****

**Universidad Autónoma de San Luis Potosí**

**Facultad de Ingeniería**

**Área de Computación e Informática**

**Programación Orientada a Objetos**

**Dr. Puente Montejano Cesar Augusto**

**Manual de Programador**

**Moreno Sabás Mayra Karina**

**Ciclo Escolar 2015-2016/I**

**7-Diciembre-2015**

PROGRAMACIÓN ORIENTADA A OBJETOS

PROPUESTA DE PROYECTO

Semestre 2015-2016/I

[No de proyecto](#h.z8dwovkpuhl8)

[Nombre del proyecto](#h.mnxc81u1z897)

[Integrantes del equipo](#h.dntok8juv9p3)

[Objetivo del proyecto](#h.bsu9evf8nxs1)

[Descripción del proyecto](#h.eivcmejh9ing)

[Descripción e imágenes de cada nivel](#h.dipcv9h52uj1)

N[ivel Principiante](#h.30shymbyey24)

[Nivel Intermedio](#h.lp6l46qd1968)

[Nivel Avanzado](#h.iwgxt4txxcht)

[Diagrama de clases UML](#h.g0vt53pc7r37)

[Características y comportamiento de cada clase](#h.mtuw3jfplyj6)

[Herencia y polimorfismo](#h.k1a372vuic1q)

[Cronograma de actividades (plan de trabajo)](#h.30zz6o5p35xo)

[Bitácora de actividades (historial)](#h.rr6kuez6yf7u)

1. No de proyecto

188

1. Nombre del proyecto

Delicious

1. Integrantes del equipo

196211

Moreno Sabás Mayra Karina

1. Objetivo del proyecto

El jugador deberá de comer la comida saludable como lo son frutas, verduras, si este llega a comer tres alimentos no saludables perderá vidas y aumentara de peso.

Descripción del proyecto

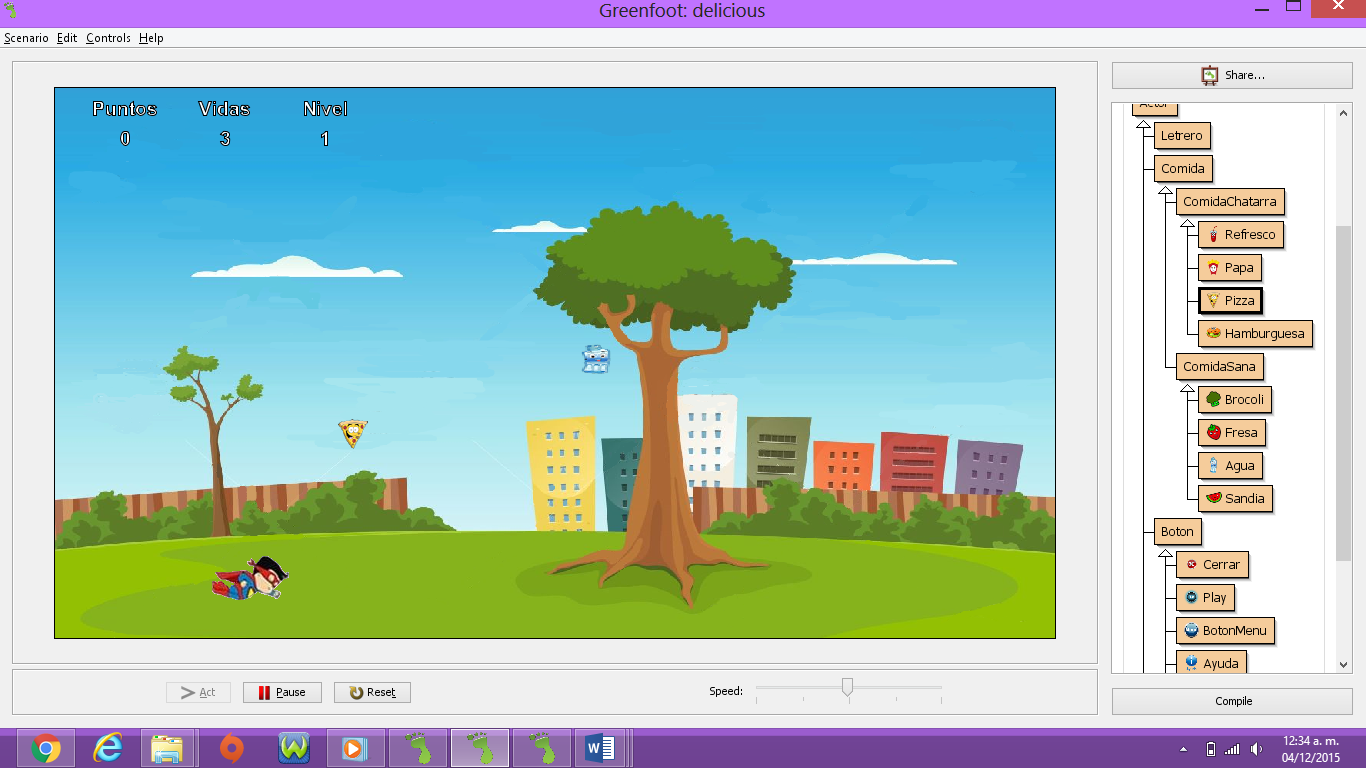
El juego trata de una persona que tendrá que comer sanamente, el jugador tratara de alcanzar a la comida sana, esta comida que consiste en frutas y verduras trataran de huir del jugador, pero también existe la comida chatarra está tratara de alcanzar al jugador para que la coma. El jugador tendrá que evitar comer comida chatarra como lo son papas fritas, refrescos, hamburguesas y pizza. Si la comida chatarra llegase a tocar 3 veces al jugador perderá una vida, además cada que la comida chatarra toque al jugador este ira engordando. Si el jugador alcanza a la comida sana ira incrementando puntos. Si el jugador logra completar el nivel se le sumaran puntos extras, además de una vida. Serán 3 niveles en cada nivel ira aumentando la velocidad en que estará moviéndose la comida. El jugador se podrá mover adelante, izquierda y derecha no podrá regresar. Al inicio el jugador tendrá 3 vidas, se tendrá que completar un total de 1000 puntos para poder avanzar al siguiente nivel. Las frutas y verduras valdrán 100 puntos y el agua 50. La comida además de quitarle vidas también le restaran 50 puntos al jugador.

1. Descripción e imágenes de cada nivel

El juego consta de tres niveles, en cada nivel la velocidad de movimiento y aparición de los objetos aumentará, al término de cada nivel se darán puntos extras y vidas extras.

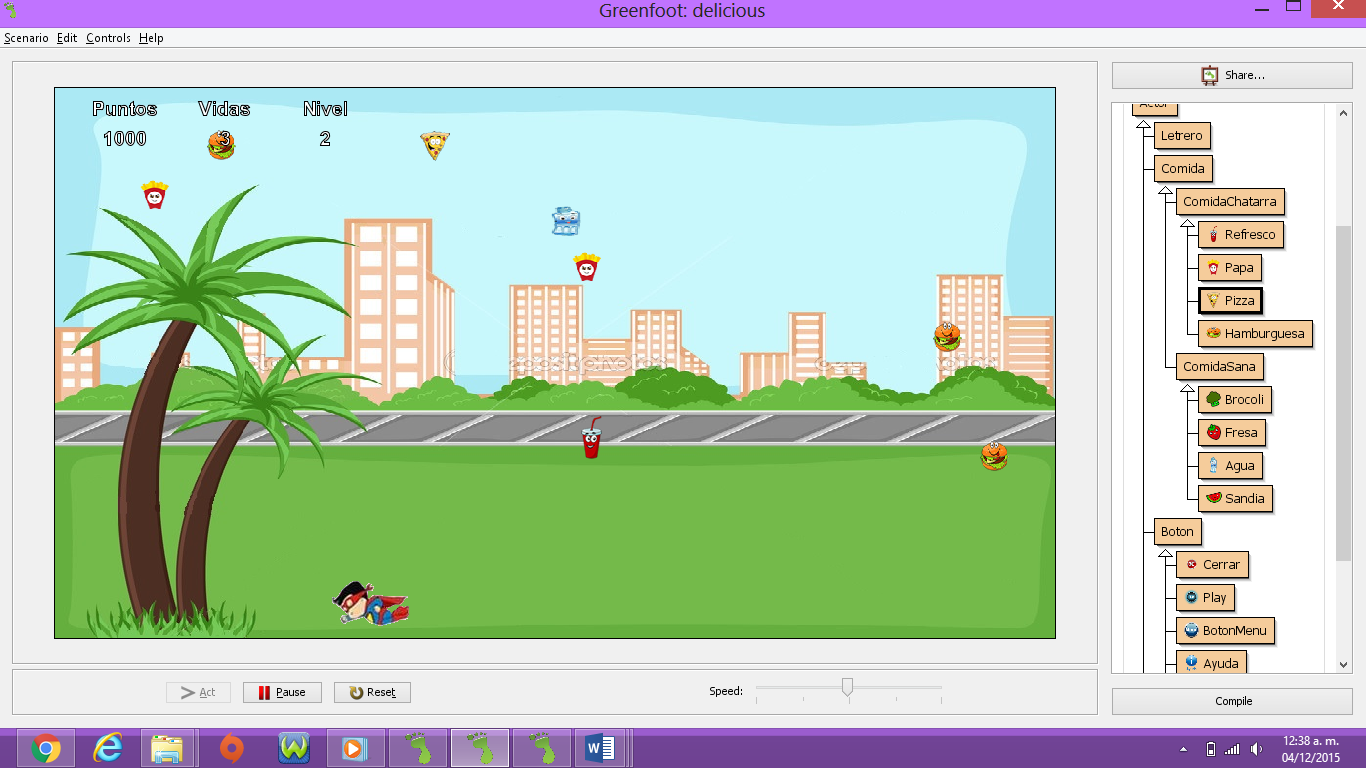
* Nivel Principiante

En este nivel la velocidad en la que aparezcan y se muevan los objetos será lenta. Se tienen que conseguir 1000 puntos para poder avanzar al siguiente nivel. Si logra juntar los 1000 puntos se le sumaran 250 puntos extras y una vida.



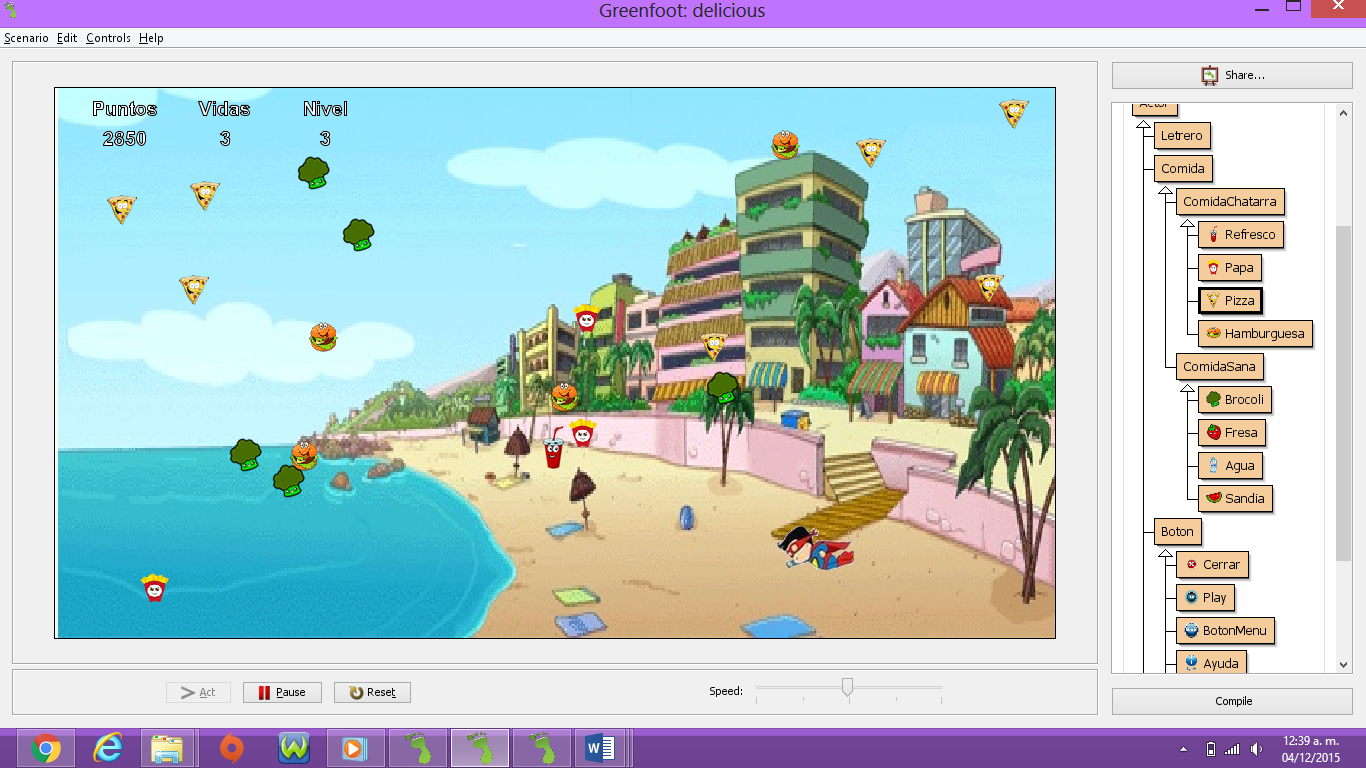
* Nivel Intermedio

Este nivel la forma de aparecer y el movimiento de los objetos será en una velocidad media, se tiene que juntar un total de 2750 puntos para poder avanzar al siguiente nivel, si se logra completar el nivel se le sumaran 500 puntos extras y una vida.

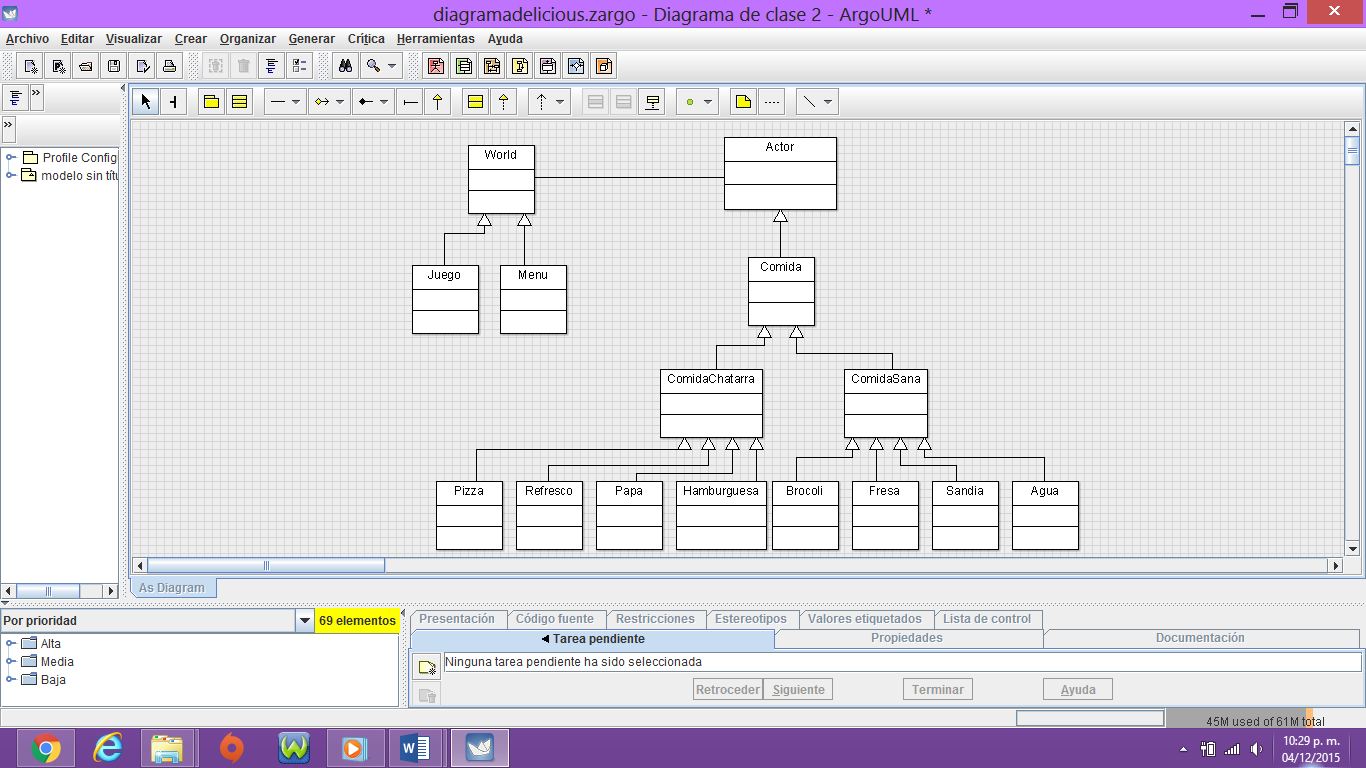


* Nivel Avanzado

En este último nivel la velocidad en la que aparecerán y moverán los objetos será más rápido, si el jugador logra llegar 4000 puntos ganara, además de que se le darán 1000 puntos extras.



1. Diagrama de clases UML



Características y comportamiento de cada clase

|  |  |
| --- | --- |
| Nombre de la clase: | Juego |
| Características: | Puntos |
|  | Vidas |
|  | Escenario |
| Comportamiento: | El fondo ira cambiando conforma vaya avanzando los niveles |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Nombre de la clase: | Niño |
| Características: | Puntos |
|  | Vidas |
|  | Imagen |
| Comportamiento: | El niño ira cambiando de imagen simulando que se mueve la capa |
|  | Además de manejar el movimiento , vidas, puntos |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Nombre de la clase: | comidaSana |
| Características: |  |
|  |  |
|  |  |
| Comportamiento: | Se estará movimiento por todo el escenario cada una con movimiento diferente, si el jugador las toca le sumaran puntos |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| Nombre de la clase: | comidaChatarra |
| Características: |  |
|  |  |
|  |  |
| Comportamiento: | Se moverá en el escenario con movimientos diferentes, si tocan al jugador le restan puntos y vidas |
|  |  |
|  |  |
|  |  |

Herencia y polimorfismo

Herencia: La herencia se ocupó en la mayoría de las clases, para heredar métodos y atributos, a subclases.

Polimorfismo: Este se ocupó dentro del juego para poder relacionar el mundo con el juego y los actores.

Cronograma de actividades (plan de trabajo)

Describir todas las actividades por realizar desde la propuesta del proyecto hasta la entrega considerando los siguientes entregables:

* + Manual del usuario
  + Manual del programador (este documento)
  + Código
  + Video
  + Link a Greenfoot

Se deben tomar en cuenta las siguientes fechas importantes:

* + fecha de entrega para examen de ordinario: 2 de diciembre
  + fecha de entrega para examen de extraordinario: 4 de diciembre
  + fecha de entrega para examen de título: 11 de diciembre

|  |  |  |
| --- | --- | --- |
| Fecha de Inicio | Fecha de Término | Actividad por realizar |
| 18/09/2015 | 19/09/2015 | Propuesta de proyecto |
| 14/10/2015 | 20/10/2015 | Diagrama de clases |
| 10/10/2015 | 30/11/2015 | Avance del proyecto |
| 5/12/2015 | 5/12/2015 | Proyecto terminado |

1. Bitácora de actividades (historial)

Esta parte será llenada durante la elaboración del proyecto. Aquí se deben describir cada una de las actividades realizadas desde la propuesta hasta la entrega del proyecto.

|  |  |
| --- | --- |
| Fecha (dd/mm/aa) | Descripción de la actividad realizada |
| 18/09/2015 | Pensar una propuesta para realizar un juego. |
| 14/10/2015 | Realizado en un programa especial para elaborar diferentes diagramas de UML |
| 30/11/2015 | Entrega de avance, se mostró el movimiento de la comida y del el jugador, además de que suma puntos, resta vidas. Funcionamiento de botones en el menú. |
| 5/2/2015 | Termino del proyecto ya funcionando |

**Código**

**Juego**

import greenfoot.\*;

/\*\*

\* Write a description of class Juego here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Juego extends World

{

private GreenfootImage fondo; //Imagen de fondo del nivel 1

private GreenfootImage fondo2; //Imagen de fondo del nivel 2

private GreenfootImage fondo3; //Imagen de fondo del nivel 3

private SimpleTimer reloj;

private int Dir;

private Niño n;

private int an;

private int punt;

private String marcadorNivel;

private int nivel;

private int vidas;

/\*\*

\* Constructor for objects of class Juego. Se inicializan las variables , carga las imagenes del esenario, el niño lo agrega al esenario

\*

\*/

public Juego()

{

// Create a new world with 600x400 cells with a cell size of 1x1 pixels.

super(1000, 550, 1);

fondo = new GreenfootImage("escenario.png");

fondo.scale(1000,550);/\* escala la imagen a las medidas del escenario en caso de exceder las dimensiones \*/

setBackground(fondo);/\* muestra la imagen en pantalla \*/

fondo2 = new GreenfootImage("nivel2.jpg");

fondo2.scale(1000,550);

fondo3 = new GreenfootImage("nivel3.jpg");

fondo3.scale(1000,550);

nivel = 1;

n = new Niño();

addObject(n,150,490);

Dir=0;

reloj = new SimpleTimer();

reloj.mark();

vidas = 3;

}

/\*\*

\* Metodo en el que llama a las acciones que ara el niño , controla la velocidad en la qeu aparecen los objetos en el esenario,

\* llama a los letreros perdiste y ganaste, cambia los esenarios de los niveles y controla las vidas, puntos y direccion.

\*/

public void act()

{

if(reloj.millisElapsed()>=3000 && nivel ==1) //velocidad en la que van a aparecer los objetos en el primer nimerl

{

reloj.mark();

agregaObjeto(Greenfoot.getRandomNumber(8)+1,Greenfoot.getRandomNumber(800)+100,Greenfoot.getRandomNumber(400)+50);

}

if(reloj.millisElapsed()>=2000 && nivel ==2)

{

reloj.mark();

agregaObjeto(Greenfoot.getRandomNumber(8)+1,Greenfoot.getRandomNumber(800)+100,Greenfoot.getRandomNumber(400)+50);

}

if(reloj.millisElapsed()>=1000 && nivel ==3)

{

reloj.mark();

agregaObjeto(Greenfoot.getRandomNumber(8)+1,Greenfoot.getRandomNumber(800)+100,Greenfoot.getRandomNumber(400)+50);

}

if(nivel == 4)

{

Greenfoot.playSound("ganaste.mp3");

Letrero ganaste;

ganaste = new Letrero(" GANASTE ");

addObject(ganaste,getWidth()/2,getHeight()/2);

Greenfoot.stop();

}

if(vidas == 0)

{

Greenfoot.playSound("gameOver.mp3");

Letrero perdiste;

perdiste = new Letrero("PERDISTE");

addObject(perdiste,getWidth()/2,getHeight()/2);

Greenfoot.stop();

}

vidas = n.getVidas();

Dir = n.getDir();

nivel = n.getNivel();

switch(nivel)

{

case 1:

setBackground(fondo);

break;

case 2:

setBackground(fondo2);

break;

case 3:

setBackground(fondo3);

break;

}

// p.setDir(Dir);

}

/\*\*

\* Metodo que va agregando la comida sana y chatarra aleatoriamente mediante un switch se pasa como parametro el numero de objeto, una coordenada en x y otra en y

\*/

public void agregaObjeto(int objeto, int x, int y){

switch(objeto){

case 1:

Agua a = new Agua();

addObject(a,x,y);

break;

case 2:

Sandia s = new Sandia();

addObject(s, x, y);

break;

case 3:

Fresa f = new Fresa();

addObject(f,x,y);

break;

case 4:

Brocoli b = new Brocoli(an,x,y);

addObject(b,x,y);

break;

case 5:

Refresco r = new Refresco(an,x,y);

addObject(r,x,y);

break;

case 6:

Hamburguesa h = new Hamburguesa();

addObject(h,x,y);

break;

case 7:

Papa pa = new Papa();

addObject(pa,x,y);

break;

case 8:

Pizza p = new Pizza();

addObject(p,x,y);

// p.setDir(Dir);

break;

default:

}

}

}

**Menú**

import greenfoot.\*;

/\*\*

\* Write a description of class Menu here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Menu extends World

{

private GreenfootSound soundBoton = new GreenfootSound("play.wav");

private GreenfootImage fondo;

private Play botonplay;

private BotonMenu botonM;

/\*\*

\* Constructor for objects of class Menu.

\*

\*/

public Menu()

{

super(1000, 550, 1, false);

fondo = new GreenfootImage("delicious.png");

fondo.scale(1000,550);/\* escala la imagen a las medidas del escenario en caso de exceder las dimensiones \*/

setBackground(fondo);/\* muestra la imagen en pantalla \*/

botonplay = new Play();

botonM = new BotonMenu();

prepare();

}

public void prepare()

{

addObject(botonplay, 300, 460);

addObject(new Ayuda(), 720, 475);

addObject(new Credito(), 510, 460);

}

/\*\*

\* Metodo que crea la ayuda al presionar el boton de la ayuda llama a este metodo y muestra la imagen que contiene la ayuda

\*/

public void creaAyuda()

{

removeObjects( getObjects(Actor.class) );

setBackground("help.jpg");

soundBoton.play();

addObject(botonM, 850,460);

}

/\*\*

\* Metodo que crea los creditosa al presionar el boton de los creditos llama a este metodo y muestra la imagen que contiene los creditos

\*/

public void creaCreditos()

{

removeObjects( getObjects(Actor.class) );

setBackground("credito.jpg");

soundBoton.play();

addObject(botonM, 850, 460);

}

}

**Letrero**

import greenfoot.\*;

/\*\*

\* Write a description of class Letrero here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Letrero extends Actor

{

/\*\*

\* Act - do whatever the Letrero wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

// Add your action code here.

}

/\*\*

\* Metodo que crea un letrero y se manda llamar en la clase juego y aparesca

\* segun la condicion ganaste o perdiste

\*/

public Letrero(String mensaje)

{

GreenfootImage letrero;

letrero = new GreenfootImage(mensaje,50,java.awt.Color.white,java.awt.Color.black);

setImage(letrero);

}

//metodos

}

**Refresco**

import greenfoot.\*;

/\*\*

\* Write a description of class Refresco here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Refresco extends ComidaChatarra

{

private int xPos;

private int yPos;

private int xRadius;

private int yRadius;

private double angle;

/\*\*

\* constructor de variables de la clase Refresco

\* parametros un angulo, posision en x y posision en y.

\*/

public Refresco(double angle, int xPos, int yPos)

{

this.angle=3.1416D/180\*angle;

this.xPos=xPos;

this.yPos=yPos;

this.yRadius=50;

this.xRadius=50;

}

public void act()

{

this.mover();

}

/\*\*

\* metodo que mueve al refresco en forma de circulo

\*

\*/

public void mover()

{

double x = (Math.cos(angle) \* xRadius + xPos);

double y = (Math.sin(angle) \* yRadius + yPos);

setLocation((int)x,(int)y);

angle+=0.1;

if(angle>=2D\*3.1416D)

angle=0;

}

/\*\*

\*iguala la posision inicial con la posision actual en x e y

\*/

public void changeCenter(int xPos, int yPos)

{

this.xPos=xPos;

this.yPos=yPos;

}

}

**Papa**

import greenfoot.\*;

/\*\*

\* Write a description of class Papa here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Papa extends ComidaChatarra

{

private int dir;

private int vel;

/\*\*

\* constructor de variables de la clase Papa

\*/

public Papa()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiaDir();

this.mover();

}

/\*\*

\* metodo que checa la direccion en que debe de ir la Papa, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiaDir()

{

switch(dir)

{

case 1:

if(getY()+10>=getWorld().getHeight())

{

dir=2;

}

break;

case 2:

if(getY()-10<=0)

{

dir=1;

}

break;

default:

}

}

/\*\*

\* mueve la papa en el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX(),getY()+vel);

break;

case 2:

setLocation(getX(),getY()-vel);

break;

default:

}

}

}

**Pizza**

import greenfoot.\*;

/\*\*

\* Write a description of class Pizza here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Pizza extends ComidaChatarra

{

private int dir;

private int vel;

/\*\*

\* constructor de variables de la clase pizza

\*/

public Pizza()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiarDir();

// if(dir2==0)

this.mover();

}

/\* public void setDir(int dir){

dir2 = dir;

}\*/

/\*\*

\* metodo que checa la direccion en que debe de ir la pizza, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiarDir()

{

switch(dir)

{

case 1:

if(getX()+10>=getWorld().getWidth())

dir=2;

if(getY()-20<=0)

dir = 4;

break;

case 2:

if(getX()-10<=0)

dir =1;

if(getY()-20<=0)

dir=3;

break;

case 3:

if(getX()-10<=0)

dir = 4;

if(getY()+10>=getWorld().getHeight())

dir =2;

break;

case 4:

if(getX()+10>=getWorld().getWidth())

dir= 3;

if(getY()+10>=getWorld().getHeight())

dir=1;

break;

default:

}

}

/\*\*

\* mueve la pizzaen el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX()+vel,getY()-vel);

break;

case 2:

setLocation(getX()-vel,getY()-vel);

break;

case 3:

setLocation(getX()-vel,getY()+vel);

break;

case 4:

setLocation(getX()+vel,getY()+vel);

break;

default:

}

}

}

**Hamburguesa**

import greenfoot.\*;

/\*\*

\* Write a description of class Hamburguesa here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Hamburguesa extends ComidaChatarra

{

private int dir;

private int vel;

private int dir2; //direccion del jugador

/\*\*

\* constructor de variables de la clase Hamburguesa

\*/

public Hamburguesa()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiaDir();

if(dir2 == 0)

this.mover();

}

public void setDir(int dir){

dir2 = dir;

}

/\*\*

\* metodo que checa la direccion en que debe de ir la hamburguesa, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiaDir()

{

switch(dir)

{

case 1:

if(getX()+40>=getWorld().getWidth())

{

setLocation(getX()-vel,getY());

dir=2;

}

break;

case 2:

if(getX()-70<=0)

{

setLocation(getX()+vel,getY());

dir=1;

}

break;

default:

}

}

/\*\*

\* mueve la hamburguesa en el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX()+vel,getY());

break;

case 2:

setLocation(getX()-vel,getY());

break;

default:

}

}

}

**Brócoli**

import greenfoot.\*;

/\*\*

\* Write a description of class Brocoli here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Brocoli extends ComidaSana

{

private int xPos;

private int yPos;

private int xRadius;

private int yRadius;

private double angle;

/\*\*

\* constructor de variables de la clase Brocoli

\* parametros un angulo, posision en x y posision en y.

\*/

public Brocoli(double angle, int xPos, int yPos)

{

this.angle=3.1416D/180\*angle;

this.xPos=xPos;

this.yPos=yPos;

this.yRadius=50;

this.xRadius=50;

}

public void act()

{

this.mover();

}

/\*\*

\* metodo que mueve al brocoli en forma de circulo

\*

\*/

public void mover()

{

double x = (Math.cos(angle) \* xRadius + xPos);

double y = (Math.sin(angle) \* yRadius + yPos);

setLocation((int)x,(int)y);

angle+=0.1;

if(angle>=2D\*3.1416D)

angle=0;

}

/\*\*

\*iguala la posision inicial con la posision actual en x e y

\*/

public void cambiaCenter(int xPos, int yPos)

{

this.xPos=xPos;

this.yPos=yPos;

}

}

**Fresa**

import greenfoot.\*;

/\*\*

\* Write a description of class Fresa here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Fresa extends ComidaSana

{

private int dir;

private int vel;

/\*\*

\* constructor de variables de la clase Fresa

\*/

public Fresa()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiarDir();

this.mover();

}

/\*\*

\* metodo que checa la direccion en que debe de ir la fresa, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiarDir()

{

switch(dir)

{

case 1:

if(getX()+10>=getWorld().getWidth())

dir=2;

if(getY()-20<=0)

dir = 4;

break;

case 2:

if(getX()-10<=0)

dir =1;

if(getY()-20<=0)

dir=3;

break;

case 3:

if(getX()-10<=0)

dir = 4;

if(getY()+10>=getWorld().getHeight())

dir =2;

break;

case 4:

if(getX()+10>=getWorld().getWidth())

dir= 3;

if(getY()+10>=getWorld().getHeight())

dir=1;

break;

default:

}

}

/\*\*

\* mueve la fresa en el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX()+vel,getY()-vel);

break;

case 2:

setLocation(getX()-vel,getY()-vel);

break;

case 3:

setLocation(getX()-vel,getY()+vel);

break;

case 4:

setLocation(getX()+vel,getY()+vel);

break;

default:

}

}

}

**Agua**

import greenfoot.\*;

/\*\*

\* Write a description of class Agua here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Agua extends ComidaSana

{

private int dir;

private int vel;

/\*\*

\* constructor de variables de la clase Agua

\*/

public Agua()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiaDir();

this.mover();

}

/\*\*

\* metodo que checa la direccion en que debe de ir el agua, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiaDir()

{

switch(dir)

{

case 1:

if(getY()+10>=getWorld().getHeight())

dir=2;

break;

case 2:

if(getY()-10<=0)

dir=1;

break;

default:

}

}

/\*\*

\* mueve el agua en el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX(),getY()+vel);

break;

case 2:

setLocation(getX(),getY()-vel);

break;

default:

}

}

}

**Sandia**

import greenfoot.\*;

/\*\*

\* Write a description of class Sandia here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Sandia extends ComidaSana

{

private int dir;

private int vel;

/\*\*

\* constructor de variables de la clase Sandia

\*/

public Sandia()

{

dir=1;

vel=3;

GreenfootImage image=getImage();

image.scale(30,30);

setImage(image);

}

/\*\*

\* Act - do whatever the Enemy3 wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

this.cambiarDir();

this.mover();

}

/\*\*

\* metodo que checa la direccion en que debe de ir la Sandia, al llegar a un extremo en el eje de las x

\* cambia su posision a la contraria

\*/

public void cambiarDir()

{

switch(dir)

{

case 1:

if(getX()+10>=getWorld().getWidth())

dir=2;

if(getY()-20<=0)

dir = 4;

break;

case 2:

if(getX()-10<=0)

dir =1;

if(getY()-20<=0)

dir=3;

break;

case 3:

if(getX()-10<=0)

dir = 4;

if(getY()+10>=getWorld().getHeight())

dir =2;

break;

case 4:

if(getX()+10>=getWorld().getWidth())

dir= 3;

if(getY()+10>=getWorld().getHeight())

dir=1;

break;

default:

}

}

/\*\*

\* mueve la sandia en el eje de las X dependiendo su direccion

\*/

public void mover()

{

switch(dir)

{

case 1:

setLocation(getX()+vel,getY()-vel);

break;

case 2:

setLocation(getX()-vel,getY()-vel);

break;

case 3:

setLocation(getX()-vel,getY()+vel);

break;

case 4:

setLocation(getX()+vel,getY()+vel);

break;

default:

}

}

}

**Play**

import greenfoot.\*;

/\*\*

\* Write a description of class Play here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Play extends Boton

{

private GreenfootSound soundBoton = new GreenfootSound("play.wav");

/\*\*

\* Metodo del boton play el cual al precionarlo dara inicio al juego

\*

\*/

public void act()

{

// Add your action code here.

MouseInfo mouse = Greenfoot.getMouseInfo();

if(mouse!=null)

{

if(Greenfoot.mouseClicked(this)){

soundBoton.play();

Greenfoot.setWorld(new Juego());

}

}

}

}

**BotónMenú**

import greenfoot.\*;

/\*\*

\* Write a description of class BotonMenu here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class BotonMenu extends Boton

{

private GreenfootSound soundBoton = new GreenfootSound("play.wav");

public BotonMenu()

{

setImage("menu.png");

}

/\*\*

\* metodo donde se checa si se presiono el boton menu

\*/

public void act(){

MouseInfo mouse = Greenfoot.getMouseInfo();

if(mouse!=null)

{

if(Greenfoot.mouseClicked(this)){

soundBoton.play();

Greenfoot.setWorld(new Menu());

}

}

}

}

**Ayuda**

import greenfoot.\*;

/\*\*

\* Write a description of class Ayuda here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Ayuda extends Boton

{

private GreenfootSound soundBoton = new GreenfootSound("play.wav");

/\*\*

\* Metodo del boton ayuda el cual al precionarlo mostrara la ayuda del juego

\*

\*/

public void act()

{

// Add your action code here.

MouseInfo mouse = Greenfoot.getMouseInfo();

if(mouse!=null)

{

if(Greenfoot.mouseClicked(this))

((Menu)getWorld()).creaAyuda();

}

}

}

**Crédito**

import greenfoot.\*;

/\*\*

\* Write a description of class Credito here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Credito extends Boton

{

/\*\*

\* Act - do whatever the Credito wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\* En este metodo se crea la funcion para que al presionar el boton de credits

\* con el mouse se active y llama al metodo de creaCreditos

\*/

public void act()

{

// Add your action code here.

click();

}

public void click()

{

MouseInfo mouse = Greenfoot.getMouseInfo();

if(mouse!=null)

{

if(Greenfoot.mouseClicked(this))

((Menu)getWorld()).creaCreditos();

}

}

}

**Niño**

import greenfoot.\*;

/\*\*

\* Write a description of class Niño here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Niño extends Actor

{

private GreenfootImage[] niño;

private int puntos;

private String marcador;

private String marcadorVidas;

private String marcadorNivel;

private int iDir;

private int contGordura;

private int contVidas;

private GreenfootSound soundBoton = new GreenfootSound("soundFood.mp3");

private int nivel;

private int x1;

private SimpleTimer reloj;

private int cambiaImagen;

private int cambio;

/\*\*

\* Constructor de la clase niño. aqui se agregan las imagenes al arreglo , ademas de inicalizar las variables

\*/

public Niño()

{

niño = new GreenfootImage[13]; //Arreglo de tipo GreenfootImage guarda todas las imagenes que pertenecen al movimietno del actor

niño[0] = new GreenfootImage("s1.png"); //delgado

niño[1] = new GreenfootImage("s3.png");

niño[2] = new GreenfootImage("s1\_izq.png"); //delgado izq

niño[3] = new GreenfootImage("s3\_izq.png");

niño[4] = new GreenfootImage("sm1.png"); //mediano

niño[5] = new GreenfootImage("sm2.png");

niño[6] = new GreenfootImage("sm1\_izq.png");

niño[7] = new GreenfootImage("sm2\_izq.png");

niño[8] = new GreenfootImage("s\_g1.png"); //gordo

niño[9] = new GreenfootImage("sg2.png");

niño[10] = new GreenfootImage("s\_g1\_izq.png");

niño[11] = new GreenfootImage("sg2\_izq.png");

contGordura = 1; //contador de la gordura 0 es delgado

contVidas = 3; //contador de vidas

nivel = 1; //niveles

iDir = 0; //direccion 0 es derecha

reloj=new SimpleTimer();

reloj.mark();

cambiaImagen = 1;

cambio=0;

}

/\*\*

\* Act - do whatever the Niño wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public void act()

{

// Add your action code here.

getWorld().showText("Puntos",70,20);

getWorld().showText("Vidas",170,20);

marcadorVidas = String .valueOf(contVidas);

getWorld().showText(marcadorVidas,170,50);

marcador = String .valueOf(puntos);

getWorld().showText(marcador,70,50);

getWorld().showText("Nivel",270,20);

marcadorNivel = String .valueOf(nivel);

getWorld().showText(marcadorNivel,270,50);

if(contVidas > 0){

if(contGordura == 4){

contVidas--;

contGordura = 1;

if(iDir==0 )cambiaImagen = 1;

else

cambiaImagen=3;

}

if(reloj.millisElapsed()>=500)//las imagenes cambian cada cierto tiempo definido

{

reloj.mark();

animacion();//llamada a funcion para movimiento de jugador

}

mover();

comeSanamente();

comeMal();

niveles();

}

}

/\*\* Metodo que se encargara del movimietno del actor y lo condiciona para que no salga del limite de la pantalla,

\* ademas de cambiar las imagenes para que se mueva la capa al momento de ir avanzando

\*

\*/

public void mover()

{ if(getX()<960 ){

if(Greenfoot.isKeyDown("right")){

move(5);

setImage(niño[ cambiaImagen -cambio ]);

iDir = 0;

if(contGordura == 1)

cambiaImagen = 1;

else

if(contGordura == 2)

cambiaImagen = 5;

else

if(contGordura == 3)

cambiaImagen = 9;

}

}

if(getX()>40 ){

if(Greenfoot.isKeyDown("left")){

move(-5);

setImage(niño[ cambiaImagen -cambio ]);

iDir =1;

if(contGordura == 1)

cambiaImagen = 3;

else

if(contGordura == 2)

cambiaImagen = 7;

else

if(contGordura == 3)

cambiaImagen = 11;

}

}

if(getY()>130){

if(Greenfoot.isKeyDown("up")){

setLocation(getX(), getY()-5);

setImage(niño[cambiaImagen]);

}

}

if(getY()<520){

if(Greenfoot.isKeyDown("down")){

setLocation(getX(), getY()+5);

setImage(niño[cambiaImagen ]);

}

}

}

/\*\*

\* Metodo que se encarga de desaparecer la comida sana cuando es tocada por el jugador y aumentar los puntos

\* que corresponden por cada alimento

\*/

void comeSanamente(){

if(isTouching(Fresa.class)){

soundBoton.play();

puntos+=200;

removeTouching(Fresa.class);

}

else if(isTouching(Sandia.class)){

soundBoton.play();

puntos+=175;

removeTouching(Sandia.class);

}

else if(isTouching(Brocoli.class)){

soundBoton.play();

puntos+=150;

removeTouching(Brocoli.class);

}

else if(isTouching(Agua.class)){

soundBoton.play();

puntos+=50;

removeTouching(Agua.class);

}

}

/\*\*

\* Metodo que restara puntos y vidas cada vez que el jugador sea tocado por la comida chatarra

\*/

void comeMal()

{

if(isTouching(Hamburguesa.class)){

soundBoton.play();

if(puntos > 200)

puntos-=200;

removeTouching(Hamburguesa.class);

if(contGordura < 4)

contGordura += 1;

}

else if(isTouching(Pizza.class)){

soundBoton.play();

if(puntos > 250)

puntos-=250;

removeTouching(Pizza.class);

if(contGordura < 4)

contGordura += 1;

}

else if(isTouching(Papa.class)){

soundBoton.play();

if(puntos > 150)

puntos-=150;

removeTouching(Papa.class);

if(contGordura < 4)

contGordura += 1;

}

else if(isTouching(Refresco.class)){

soundBoton.play();

if(puntos > 100)

puntos-=100;

removeTouching(Refresco.class);

if(contGordura < 4)

contGordura += 1;

}

}

/\*\*

\* metodo el cual se encarga de mandar la direccion a la clase juego

\*/

public int getDir(){

return iDir;

}

/\*\*

\* Metodo que cambia de nivel segun los puntos ira cambiando de nivel

\*/

public void niveles(){

if(puntos >= 1000 && nivel ==1){ //pasar nivel 2

nivel = 2;

puntos += 250;

contVidas += 1;

}

if(puntos >= 2250 && nivel == 2){ //pasar nivel 3

puntos += 500;

contVidas += 1;

nivel = 3;

}

if(puntos == 4000 && nivel == 3){ // pra ganar

puntos += 1000;

nivel = 4;

}

}

/\*\*

\* metodo que se encarga de mandar los niveles a la clase juego

\*/

public int getNivel(){

return nivel;

}

/\*\*

\* Metodo que se encarga de mandar las vidas a la clase juego

\*/

public int getVidas(){

return contVidas;

}

/\*\*

\* Metodo en el cual se hace la animacion de la capa para que simule que esta en movimiento la capa

\*/

public void animacion(){

if(cambio==0)

cambio =1;

else

cambio=0;

}

}

**SimpleTimer**

/\*\*

\* A simple timer class that allows you to keep track of how much time

\* has passed between events.

\*

\* You use this class by creating a timer as a member field in your actor (or whatever):

\* <pre>

\*

\* private SimpleTimer timer = new SimpleTimer();

\* </pre>

\*

\* Then when you want to start the timer (for example, when a shot is fired), you call the mark() method:

\*

\* <pre>

\*

\* timer.mark();

\* </pre>

\*

\* Thereafter, you can use the millisElapsed() method to find out how long it's been since mark()

\* was called (in milliseconds, i.e. thousandths of a second). So if you want to only allow the player to fire a shot every second, you

\* could write:

\*

\* <pre>

\*

\* if (timer.millisElapsed() > 1000 && Greenfoot.isKeyDown("space"))

\* {

\* // Code here for firing a new shot

\* timer.mark(); // Reset the timer

\* }

\* </pre>

\*

\* @author Neil Brown

\* @version 1.0

\*/

public class SimpleTimer

{

private long lastMark = System.currentTimeMillis();

/\*\*

\* Marks the current time. You can then in future call

\* millisElapsed() to find out the elapsed milliseconds

\* since this mark() call was made.

\*

\* A second mark() call will reset the mark, and millisElapsed()

\* will start increasing from zero again.

\*/

public void mark()

{

lastMark = System.currentTimeMillis();

}

/\*\*

\* Returns the amount of milliseconds that have elapsed since mark()

\* was last called. This timer runs irrespective of Greenfoot's

\* act() cycle, so if you call it many times during the same Greenfoot frame,

\* you may well get different answers.

\*/

public int millisElapsed()

{

return (int) (System.currentTimeMillis() - lastMark);

}

}

**Link Greenfoot**

<http://www.greenfoot.org/scenarios/15587>

**Youtube**

<https://www.youtube.com/watch?v=7XJH8BGe6Xg>