Facultad Regional
UTN VILLA MARIA

EVALUACIÓN PARADIGMAS

PARTE GRUPAL

Fecha: 24/09/2020

Profesor: Rinaldi Mario.

Grupo:

- Achetta Mauricio (13356)
- Aimbinder Tiago (13778)
- Cordoba Aylen (13638)
- Simonin Eloy (13727)
- Zoy Eder (13620)

Punto 2: Principales Características de los escenarios Laboratorio Newton

Newton Labs 1:

En este escenario contamos con dos SuperClases principales: World y Actor

Space es la calse en la que se genera el espacio. Se utiliza la palabra reservada súper, y se le pasa como parámetro las dimensiones que se quiere que tenga el mundo.

```
/**
 * Create space.
 */
public Space()
{
    super(960, 620, 1);

    // Uncomment one of the following method calls if you want the object

    //sunAndPlanet();
    //sunAndTwoPlanets();
    //sunPlanetMoon();
```

super(altura, anchura, resolución)

Además, en esta clase tenemos métodos los cuales están comentados y no son ejecutados. Lo que hacen es crear Objetos y darles un cierto movimiento.

Por otra parte, Dentro de la **superclase Actor** se hereda **SmoothMover, la cual es una clase de tipo abstracta**, allí dentro se inicializan los métodos, pero no se lleva a cabo la implementación.

Body es una subclase de SmoothMover y esta clase permite crear cuerpos, los cuales son colocados en el espacio, pero estos no tienen movivilidad. Todo objeto creado hereda las propiedades de la superclase SmoothMover.

También contamos con una clase auxiliar **Vector.** Esta clase vector nos provee de toda la información del movimiento de los cuerpos.

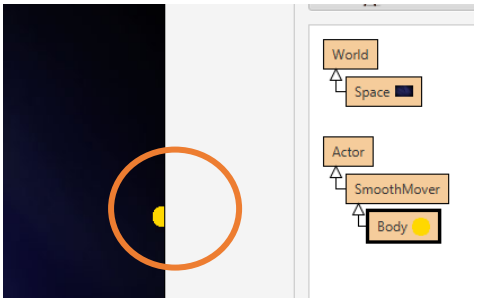
No se presenta gran funcionalidad en este escenario.

Newton Labs 2:

En este escenario ocurre exactamente lo mismo que el anterior, solo que tenemos modificaciones en la clase **Body** donde los objetos creados **presentan movimientos** lineales. Una vez que el objeto llega a un costado del espacio no pasa nada más.

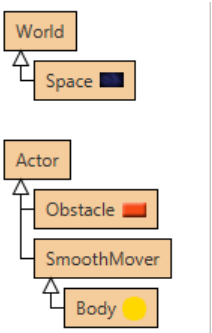
```
/**
 * Act. That is: apply the gravitation forces from
 * all other bodies around, and then move.
 */
public void act()
{
    applyForces();
    move();
}
```

Pero si dos cuerpos se chocan, estos presentan movimientos al azar por todo el escenario y con aceleraciones variadas, según las veces que se toquen con otro cuerpo.



Newton Lab 3:

Este escenario ya plantea mas funcionalidades. Apareciendo una nueva clase Obstacle.



Se crean 5 objetos al azar junto a una fila de obstáculos, dichos obstáculos tienen un sonido especifico.

```
private String[] soundFiles = { "2c", "2d", "2e", "2f", "2g", "2a", "2b", "3c", "3d", "3e", "3f", "3g", "3a", "3b" };
```

Si los objetos chocan un obstáculo, estos emiten su sonido correspondiente.

En el constructor de la clase Body, nos especifica la masa, velocidad y color de los cuerpos que se crenn.

```
/**
 * Construct a Body with a specified size, mass, velocity and color.
 */
public Body(int size, double mass, Vector velocity, Color color)
{
    this.mass = mass;
    addToVelocity(velocity);
    GreenfootImage image = new GreenfootImage (size, size);
    image.setColor (color);
    image.fillOval (0, 0, size-1, size-1);
    setImage (image);
}
```

Cuando las bolitas chocan con los obstáculos lo que hacen es cambiar de color. Los colores son aleatorios.

Mediante el método **BounceAtEdge** hace que si una pelota toca un borde rebote.

```
/**
 * Act. That is: apply  the gravitation forces from
 * all other bodies around, and then move.
 */
public void act()
{
    applyForces();
    move();
    bounceAtEdge();
}
```

```
/**
 * Check whether we have hit the edge of the universe. If so, bounce off it.
 */
private void bounceAtEdge()
{
    if (getX() == 0 || getX() == getWorld().getWidth()-1) {
        setLocation((double)getX(), (double)getY());
        invertHorizontalVelocity();
        accelerate(0.9);
    }
    else if (getY() == 0 || getY() == getWorld().getHeight()-1) {
        setLocation((double)getX(), (double)getY());
        invertVerticalVelocity();
        accelerate(0.9);
    }
}
```

Si las bolitas se chocan entre si hace que se muevan para un lado en forma vertical y horizontal con cierta aceleración.

```
/**
 * Apply the force of gravity of a given body to this one.
 * (Force is applied for one time unit; dt==1.)
 */
private void applyGravity(Body other)
{
    double dx = other.getExactX() - this.getExactX();
    double dy = other.getExactY() - this.getExactY();
    Vector dv = new Vector (dx, dy); //sets direction correctly; length still
    double distance = Math.sqrt (dx*dx + dy*dy);
    double force = GRAVITY * this.mass * other.mass / (distance * distance);
    double acceleration = force / this.mass;
    dv.setLength (acceleration);
    addToVelocity (dv);
}
```

```
/**
 * Apply the forces of gravity from all other celestial bodi
 */
private void applyForces()
{
    List<Body> bodies = getWorld().getObjects(Body.class);

    for (Body body : bodies)
    {
        if (body != this)
        {
            applyGravity(body);
        }
    }

    // ensure that we don't get too fast: If the current spe
    if (getSpeed() > 7)
    {
        accelerate (0.9);  // acceleration with factor < 1 i
    }
}
```

En conclusión, los cuerpos creados experimentan diferentes propiedades según su comportamiento. Si estos chocan con algún obstáculo u otro cuerpo cambiara su velocidad, dirección de movimiento, aceleración, colores, etc.

2.

MODIFICACIÓN SIGNIFICATIVA:

Lo que se pensó para esta modificación era la implementación de nuevos métodos, creados por nosotros con conocimientos adquiridos en las clases anteriores.

Lo que se hizo fue modificar la ubicación de los obstáculos haciendo que estos se ubiquen en posiciones random mediante el método **Greenfoot.getRandonNumber**, colocándolo en las coordenadas x,y.

En la clase Space se crean los objetos que inicializan al juego.

Se crean 10 Obstaculos, posicionados al azar:

```
int i = 0;
while (i < 10)
{
    addObject (new Obstacle (soundFiles[i] + ".wav"), Greenfoot.getRandomNumber(300) + i*60, Greenfoot.getRandomNumber(i+200));
    i++;
}
```

Además, se crean 5 cuerpos, los cuales serán los protagonistas de nuestro juego.

Estos cuerpos tendrán las mismas propiedades que en el escenario original.

```
public Space()
{
    super(960, 620, 1);

    createObstacles();
    randomBodies(5);
}
```

Dentro de la clase Obstáculo, se llamara al método **“comer()”.** Este método lo que hará es que al momento de que un cuerpo toca un obstáculo, este lo eliminara.

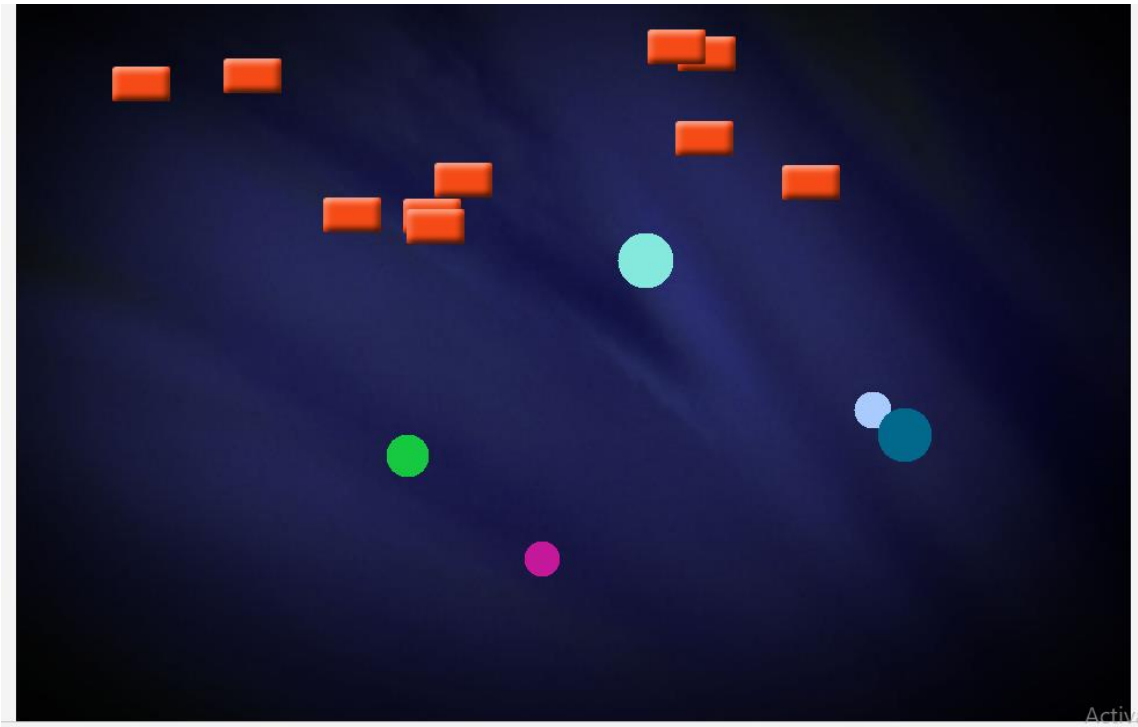
```
/**
 * Each act cycle, check whether we were hit. If we were, play our so
 */
public void act()
{

    if (!isTouching(Body.class))  // not touched anymore
    {
        touched = false;
        setImage ("block.png");

    }else if (isTouching(Body.class))   // just being touched now
    {
        touched = true;
        setImage ("block-light.png");
        comer();
    }
```

Metodo **“comer();”**

Remueve a Body

El fin del juego es cuando ya no quedan cuerpos por ser comidos.

```
public void comer(){

    Greenfoot.playSound(sound);
    removeTouching(Body.class);
    cont++;
    if (cont == 5){
        Greenfoot.stop();
        //Grenfoot.playSound(
    }
}
```



Junto a la entrega del trabajo se adjuntará un video, mostrando el funcionamiento del juego.

Punto 3: Códigos en URI

Para resolver los siguientes ejercicios se utilizo la sintaxis de JAVA.

Problema 1:

```java
package ProblemasUri;
import java.util.Scanner;
public class ProblemaUno {
    public static void main(String[] args) {

        Scanner Num = new Scanner(System.in);
        //Vector con la cantidad de Leds que se necesitan por número
        int ValidarLed[] = {6, 2, 5, 5, 4, 5, 6, 3, 7, 6}; //0,1,2,3,4,5,6,7,8,9
        int entrada = 0;
        int resto = 0;
        int totalLed = 0;
        int pruebas = 0;
        int i=0;
        ////////////////////////////////////////////////////////////////////
        System.out.println("Ingrese la cantidad de pruebas:");
        pruebas= Num.nextInt(); //Se ingresan la cantidad de casos de prueba
        if (pruebas>0 && pruebas<2000) {
            //Se ejecutaran la cantidad de pruebas elegidas por el usuario
            while (i<pruebas) {
                totalLed=0;
                System.out.println("Ingrese el número:");
                entrada = Num.nextInt(); //Se ingresa el numero a analizar
                //Se comprueba si el número ingresado esta en el rango correcto.
                while(entrada < 1 || entrada > 10100){
                System.out.println("Entrada incorrecta");
                }
                while(entrada>10){ //Si el numero es mayor que 10 sera de 3 cifras
                    resto = entrada % 10; //Se le saca su modulo
                    entrada = entrada/10;
                    if(resto != 0){  //Si el resto es distinto de cero
                       totalLed += ValidarLed[resto]; //El total de led sera igual
                    } else {
                       totalLed += ValidarLed[resto];
                    }
                }
```

```
                }
                if(entrada<10){ //Cuando el número sea menor a 10, simplemente se lo
                    resto = entrada;
                    if(resto != 0){
                       totalLed += ValidarLed[resto];
                    } else {
                       totalLed += ValidarLed[resto];
                    }
                    //Se imprimen resultados.
                    System.out.println("Se necesitan:");
                    System.out.println(totalLed + " leds");
                    i++; //Acumulador de los casso de prueba ya hechos
                }
            }
        }
        else {
            System.out.println("El número de pruebas esta fuera del rango");
        }
    }
}
```

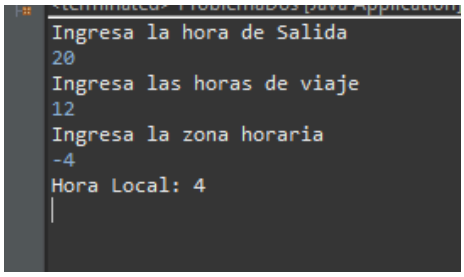
Caso de Prueba:

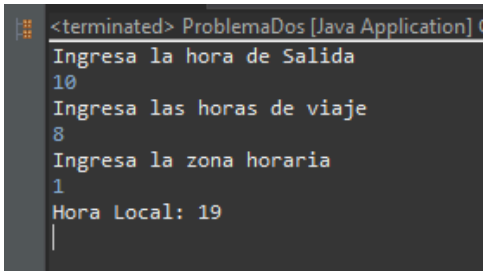
```
<terminated> ProblemaUno [Java Application] C:\P
Ingrese la cantidad de pruebas:
3
Ingrese el número:
115380
Se necesitan: 27 leds
Ingrese el número:
2819311
Se necesitan: 29 leds
Ingrese el número:
23456
Se necesitan: 25 leds
```

Problema 2:

```
package ProblemasUri;
import java.util.Scanner;
public class ProblemaDos {

    public static void main(String[] args){
            int hora=0;
            Scanner llegada = new Scanner(System.in);

            //Se ingresa Hora de Salida
            System.out.println("Ingresa la hora de Salida");
             int s = llegada.nextInt();
             if (s<=23 && s>=0) { //Se comprueba el rango

                 //Se ingresa horas de Viaje
                 System.out.println("Ingresa las horas de viaje");
                int t = llegada.nextInt();
                if (t<=12 && t>=1) { //Se comprueba rango

                    //Se ingresa zona horaria
                    System.out.println("Ingresa la zona horaria");
                    int f = llegada.nextInt();
                    if (f<=5 && f>=-5) { //Se comprueba rango
                        //Se suman todas las horas = Cantidad total de horas de viaje
                        hora = s + t + f;

                   } else { System.out.println("Rango Invalido"); }

                } else { System.out.println("Rango Invalido"); }

             } else { System.out.println("Rango Invalido"); }

             if(hora >= 24){ //Si hora es mayor o igual que 24
                 hora -= 24; ///Resta y asignacion: A la hora se le resta 24
             } else if(hora < 0){ //Si hora es menor 0
                 hora += 24; //Se le suma 24
             }
             System.out.println("Hora Local: " + hora); //Se imprime la hora

```

Caso de Prueba:





***Junto a la entrega se adjunta un video validando los códigos, ya que URI no los valida.**