**Endangered Animal**

This is a game about endangered animal. There will be random animals going from left to right; our aim is to catch as many animals as we can but not the endangered animals. The game will end if we caught too many endangered animals.

**World**

First of all, we need to create a background for our world.  
On the right hand side of the screen, right click on the class World. There will be a list pop up and click on the ‘new subclass..’ option.  
A new screen will pop up. Enter any name you want for the world (I will call it myWorld) and select an image in the backgrounds category. Click ok then we are done! We just created a new world!  
  
To edit myWorld(or whatever you name it), right click on the class myWorld and select ‘open editor’. A screen will pop up and this is where we program what we want! There is a method with the same name as your world (In this case is myWorld) and inside of it there is a code ‘super(600, 400, 1)’. We can change the size of the world by changing the first two numbers. From now on I will set it to ‘super(1000, 600, 1)’ as below.

import greenfoot.\*;

public class myWorld extends World

{

public myWorld()

{

super(1000, 600, 1); //Change the size of the world  
}

}

Other words shown on your code is just to tell you what to do. You can also use this to remind yourself what something does.

For example, the sentence ‘//Change the size of the world’ is just to remind me that this code change the size of the world. It does not affect the code, it does not matter if it is here or not.

When we compile it, we will get the world with the image we selected.

Note: All the myWorld in the code might not be the same as yours. It depends on how did you name the world.

Exercise:

1. Experience yourself, change the world into different sizes.

**Actor**

Now we have a world, its time to put something inside. All objects we put inside the world is called Actor.

Create a subclass under the class Actor and name it human, select any image in the people category. We would like to put the human inside the world, but how?

To do this, we have to edit human. When you open human to the coding screen, basically there is nothing in it. In order to create a human object, simply add the code (constructor) below into the class human:

public human()

{

}

Then the overall code will be (I will only show the part(s) of the code that I edited in the rest of the book):

import greenfoot.\*;

public class human extends Actor

{

public human()

{  
}

public void act()

{

}

}

After we compiled it, when we right click on the human class, there will be an new option called ‘new human()’. Click on it and the image you selected before pop up, now you can put it anywhere in the world.

Exercise:

1. Create two subclasses (Under Actor) ground and floor using the image

‘brick.png’ under objects category. Edit them in the same way so that you can put it inside the world.

We will be using these two subclasses later.

**Running**

It is boring to just put the objects inside the world without doing anything. Let us make the human move!

Here we have to introduce a few methods that already exists inside greenfoot, we will be using it to make the human move:

1. getX() – This is used to get the X-coordinate of the object inside the world.
2. getY() – This is used to get the Y-coordinate of the object inside the world.
3. setLocation(int, int) – This is used to put the object into the location we want using X,Y-coordinate.
4. Greenfoot.isKeyDown(“Key”) – This is used to find out what did the user press on the keyboard.

Create two methods called moveLeft and moveRight. These will be used to make the human move left or right.

public void moveLeft()

{

setLocation(getX() – 5, getY()); //Put the image into new location

}

However, we can define the value 5 into a variable at the beginning of the class so that if we want to change the speed of moving, we don’t have to find this method and change the value.

public class human extends Actor

{

private speed = 5;

.

.

.

.

public void moveLeft()

{

setLocation(getX() – speed, getY());

.

.

.

The getX() and getY() get the current coordinate of the human. We set the human into a new location with smaller X-coordinate and leave Y-coordinate unchanged (Not going up and down).  
This will make it looks like moving to the left as X-coordinate getting bigger from left to right.

Try to make the moveRight method as an exercise later.  
  
However, how would the human know it should go left or right? That’s right, we have to check what did the user press on the keyboard. For this, lets create a method called checkKey.

public void checkKey()

{

if (Greenfoot.isKeyDown(“left”)) //Check if user pressed left key

{

moveLeft(); //If pressed left key move to the left

}

else if (Greenfoot.isKeyDown(“right”)) //Check if user pressed right key

{

moveRight(); //If pressed right key move to the right

}

}

Now we are one step away. Did you notice there is a method called act inside the human class? Put the checkKey method in it to make it calls this method when the program runs.

public void act()

{

checkKey();

}

What this does is that when the program runs, it will call the checkKey method and run it. Then the checkKey method will check what the user pressed. If pressed left key, call moveLeft method and run it, which finally move to the left. Same thing happens when user pressed the right key.

After finishing the moveRight method, compile it and put the human in the world. Press the run button at the bottom of the screen. You can now control the human to move left or right!

Exercise:

1. Create the moveRight method. It is similar to the moveLeft method.
2. Change the speed of the human so that it moves faster/slower.

**Changing the Size of an Object**

This is different from changing the size of the world.  
Open the ground subclass and remove the act method.

Input the code below under the class:

private GreenfootImage ground = new GreenfootImage(“brick.png”);

What this does is create an object called ground with the image ‘brick.png’. Now add a constructor so that we can put it in the world. Do you remember how to do it?

public ground()

{

}

In order to change the size of the object, we have to do something inside the constructor:

public ground()

{

ground.scale(150, 10);

setImage(ground);

}

ground.scale(int width, int height) changes the size of the image ‘brick.png’, setImage(Image) sets ‘brick.png’ be the image of this class.

Now when you compile it, you will get a bigger ground image.  
  
Exercise:

1. Change the size of the class floor so that it fits the width of your world.
2. Can you also change the size of human? Try it out!

**Jumping and Falling**

Put a ground object at a random place inside the world and put a human object on top of it. Try to move the human. Does it falls when not walking on the ground? No! So we have to make it falls when leaving the ground.

Open the human class once again. Now we create a method called fall.

public void fall()

{

setLocation (getX(), getY() + fallingSpeed);

fallingSpeed = fallingSpeed++; //the value of fallingSpeed keep increasing,

// which mean fall faster and faster

}

As Y-coordinate gets bigger going down, if we increase the value of Y-coordinate the image will go down.

Like before, fallingSpeed is just a variable that I defined at the beginning of the class.

Now call this method in the act method.

public void act()

{

checkKey();

fall();

}

Now compile it and put a human in the world. When you pressed run the human will fall to the bottom of the screen.

Put a ground under the human and try again. What happen? The human still falling! Why? This is because we didn’t tell the human to stop falling when it touches the ground. Let us try to add this function in it.

As before, open the human class. This time we create a method to check if the human touching the ground. Call it onGround.

public boolean onGround()

{

Actor under = getOneObjectAtOffset (0, getImage().getHeight()/2,

ground.class);

return under != null;

}

getOneObjectAtOffset(int dx, int dy, Object) is a greenfoot method that return the Object that is located at the location dx and dy. What this method does is return true when there is ground under human.

After we have a method to check if there is ground under human, now we can create a method to tell the human what to do when it is true or false.

public void checkFall()

{

if(onGround())

{

fallingSpeed = 0; //Recall in the fall method, when fallingSpeed = 0

//the image will just stay where it is

}

else

{

fall(); //If not on the ground, fall

}

}

Last step, we just have to call checkFall() method in act() method. Also remove the fall() method inside act() method otherwise the human would just fall no matter what.

public void act()

{

checkKey();

checkFall();

}

Now compile and run it, the human will walk on the ground and fall when not.  
  
However, we don’t just want to go down, we also want to get up. Hence we need to create a jump method.

public void jump()

{

fallingSpeed = -jumpSpeed;

fall();

}

jumpSpeed is just a variable define at the beginning of the class. Try it out and find the value you think is the best.  
By setting the fallingSpeed to negative value, when running the fall() method, the human will first go up since the Y-coordinate gets smaller, then go back down since the fallingSpeed keeps getting bigger.

We would like to make the human jumps when we press a key. How can we do it?

Exercise:

1. Add the ability to jump for the human when pressing the key “space”.
2. Create onFloor() method which similar to the onGround() method.
3. Edit the checkFall() method so that the human also stop falling when on floor.

**Animal**

It is time to add animals into the world! First create a subclass called animal under Actor. Then create a subclass normal under animal. After that create a subclass pig and select an image for pig.

Now let us tell the pig to move when the world runs.  
  
public class pig extends animal

{

public void act()

{

move(2);

}

}

move(int) is another method in greenfoot that tell the object to move from left to right.

Now compile it and see what the pig does. Yeah! It moves towards the right, but get stick on the right hand side of the screen.

What if there are many pigs on the screen? They would all stick on the right hand side of the screen which is not what we want to happen. How can we fix this?  
Easy, just remove the pig when it reaches right hand side of the screen. But how?

if (getX() >= getWorld().getWidth() - getImage().getWidth()/2)

{

getWorld().removeObject(this);

}

This is the code we needed inside the act() method of the pig.

The condition is that if the X-coordinate of the pig (getX()) greater than or equal to (>=) the width of the world minus the half of the width of the pig (getWorld().getWidth() - getImage().getWidth()/2) then remove the pig (getWorld().removeObject(this)).

Note: getWorld().getWidth() is getting the width of the world and getImage().getWidth is getting the width of the pig. You have to tell the program what are they looking for.

Now if you compile and run it, the pig will disappear when it reaches the right.

Exercise:

1. List some endangered animals. Find three of them inside the image categories.
2. Find two animals, which are not endangered inside the image categories. Create two subclasses, one for each, under the normal class. Edit them so that they act the same as the pig class.
3. Create a subclass called endangered under the animal class. Create three subclasses for the three endangered you listed in 10, under the endangered class. Edit them so that they act the same as the pig class.
4. (Hard) Edit the human class, so that it can catch the animals. ie Make the animals disappear when human touches them.

Hints: Recall the work before. How did we find out if human touches other objects like ground or floor? How did we make the animals disappear if human touches them?

**Random Action**

We don’t want to keep add animals into the world by ourselves while playing the game. We should make the program to do it for us. But when? Which animal to add? Where should it appear? How many should we add?  
All the problems can be solve by using random function inside greenfoot.(getRandomNumber(int)).

getRandomNumber(int) will generate any number from 0 to one less then the int. For example, getRandomNumber(10) will generate number from 0 to 9.

This time, open the world class, create an act() method. We will set up a timer, which defined at the beginning of the class. If the time can be divided by 100, then we have a chance to create a animal.  
  
public void act()

{

if(timer % 100 == 0) // if timer can de divided by 100

{

if(Greenfoot.getRandomNumber(6) == 0) //I use 6 because we

// have 6 animals

{

addObject(new pig(), getWidth()/2, getHeight()/2);

//Create a pig in the middle of the world

timer++; //The value of timer keep increasing by 1

}

}

else

{

timer++; //Otherwise the value of timer will be constant

//ie if timer won’t increase and it stays at 5, then

// no animal will be created

}

}

If you compile it and run it, there will be pig randomly generated in the middle of the world and move to the right and disappear.

Exercise:

1. Edit the code above and make the pig generate at the very left of the world but random Y-coordinate.
2. Edit the code above so that it can generate different types of animals.

Also at the very left of the world but random Y-coordinate.

**Set Up the World**

It is annoying to set up the world every time you play it right?

We can just set it up inside the world class so that we don’t have to do it over and over again.

We will be using addObject(Object, int dx, int dy) again, which add the object at given coordinates.

We will be editing the constructor, myWorld() is this case, to set up everything.

public myWorld()

{

super(1000, 600, 1);

addObject(new human(), 550, 270);

}

Now if you compile it, you can already see a human stand in the middle of the world.

Exercise:

1. Edit the world so that you have some grounds set up in such a way that you would be able to jump up and down to catch the animals. Also set up the floor and the very bottom of the world.

**Score**

Now we need something to count our score. We don’t have to do much since there is a class called Counter inside greenfoot. Right click on the greenfoot icon on the tab bar and click edit, import class. A list will pop up and we just have to choose Counter. This is a class allow us to add or subtract easily.

Next, go back to myWorld class and create an object called counter at the beginning of the class.

public class myWorld extends World

{

private int timer = 0;

Counter counter = new Counter(“Score: “);

.

.

After counter is created, we can use the method inside the Counter class in myWorld class. The part “Counter(“Score: “)” also allow us the display the score on the screen. We will do this later.

Now we have to make two methods under the myWorld class in order to add and subtract the score. I will show you how to do the add method and you can try to finish the subtract method for exercise.

public void addScore()

{

counter.add(20); //add 20 to the score when this is called

}

It is simple as that. We tell the program to add 20 into the Counter call named counter. However, when we want to add the score? Of course when the human caught the normal animals. To do this we have to open the human class once again and go to the checkAnimal() method.

public void checkAnimal()

{

Actor endangered = getOneObjectAtOffset (0, getImage().getHeight()/2, endangered.class);

Actor normal = getOneObjectAtOffset (0, getImage().getHeight()/2, normal.class);

if (endangered != null)

{

getWorld().removeObject(endangered);

}

else if (normal != null)

{

getWorld().removeObject(normal);

((myWorld) getWorld()).addScore(); //call the addScore method inside

//myWorld class

}

}

So now when we compile and play the game, the score will increase when you caught one normal animal. But we couldn’t see it right? Let us make a scoreboard to show the result.

Exercise:

1. Finish the subtract method in myWorld class and call it inside the checkAnimal() method in human class so that you subtract the score when you caught endangered animal.

The method to subtract the score is ‘counter.subtract(int)’.

1. Create a Counter called EndangeredAnimal = new Counter(“Endangered Animals Caught: “) in the myWorld class.

Create a method inside the myWorld class to add one into the counter Endangered Animal.

1. Edit the checkAnimal() method insdie the human class once again so that it add one to the counter EndangeredAnimal when you caught an endangered animal.

**Showing the score on the screen**

We have to show the score and number of endangered animals caught on the screen so that player can keep trick of the game.

We just have to edit the constructor of the myWorld class then we would be able to see the score on the screen.

However, let us create a Scoreboard class like the way we created a Counter class. We will be using it later.

public myWorld()

{

super(1000, 600, 1);

addObject(new ground(), 300, 200);

addObject(new ground(), 800, 200);

addObject(new ground(), 550, 500);

addObject(new ground(), 550, 300);

addObject(new ground(), 550, 100);

addObject(new ground(), 300, 400);

addObject(new ground(), 800, 400);

addObject(new ground(), 0, 600);

addObject(new floor(), 500, 600);

addObject(new human(), 550, 270);

setPaintOrder(ScoreBoard.class, Counter.class);//Just to set up for end game

addObject(counter, 100, 560);

}

This isn’t anything new, we just add the object counter at the coordinates (100, 560).

Exercise:

1. Do the same thing so that it shows the number of endangered animals caught on the screen. Add it at the bottom right of the screen.

**Game Over**

The game is basically done, but we don’t want to play it forever: We want to make it stops at some point.  
Hence we can make a method inside the human class.

public void gameOver()

{

getWorld().addObject(new ScoreBoard(((myWorld) getWorld()).getValue()), getWorld().getWidth()/2, getWorld().getHeight()/2);

// Show the score at the end of the game on the middle of the screen

Greenfoot.stop();

}

Then call this method inside the act() method of human class.

public void act()

{

checkKeys();

checkFall();

checkAnimal();

gameOver();

}

Now if you compile it and run it, the game would just stop no matter what. This is because the gameOver() method tell greenfoot to stop no matter what. We have to change it so that it stops only certain condition is reached. Let say caught 5 endangered animals.

First of all we have to make a variable to counter how many endangered animals the human caught.

public class human extends Actor

{

private int speed = 5;

private int fallingSpeed = 0;

private int jumpSpeed = 10;

private int numberOfEndangeredAnimal = 0;

.

.

.

Now we just have to edit the checkAnimal() method and gameOver() method to make this work. Think about it! We are nearly there. After this we are totally done!

Exercise:

1. Edit the checkAnimal() method so that when an endangered animal is caught, the int numberOfEndangeredAnimal goes up by one.
2. Edit the gameOver() method to check if the numberOfEndangeredAnimal is 5 or not. If yes, then tell the game to stop.