Universidad Autónoma de San Luis Potosí

Facultad de ingeniería

Área de Ciencias de la Computación

Materia: Tecnología Orientada a Objetos

Manual del Programador

Equipo #3

Pandemic

Sebastián Carrillo Liñán	295197
Josué Israel Esquivel Garza	281876
Adán Alejandro Sánchez de Santiago	298491
Kevin Ávila Rodríguez	300136
Armando Ortega Flores	294680

Índice

Objetivo

Este es el trabajo de fin de semestre de la materia de Tecnología Orientada a Objetos en el cual el objetivo es hacer un videojuego en la plataforma de Greenfoot en donde se puedan plasmar los conocimientos aprendidos en la materia durante todo el semestre.

Introducción

La idea principal del juego esta basada en la situación mundial actual del virus del COVID-19 en donde el personaje tiene que cuidarse de las personas infectadas que arrojan bacterias y el se defiende arrojando cubre bocas y recogiendo elementos de ayuda como gel anti bacterial que te le da mas fuerza a los proyectiles y otro elemento de ayuda es un cubre bocas que hace que le suba la vida.

Los enemigos se moverán de lado a lado de la pantalla apareciendo en lugares aleatorios, y arrojando proyectiles al lado que se encuentre el jugador, habrá diferentes enemigos, teniendo la misma función pero será mas difícil eliminarlos.

Esto lo hará durante 3 diferentes niveles en los que se determinan por el numero de enemigos que eliminas.

En el cuarto y ultimo nivel el enemigo será diferente, lo que cambia es que se moverá de arriba hacia abajo, sus proyectiles harán mayor daño y será mas complicado eliminarlo

World Classes

Menu

La clase Menu es una subclase de World.

En el constructor se le dan las dimensiones y se manda a llamar al método PreparaMundo() en donde se agregan los objetos Star, Back Help, y select.

```
import greenfoot.*;  // (World, Actor, GreenfootIm

public class Menu extends World
{
    private int opcion=0;// 0=star 1=back 2=help
    public Menu()
    {
        super(1150, 650, 1);
        PreparaMundo();
    }

    private void PreparaMundo()
    {
      addObject(new Star(),1000,500);
      addObject(new Back(),1000,560);
      addObject(new help(),1000,620);
      select s=new select();
      addObject(s,700,500);
    }
}
```

Control

La clase control es una subclase de World.

En el constructor se le dan las dimensiones y se manda a llamar el método PreparaMundo() en donde se agrega el objeto arrow.

```
import greenfoot.*;  // (World, Actor, GreenfootIr

public class control extends World
{
    /**
     * Constructor for objects of class control.
     *
     */
    public control()
    {
        super(1150, 650, 1, false);
        PreparaMundo();
    }

    private void PreparaMundo()
    {
        arrow a=new arrow();
        addObject(a,1000,50);
    }
}
```

MyWorld

La clase MyWorld es una subclase de World.

Al inicio de la clase se encuentran las declaraciones de variables y creación de objetos.

```
import greenfoot.*;
import java.util.ArrayList;

public class MyWorld extends World
{
    private int scenarioNumber = 0;
    private int enemyListTop = 0;
    private int enemyListMax = 10;
    private int healthPackageMax = 0;
    private int dammageBoost = 0;
    private boolean bossSpawned = false;
    Score score = new Score();
    DammageCounter dammageImage = new DammageCounter();
    Player mainPlayer = new Player();
```

Después esta el constructor en donde se le dan las dimensiones y manda a llamar al método initializeworld().

```
    public MyWorld()
    {
        super(1200, 800, 1, false);

        initializeworld();
    }
```

En el método initializeworld() se agregan los objetos mainPlayer, score y dammageImage.

```
    public void initializeworld()
    {
        addObject(mainPlayer, 20, 600);
        addObject(score, 120, 40);
        addObject(dammageImage, 120, 75);
    }
```

En el método act se mandan a llamar los métodos spawnEnemies(), spawnItem() y checkLevel().

```
    public void act()
    {
        spawnEnemies();
        spawnItem();
        checkLevel();
    }
```

En el método getScore() lo único que hace es regresar los puntos que lleva el jugador.

```
    public Score getScore()
    {
        return score;
    }
```

El método spawnEnemies(), lo primero que hace es checar en que escenario está según la variable scenarioNumber el cual empieza en 0.

Dentro del escenario 0 se crea un número máximo de enemigos de 5 y se crean los enemigos de tipo 1.

```
public void spawnEnemies()
{
    if(scenarioNumber == 0)
    {
        if((Greenfoot.getRandomNumber(1000) < 20) && (enemyListTop < 5))
        {
            Enemy p = new Infected1(1, 2);
            enemyListTop++;
            addObject(p, 1220, Greenfoot.getRandomNumber(500) + 300);
        }
    }
```

Dentro del escenario 1 se crea un número máximo de enemigos de 7 y se genera un número aleatorio el cual si es menor a 4 el tipo de enemigo será del tipo 2 el cual genera un daño de 120, de lo contrario se generarán enemigos de tipo 1.

```
else if(scenarioNumber == 1)
{
    if((Greenfoot.getRandomNumber(1000) < 20) && (enemyListTop < 7))
    {
        int aux = Greenfoot.getRandomNumber(10);
        Enemy p = null;
        if(aux < 4) p = new Infected2(2, 5, 120);
        else p = new Infected1(1, 5);
        enemyListTop++;
        addObject(p, 1220, Greenfoot.getRandomNumber(500) + 300);
    }
}
```

Dentro del escenario 2 se crea un número máximo de enemigos de 9 y se genera un número aleatorio el cual si es menor a 5 el tipo de enemigo será del tipo 2 el cual hace un daño de 130, de lo contrario se generarán enemigos de tipo 1.

```
else if(scenarioNumber == 2)
{
    if((Greenfoot.getRandomNumber(1000) < 20) && (enemyListTop < 9))
    {
        int aux = Greenfoot.getRandomNumber(10);
        Enemy p = null;
        if(aux < 5) p = new Infected2(2, 5, 130);
        else p = new Infected1(1, 6);
        enemyListTop++;
        addObject(p, 1220, Greenfoot.getRandomNumber(500) + 300);
    }
}
```

Dentro del escenario 3 se crea un número máximo de enemigos de 11 y se genera un número aleatorio el cual si es menor a 7 el tipo de enemigo será del tipo 2 el cual hace un daño de 150, de lo contrario se generarán enemigos de tipo 1.

```
else if(scenarioNumber == 3)
{
    if((Greenfoot.getRandomNumber(1000) < 20) && (enemyListTop < 11))
    {
        int aux = Greenfoot.getRandomNumber(10);
        Enemy p = null;
        if(aux < 7) p = new Infected2(2, 6, 150);
        else p = new Infected1(1, 7);
        enemyListTop++;
        addObject(p, 1220, Greenfoot.getRandomNumber(500) + 300);
    }
}
```

Dentro del escenario 4 (escenario final) checa si no se ha llegado al jefe final anteriormente, si no lo ha enfrentado se agregará a la pantalla el jefe final y se marcará que ya se enfrento

```
else if(scenarioNumber == 4)
{
    if(bossSpawned == false)
    {
        addObject(new Boss(3, 1, 600), 1100, 600);
        bossSpawned = true;
    }
}
}
```

Después sigue el método llamado spawnItem(), en donde se checa si la vida es menor a 100 y el número de ítems de paquetes de ayuda son menores a 2, se agregará un ítem y se registra el número de ítems en el escenario.

```
public void spawnItem()
{
    if((Greenfoot.getRandomNumber(1000) < 20) && (mainPlayer.getLife() < 100) && (healthPackageMax < 2))
    {
        Item item = new HealthPack(1);
        addObject(item,Greenfoot.getRandomNumber(1200), Greenfoot.getRandomNumber(500) + 300);
        healthPackageMax++;
    }
```

Después se checa si la vida es menor a 100 y el número de ítems de mejora de daño son menores a 1, se agregará un ítem y se registra el número de ítems en el escenario.

Después checa si la vida es menor a 150 (se permite que el judador tenga mas vida) y está en el escenario 4, se permitirá que haya más ítems de ayuda y mejoras de daño.

```
if((Greenfoot.getRandomNumber(1000) < 20) && (dammageBoost < 1))
{
    Item item = new Dammage(2);
    addObject(item,Greenfoot.getRandomNumber(1200), Greenfoot.getRandomNumber(500) + 300);
    dammageBoost++;
}
if((Greenfoot.getRandomNumber(1000) < 10) && (mainPlayer.getLife() < 150) && (scenarioNumber == 4))
{
    Item health = new HealthPack(1);
    Item dammage = new Dammage(2);
    addObject(health, Greenfoot.getRandomNumber(1000), Greenfoot.getRandomNumber(500) + 300);
    addObject(dammage, Greenfoot.getRandomNumber(1000) , Greenfoot.getRandomNumber(500) + 300);
    dammageBoost++;
    healthPackageMax++;
}
}
```

En el método de checkLevel() lo que hace es que confirma el numero de enemigos eliminados y en que escenario te encuentras, en el momento que sea verdadero se manda a llamar al método resetLevel()

En el nivel 0 se tienen que matar 5 enemigos para que se mande a llamar al método mencionado, para el nivel 1 se tienen que haber matado a 12 enemigos en total es decir en el nivel 1 tienen que matar a 7, en el nivel 2 tienen que haber matado 21 enemigos y en el nivel 3 tienen que haber matado a 32 enemigos en total.

```
public void checkLevel()
{
    if((mainPlayer.getEnemiesKilled() == 5) && (scenarioNumber == 0)) resetLevel();
    else if((mainPlayer.getEnemiesKilled() == 12) && (scenarioNumber == 1)) resetLevel();
    else if((mainPlayer.getEnemiesKilled() == 21) && (scenarioNumber == 2)) resetLevel();
    else if((mainPlayer.getEnemiesKilled() == 32) && (scenarioNumber == 3)) resetLevel();
}
```

En el método resetLevel() se regresa a 0 el numero de paquetes de ayuda y de mejora de daño, se aumenta el numero del escenario, el numero de enemigos se declara en 0 y se manda a llamar al método changeImageBackground().

```
public void resetLevel()
{
    dammageBoost = 0;
    healthPackageMax = 0;
    scenarioNumber++;
    enemyListTop = 0;
    changeImageBackground();
}
```

En el método changeImageBackground() empieza declarando la imagen en NULL porque no sabe a cual va a cambiar, después se checa que escenario es el actual y

```
public void changeImageBackground()
{
    GreenfootImage bg = null;
    if(scenarioNumber == 0) bg = new GreenfootImage("bg 4_00632.png");
    else if(scenarioNumber == 1) bg = new GreenfootImage("escenario1.png");
    else if(scenarioNumber == 2) bg = new GreenfootImage("escenario2.png");
    else if(scenarioNumber == 3) bg = new GreenfootImage("escenario3.png");
    else bg = new GreenfootImage("bg 10_00362.png");
    bg.scale(getWidth(), getHeight());
    setBackground(bg);
}
```

se pone la imagen, después se le pone la escala adecuada y se pone la imagen adecuada en la pantalla.

En el método de getDammage() se accede a la instancia del jugador y regresa el daño.

En el método de addEnemiesKilled() se envían los enemigos eliminados a la instancia del jugador.

En el método de getPlayerX() es para obtener las coordenadas en x del jugador

```
    public int getDammage()
    {
        return mainPlayer.getDammage();
    }

    public void addEnemiesKilled()
    {
        mainPlayer.addEnemiesKilled();
    }

    public int getPlayerX()
    {
        return mainPlayer.getPX();
    }
}
```

Game Over

La clase GameOver es una subclase de World

Lo primero que hace es poner una imagen en la pantalla

En el constructor se le dan las dimensiones en lo que lo tiene que hacer y se reproduce una canción.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseI
import java.awt.Color;

public class GameOver extends World
{

    /**
     * Constructor for objects of class GameOver.
     *
     */
    private GreenfootImage gameOver = new GreenfootImage("GAMEOVER1.jpeg");
    public GameOver()
    {

        super(1200, 800, 1);
        Greenfoot.playSound("national-anthem-of-ussr.mp3");
    }
}
```

Game Win

La clase de GameWin es una subclase de World, en donde el constructor le da las dimensiones de la pantalla

```
import greenfoot.*;  // (World, Actor, GreenfootIm

/**
 * Write a description of class GameWin here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class GameWin extends World
{

    /**
     * Constructor for objects of class GameWin.
     *
     */
    public GameWin()
    {
        super(1200, 800, 1);
    }
}
```

Actor Classes

Bullet

La clase bullet es una subclase de Actor

Al inicio se declara una variable booleana que dicta la dirección siendo true para la izquierda y false para la derecha.

En el constructor se recibe el valor de dirección, en este se le asigna el tamaño de la imagen y se le asigna la dirección a la variable declarada en la misma clase.

En el método act() se manda a llamar projectilMove() y removeFromWorld()

En el método projectilMove() checa si la dirección es false y se pone en la ubicación y se suma la x para avanzar, y de ser verdadero se le resta en x .

En el método removeFromWorld() checa si esta fuera de los límites de la pantalla y se elimina el objeto.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInf

public class Bullet extends Actor
{
    private boolean direction; // true->left && false->right

    public Bullet(boolean direction)
    {   getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
        this.direction = direction;
    }

    public void act()
    {
        projectileMove();
        removeFromWorld();
    }

    public void projectileMove()
    {
        if(direction == false)setLocation(getX() + 7, getY());
        else setLocation(getX() - 7, getY());
    }

    public void removeFromWorld()
    {
        if((getX() >= 1200) || (getX() <= 0)) getWorld().removeObject(this);
    }
}
```

Enemy Bullet

La clase EnemyBullet es una subclase de Actor

Al inicio se declaran 3 variables siendo el daño, el tipo de enemigo que esta disparando y la dirección (booleana y funciona igual que en la clase Bullet).

En el constructor se recibe la dirección y el tipo de enemigo. Primero se crea la imagen, se asigna la dirección y el tipo de enemigo y se le asigna el daño que hace.

En el método act() se manda a llamar projectilMove() y removeFromWorld()

En el método projectilMove() checa si la dirección es false y se pone en la ubicación y se suma la x para avanzar, y de ser verdadero se le resta en x .

En el método removeFromWorld() checa si esta fuera de los límites de la pantalla y se elimina el objeto.

En el método getBulletDamage() regresa el daño que hace la bala.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInf

public class EnemyBullet extends Actor
{
    private boolean direction; // true->left && false->right
    private int dammage;
    private int enemyTypeShooted;

    public EnemyBullet(boolean direction, int enemyType)
    {
        getImage().scale(getImage().getWidth()/3,getImage().getHeight()/3);
        this.direction = direction;
        enemyTypeShooted = enemyType;
        dammage = (enemyType * 10);
    }

    public void act()
    {
        projectileMove();
        removeFromWorld();
    }

    public void projectileMove()
    {
        if(direction == false)setLocation(getX() + 7, getY());
        else setLocation(getX() - 7, getY());
    }

    public void removeFromWorld()
    {
        if((getX() >= 1200) || (getX() <= 0)) getWorld().removeObject(this);
    }

    public int getBulletDammage()
    {
        return dammage;
    }
}
```

Player

La clase Player es una subclase de Actor.

Al inicio de la clase se declaran todas las diferentes variables que se utilizan.

En el constructor se le da la escala de la imagen y se inicia una canción

En el método de act() se manda a llamar a moveArround(), fireGun() y getHittedByEnemy()

```
import greenfoot.*;

public class Player extends Actor
{
    private String player_name;
    private int steps = 4;
    private boolean canFire = true;
    private int life = 100;
    private boolean direction; //true->left && false->right
    private boolean stunned = false;
    private int dammage = 10;
    private int enemiesKilled = 0;
    private GreenfootImage runIzq = new GreenfootImage("Prota-Izq-Walk_00000.png");
    private GreenfootImage runDer = new GreenfootImage("Prota-Der-Walk_00000.png");
    private int frame =1;//decimos que la imagen va de lleno a derecha
    public Player()
    {
        getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
        Greenfoot.playSound("duel-of-the-fates.mp3");
    }

    public void act()
    {
        moveAround();
        fireGun();
        getHittedByEnemy();
    }
```

En el método moveArround() se asignan los botones que se usarán para mover al personaje checando la tecla que se presiona, y el limite que se le permite estar al personaje.
Al presionar “w” se asigna la ubicación en X y Y y en Y se le restan los pasos (siendo 4 declarados anteriormente).
Al presionar “s” se asigna la ubicación en X y Y y en Y se le suman los pasos.

```
public void moveAround()
{
    String shootButton = null;
    if((Greenfoot.isKeyDown("w") || Greenfoot.isKeyDown("W")) && (getY() > 300))
    {
        setLocation(getX(), getY() - steps);
    }
    if((Greenfoot.isKeyDown("s") || Greenfoot.isKeyDown("S")) && (getY() < 780))
    {
        setLocation(getX(), getY() + steps);
    }
    if((Greenfoot.isKeyDown("a") || Greenfoot.isKeyDown("A")) && (getX() > 20))
    {
        direction = true;
        move(-steps);
        //animateL();
    }
    if((Greenfoot.isKeyDown("d") || Greenfoot.isKeyDown("D")) && (getX() < 1180))
    {
        direction = false;
        move(steps);
        //animateR();
    }
    if((Greenfoot.isKeyDown("e") || Greenfoot.isKeyDown("E"))) pickUpItem();
}
```

Al presionar “a” se asigna true a la dirección, es decir que va a la izquierda, y se le restan los pasos sobre la x.
Al presionar “d” se asigna false a la dirección, es decir que va a la derecha, y se le suman los pasos sobre la x.

Si se presiona la tecla “e” se manda a llamar al método pickUpItem()

En el método fireGun() checa si se esta presionando la tecla “ESPACE” y es true la acción de que puede disparar, de caso de ser cierto se agrega la imagen de la bala creando un objeto Bullet, después se pone como false la acción de que puede disparar y hay un sonido, en el caso que no se presione la tecla “ESPACE” es true la acción de disparo.

```
public void fireGun()
{
    if(Greenfoot.isKeyDown("space") && canFire == true)
    {
        getWorld().addObject(new Bullet(direction), getX(), getY());
        canFire = false;
        Greenfoot.playSound("shot.mp3");
    }
    else if(!Greenfoot.isKeyDown("space")) canFire = true;
}
```

En el método pickUpItem() se crea una instancia de la clase Item en la variable itemPicked, en el caso de que no sea falso itemPicked, se asigna al uso del item de la subclase de item itemFunction. Si itemUse es igual a uno es el paquete de ayuda que se suman puntos a la vida del jugador y se reproduce un sonido. Si itemUse es igual a 2, se le suma al daño que hace el jugador y se reproduce un sonido. Después se elimina la imagen de la pantalla.

```
public void pickUpItem()
{
    Item itemPicked = getOneIntersectingObject(Item.class);
    if(itemPicked != null)
    {
        int itemUse = itemPicked.itemFunction();
        if(itemUse == 1)
        {
            HealthPack item = getOneIntersectingObject(HealthPack.class);
            life += item.healthRecover();
            Greenfoot.playSound("HealthUp.mp3");
        }
        else if(itemUse == 2)
        {
            Dammage item = getOneIntersectingObject(Dammage.class);
            dammage += item.getDammage();
            Greenfoot.playSound("PowerUp.mp3");
        }
        getWorld().removeObject(itemPicked);
    }
}
```

En el método animateR() checa si frame es igual a uno y definir la imagen con runDer para después definir frame como 0

```
public void animateR()
{
    if(frame ==1)
    {
    setImage(runDer);
    frame =0;
    }
}
```

En el método animateL() checa si frame es igual a 0 y definir la imagen con runIzq para después definir frame como 1.

```
public void animateL()
{
     if(frame ==0)
    {
    setImage(runIzq);
    frame =1;
    }
}
```

El método getHittedByEnemy() confirmacon la variable hit si hay intersección con el objeto EnemyBullet, si hit no es null, la baja la vida dependiendo del daño que hace la bala, y se elimina la imagen de la pantalla, y se manda a llamar al metodo checkLife().

```
public void getHittedByEnemy()
{
    EnemyBullet hit = getOneIntersectingObject(EnemyBullet.class);
    if(hit != null)
    {
        life -= hit.getBulletDammage();
        getWorld().removeObject(hit);
    }
    checkLife();
}
```

En el método checkLife() checa si life es menor o igual a 0, en caso de ser cierto se remueve el personaje y se cambia el mundo a GameOver() y se detiene el programa.

En el método getLife() te regresa la vida del jugador.

En el método de getDammage() te regresa el daño que hace el jugador.

En el método de getEnemiesKilled() te regresa el número de enemigos eliminados.

En el método de addEnemiesKilled() se aumenta en uno el número de enemigos eliminados.

En el método gatPX() te regresa la ubicación en x.

```
public void checkLife()
{
public int getLife()
{
    return life;
}

public int getDammage()
{
    return dammage;
}

public int getEnemiesKilled()
{
    return enemiesKilled;
}

public void addEnemiesKilled()
{
    enemiesKilled++;
}

public int getPX()
{
    return getX();
}
}
```

Select

La clase select es una subclase de Actor

Primero se declara una variable opción en 0

El constructor re escala la imagen del selector.

En el metodo de act() se manda a llamar el metodo de mover().

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)


public class select extends Actor
{
    private int op=0;
    public  select(){
        getImage().scale(getImage().getWidth()/3,getImage().getHeight()/3);
    }

    public void act()
    {
        mover();
    }
```

En el método de mover() se declaran las variables x y y con las coordenadas del objeto. Si se presiona la tecla “s” se aumenta uno en y (baja) y si se presiona “w” se resta uno en y (sube). Después se limita en y entre 490 y 620 que es la ubicación del menú. Después se obtiene la ubicación, checa si se presiona la tecla “ENTER” y si esta entre las coordenadas 500 y 420 se inicia el juego llamando a MyWorld(), si esta entre 560 y 520 se detiene el juego y si esta entre 620 y 580 se manda a llamar el mundo de control().

```
    public void mover()
    {
        int x=getX(),y=getY();
        if(Greenfoot.isKeyDown("S"))y++;
        if(Greenfoot.isKeyDown("w"))y--;
        if(y<490)
        {
            y=490;
        }
        if(y>=620)
        {
            y=620;
        }
        setLocation(x,y);
        if(Greenfoot.isKeyDown("enter")){
            if(y<500 && y >420)
            {
                Greenfoot.setWorld(new MyWorld());
            }else if(y<560 && y>520)
            {
                Greenfoot.stop();
            }else
            if(y<620 && y>580)
            {
                Greenfoot.setWorld(new control());
            }
        }
    }
}
```

Arrow

La clase arrow es una subclase de Actor

En el constructor se re escala la imagen

En el método act() se manda a llamar al metodo mover()

En el método mover() checa si se esta presionando la tecla de “ESPACE”, en el caso de ser cierto, se manda a llamar al mundo de Menu().

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and Mou

/**
 * Write a description of class arrow here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class arrow extends Actor
{

    public arrow()
    {
        getImage().scale(getImage().getWidth(),getImage().getHeight());
    }
     public void act()
    {
        mover();
    }

    public void mover()
    {
        if(Greenfoot.isKeyDown("space"))
        {
            Greenfoot.setWorld(new Menu());
        }
    }
}
```

Dammage Counter

La clase DammageCounter es una subclase de Actor.

Primero se inicializa la variable dammage.

En el constructor se crea una imagen con el texto “Dammage: “, con un tamaño de 30 y de color rojo sin relleno (null).

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.awt.Color;

public class DammageCounter extends Actor
{
    private int dammage;
    public DammageCounter()
    {
        setImage(new GreenfootImage("Dammage: ", 30, Color.RED, null));
    }

    public void act()
    {
        getCounterDammage();
        setImage(new GreenfootImage("Dammage: " + dammage, 30, Color.RED, null));
    }
```

En el método de act() manda a llamar al método getCounterDammage() y se crea una imagen con el texto “Dammage: “ y el numero de daño, de tamaño 30, de color rojo y sin relleno.

En el método getCounterDammage() donde se define una instancia del mundo MyWorld() y accede al daño del jugador.

```
    public void getCounterDammage()
    {
        MyWorld world = (MyWorld)getWorld();
        dammage = world.getDammage();
    }
}
```

Item

La clase Item es una subclase de Actor

Primero se declara una variable itemType y se iguala a 1.

En el constructor recibe itemType y lo asigna a la variable del mismo nombre, después re escala la imagen.

En el método itemFunction() regresa el tipo de item.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseIn

public class Item extends Actor
{
    private int itemType = 1;

    public Item(int itemType)
    {
        this.itemType = itemType;
        getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
    }

    public void act()
    {
        // Add your action code here.
    }

    public int itemFunction()
    {
        return itemType;
    }
}
```

Dammage

La clase Dammage es una subclase de Item.

Primero se declara una variable dammageBoost y se iguala a 5.

El constructor recibe itemType y se manda a la superclase.

El método de getDammage() regresa el valor en dammageBoost.

```
import greenfoot.*;  // (World, Act

/**
 * Write a description of class Dam
 *
 * @author (your name)
 * @version (a version number or a
 */
public class Dammage extends Item
{
    private int dammageBoost = 5;

    public Dammage(int itemType)
    {
        super(itemType);
    }

    public int getDammage()
    {
        return dammageBoost;
    }
}
```

Health Pack

La clase HealthPack es una subclase de Item.

Primero se declara la variable health y se iguala a 10.

En el constructor se recibe itemType y se manda a la superclase.

En el método healthRecover() regresa el valor en health.

```
import greenfoot.*;  // (World, Actor,

public class HealthPack extends Item
{
    private int health = 10;

    public HealthPack(int itemType)
    {
        super(itemType);
    }

    public void act()
    {
        // Add your action code here.
    }

    public int healthRecover()
    {
        return health;
    }
}
```

Score

La clase Score es una subclase de Actor.

Primero se declara una variable score en 0.

En el constructor se re escala la imagen.

En el metodo act() se crea una imagen con un texto “Score: “ de tamaño 50, de color verde y sin relleno

En el metodo addScore() recibe score_to_add el cual se le suma a score.

El metodo getScore() regresa el valor en score.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.awt.Color;

public class Score extends Actor
{
    int score = 0;
    public Score()
    {
        setImage(new GreenfootImage("Score: ", 50, Color.GREEN, null));
    }

    public void act()
    {
        setImage(new GreenfootImage("Score: " + score, 50, Color.GREEN, null));
    }

    public void addScore(int score_to_add)
    {
        score += score_to_add;
    }

    public int getScore()
    {
        return score;
    }
}
```

Enemy

La clase Enemy es una subclase de Actor.

Primero se declaran las variables.

En el primer constructor se recibe el tipo de enemigo y la velocidad y se asignan a las variables enemyType y speed.

En el segundo constructor se recibe el tipo de enemigo, la velocidad y la vida y se asignan a las variables enemyType, speed y life.

En el metodo receiveDamage() recibe la vida que se va a reducir, el valor que entró se le resta a la variable life y se manda a llamar al metodo checkLife().

```
import greenfoot.*;

public class Enemy extends Actor
{
    private boolean removed = false;
    private boolean direction = true; //true->left &&
    private int life = 100;
    private int enemyType;
    private int speed;

    public Enemy(int enemyType, int speed)
    {
        this.enemyType = enemyType;
        this.speed = speed;
    }

    public Enemy(int enemyType, int speed, int life)
    {
        this.enemyType = enemyType;
        this.speed = speed;
        this.life = life;
    }

    public void receiveDamage(int lifeToReduce)
    {
        life -= lifeToReduce;
        checkLife();
    }
```

El metodo checkLife() checa si la vida es mayor igual a 0, de ser cierto crea una nueva instancia del mundo MyWorld, crea una instancia de la barra de puntos partir de la instancia del mundo, al mundo le añade un enemigo eliminado, se suman los puntos por el tipo de enemigo eliminado, se elimina el objeto del mundo y se reproduce el sonido de muerte.

```
    public void checkLife()
    {
        if(life <= 0)
        {
            MyWorld myWorld = (MyWorld)getWorld();
            Score score = myWorld.getScore();
            myWorld.addEnemiesKilled();
            score.addScore(enemyType * 10);
            getWorld().removeObject(this);
            playDeathSound();
            if(enemyType==3){
                Greenfoot.setWorld(new GameWin());
                Greenfoot.stop();
            }
        }

    }
```

El método moveEnemyLeft() se obtiene la ubicación y se le resta la velocidad, después se llama al método de checkDirection().

El método moveEnemyLeft() se obtiene la ubicación y se le suma la velocidad, después se llama al método de checkDirection().

El método checkDirection()checa si la ubicación en X es menor igual a 0, la dirección se declara false (derecha), y si es mayor igual a 1200 se declara true (izquierda).

```
public void moveEnemyLeft()
{
    setLocation(getX() - speed, getY());
    checkDirection();
}

public void moveEnemyRight()
{
    setLocation(getX() + speed, getY());
    checkDirection();
}

public void checkDirection()
{
    if((getX() <= 0)) direction = false;
    else if(getX() >= 1200)  direction = true;
}
```

En el método playSound() checa que tipo de enemigo es y suena un sonido diferente para cada uno.

```
public void playDeathSound()
{
    if(enemyType == 1) Greenfoot.playSound("ZombieDeath.mp3");
    else if(enemyType == 2) Greenfoot.playSound("Infected2Hitted.mp3");
}
```

En el método changeDirection() checa la dirección, false-derecha, true-izquierda, y cambia la dirección.

```
public void changeDirection()
{
    if(direction == false) direction = true;
    else direction = false;
}
```

En el método enemyShoots() se genera un numero aleatorio, se obtiene la dirección del jugador con el método getPlayerDirection(), y depende de que lado esté se generara una bala en el mundo.

```
public void enemyShoots()
{
    if(Greenfoot.getRandomNumber(1000) < (enemyType * 2))
    {
        boolean playerDirection = getPlayerDirection();
        getWorld().addObject(new EnemyBullet(playerDirection, enemyType), getX(), getY());
    }
}
```

En el método getEnemyType() regresa el tipo de enemigo.

En el método de getPlayerDirection() se crea una instancia del mundo MyWorld y se declara una variable aux igualada a 0, la ubicación del jugador en X en el mundo se asigna a aux, si la ubicación del jugador es menor a la del enemigo regresa true, de ser contrario regresa falso.

El metodo getDirection() regresa la direccion del enemigo.

En el metodo de getspeed() regresa la velocidad del enemigo.

En el metodo de getDammage() regresa el producto del tipo de enemigo y 10.

```
    public int getEnemyType()
    {
        return enemyType;
    }

    public boolean getPlayerDirection()
    {
        MyWorld myWorld = (MyWorld)getWorld();
        int aux = 0;
        aux = myWorld.getPlayerX();
        if(aux < getX()) return true;
        else return false;
    }

    public boolean getDirection()
    {
        return direction;
    }

    public int getspeed()
    {
        return speed;
    }

    public int getDammage()
    {
        return (enemyType * 10);
    }
}
```

Boss

La clase de Boss es una subclase de Enemy.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo

/**
 * Write a description of class Boss here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Boss extends Enemy
{
    public Boss(int type, int speed, int life)
    {
        super(type, speed, life);
        getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
    }
```

El constructor recibe el tipo de enemigo, velocidad y vida, donde lo regresa a la superclase y re escala la imagen.

En el método act() se manda a llamar al metodo moveBoss(), enemyShoots() y hitBullet().

En el método de moveUp() se asigna la ubicación y se le resta la velocidad en y.

En el método de moveDown() se asigna la ubicación y se le suma la velocidad en y.

En el método de checkDirectionToMove() checa si el Boss esta menor igual a 300 o mayor igual a 700, para cambiar de dirección con el metodo de changeDirection().

En el método de moveBoss() checa si getDirection() es true y manda a llamar al método moveUp(), de ser contrario llamar al método moveDown() y después manda a llamar al método checkDirectionToMove().

```
public void act()
{
    moveBoss();
    enemyShoots();
    hitBullet();
}

public void moveUp()
{
    setLocation(getX(), getY() - getspeed());
}

public void moveDown()
{
    setLocation(getX(), getY() + getspeed());
}

public void checkDirectionToMove()
{
    if(getY() <= 300) changeDirection();
    else if(getY() >= 700) changeDirection();
}

public void moveBoss()
{
    if(getDirection() == true) moveUp();
    else moveDown();
    checkDirectionToMove();
}
```

En el metodo hitBullet() busca contacto con un objeto de la clase Bullet() asignado en la variable hit, si hit no es null, se crea una instancia del mundo, se elimina el objeto y se manda a llamar al metodo receiveDammage() enviando la instancia del mundo.

```
public void hitBullet()
{
    Actor hit = getOneIntersectingObject(Bullet.class);
    if((hit != null))
    {
        World world = getWorld();
        MyWorld myWorld = (MyWorld)world;

        getWorld().removeObject(hit);

        receiveDamage(myWorld.getDammage());
        //Greenfoot.playSound("Infected2.mp3");
    }
}
}
```

Infected1

La clase Infected1 es una subclase de Enemy.

En el constructor se recibe el tipo de enemigo y la velocidad, y se regresa a la superclase, y se re escala la imagen.

En el método de act() checa la dirección a la que esta volteando el enemigo (true izquierda, false derecha), después se manda a llamar al método enemyShoots() y hitBullet().

En el método hitBullet() se checa si hay contacto con un objeto de la clase Bullet() y se asigna a la variable hit, si hit no es null, se crea una instancia del mundo y se elimina el objeto, se manda a llamar al método receiveDammage() y se reproduce un sonido.

```
import greenfoot.*;

public class Infected1 extends Enemy
{
    public Infected1(int type, int speed)
    {
        super(type, speed);
        getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
    }

    public void act()
    {
        if(getDirection() == true) moveEnemyLeft();
        else moveEnemyRight();
        enemyShoots();
        hitBullet();
    }

    public void hitBullet()
    {
        Actor hit = getOneIntersectingObject(Bullet.class);
        if((hit != null))
        {
            World world = getWorld();
            MyWorld myWorld = (MyWorld)world;
            getWorld().removeObject(hit);
            receiveDamage(myWorld.getDammage());
            Greenfoot.playSound("ZombieHitted.mp3");
        }
    }

}
```

Infected2

La clase Infected2 es una subclase de Enemy.

En el constructor se recibe el tipo de enemigo y la velocidad, y se regresa a la superclase, y se re escala la imagen.

En el método de act() checa la dirección a la que está volteando el enemigo (true izquierda, false derecha), después se manda a llamar al método enemyShoots() y hitBullet().

En el método hitBullet() se checa si hay contacto con un objeto de la clase Bullet() y se asigna a la variable hit, si hit no es null, se crea una instancia del mundo y se elimina el objeto, se manda a llamar al método receiveDammage() y se reproduce un sonido.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseIn

public class Infected2 extends Enemy
{
    public Infected2(int type, int speed, int life)
    {
        super(type, speed, life);
        getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5);
    }

    public void act()
    {
        if(getDirection() == true) moveEnemyLeft();
        else moveEnemyRight();
        enemyShoots();
        hitBullet();
    }

    public void hitBullet()
    {
        Actor hit = getOneIntersectingObject(Bullet.class);
        if((hit != null))
        {
            World world = getWorld();
            MyWorld myWorld = (MyWorld)world;
            getWorld().removeObject(hit);
            receiveDamage(myWorld.getDammage());
            Greenfoot.playSound("Infected2.mp3");
        }
    }
}
```

Complicaciones

Un problema que se presentó fue que los personajes que aparecen en la pantalla originalmente iban a ser de tipo gif para que se viera mejor pero no encontramos la forma de implementar ese tipo de imágenes y nos conformamos con imágenes regulares

Cuando decidimos usar imágenes regulares para los sprites, las imágenes eran demasiado grandes para ponerlas en el juego y se solucionó con la función getImage().scale(getImage().getWidth()/5,getImage().getHeight()/5); que lo que hace es re escalar la imagen a un tamaño deseado.

Ya casi al final del juego, cuando se presenta el jefe final nos encontramos con el problema de que se veía mal y no se podía jugar el nivel final, y se soluciono utilizando los backgrounds correctos.

En resumen lo que mas nos dio problema fueron los sprites que no se podían implementar de la forma que queríamos y después aparecían de formas inesperadas.

Conclusión

Con este trabajo se comprendió de una mejor forma el uso de objetos en programación con Java implementándolo con el objetivo de crear un videojuego trabajando en equipo y aplicando conocimientos aprendidos en clase de Tecnologías Orientadas a Objetos

Referencias

https://www.youtube.com/watch?v=09SViWwRepE&list=PLmwzeqwf733-OwH_ZlPMlQAS4DAaaKzKV&index=2

Haungs, M. (2020). Creative Greenfoot by Michael Haungs (2014-12-24). Packt Publishing - ebooks Account.