SpaceInvaders Tutorial

Setting up our World

Open Greenfoot and double-click on MyWorld. - Now in the editor, everywhere that there is the word MyWorld replace it with SpaceInvadersWorld. (Note the capitalization). - Change the size to 1024 by 800. // super(1024, 800, 1); - Lets set a nice space type background as well.

Saving our Work and adding some icons and sounds

- Take note of the directory where your work will be saved. When you do a save as … You should get a dialog that tells you where everything is currently being saved to. Write it down for later. However, don't complete the save As … - just do a save for now.

Reminder - Animation

Remember that animation is achieved by placing an object at an x, y location on the screen. Everytime the screen is redrawn if that x and/or y coordinate has changed the object will appear as if it had moved there. This can be thought of much like movie frames in film.

Understanding Objects

- Assume we have a class of Person. Each person can know their name, age, and team that they belong to. They might also know how to do the following: sit(), stand(), moveForward(someNumberOfSteps) //More on functions later
- Let's look at this on Miro https://miro.com/app/board/uXjVO6NNQBE=/?moveToWidget=3458764526

Understanding Actors

Actors are the basic building blocks within Greenfoot. Explore how to create an Actor (Rocket) - Set the icon - Now we will use the Greenfoot environment to look at some of the methods inherited from Actor - turn(), getRotation(), getX(), getY(), act()

Note that all Actors will have an act() method. This is where all (most) of your programming will occur.

How to know what methods that Actors can respond to.

From your **main screen** in the Greenfoot environment, select Help then Greenfoot Class Documentation.

How To Do User Input

There is another class in the Greenfoot environment named - well - **Greenfoot**. This class provides some useful functions for communicating with the player of the game. One of these methods is the **isKeyDown** method.

So, after we understand how this method works, we can now make our act method look like this. We haven't discussed the **if** statement yet, but for now, I think you will get the idea.

```
public void act()
{
    if (Greenfoot.isKeyDown("left")) {
        move (-5);
    }
}
```

That works great - our Rocket now moves to the left. Enhance the code so that the Rocket will also move to the right.

First Exercise - Create some objects

- Create a Rocket subclass (from Actor)
  - set the icon
  - make your Actor move across the screen from right to left 8 pixels when the left or right arrow keys are pressed.

Our first Bug...

It seems that our Rocket is facing the wrong direction. Actually, this is the right direction as by default all objects are to be facing east (0 degrees) but I digress. To fix this bug we need to figure out a different strategy. Well, luckily we know that we can turn our object by any number of degrees or we can set it's rotation (setRotation()) but we only need to do this once - not everytime the Rocket is asked to act(). As it turns out, all actors are also sent a special method call addToWorld(World world) once (and only once) whenever they are added to the world. This is convenient to do some one time initialization. Let's see how this works.

```
public void addedToWorld(World world) {
    setRotation(270);
}
```

Note: This must come after the act() method (After the last **}** in **act**) but before the last **}** in the class Rocket.

This method is called by the Greenfoot system only if it exists in the class and again, it is only called once.

This has solved one problem for us, but has now introduced a problem with the way we move.

A different way to move

We have a problem - we want our Rocket to move east or west but it is facing north. Now we have to think about the coordinate system. Assume that the rocket is sitting somewhere near the bottom of the screen at let say 200, 800 (x = 200, y = 800). If the left arrow key is pushed, we would like the rocket to be at a new location of x = 192, y = 800. Of course, if the right arrow key had been pushed we would have expected the Rocket to be at x = 208, y = 800 and if the right arrow key was pushed again then the Rocket would be at x = 216, y = 800 and so on. Notice the y coordinate does not change because we only want the rocket to move left or right not up and down. So let's see how this might be in code.

```
    public void act()
    {
        int currentX;
        int newX;
        int newY;
        currentX = getX();
        newX = getX();  // Assume the new x will be the same as the old
        if (Greenfoot.isKeyDown("left")) {
            newX = currentX - 8; //We need to change
        }
        if (Greenfoot.isKeyDown("right")) {
            newX = currentX + 8;
        }
        newY = getY();  // The y position will not change
        setLocation(newX, newY);

    }
```

Let's picture this in memory

```
            +-----------+
currentX    |   300     |   // result from getX()
            +-----------+

            +-----------+
newX        |   300     |   // result from getX()
            +-----------+

            +-----------+
newY        |   800     |   // result from getY()
            +-----------+
```

Exercise

- Make the changes to your Rocket class.
- Create a new class Alien.
- Make your Alien move across the screen from right to left by 3 pixels or so.

Understanding Variables in Java

Variables are things that can hold on to values that might change over the course of your program. Variables can be thought of as labelled "pockets" or "boxes" to hold things until you later want to retrieve the things or change the things. When we **declare** a variable we must also declare what **type** of object this variable is going to hold on to. For example, if I want a variable to hold on to the current x coordinate of something I might declare it like this.

```
int x;
```

This says that x is a variable (named box) that can hold onto any integer value. (Integers are whole numbers - positive or negative).

Remember that any time you declare a variable you always declare the type first. So the general form for declaring variables is

someType variableName

where some common **someTypes** are int, String, Object, or Rocket.

And remember to think of it as a named box where a value can be put in or taken out.

```
  +-----------+
x |   300     |
  +-----------+
```

The way that we put things into the box is by assigning a value to the box. (i.e the value 300 was put into the above box by the following code)

```
x = 300;    //Note the semicolon at the end;
```

and the way that we get values out of the box later is by refering to the box name.

```
y = x + 1;
```

IMPORTANT NOTE: When there is an "=" sign you need to think of it as an assignment operation not as a mathematical equality sign so the above should be said as "Take whatever is in box x, add 1 to it and put it in box y"

This would result in the following:

```
  +-----------+
x |   300     |
```

```
  +-----------+

  +-----------+
y |   301     |
  +-----------+
```

Variables must be defined within the class you are working in.

```
public class YourClass extends Actor
{
    //Variables must be defined here before the act method.
    //This is not always true, but for now - just trust me. :-)

    int numberOfPixels = 5;

    public void act() {
         move(5);   // hard-coded 5
         // is the same as ...
         move(numberOfPixels);
    }
}
```

Understanding the if statement

The "If" statement

The if statement must occur within your act() method.

```
...
if ( someBooleanExpression )
{
    statement1;
    statement2;
        ...
    statementN;
}
else
{
    elseStatement1;
    elseStatement2;
        ...
    elseStatementN;
  }
}
```

Let's break that down - firstly what is someBooleanExpression. It is an expression that after it is evaluate results in a "true" or "false" value. For example

```
4 <= 6;  // this results in a "true" value
```

or

```
int y = 10;
y <= 6;  // this results in a "false" value.
```

or say that getX() is a function that currently returns 1 (If the Actor was against the left side of the screen) then

```
getX() <= 1    //this results in a "true" value.
```

Combining this we now have a valid "if" statement

```
if (getX() <= 1) {
    // Do something that makes sense here
    // Like ... Let's bounce
}
```

```
    public void act()
    {
        int currentX = getX();
        int newX = currentX + x;
        if (newX <= 1) {
            x = -x;
        }
        if (newX >= getWorld().getWidth() - 1) {
            x = -x;
        }
        setLocation(newX, currentY);
    }
```

The above snippet should help you modify your Alien class so that it bounces. Make the above changes to your Alien class. Once you have that working, think about the following: How would you make the Aliens move down the screen (when they get to one side or the other) closer and closer to the Rocket until they crash into the Rocket??

If you think that you know how to do that, please try on your own.

```
    public void act()
    {
        int currentX = getX();
        int newX = currentX + x;
        int currentY = getY();
        if (newX <= 1) {
            x = -x;
            currentY = getY() + 30; //
        }
        if (newX >= getWorld().getWidth() - 1) {
            x = -x;
```

```
            currentY = getY() + 30;
        }
        setLocation(newX, currentY);
    }
```

The Bullet Class

For now we have enough to create a Bullet class. - Assign it an appropriate icon (Use one of the small ball classes for now. We can tweak icons later.) - By default it should also have its rotation facing north. (up) - When it is on the screen it should move with velocity of 10. - If it hits the top of the screen it should be gone. To remove an object from the screen is to remove it from the world and that looks like this:

```
SpaceInvadersWorld world = getWorldOfType(SpaceInvadersWorld.class);
world.removeObject (this);
```

Hint: removing objects from the world has to be the last thing in an act method.

Now drop this Bullet on the screen and test that it behaves as expected. We will get it to fire from the rocket in the next section. However, notice that this class is a little different from all the other classes because it doesn't already exist on the screen from the first moment. Normally, this would be created on the fly - (I.e - when someone hits the space bar)

Dynamic Object Creation

Now that we have all the behaviour working in the bullet class it would be nice if we could hit the space bar to fire a bullet (or missile or phaser). To create a Bullet on the fly, we need a snippet of code that looks like this:

```
Bullet bullet = new Bullet();
getWorld().addObject(bullet,???, ???); //Where (x,y) should it be added?
```

Well we know that the x coordiant will be the same as the x coordinate of the Rocket and the y coordinate (slightly modified) will be as well. So this following snippet might do the trick.

```
Bullet bullet = new Bullet();
getWorld().addObject(bullet,getX(), getY() - 10);
```

Exercise - Add the dynamic creation of the Bullet to the Rocket class

Hints: - you know how to do if statements - you know how to get keyboard input.

Problems: It looks like a lot of bullets are being fired. There should only be one. You will have to import Java.util.List below.

```
List <Bullet> bullets = getWorld().getObjects(Bullet.class);
if (bullets.isEmpty())
{
    if (Greenfoot.isKeyDown("space"))
    {
        Bullet bullet = new Bullet();
        getWorld().addObject(bullet,getX(), getY() - 10 ;
    }
}
```

Collision Detection.

Now all that we need to know is whether the Alien is hit by a bullet. This is called collision detection and the way that this works is by asking the Actor whether there is one intersecting object currently touching the Actor (Alien). If there is, that object will be returned in a variable otherwise **null** will be returned.

So within the Alien class, in the act method after everything else (but still in the act method) add these lines of code.

```
Bullet bullet = (Bullet) getOneIntersectingObject(Bullet.class);
if (bullet != null)
{
    SpaceInvadersWorld world = getWorldOfType(SpaceInvadersWorld.class);
    // world.addScore(10);
    world.removeObject(bullet);
    world.removeObject(this);
}
```

Adding Scores

- Import the Label class

Adding more Aliens - Understanding for loops

Some useful Snippets of Code.

Getting the World

Whenever you need to get information to or from the SpaceInvadersWorld you need to do it with this line of code.

```
SpaceInvadersWorld world = getWorldOfType(SpaceInvadersWorld.class);
```

Now you can invoke methods that exist within your SpaceInvadersWorld class like such

```
SpaceInvadersWorld world = getWorldOfType(SpaceInvadersWorld.class);
int width = world.getWidth();
```

Printing to the console.

```
System.out.println ("The value of x is" + getX()");
```