TP 24/09/2020

INTEGRANTES: Yordan Moises Vazquez Tomas Mairone Nicolas

PUNTO 2A:

NEWTON LABS 1:

Funcionamiento: Solamente se puede colorcar las cantidades de Bodys que se desea de color amarillo, no va a tener ninguna acción.
Objetos: Espacio, Body y se pueden agregar hasta 2 planetas , 1 sol y 1 luna
Métodos:

Clase SmoothMover(abstracta):

- SmoothMover(): Predetermina el constructor
- move():
- setLocation(doublé x, doublé y): Asigna una nueva ubicación para este objeto
- getExactX()
- getExactY()
- addToVelocity(Vector boost)
- accelerate(double factor)
- getSpeed(): Devuelve la velocidad de este actor
- invertHorizontalVelocity()
- invertVerticalVelocity()

Clase Body:

- Body(): Construye un body definiendo tamaño, masa, velocidad y color
- act()
- getMass(): Devuelve la masa de este cuerpo

Clase Space:

- Space(): Crea un espacio
- sunAndPlanet(): Crea un sol y un planeta
- sunAndTwoPlanets(): Crea un sol y dos planetas
- sunPlanetMoon(): Crea un sol , un planeta y una luna
- removeAllObjects()

NEWTON LABS 2:

Funcionamiento: Lo que pudimos observar en este escenario es que solamente cambia el movimiento del cuerpo con respecto al Newton Labs 1, desplazándose hacia la derecha
Objetos: Espacio, Body, se pueden agregar hasta 2 planetas, 1 sol y 1 luna.
Métodos:

Clase Body:

- Body(): Construye un body definiendo tamaño, masa, velocidad y color
- act()
- applyForces(): Aplica la fuerza de gravedad para todos los cuerpos celestiales en el universo.
- applyGravity(Body other)
- getMass(): Devuelve la masa de este cuerpo

Clase SmoothMover(abstracta):

- SmoothMover(): Predetermina el constructor
- move():Se mueve en la dirección del vector de velocidad. Esto simula el movimiento en una unidad de tiempo (dt == 1).
- setLocation(doublé x, doublé y): Asigna una nueva ubicación para este objeto
- getExactX()
- getExactY()
- addToVelocity(Vector boost)
- accelerate(double factor)
- getSpeed(): Devuelve la velocidad de este actor
- invertHorizontalVelocity()
- invertVerticalVelocity()

Clase Space:

- Space(): Crea un espacio
- sunAndPlanet(): Crea un sol y un planeta
- sunAndTwoPlanets(): Crea un sol y dos planetas
- sunPlanetMoon(): Crea un sol , un planeta y una luna
- removeAllObjects()

NEWTON LABS 3:

Funcionamiento: Lo que pudimos observar en este escenario, es que apenas inicia ya están creados 5 cuerpos celestes y una línea de “obstáculos” con distintos sonidos incluidos dentro de cada uno. Estos cuerpos se van a mover en una dirección aleatoria, cuando se choquen uno con el otro van a salir disparados en la dirección contraria a la que venían, lo mismo pasa si choca contra una pared. Si uno de estos cuerpos pasa por un obstáculo emite el sonido correspondiente.
Objetos: Espacio, body, obstáculos y 5 cuerpos celestes.

Métodos:

Clase Body:
- Body(): Construye un body definiendo tamaño, masa, velocidad y color
- act()
- bounceAtEdge(): Comprueba si hemos llegado al borde del universo, si es asi rebota.
- applyForces(): Aplica la fuerza de gravedad para todos los cuerpos celestiales en el universo.
- applyGravity(Body other)
- getMass(): Devuelve la masa de este cuerpo

Clase SmoothMover(abstracta):
- SmoothMover(): Predetermina el constructor
- move():Se mueve en la dirección del vector de velocidad. Esto simula el movimiento en una unidad de tiempo (dt == 1).
- setLocation(doublé x, doublé y): Asigna una nueva ubicación para este objeto
- getExactX()
- getExactY()
- addToVelocity(Vector boost)
- accelerate(double factor)
- getSpeed(): Devuelve la velocidad de este actor
- invertHorizontalVelocity()
- invertVerticalVelocity()

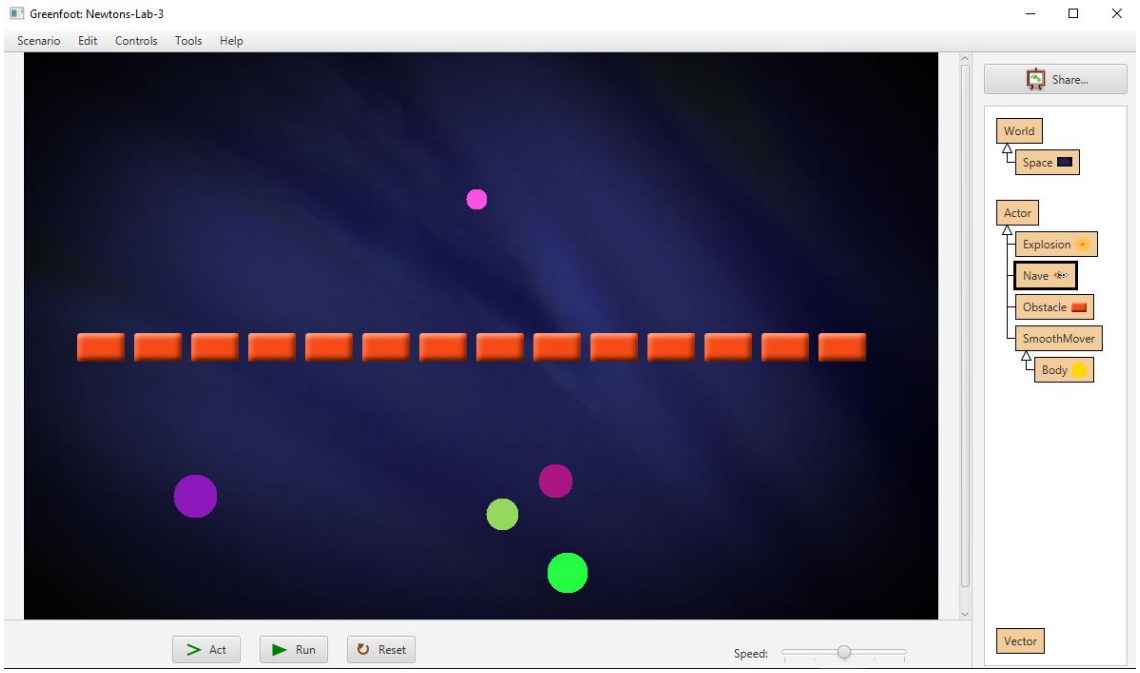
Clase Space:
- Space(): Crea un espacio
- createObstacles(): Crea los obstaculos
- randomBodies(int number): Crea los cuerpos celestes aleatorios

Clase Obstacule:
- Obstacle(String soundFile): Crea un obstáculo con un sonido asociado
- act(): Comprueba que en cada ciclo se ha golpeado a un obstáculo, si lo hizo toca un sonido.
- playSound(): Reproduce un sonido

PUNTO 2B: Lo que le agregaremos al escenario Newton Lab 3 son dos subclases de Actor llamadas “NaveEspacial” y otra “Explosion”, NaveEspacial tendrá una imagen de una nave , se desplazara con un movimiento constante y la dirección la manejara el usuario con sus teclas. Además, cuando esta nave choque con un cuerpo celeste explotara. Dentro de la nave explosión, crearemos una explosión con imágenes y sonidos similares a una.

Fotos del código:

Mostramos las subclases de Actor:



Mostramos el código de la clase NaveEspacial

```
public class Nave extends Actor
{
    public void act()
    {
        move(5);
        checkKeypress();
        checkCollision();
    }
    public void checkKeypress()
    {
        if (Greenfoot.isKeyDown("left"))
        {
            turn(-4);
        }
        if (Greenfoot.isKeyDown("right"))
        {
            turn(4);
        }
    }
    private void checkCollision()
    {
        Body a = (Body) getOneIntersectingObject(Body.class);
        if (a != null) {
            getWorld().addObject(new Explosion(), getX(), getY());
            getWorld().removeObject(this);
        }
    }
}
```

Mostramos el código de la clase Explosion:

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.util.*;
public class Explosion extends Actor
{
    private final static int IMAGE_COUNT= 8;
    private static GreenfootImage[] images;
    private int imageNo = 0;
    private int increment=1;
    public Explosion() {
        initialiseImages();
        setImage(images[0]);
        Greenfoot.playSound("Explosion.wav");
    }
    public synchronized static void initialiseImages()
    {
        if (images == null) {
            GreenfootImage baseImage = new GreenfootImage("explosion-big.png");
            int maxSize = baseImage.getWidth();
            int delta = maxSize / IMAGE_COUNT;
            int size = 0;
            images = new GreenfootImage[IMAGE_COUNT];
            for (int i=0; i < IMAGE_COUNT; i++) {
                size = size + delta;
                images[i] = new GreenfootImage(baseImage);
                images[i].scale(size, size);
            }
        }
    }
    public void act()
    {
        setImage(images[imageNo]);

        imageNo += increment;
        if (imageNo >= IMAGE_COUNT) {
            increment = -increment;
            imageNo += increment;
        }

        if (imageNo < 0) {
            getWorld().removeObject(this);
        }
    }
}
```

PUNTO 3

A:

```
#include <stdio.h>

#include <string.h>

int main () {
   int i, length ,n;
   char str[100];
   int total;
   scanf("%d",&n);
   while(n--){
      scanf("%s",str);
      length = strlen(str);

      for(i = 0, total = 0; i < length; i++) {
         if(str[i] == '0')
            total += 6;
         else   if(str[i] == '1')
            total += 2;
         else if(str[i] == '2')
            total += 5;
         else  if(str[i] == '3')
            total += 5;
         else   if(str[i] == '4')
            total += 4;
         else   if(str[i] == '5')
            total += 5;
         else  if(str[i] == '6')
            total += 6;
         else  if(str[i] == '7')
            total += 3;
         else  if(str[i] == '8')
            total += 7;
         else  if(str[i] == '9')
            total += 6;
      }
      printf("%d leds\n", total);
   }
   return 0;
}
```

B:

```
#include <iostream>

using namespace std;

int main() {

   int a,b,c;

   cin>>a>>b>>c;

   if(a==0)

   a=24;

   int sum;

   sum=a+b+c;

   if(sum>24)

   {

     sum=sum-24;

     cout<<sum<<endl;

   }

   else if(sum==24)

   cout<<0<<endl;

   else

   cout<<sum<<endl;

   return 0;

}
```