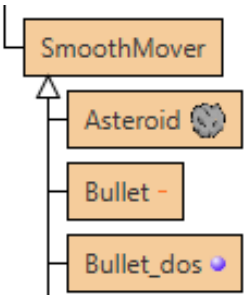
Resolución trabajo práctico clase 28/09 escenario “Asteroids”

Punto 1

Nuestra solución a este punto fue agregar una nueva clase denominada “Bullet_dos”, la cual tiene la misma funcionalidad que la clase “Bullet”, solo se le cambia la imágen. Además más adelante será utilizada por el segundo jugador que añadiremos posteriormente.



Punto2

La solución que encontramos a este punto fue invertir el “boost” al apretar la flecha hacia abajo (la funcionalidad para la flecha arriba ya existía), cambiándose el color del “fuego” a azul para indicar la animación del movimiento, aquí debajo dejamos imágenes de la implementación y el código.

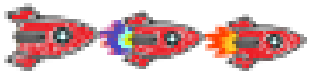
```
private void checkKeys()
{
    ignite(Greenfoot.isKeyDown("up"), Greenfoot.isKeyDown("down"));
    desacelerate(Greenfoot.isKeyDown("down"), Greenfoot.isKeyDown("up"));
```

```
private void ignite(boolean boosterOn, boolean frenandoOn)
{
    if (boosterOn) {
        setImage(rocketWithThrust);
        addToVelocity(new Vector(getRotation(), 0.3));
    } else if (!frenandoOn){
        setImage(rocket);
    }
}
```

```
private void desacelerate(boolean frenandoOn, boolean boosterOn) {
    if (frenandoOn) {
        setImage(rocketSlowingDown);
        addToVelocity(new Vector(getRotation(), -0.3));
    } else if (!boosterOn) {
        setImage(rocket);
    }
}
```

Punto 3

Para este punto agregamos una nueva clase llamada “Rocket2”, esencialmente el código es igual al de “Rocket”, solamente le modificamos las teclas con las que se mueve y la imagen (le damos un color distinto para diferenciarlo del primero). Debajo el código.



```
ignite(Greenfoot.isKeyDown("w"), Greenfoot.isKeyDown("s"));
desacelerate(Greenfoot.isKeyDown("s"), Greenfoot.isKeyDown("w"));
teleport(Greenfoot.isKeyDown("v"));
if (Greenfoot.isKeyDown("a"))
{
    turn(-5);
}
if (Greenfoot.isKeyDown("d"))
{
    turn(5);
}
if (Greenfoot.isKeyDown("x"))
{
    fire();
}
if (Greenfoot.isKeyDown("z"))
{
    startProtonWave();
}
```

Punto 4

Agregamos una funcionalidad la cuál hace que la nave se ubique en la posición inversa a la que se encuentra dando el efecto de una “teletransportación”, para hacerlo utilizamos un boolean como bandera “isTeleporting” para evitar que la funcionalidad genere un “spam” de teletransportaciones y solo se ejecute una vez cuando se presiona la tecla.

Para el Jugador 1 se utiliza la tecla “c” y para el jugador 2 la tecla “v”.

```
teleport(Greenfoot.isKeyDown("c"));
```

```
teleport(Greenfoot.isKeyDown("v"));
```

```
private void teleport(boolean teleportOn) {
    if (teleportOn && !isTeleporting) {
        this.setLocation(600 - getX(), 500 - getY());
        isTeleporting = true;
    } else if (!teleportOn) {
        isTeleporting = false;
    }
}
```

Punto 5

Para solucionar este problema creamos una clase distinta llamada “RockDorada” la cual en esencia es una copia de la clase “asteroids” con la diferencia de que es dorada, más veloz y que al romperse no se divide en rocas más pequeñas, si no qué otorga los puntos al instante. Evidentemente son mayores que los otorgados al romper un asteroid.

```
private int points = 25;
```

```
private void breakUp()
{
    Greenfoot.playSound("Explosion.wav");
    Space space = (Space) getWorld();
    space.addPoints(this.points);
    getWorld().removeObject(this);
}
```

Además, para que el tablero vaya presentando en tiempo real el puntaje, tuvimos que crear en la clase “Space” el siguiente método.

```
public void addPoints(int pointsAdd) {
    scoreCounter.add(pointsAdd);
}
```

Punto 6

Para aplicar esta funcionalidad, a la clase “Rocket” y “Rocket2”, le agregamos una variable publica de tipo entero llamada “lives”, la cuál utilizamos en el siguiente método

```
private void checkCollision()
{
    Asteroid a = (Asteroid) getOneIntersectingObject(Asteroid.class);
    RockDorada r = (RockDorada) getOneIntersectingObject(RockDorada.class);
    if (a != null || r!=null)
    {
        Space space = (Space) getWorld();
        space.addObject(new Explosion(), getX(), getY());
        if (lives > 1) {
            lives--;
            space.resetGame();
        } else {
        space.removeObject(this);
        space.gameOver();
    }
    }
}
```

```
public void resetGame() {
    removeAsteroids();
    for (Rocket jugador1: getObjects(Rocket.class)) {
        jugador1.setLocation(450, 450);
        jugador1.accelerate(0);
        int vidasj1 = jugador1.lives;
        String vidas1 = "Jugador 1: " + Integer.toString(vidasj1) + " vidas.";
        showText(vidas1,225,475);
    }
    for (Rocket2 jugador2: getObjects(Rocket2.class)) {
        jugador2.setLocation(450, 475);
        jugador2.accelerate(0);
        int vidasj2 = jugador2.lives;
        String vidas2 = "Jugador 2: " + Integer.toString(vidasj2) + " vidas.";
        showText(vidas2,475,475);
    }
    addAsteroids(3);
}
```

```
public void removeAsteroids() {
    removeObjects (getObjects(Asteroid.class));
    removeObjects (getObjects(RockDorada.class));
}
```

En resumen, estos métodos comprueban si al morir al jugador le quedan suficientes vidas como para seguir jugando y de ser así, resetea el escenario, creando nuevos asteroides y posicionando a los jugadores sin aceleración en la parte inferior derecha del escenario. Al hacerlo muestra un texto en la parte inferior de la pantalla donde se indica la cantidad de vidas restantes.

Si el jugador se queda sin vidas entonces se llama al método “space.gameOver()” el cual detiene el juego.