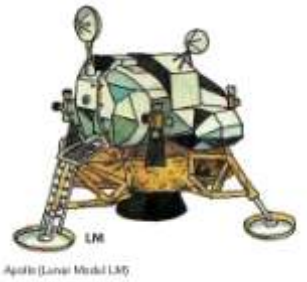
Extreme Environments: Building a Lunar Lander Simulator

Project Overview



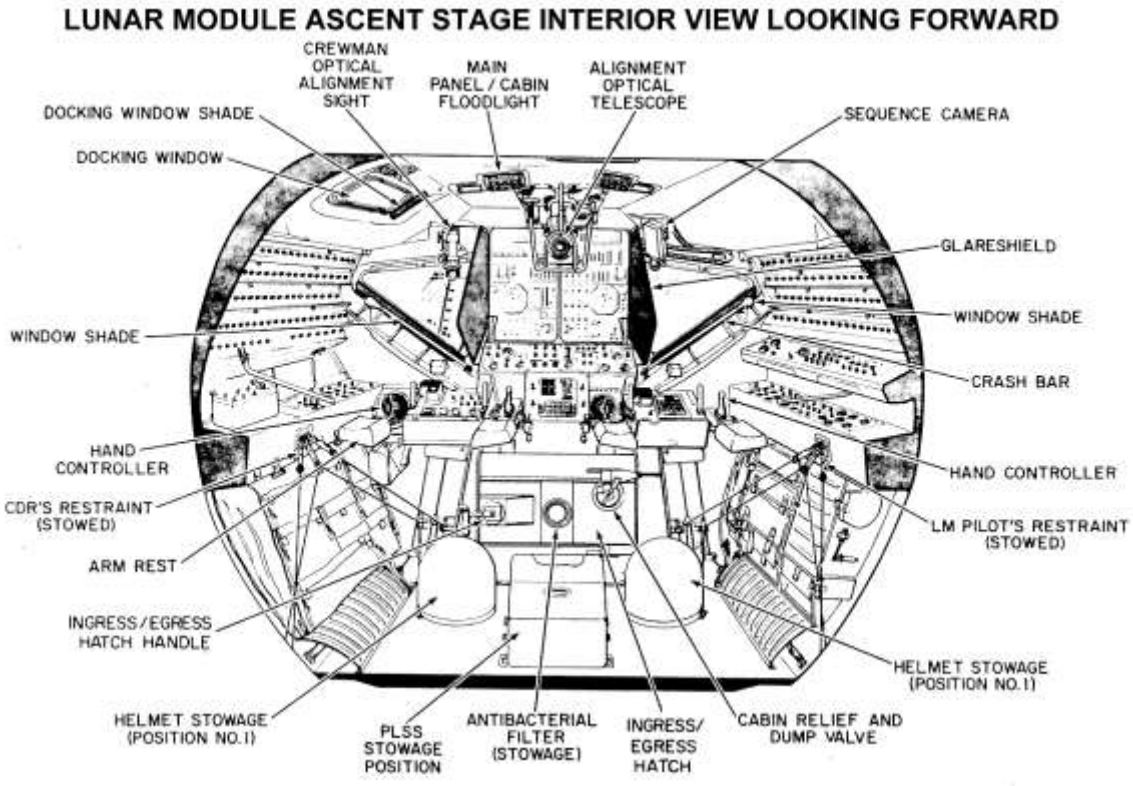
Apollo (Lunar Model LM)

For this program we are going to build a lunar lander to explore the extreme environment of the moon. In 1969, the very first lunar lander took man to the moon and back to his spaceship for the very first time. There were several problems scientist had to overcome when building this first lunar lander.

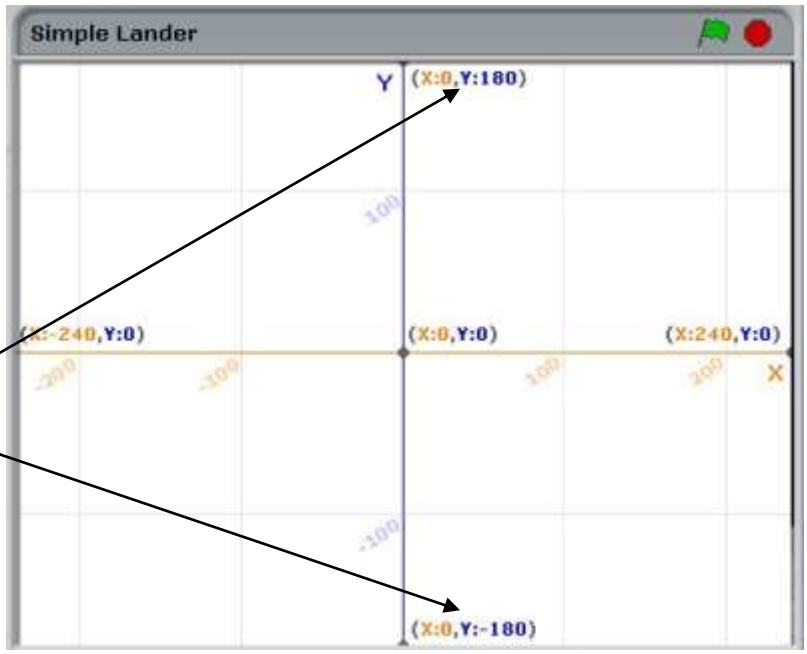
First, it had to be lightweight yet strong enough to survive its landing on the moon. It had to carry sufficient fuel so that it could land without crashing and return to the mother ship. The moon has no air but it does have gravity that pulled the lunar lander to the surface. When the lander was landing it could travel side to side with nothing able to stop it except for the thrusts from a rocket. Yet the lunar lander need to have enough fuel to land at a slow enough speed not to crash into the moon.

For this program you will need to design your lander to control for it side to side motion as well as, it's speed of descent. On the moon an object that is traveling side to side will continue to do so unless rockets are fired to stop it. This is known as **inertia**, a law of physics that says something set in motion keeps moving that direction unless something else stops it. On the earth air eventually slows all objects, but on the moon there is no air to slow something down.

You will also need to control a bottom rocket on your lander that will slow your descent and stop you from crashing into the moon at a speed that will destroy your lander.



Technical Challenges

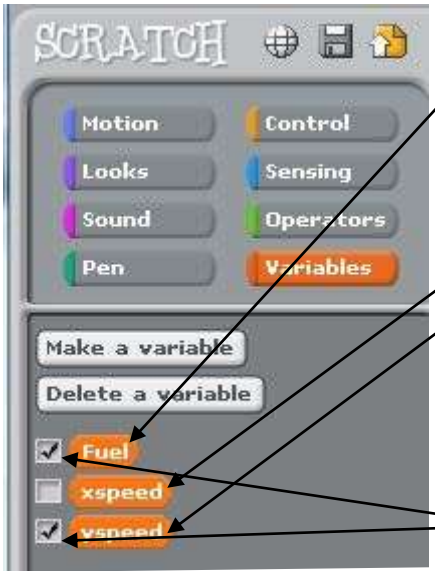


When scientists and engineers were designing the first real lunar lander there were several technical challenges they need to overcome. We must overcome the same challenges in building our lunar lander. When constructing software in SCRATCH remember that all objects on the stage move in relation to a series of **X and Y coordinates**. Objects going up the screen towards the top are going to increase in **Y value**. ***The thrust of a rockets will increase the value of Y of our lander*** on the screen. ***Gravity, is pulling our lander to a negative (or minus) Y value to the bottom of the screen.*** Remember that when we designed our gravity system in SCRATCH, objects speed up as a fall back to earth from the top of the screen to the bottom of the screen. Thrust must be used by our lander to slow down the pull of gravity so that our lander does not crash before running out of fuel.

Also, our lander must be able to move side to side in order to land it on the landing pad. Remember that inertia on the moon says that if an object is moving from one side to another you must apply thrust in the opposite direction in order to slow it down. For your program to simulate inertia you must use **recursion** (looping) in your software so that your program will make decisions about the pull of gravity and how the force of inertia inertia will act on your lander regardless of what the person piloting your lander does. In other words, the **X value** of your lander will continue to increase or decrease in value once thrust has been applied in one direction or the other.

For this program your job will be to create a lander that takes off on one landing pad and is able to land softly on another. With that let's get started on writing our program.

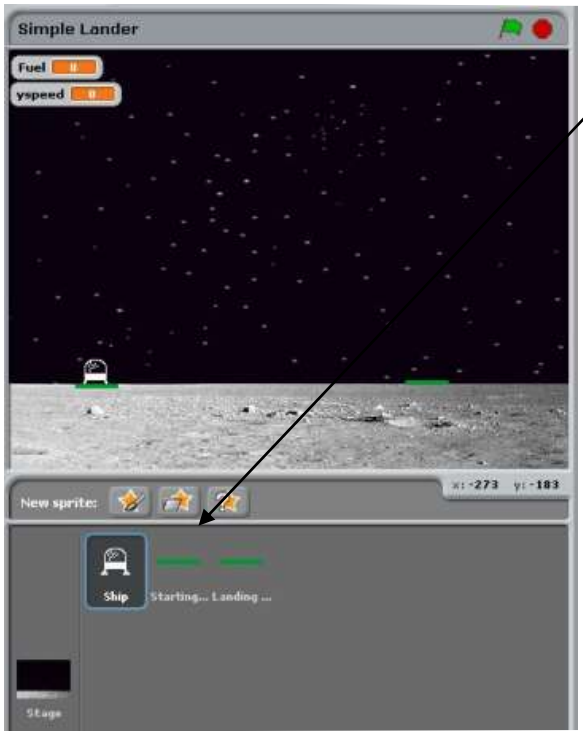
Setting Up Variables



To start your program off we are going to create three variables. *The first variable we will call* ***fuel***. This variable will be used to calculate the amount of fuel remaining in your lander that can be used to apply rockets with. *The next variable we will call* ***X speed***. The Xspeed variable will be used to calculate how fast the lander is traveling in a left or right direction on the screen. Rockets on each side of the lander can be used to increase the left or right speed of the lander BUT the opposite rockets must be applied when piloting as there is nothing to stop the lander from continuing left or right on the moon (**inertia**). *The third variable we will create is called* ***Yspeed***. Yspeed is used to calculate the speed the lander is traveling up or down in relation to the moon. If rockets are applied in the value of Yspeed will increase as the lander pushes up from the moon. If rockets are stopped the value of Y speed will decrease as gravity pulls the lander back to the moon. Rockets must be applied so that the lander falls at a slow enough speed through the pull of gravity to land on the pad but not so fast that breaks the lander. I will also *check* the fuel and Y speed variables so that they can be viewed on

the screen. This will allow me to see how much fuel I'm using in my landing attempt and the speed of my lander as it goes up and comes down (descends).

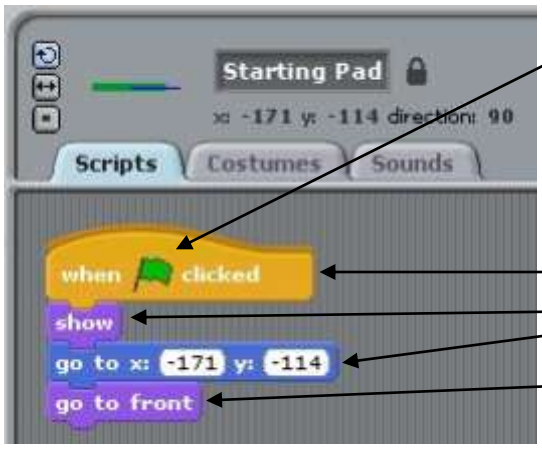
Setting Up Sprites and Costumes

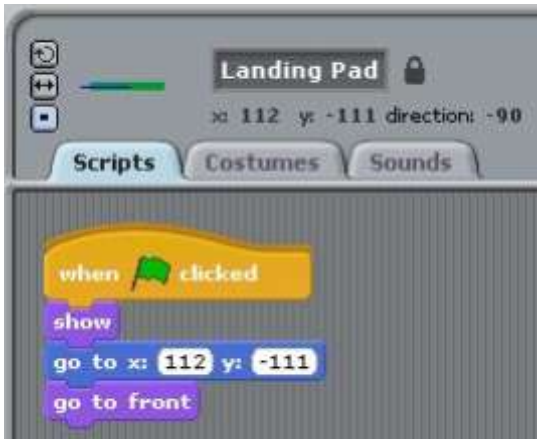


We will now set up our initial sprites and provide costumes to those sprites. *I will start by creating three sprites.* The first will be my lander and I will name it ***Ship.*** The next two I will name ***Starting Pad*** and ***Landing Pad*** and will make these objects my lander can sit on.

For the next step, I will provide a **Costume** to my background. This can be any costume that covers the background that you like. For mine I have chosen a ***Moonscape*** although yours does not have to be this. *Next, I will provide my* ***lander*** *with three different costumes*. The first one I will call ***noThrust***, the second one ***Landed***, and the third one ***Crashed***. While what these look like is up to you, the names are significant within the program. While you may change the names of the costumes, you will also need to adjust your program accordingly.

Writing the Starting & Landing Scripts





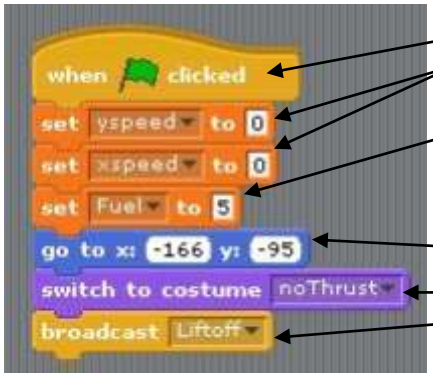
For my program I have chosen to use the **green flag** to start each attempt with the lander. As such, each script will reset when the green flag is clicked.

I start by writing my script for the Starting and Landing pads which are very similar. When the ***green flag is clicked*** I have both my pads make sure they are ***showing*** themselves on the screen, they ***set the location they should be (X & Y Position***) at and move themselves ***to the front***. By moving themselves to the front this ensures they can be "sensed"by a blue sensing block from the lander (later when we write this script).

Remember that to set an initial location of an object, move the object on the screen in the place you want it, then double-click on that object. You can then drag a motion block out and location it was in when you double-click it will be set on that block.

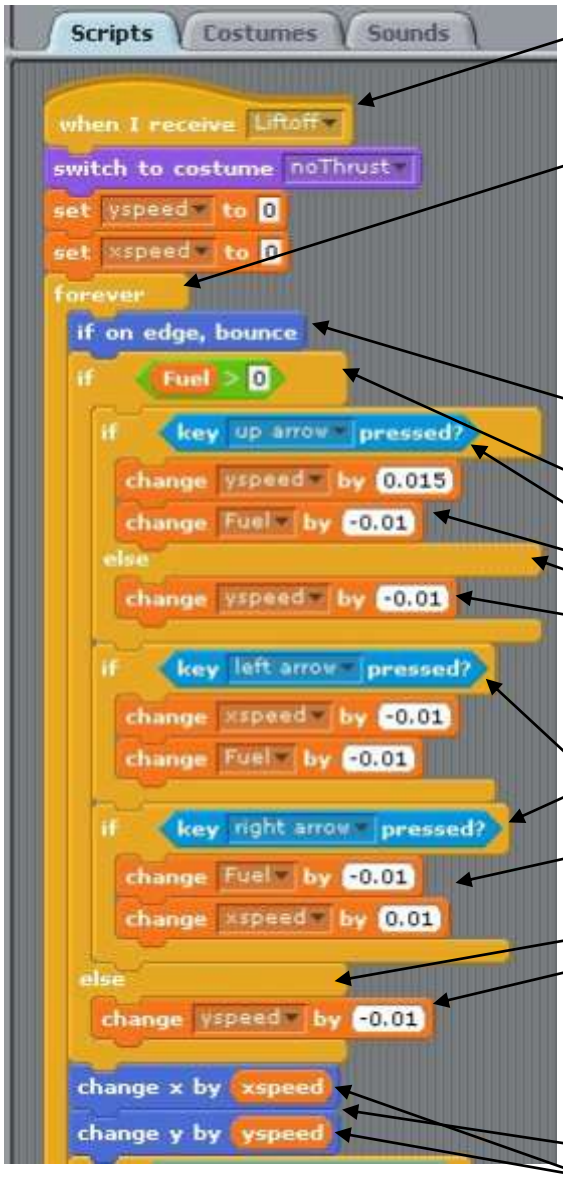
Writing the Lander Set-up Script

Most of the work in this program is done by the script attached to the lander. In the sprites window select the ***Lander sprite*** and then click on the *scripts tab*. The first script we are going to write is to set the lander up when the green flag is first clicked.



When the green flag is clicked we want to set the **Yspeed and Xspeed variables** values to zero. This is because our lander is not yet moving. We want to set our **fuel variable to 5** to give our lander its initial amount of fuel. We will subtract from this value later on the program as the rockets are used. We'll use a motion block to set our initial lander location on top of the start pad **and switch our costume to noThrust** (the initial costume). Then broadcast a message called **liftoff** that tells our program it is ready to start.

Writing the Lander Rockets & Gravity Scripts



The **liftoff message** sent by clicking the green flag is received by a script that controls the lander's thrust gravity and ability to crash. The important part of this script is the control block called **forever**. This is the ***recursion (looping)*** part of the script that checks conditions to see if thrust (rockets) are being applied, the lander is falling back to earth with gravity or whether it has crashed. **After the forever** block the loop starts by checking to see if the lander is touching the edge of the screen. If the lander is on the **edge of the screen it bounces it off the edge** and back into the sky. The script then checks to see if the lander's **fuel variable is greater than zero**. If it is, it then checks to see if the up arrow key is pressed. If this is true it increases the lander's Yspeed variable slightly and reduces its fuel variable slightly thus, making the lander travel up with ever-increasing speed as long as it has fuel. If the up arrow key is not pressed **(else),** it decreases its speed slightly each time through the loop. The same thing is done with **right and left arrow keys** if these are pressed thrust is applied and **the Xspeed increases or decreases** so the lander can move side to side and fuel is also decreased. A condition is also set so that if fuel is zero **(else)**, Yspeed is decreased causing the lander to always fall back to the moon once the lander's Yspeed variable goes below 0.

At the bottom of this if statement is a **motion block that changes X and Y values** of the lander **for the amount of Xspeed and Yspeed variables** that was set in the if statements above***. This is the part of the program that actually moves the lander on the screen. The important part here is the Yspeed variable.*** If the Yspeed variable is a number above zero then the lander will travel up. If the Yspeed variable is a number below zero (negative value) then the lander is being pulled back to the moon by gravity at an ever-increasing speed (-0.01 each time through the loop). By controlling the speed at which the lander fall to the moon, allows you to landed safely on the moon. ***What you need to do then is have a controlled crash.***

The next part of the loop determines if the lander is taking off, has landed safely, or has crashed. The **if touching starting pad sensing block** sets the initial conditions for the lander if it is on the starting pad. The next if statement determines if the lander has landed on the landing pad safely. **If** the lander is **touching the Landing Pad** **and** **if** it's **Yspeed variable** when it hit the **pad is not less than -.7** it will **switch to the costume called Landed** and **say you won** **stopping all scripts**. In this case -.7 is the maximum speed the lander is allowed to fall to the pad without being destroyed.

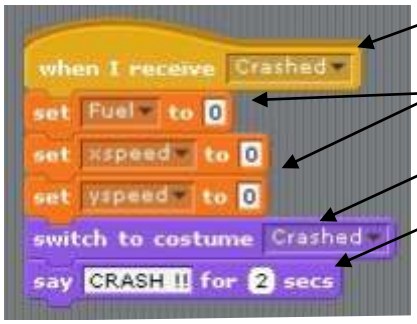
The next if touching landing pad sensing block statement checks to see if the lander has crashed. If the lander's Yspeed variable **is less than -.7** (meaning the lander is traveling too fast when it hit the pad) it creates a **broadcast called crashed** and **stops all the scripts**.

The next if statement checks to see if the **Yposition of the lander is less than -96**. This means the lander did not hit either of the pads and has fallen below its starting position on the moon. When this happens a broadcast of crashed is also sent and all scripts are stopped.

As with all types of recursion this is the bottom of the loop and the lander will continually check through this loop to see which conditions are true and run the pieces of the program that are true at any given point in time. This is how recursion works and allows the program to make decisions and have things happen (e.g.: having the lander fall to earth) even when a person is not controlling (piloting) the lander.

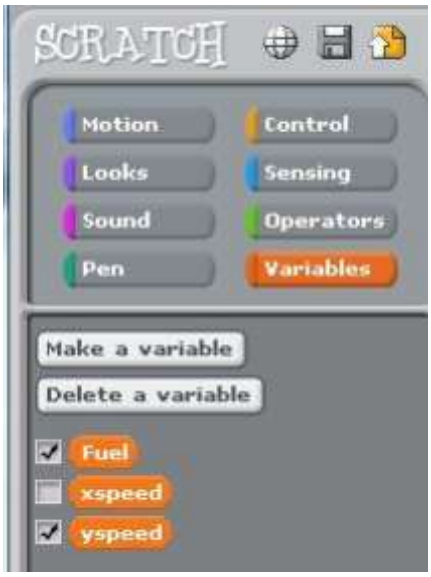
Writing the Crashing Script

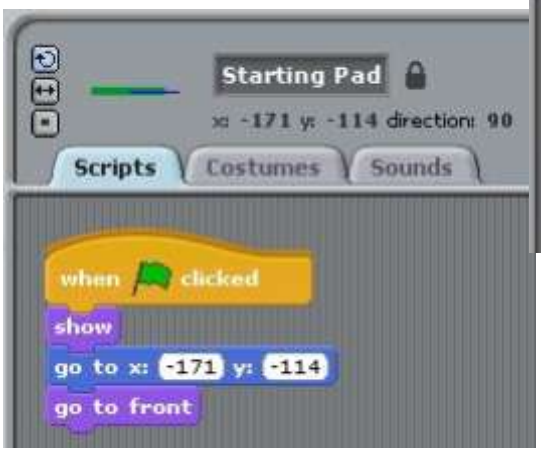
The last script attached to the lander is the crashed script. When the recursive lander script sends out a crashed message either because the lander fell too quickly onto the landing pad or hit the ground this script receives the crashed message. It starts by setting the Fuel, Xspeed and Yspeed variables to zero as the lander has now crashed. It then switches the lander's costume to a crashed costume and says crashed for 2 seconds on the screen ending the simulation.

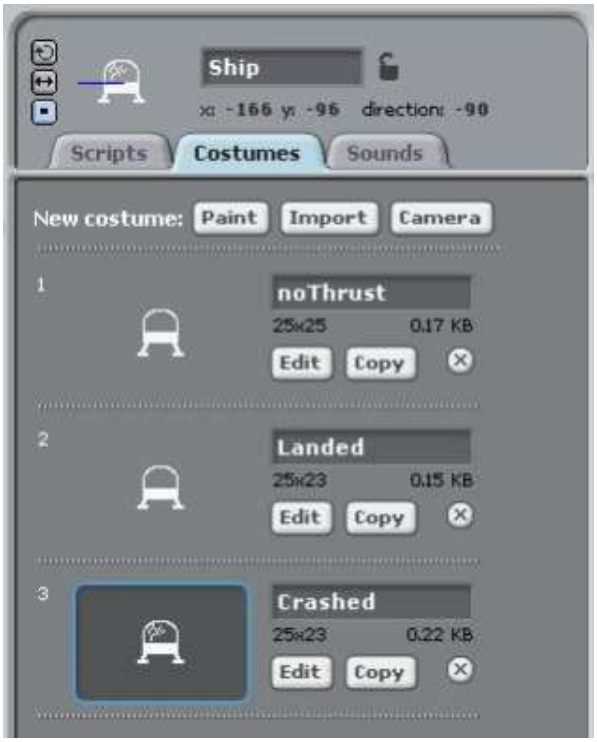


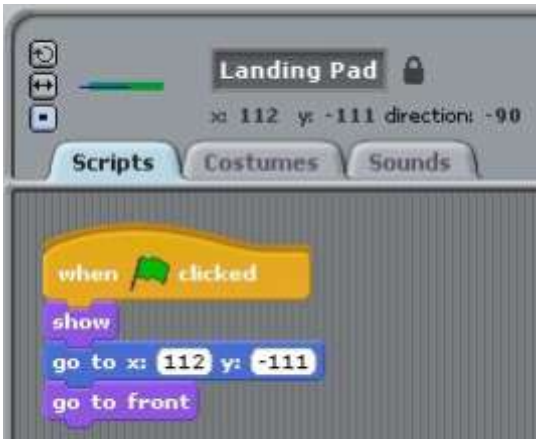
To restart the lander simply click on the green flag at the top and all the conditions will be set so the lander can launch again.

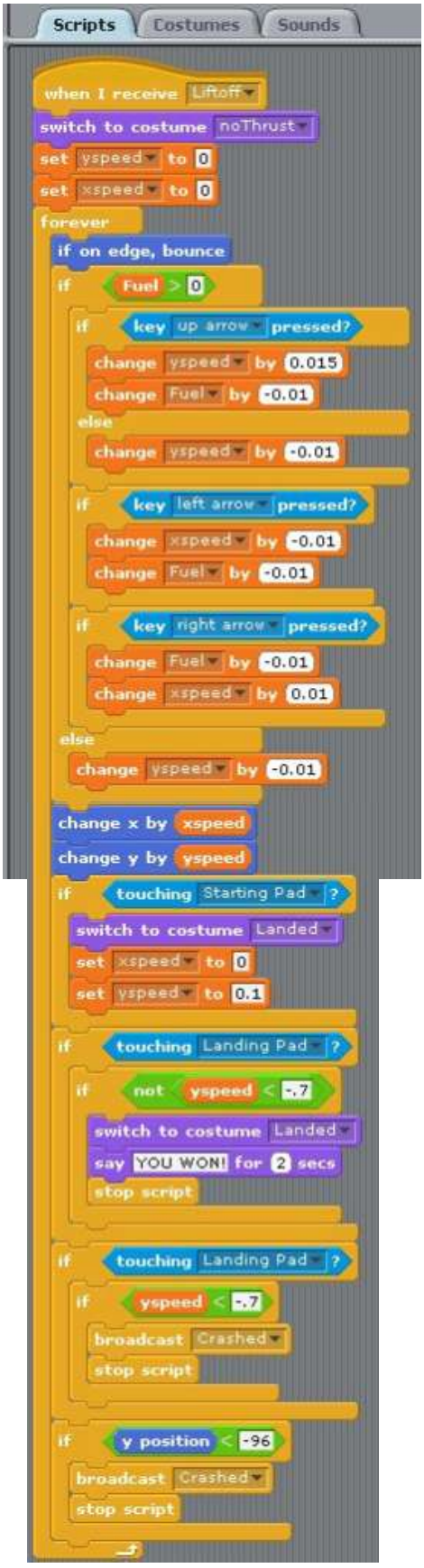
Complete Script Listing

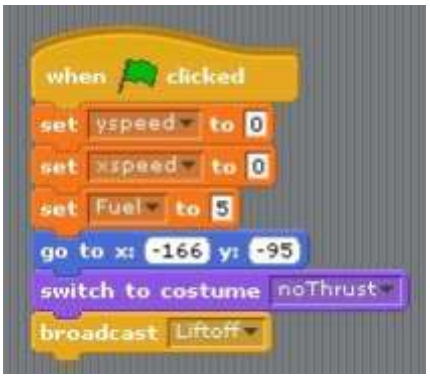


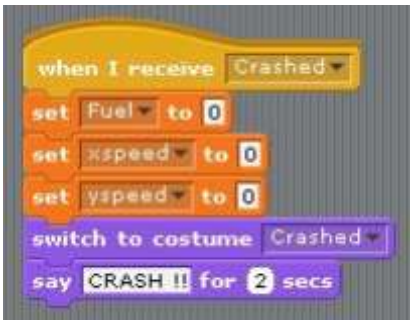












Complete Lander Script Listing