Describir las principales características de los 3 Newton Labs (funcionamiento, clases, objetos, métodos, etc)

# Newton Lab 1

Al abrir el escenario nos encontramos con diferentes clases nuevas:

* Space: Clase heredada de World, genera un objeto World pasando como parámetros (960,620,1). Además, vemos que contiene tres métodos “sunAndPlanet(), sunAndTwoPlanets() y sunPlanetMoon()” para instanciar varios objetos en el escenario. Y, por último, un método removeAllObjects() que como su nombre indica, permite remover del escenario todos los objetos.
* SmoothMover: Una clase abstracta heredada de Actor, la cual tiene varios métodos y atributos de soporte que luego le serán heredados a los objetos Body. Posee como atributos un vector privado velocity, y dos double exactX y exactY, también privadas. El constructor default de esta clase llama a this(new Vector()), mientras que el constructor iniciado con una velocidad dada toma un Vector velocity que se guarda en un atributo de la clase llamado velocity. Vemos que contiene un atributo move() el cual determina según el vector velocity la nueva localización del objeto y la setea llamando al segundo método de esta clase, setLocation() que toma como parámetros un double y un double , guardando estas variables en los atributos del objeto exactX y exactY, respectivamente, y llamando al método de su superclase setLocation() pasando como parámetro y convertidos a int. Vemos que también existe una sobrecarga de métodos planteando setLocation() pero con parámetros int y int . Además, esta clase contiene los getters para el exactX y el exactY. Esta clase incorpora además los métodos addToVelocity() que toma un Vector boost y llama a velocity.add(boost); accelerate() que toma un double factor y llama a velocity.scale(factor) pero también verifica si la longitud del vector velocity es mayor a 0.15 y de ser cierto, llama a velocity.setNeutral(); getSpeed() que devuelve la longitud del vector velocity; invertHorizontalVelocity() y invertVerticalVelocity(), que llaman a velocity.revertHorizontal() y velocity.revertVertical() respectivamente.
* Body: Clase heredada de SmoothMover, denota cualquier tipo de objeto en el espacio que tiene una masa. Posee como atributos dos constantes privadas, estáticas y final, double GRAVITY y Color defaultColor, que son igual a 5.8 y new Color(255, 216, 0) respectivamente; además una variable privada double mass. Tiene varios constructores: el primero no toma ningún argumento y llama al otro constructor pasándole como variables (20, 300, new Vector(0, 1.0), defaultColor); y el otro constructor que toma por argumento int size, double mass, Vector velocity y Color color y crea un óvalo con el color especificado, además de guardar sus atributos mass, velocity y setear su imagen. Vemos que su método act() aún no esta implementado y posee además un método getter para la variable mass.
* Vector: Esta clase crea objectos de vectores en dos dimensiones. No se hereda de ninguna otra clase. Tiene por atributos tres double , y , además de un int . Al llamar al constructor default, no se aplica ningún atributo al mismo, convirtiéndolo en un vector neutral. Si creamos un vector pasando como parámetros un int y un length, los almacena en los atributos respectivos del objecto y llama al método updateCartesian(). Si creamos un vector pasando como parámetros dos double y ; los almacena respectivamente y llama a updatePolar(). Nos encontramos además con el método setDirection(), el cual settea la dirección y llama a updateCartesian(). Otro método que se incorpora en la clase es add(), el cual toma como parámetro otro Vector y suma sus componentes y ; llamando a updatePolar(). El vector setLength() toma como parámetro un double y lo guarda en el atributo, llamando a updateCartesian(). El método scale() toma como parámetro un double factor y multiplica la longitud del objeto por el factor, luego llama a updateCartesian(). El método setNeutral() deja el vector como si lo hubiéramos creado con el constructor default. Los métodos revertVertical() y revertHorizontal() invierten el atributo y respectivamente, llamando a updatePolar() para finalizar. También se proveen una serie de getters para el , , y . Por último, los métodos updatePolar() y updateCartesian() hacen uso de funciones matemáticas para obtener la dirección, la longitud, el y él a partir de los datos ya conocidos; para tratar al vector de forma polar o de forma cartesiana.

En el apartado de funcionamiento, podemos ver al ejecutar el método sunPlanetMoon() los planetas no tienen ningún efecto entre sí, ya que se quedan quietos.

# Newton Lab 2

Al abrir el escenario por primera vez, encontramos los siguientes cambios:

* En la clase Space: El método removeAllObjects() paso a llamarse removeObjects() y ahora toma como parámetro getObjects(null).
* En la clase SmoothMover: No notamos ningún cambio nuevo.
* En la clase Body: Vemos que ha habido varios cambios:
  + En el método act(): Se ha agregado la llamada a dos métodos applyForces() {nuevo de la clase Body} y a move() {heredado de la clase SmoothMover}
  + Nuevo método act(): Crea una lista de todos los bodies generados en World y llama a la función applyGravity() en el body.
  + Nuevo método applyGravity(): Crea dos variables double y las cuales utiliza para crear un vector ; además consigue la distancia calculando la raíz cuadrada de . Luego calcula la fuerza con , y la utiliza para calcular la aceleración, la cual es aplicada al vector , finalizando el método llamando a addToVelocity().
* En la clase Vector: No parece haber cambios.

En el apartado de funcionamiento, podemos ver al ejecutar el método sunPlanetMoon() que los mismos se ven afectados por las fuerzas de la gravedad, simulando el sistema solar de la Tierra, la Luna y el Sol. Definitivamente no es perfecto ya que la Luna termina por colapsar con la Tierra, lo cual manda a la misma cerca del sol disparada y a la misma Luna directo al centro del Sol.

# Newton Lab 3

Este escenario implementa una serie de nuevos cambios:

* En la clase Space:
  + Nuevos atributos: Se agrego un String[] soundFiles conteniendo { "2c", "2d", "2e", "2f", "2g", "2a", "2b", "3c", "3d", "3e", "3f", "3g", "3a", "3b" }.
  + Métodos eliminados, para hacer lugar a nuevos: En el constructor de la clase ahora se llama a los métodos createObstacles() y randomBodies() que reemplazan a los métodos ya definidos sunAndPlanet(), sunAndTwoPlanets() y sunPlanetMoon(). El método createObstacles() que instancia n número de objetos Obstacle (nueva clase) donde n es igual a la cantidad de elementos en el String[] soundFiles, pasando como parámetros:
    - .
    - ++
  + El método randomBodies() genera un número de objetos Body (determinado por su parámetro int number); con:
    - size entre 20 y 50 (randomizado).
    - mass equivalente a siete veces el size.
    - direction aleatoria entre 0 y 360.
    - módulo de velocidad entre 0 y 1.5.
    - color RGB aleatorio.
    - posición aleatoria entre 0 y su getWidth() y getHeight().
* En la clase SmoothMover:
  + No parece haber cambios en esta clase.
* En la clase Body:
  + Atributo GRAVITY: Cambiado a 7.8.
  + Atributo color: Renombrado a defaultColor.
  + Método applyForces():
    - Agregada una verificación para que si getSpeed() es mayor a 7, se desacelera por 0.9.
  + Nuevo método bounceAtEdge():
    - Este método incorpora la funcionalidad al body para que si este esta tocando cualquier borde del mapa, se invierta la velocidad en la dirección opuesta a la pared, y además se desacelere por 0.9.
* En la clase Vector:
  + No parece haber cambios.
* Nueva clase “Obstacle”:
  + Posee como atributos un String sound y un boolean touched = False, ambos privados.
  + Su constructor toma por parámetros un String soundFile y lo guarda en el atributo sound del objeto.
  + En el método público act() se implementa la funcionalidad de que si esta en contacto con cualquier otro objeto Body se reproducirá un sonido y se seteará la imagen haciendo uso del boolean touched y la variable Actor body donde se guarda el body que está interceptando al Obstacle.

En cuanto al apartado de funcionamiento, al correr el código vemos que se crea un escenario con una línea horizontal de 14 Obstacles al medio del mismo, con 5 bodies generados de manera aleatoria. Los mismos comienzan a ser afectados por la gravedad y al interceptar los Obstacles emiten un sonido distinto según cual tocaron, como si fuera un Piano.

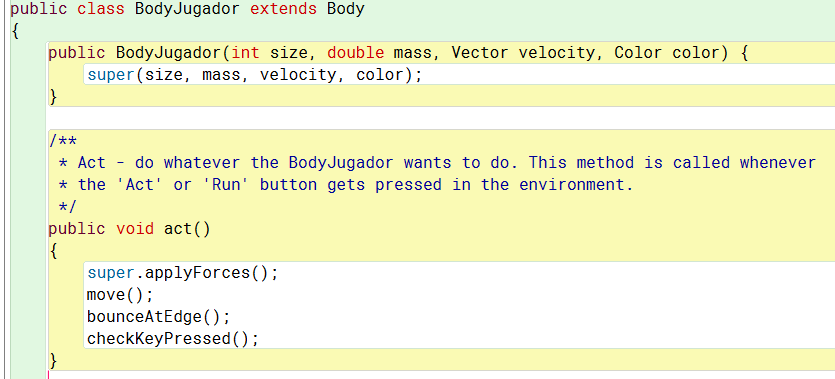
# A partir del Newton Lab 3, incorporar una modificación significativa a elección

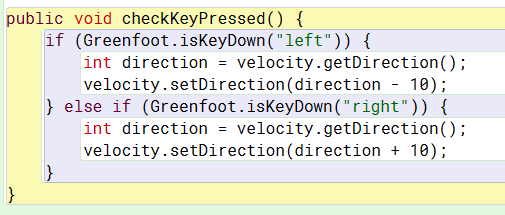
Agregamos una nueva subclase de Body llamada BodyJugador, que hereda todas las funcionalidades de Body. Además, desarrollamos métodos que permiten que el mismo rote sobre su eje al presionar las teclas “left” y “right”, para lo cual tuvimos que convertir varias variables de Vector y de Body a protected para poder utilizarlas dentro de nuestra clase sin restricciones con sus métodos getter y setter.

Este jugador se diferencia de los demás ya que aparece en la esquina superior izquierda del escenario y como los demás bodies generados automáticamente tiene masa, tamaño, dirección y velocidad aleatorios en base a su masa y tamaño.

A pesar de poder controlar su dirección, se vuelve complicado mover al objeto debido a los diferentes cuerpos creados en el escenario, los cuales aplican su gravedad sobre el mismo obligándole a cambiar constantemente de dirección y velocidad.

Código





Demostración:

{ver vídeo}

Ejemplo:

