycode) {
            case Input.Keys.DOWN: if(Gdx.input.isKeyPressed(Input.Keys.UP)) Mundo.dedo.adelante();
                                else Mundo.dedo.para();
                                break;
            case Input.Keys.UP: if(Gdx.input.isKeyPressed(Input.Keys.DOWN)) Mundo.dedo.atras();
                              else Mundo.dedo.para();
                              break;
        }

        return true;
    }
}

```

## Personaje.java

```

abstract class Personaje {
    float x, y, ancho, alto;
    float velocidad;
    int direccion;
    Rectangle hitBox;

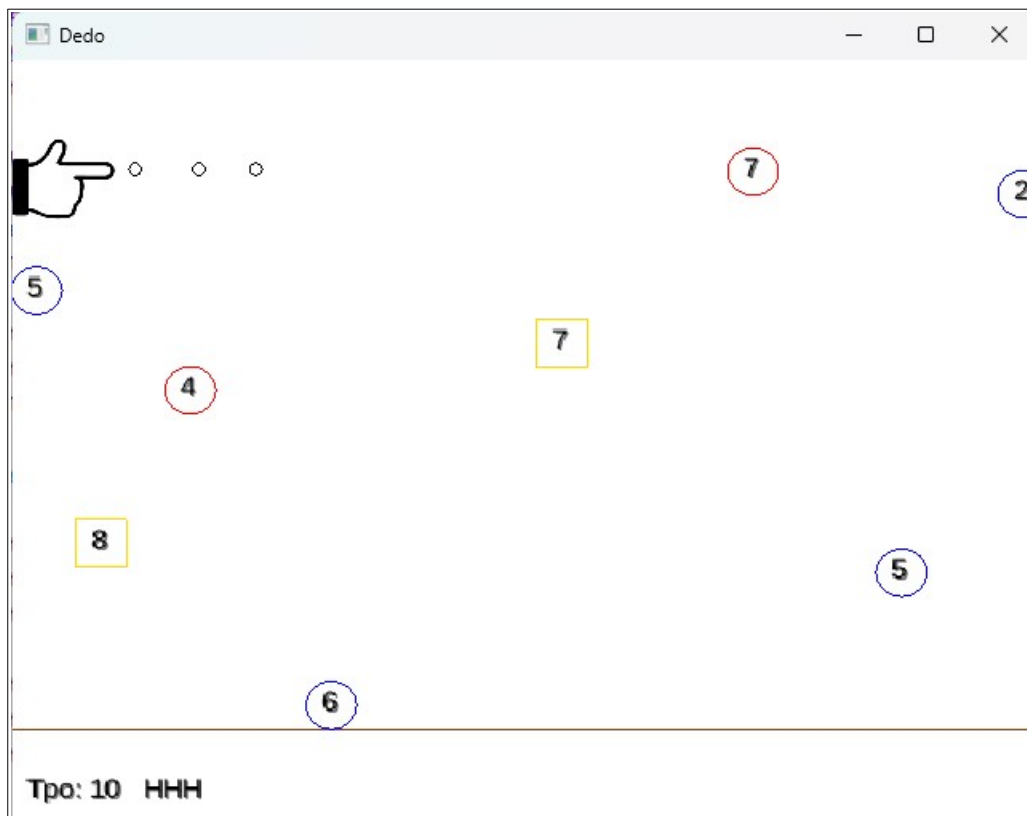
    public Personaje(float x, float y, float ancho, float alto, float velocidad, int dirección) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.velocidad = velocidad;
        this.direccion = dirección;
        this.hitBox = new Rectangle(x, y, ancho, alto);
    }

    public abstract void actualiza(float delta);
    public abstract void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente);

    public void adelante() { direccion = 1; }
    public void atras() { direccion = -1; }
    public void para() { direccion = 0; }

    public Rectangle getHitBox() { return hitBox; }
}

```



## Dedo.java

```

public class Dedo extends Personaje {

    Texture texture;

    public Dedo() {
        super(0,0, Mundo.DEDO_ANCHO,Mundo.DEDO_ALTO,Mundo.DEDO_VELOCIDAD,0);
        texture=new Texture("dedo.png");
    }

    public void reset() {
        x=hitBox.x=0;
        y=hitBox.y=(Mundo.ALTO_MENOS_DEDO-Mundo.ALTO_BARRA_INFO)/2;
        para();
    }

    public float getY() { return y; }

    public void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
        if(!Mundo.enHiperEspacio())
            sb.draw(texture,x,y,ancho,alto);
    }

    public void actualiza(float delta) {
        if(!Mundo.DEDO_MOVIMIENTO_DURANTE_HIPERESPACIO && Mundo.enHiperEspacio()) return;
        y += velocidad * delta * direccion;
        if (y < Mundo.ALTO_BARRA_INFO)
            y = Mundo.ALTO_BARRA_INFO;
        else if (y > Mundo.ALTO_MENOS_DEDO)
            y = Mundo.ALTO_MENOS_DEDO;
        hitBox.y=y;
    }
}

```

## Bala.java

```

public class Bala extends Personaje {
    Texture texture;

    public Bala(float x,float y) {
        super(x, y, Mundo.BALA_ANCHO, 0, Mundo.BALA_VELOCIDAD, 1);
    }

    public void dibuja(SpriteBatch sb, ShapeRenderer sr,BitmapFont fuente) {
        sr.setColor(Color.BLACK);
        sr.circle(x, y, Mundo.BALA_ANCHO);
    }

    public void actualiza(float delta) {
        x += velocidad * delta * direccion;
        if (x > Mundo.ANCHO) Mundo.eliminaBala(this);
        hitBox.x=x;
    }
}

```

## Enemigo.java

```
abstract public class Enemigo extends Personaje {
    int numVidas= Mundo.getRandomInterval(Mundo.ENEMIGO_MIN_VIDAS,Mundo.ENEMIGO_MAX_VIDAS);
    float xCentro,yCentro;

    public Enemigo(float x, float y, float ancho, float alto, float velocidad, int dirección) {
        super(x, y, ancho, alto, velocidad, dirección);
    }

    public void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
        fuente.setColor(Color.BLACK);
        fuente.draw(sb,numVidas+"",x+(ancho*.3f),y+(alto*.8f));
    }

    public boolean impactoDefinitivo() { return --numVidas==0; }
}
```

## Rectangulo.java

```
public class Rectangulo extends Enemigo {
    public Rectangulo() {
        super(Mundo.ANCHO,Mundo.getYaparicionCirculo(), Mundo.ENEMIGO_ANCHO, Mundo.ENEMIGO_ALTO,
Mundo.ENEMIGO_VELOCIDAD, 1);
    }

    @Override public void actualiza(float delta) {
        x -= delta * velocidad * direccion;
        hitBox.x=x;
        if(x<-ancho) Mundo.eliminaEnemigo(this);
    }

    @Override public void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
        super.dibuja(sb,sr,fuente);
        sr.setColor(Color.GOLD);
        sr.rect(x,y,ancho,alto);
    }
}
```

## Circulo.java

```
public class Circulo extends Enemigo {
    boolean baja;

    public Circulo() {
        super(Mundo.ANCHO,Mundo.getYaparicionCirculo(), Mundo.ENEMIGO_ANCHO, Mundo.ENEMIGO_ALTO,
Mundo.ENEMIGO_VELOCIDAD, 1);
        baja=Mundo.random.nextBoolean();
    }

    @Override public void actualiza(float delta) {
        if(x<0) {
            y += delta * velocidad * (baja ? -1 : 1);
            if(y<Mundo.ALTO_BARRA_INFO || y>Mundo.ALTO)
                Mundo.eliminaEnemigo(this);
            hitBox.y=y;
        }
        else {
            x -= delta * velocidad * direccion;
            hitBox.x=x;
        }
    }

    @Override public void dibuja(SpriteBatch sb, ShapeRenderer sr, BitmapFont fuente) {
        super.dibuja(sb,sr,fuente);
        sr.setColor(baja?Color.RED:Color.BLUE);
        float radio=Mundo.ENEMIGO_ANCHO/2;
        sr.circle(x+radio,y+radio, radio);
    }
}
```