LAB 1:

Manav Pavitra Singh
SID: 009328254

Description of classes which I have modified:

GumballMachine Class:

```
import greenfoot.*;
import java.util.List;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
This class is the starting class, in which I have implemented the code in the act
method. The flow of execution starts from here such as removing the coin and then
displaying the message “Have a coin!” on the screen. This satisfies our first
requirement. From this class the flow of execution passes to two classes: one is the
message class by which we display the message “Have a coin!” using a
parameterized constructor and the other, the Inspector class which takes over the
flow of control from this class.
 */
public class GumballMachine extends Actor
{
   private Coin coin;
   public GumballMachine()
   {
      GreenfootImage image = getImage() ;
      image.scale( 350, 400 ) ;
   }

   public void act()
   {
      Actor tempCoin;

tempCoin = getOneObjectAtOffset(0,0,Coin.class);/*Gives the position of  objects of
                                                  Coin class  relative to
                                                  GumballMachine. 0,0
                                                  determines the difference in
                                                  coordinates between the two
                                                  classes*/
      if(tempCoin != null)
      {
         World world;
         world = getWorld();
         world.removeObject(coin);                    //Removes the coin
         world.addObject(new Message("Have a coin!"), 495,68);
```

```
        coin = (Coin) tempCoin;        /*Here I have type casted the object tempCoin of
Actor class into a class variable in order to satisfy the not null condition so that
when I use the mousePressed method it doesn't take coin as null. This is done
because as we know the Act method keeps on running like a continuous thread. So,
by the time I press the mouse coin is 'null' again. */
      }
      /* So, when the mouse button is pressed on the greenfoot machine (use of this),
the list of all of the objects of Inspector class comes into the list. Then the object at
the '0th' index of list is accessed and referred by a variable 'i'. This creates the
reference to Inspector class and the flow of execution is now transferred to
Inspector class */

      if(Greenfoot.mousePressed(this) && coin != null)
      {
        List<Inspector> inspectorList = getIntersectingObjects(Inspector.class);
        Inspector i= inspectorList.get(0);
        i.isValidCoin(coin);
        coin = null;
      }
   }
}
```

Message Class:

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/*
I have created this class as a subclass of the Actor class. This class contains a
parameterized constructor in which a string is passed and it creates an image on the
screen with a message displaying on it. This class is used for displaying required
messages. It is used by the Gumball Machine class to display the message "Have a
coin!" and it is used by the Inspector class to display the message "Fake quarter" and
"It's a penny" in case of fake quarter and penny respectively. This satisfies our
second condition
 */
public class Message extends Actor
{
   public Message(String text)
   {
      GreenfootImage image = getImage() ;
      image.scale( 120, 120 ) ;
      image.drawString(text, 24, 56);
   }
   /*
    * Act - do whatever the Message wants to do. This method is called whenever
    * the 'Act' or 'Run' button gets pressed in the environment.
```

```
   */

   public void act()
   {
      // Add your action code here.
   }
}
```

Inspector Class:

```
import greenfoot.*;
import java.util.List;

/*
This class contains a Boolean method isValidCoin(). It is called in the
GumballMachine class. When mouse is pressed over the gumball machine i.e. the
crank is turned, this method determines whether the removed coin or the coin used
was the fake quarter, or a penny. In case of real quarter it simply ejects the gumball
from where the control is transferred to the RandomPicker and GreenPicker. Hence,
we move to our third condition.
 */
public class Inspector extends Alien
{
   /**
    * Act - do whatever the Inspector wants to do. This method is called whenever
    * the 'Act' or 'Run' button gets pressed in the environment.
    */

   public void act()
   {
      // Add your action code here.
   }
/* This method determines the whether the quarter was real or not. It uses nested if
and else to do so. "instanceof operator is used to compare the object with the
class.*/
   public boolean isValidCoin(Actor coin)
   {
         World world;
         world=getWorld();
          if (coin instanceof Quarter)
             {
                if(coin instanceof FakeQuarter)
                {

                world.addObject(new Message("Fake Quarter"), 495, 68);
```

```
            }
            else{
            List<Picker> pickerList=world.getObjects(Picker.class);
            Picker p = pickerList.get(Greenfoot.getRandomNumber(2))
```

/* getRandomNumber is used to return a random value depending upon which the flow of control can then be assigned to either RandomPicker or GreenPicker. Also, here we have used pickBall method. This method is defined in the picker class but it is overridden by the same method in the RandomPicker and the GreenPicker class (child classes of Picker class). So, from here the flow of execution can be transferred to any of the two child classes of Picker class at runtime. */

```
             p.pickBall();
            world.addObject(new Message("Gumball Ejected!"), 495,68);


             }

          }


       else{

          world.addObject(new Message("It's a penny!"), 495,68);

          }
      return true;

  }
}
```

Picker class:

```
import greenfoot.*;


public class Picker extends Alien
{
  /*
```

The only change I made in this class is an empty method pickBall. I just defined it here. It is being overridden by any of the two child classes' method at runtime.

```
   */


  public void act()
  {
```

```
      // Add your action code here.
   }
   public void pickBall()
   {
   }
}
```

RandomPicker class:

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/*
The function of this class is to generate a random color (any of red, blue and green)
gumball. As mentioned earlier, pickBall() method is also called at runtime
depending upon the random number generated in the Inspector class. I have used
the same random number method to eject a random color gumball during the
runtime. I have used the switch statement and created one case per color in order to
do that.
 */
public class RandomPicker extends Picker
{

   /**
    * Act - do whatever the RandomPicker wants to do. This method is called
whenever
    * the 'Act' or 'Run' button gets pressed in the environment.
    */
   public void pickBall()
   {
      int x=Greenfoot.getRandomNumber(3);

     World world=getWorld();
      switch(x){

         case 0:
         {
            Gumball g = new BlueGumball();

            world.addObject(g, 367, 461);

         }
         break;
         case 1:
         {
```

```
            Gumball g = new RedGumball();

            world.addObject(g, 367, 461);

          }
          break;
          case 2:
          {
             Gumball g = new GreenGumball();

             world.addObject(g, 367, 461);

          }
          break;
          default:
          {}
       }
    }


    public void act()
    {
       // Add your action code here.
    }
}
```

GreenPicker class:

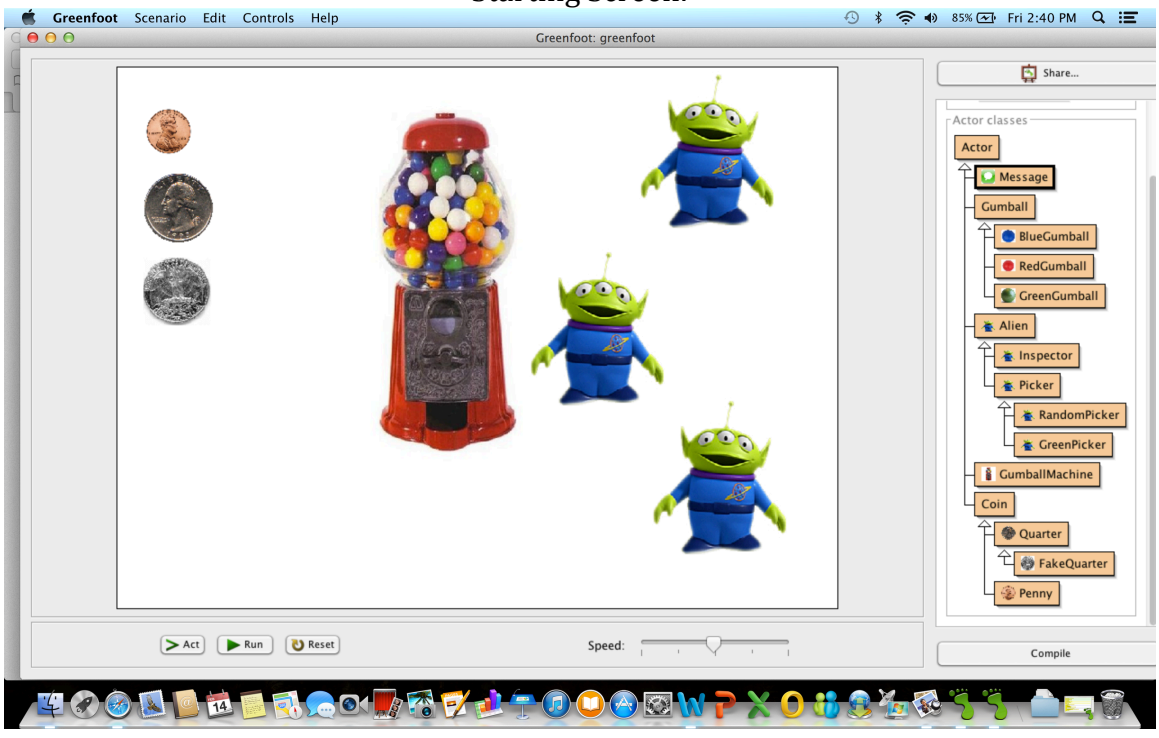
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/*
This class is just used to eject the green ball when its pickBall method is called at
runtime, as discussed earlier.
*/
public class GreenPicker extends Picker
{
    /**
     * Act - do whatever the GreenPicker wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void pickBall()
    {
       Gumball g = new GreenGumball();
       World world = getWorld();
       world.addObject(g, 367, 461);
```

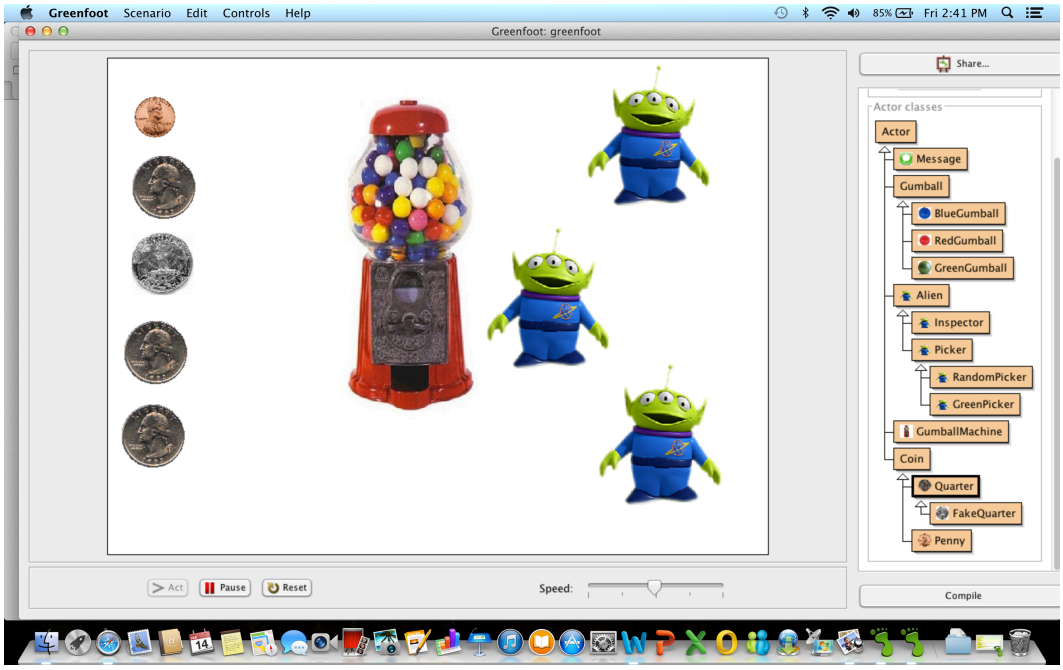
```
    }
    public void act()
    {
        // Add your action code here.
    }
}
```

Screenshots:

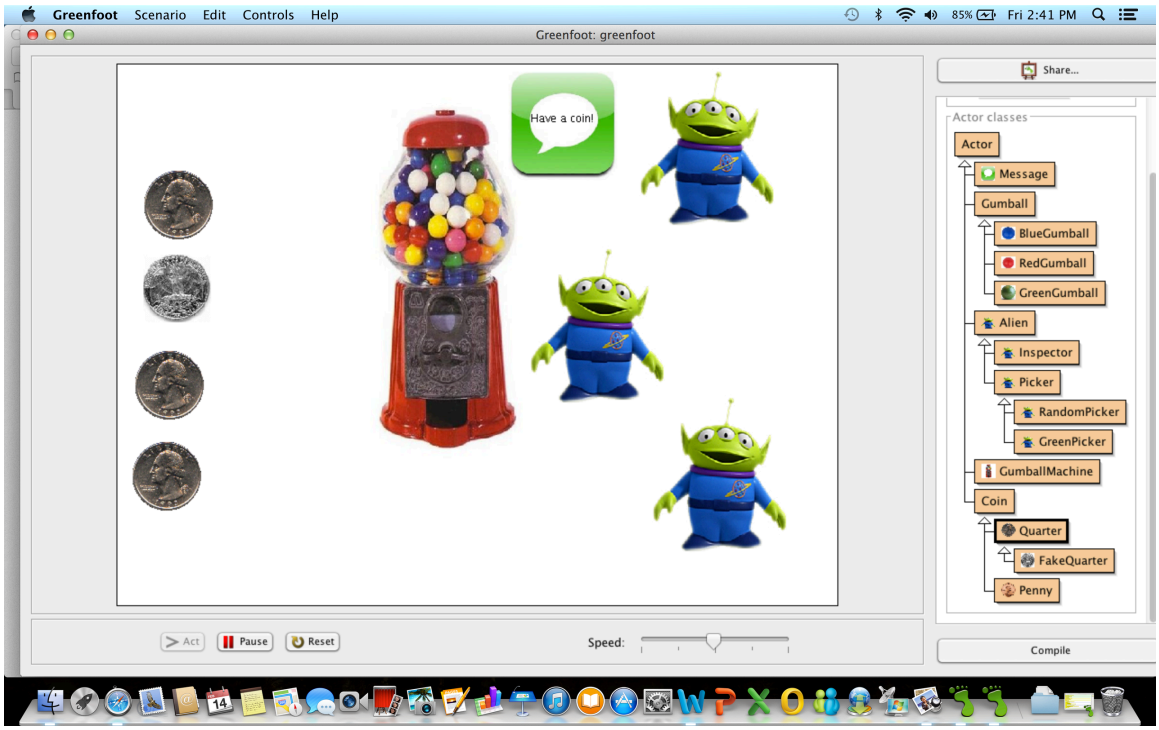
Starting Screen:



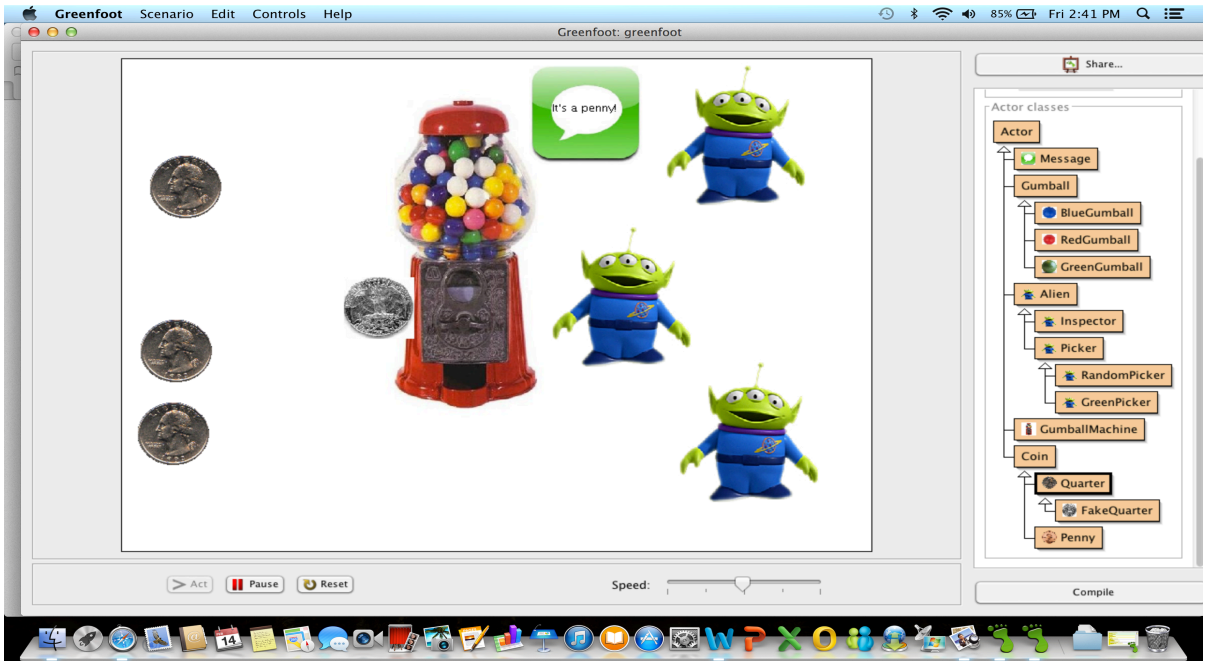
I have added some more quarters, in order to know if it's ejecting gumball randomly:



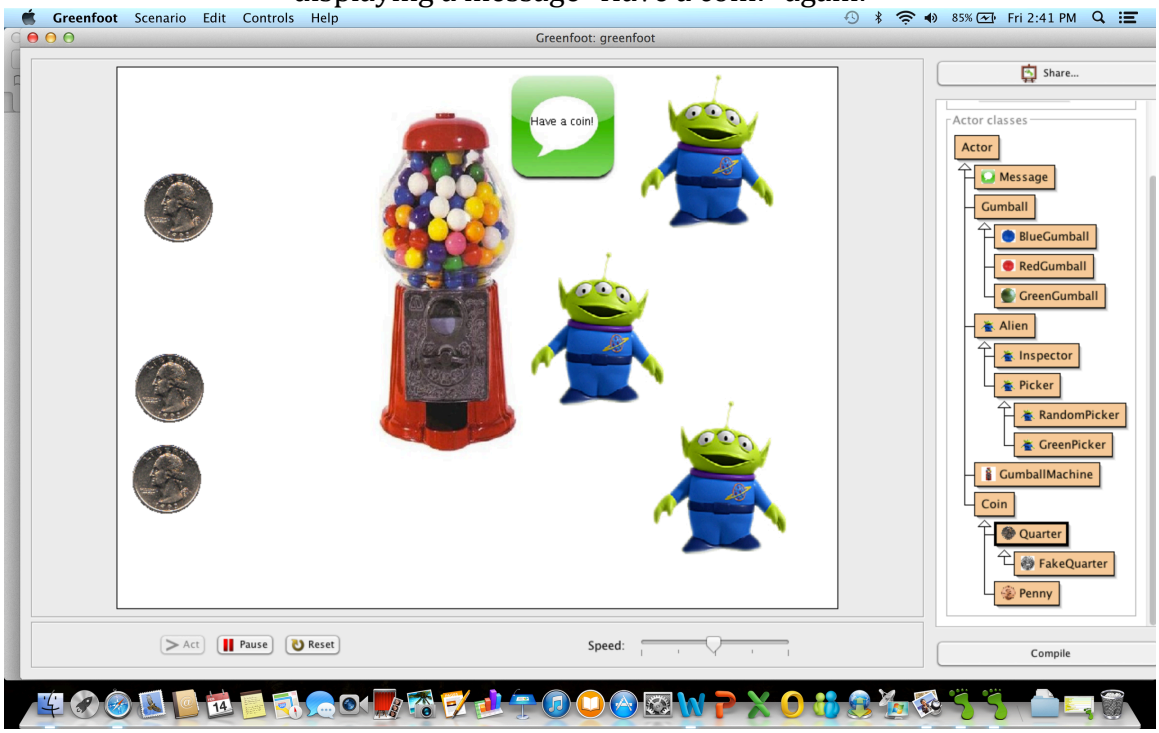
A penny is inserted and Gumball is displaying message “Have a coin!”, first condition is being satisfied here:



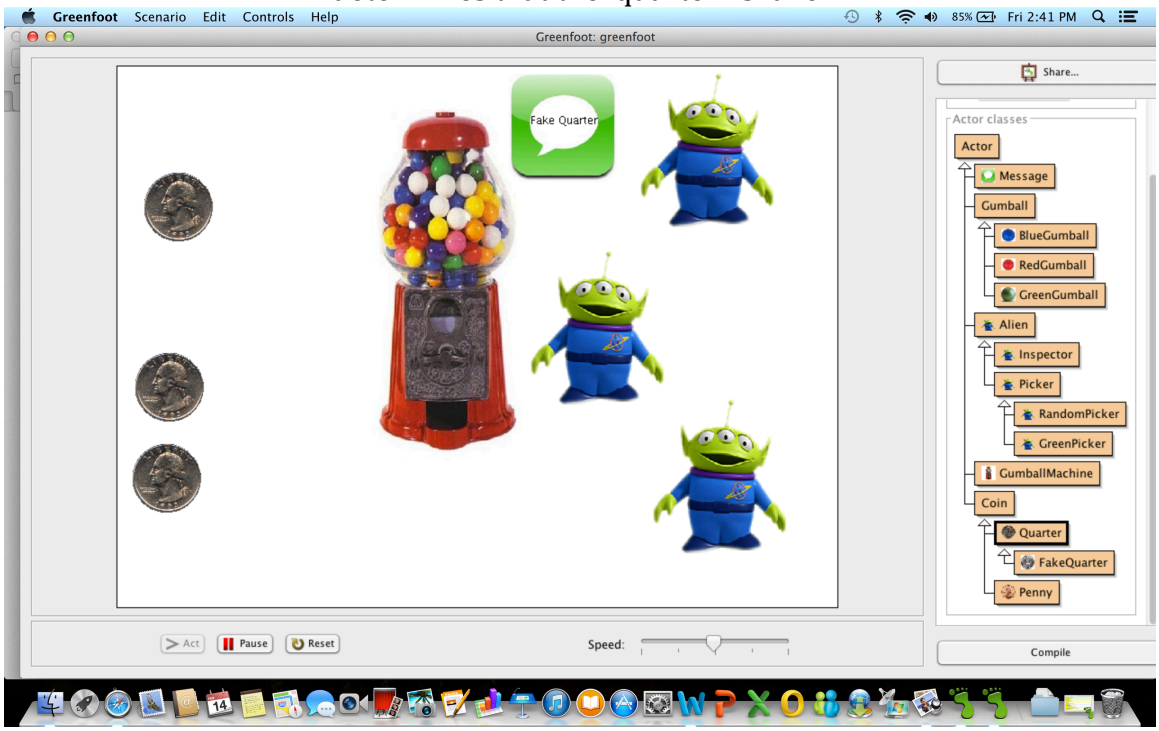
Message being displayed It's a penny and now in next shot Fake quarter is being inserted, Inspector tells it's a penny, satisfying second condition:



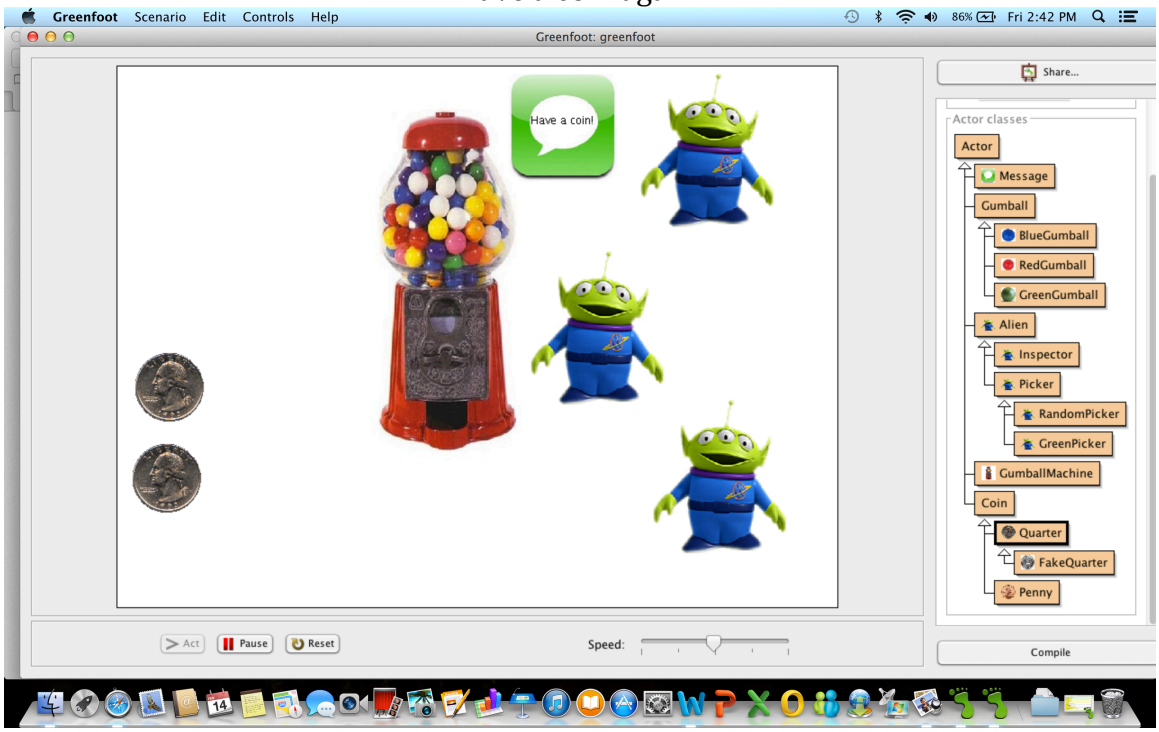
Since I have just inserted the "Fake Quarter" and not turned the crank yet, it's displaying a message "Have a coin!" again:



After turning the crank it's showing the message "Fake Quarter", Inspector determines that the 'quarter' is fake:

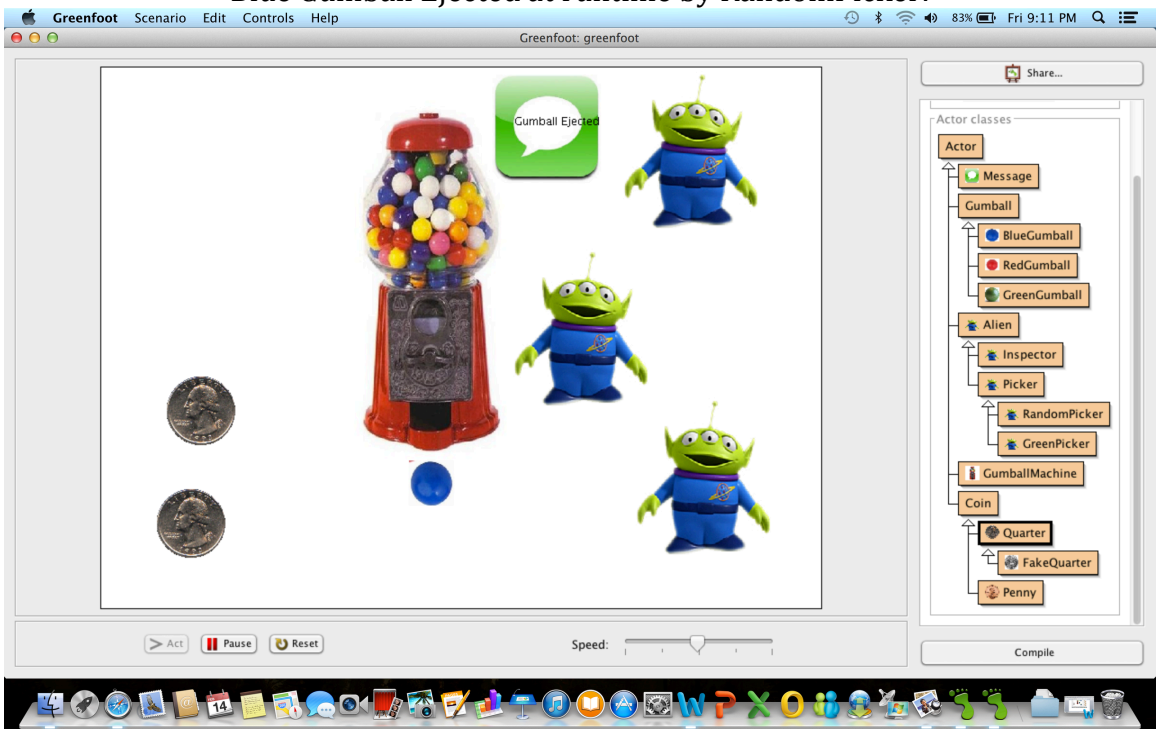


A quarter is being inserted and the gumball machine is displaying the message "Have a coin again":



Gumball ejected message is displayed and a green gumball is ejected randomly, at runtime any of the two pickers have ejected the gumball satisfying the third condition:

Blue Gumball Ejected at runtime by RandomPicker:



Red Gumball Ejected by RandomPicker: