Create an Infrared Controlled Mouse with Arduino Micro, Leonardo, or Yun

Created by Chris Young

Last updated on 2018-08-22 03:46:58 PM UTC

Guide Contents

Overview

Introduction

In this tutorial we're going to use the mouse and keyboard emulation abilities of a variety of Arduino devices combined within infrared receiver to make a remote control mouse with some keyboard commands thrown in for good measure.

There are a variety of reasons why you might want to create an infrared remote control mouse or keyboard. Suppose you're making a PowerPoint presentation and you don't want to be stuck standing next to the laptop as you click through the slides. Or perhaps you've set up your Raspberry Pi as a media server or you have some sort of Home Theater PC connected to your TV.

However for me the ability to do mouse and keyboard remotely is much more personal. As a result of my disability, I cannot use a traditional mouse. I control my computer using dictation software. However the mouse commands for that software have limits or are somewhat inconvenient to use. I can however use a traditional TV remote by poking at the buttons with a stick in my mouth. I originally developed this project because it helps me overcome my disability and allows me to do productive things like write tutorials for the Adafruit Learning System. If you're interested, here is a link to my personal blog about how this project was originally developed (https://adafru.it/eUy). I use it every day and I could not have written this tutorial without it.

Hardware Needed

We originally intended this tutorial to have support for Pro Trinket as well as any ATmega32u4 based boards such as the Arduino Leonardo, Arduino Micro, and Arduino Yun. However there appeared to be some conflicts between IRLib infrared library and the Pro Trinket mouse and keyboard libraries. If we can resolve those problems at a later date we will update this tutorial. For now we are sticking to the ATmega32u4 based boards. We will be using Arduino Micro however the code will work identically with an Arduino Leonardo and the Leonardo portion of the Arduino Yun.

You will need an IR sensor such as the TSOP38238 shown at the right.

You will need some sort of infrared remote control. All of our example code has been written for the Adafruit Mini Remote Control shown on the right but you can modify the code to use any ordinary TV remote if it uses a protocol supported by IRLib.

Although it is not 100% necessary, we will also add 2 NeoPixel's as indicator lights that will help you know whether or not you are in mouse mode or keyboard mode and if you have pressed the Control, Alt, or Shift buttons. The NeoPixel also indicates whether or not you are dragging the mouse or clicking it.

Software Libraries

The mouse and keyboard libraries for the Leonardo, Micro, and Yun are already installed as part of the standard Arduino IDE. You can find more about these libraries at http://arduino.cc/en/Reference/MouseKeyboard (https://adafru.it/ejW)

You also need the IRLib2 Infrared Library for Arduino available on GitHub. (https://adafru.it/vxb). For an introduction to using this library, see our previous tutorial Using an Infrared Library for Arduino (https://adafru.it/vxc).

NOTE: These tutorials have been updated for use with IRLib 2.x which is not backwards compatible with the original IRLib 1.x.

If you're using NeoPixels as indicator lights you will need down eight the Adafruit NeoPixel Library from GitHub (https://adafru.it/aZU).

Both of these libraries can be downloaded using the buttons below.

https://adafru.it/vxa

https://adafru.it/vxa

https://adafru.it/cDj

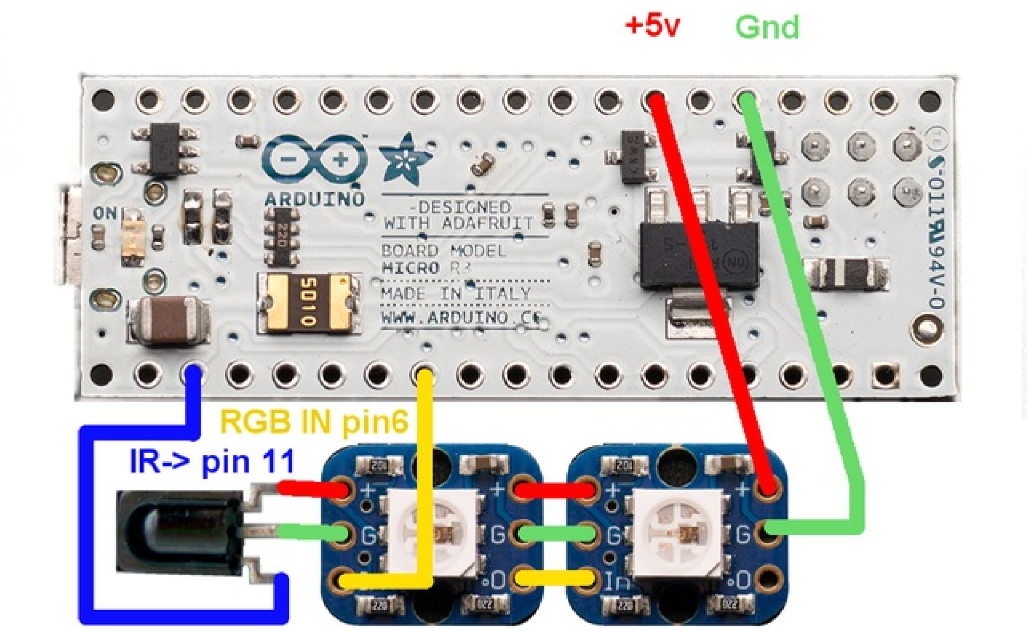
https://adafru.it/cDj

See the tutorial All About Arduino Libraries (https://adafru.it/dNR) on installing the code into your Arduino Integrated Development Environment (IDE) for your use.

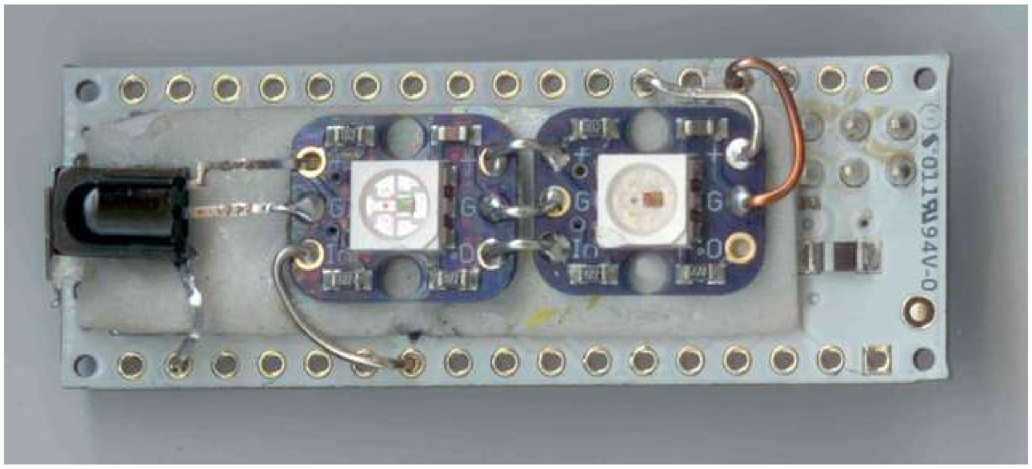
Assembling the Hardware

Wiring the Arduino Micro

Here is a schematic of the Arduino Micro showing the IR receiver connected to pin 11. We also have 2 NeoPixels wired together wired to pin 6. Be sure to connect to the input side of the first NeoPixel and connect its output to the input of the second NeoPixel. All three devices have +5v and Gnd lines connected to them. The image shows the underneath side of the Arduino Micro because in this instance, it will be easier to wire everything up from the underneath side.

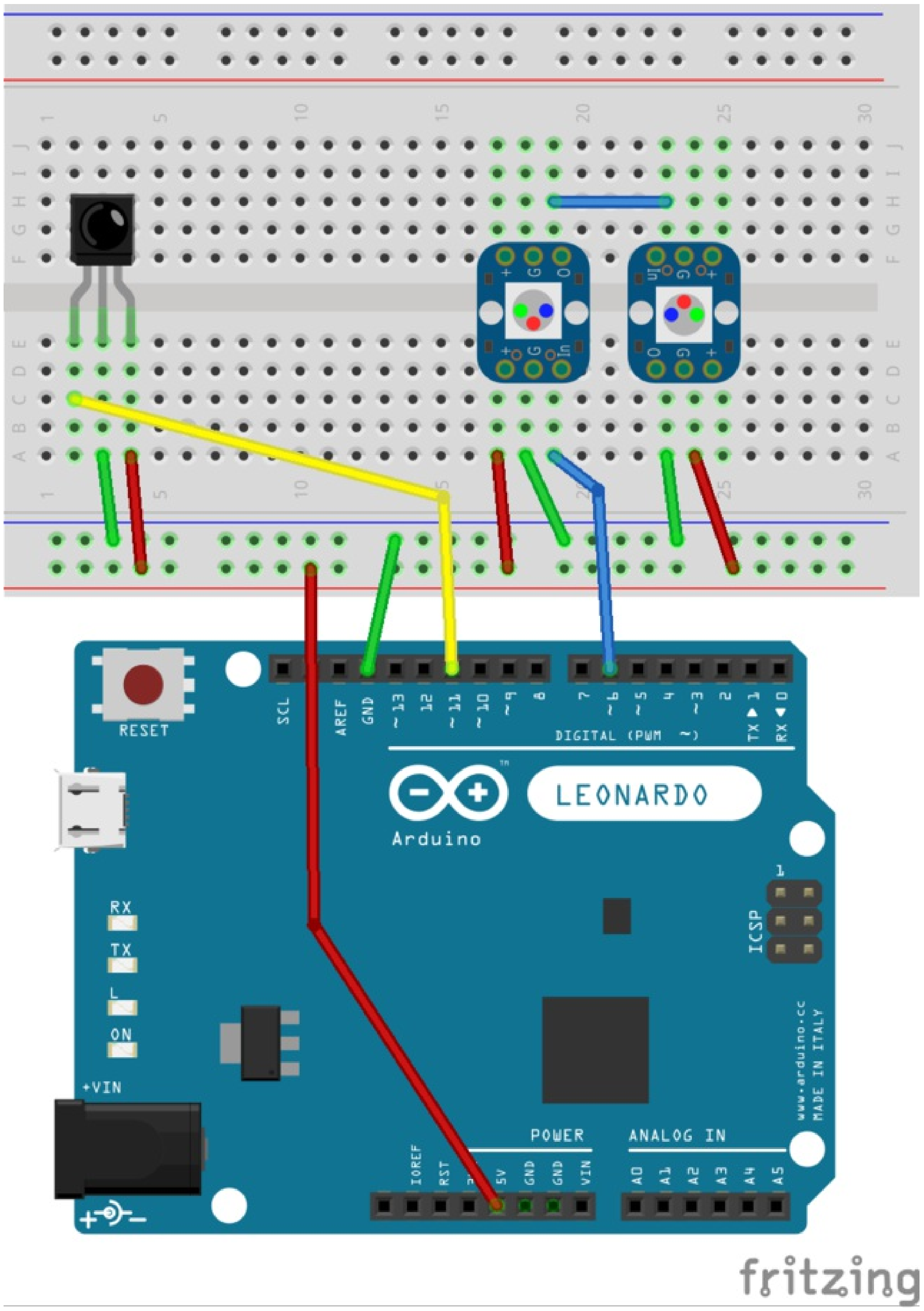


Here is a photo of the unit assembled. We placed a piece of double stick tape on the back of the Arduino Micro and placed the parts on the tape. The wires are solid uninsulated wires so you have to be very careful that they do not short out one another or to the Arduino Micro itself. Ultimately you might want to wrap electrical tape around the entire package leaving holes for the IR receiver and the NeoPixels. Or perhaps you might encase the entire assembly in sugru putty.



Wiring Leonardo or Yun

Because the Leonardo and Yun also use the ATmega32u4 processor, they will work identically to the Arduino Micro. You would wire the receiver to pin 11. The NeoPixel signal wire to pin 6. Also run power and ground to all the devices. Although the Leonardo is shown below, the Yun pinouts are the same as the Leonardo.

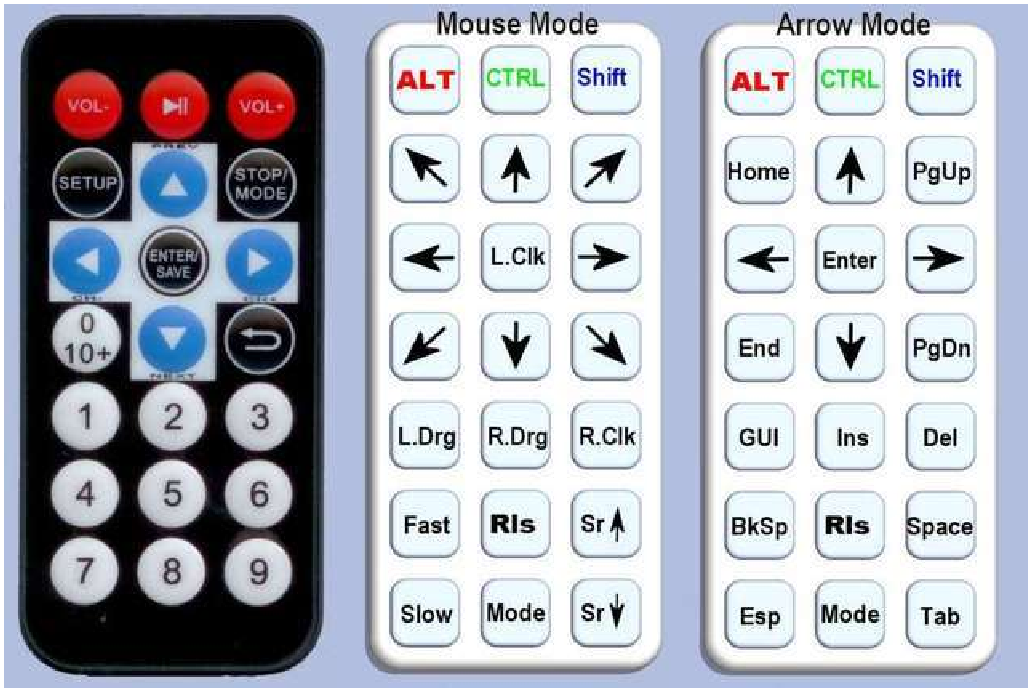


Configuring the Device

If all we need to do is move the mouse around and click it using the left click button then you really only need to assign five buttons on your remote to the up, down, left, and right mouse movement functions and the click function. Most remotes have a series of four arrow keys with a select button in the center that can be used for clicking. We however are going to go above and beyond and assign as many functions as we can to the Adafruit Mini Remote Control. We are also going to define two different modes of use: one of them for mouse commands and the other for a limited set of keyboard commands mostly arrow keys. The full source code can be found in the next page of this tutorial.

Using the Adafruit Mini Remote Control

Here is an illustration of the commands that we are assigning to the 21 buttons available on our remote. Some of the commands are duplicated in both mouse mode and keyboard mode. You switch modes by pressing the "8" button which is at the lower center of the remote.



If you are using a different remote, you will need to come up with your own layout assigning buttons to functions. You may not want to assign all the buttons we have assigned or you may come up with additional ones that you would like to use.

Below is a section of the code we will use. It defines the 32-bit hex values that are transmitted by the various buttons on the remote. A complete source code listing is given on the last page of the tutorial.

```
//Codes used by both mouse and arrow modes
#define CodeAlt   0xfd00ff //Vol-
#define CodeCtrl 0xfd807f //Play/Pause
#define CodeShift 0xfd40bf //Vol+
#define CodeRls   0xfda857 //5
#define CodeMode 0xfd9867 //8

//Used by mouse and arrow but have different meanings
#define CodeUp 0xfda05f //Up arrow
#define CodeDown 0xfdb04f //Down arrow
#define CodeLeft 0xfd10ef //Left arrow
#define CodeRight 0xfd50af //Right arrow

//Used only in mouse mode
#define CodeUpLf 0xfd20df //Set up
#define CodeUpRt 0xfd609f //Stop/mode
#define CodeLfClk 0xfd906f //Enter/save
#define CodeDnLf 0xfd30cf //0 10+
#define CodeDnRt 0xfd708f //Repeat
#define CodeLfDrag 0xfd08f7 //1
#define CodeRtDrag 0xfd8877 //2
#define CodeRtClk 0xfd48b7 //3
#define CodeFaster  0xfd28d7 //4
#define CodeScrUp 0xfd6897 //6
#define CodeSlower 0xfd18e7 //7
#define CodeScrDn 0xfd58a7 //9

//Arrow mode only
#define CodeHome 0xfd20df //Set up
#define CodePgUp 0xfd609f //Stop/mode
#define CodeEnter 0xfd906f //Enter/save
#define CodeEnd 0xfd30cf //0 10+
#define CodePgDn 0xfd708f //Repeat
#define CodeGUI 0xfd08f7 //1
#define CodeInsert 0xfd8877 //2
#define CodeDelete 0xfd48b7 //3
#define CodeBkSp 0xfd28d7 //4
#define CodeSpace 0xfd6897 //6
#define CodeEsc 0xfd18e7 //7
#define CodeTab 0xfd58a7 //9
```

Five of the buttons are available in both mouse and keyboard modes. The three buttons across the top are toggle keys for the Alt, Control, and Shift keys. When you press them once, it holds that modifier key until you press again. You can also press "5" marked "Rls" for "Release". It releases all of the toggle keys as well as releasing any held down mouse buttons for dragging. In addition to the three modifier keys and the release button, the mode switch button "8" is also available in both modes.

Because the keys are toggle keys, I found it easy to forget whether I left the shift key held or not. That's why we need the NeoPixels. The first NeoPixel tells you which mode you are in. If the pixel is off you are in mouse mode. If it is green you are in keyboard mode. If it is red you are holding or dragging the left mouse button. If it is blue you are holding or dragging the right mouse button.

The second NeoPixel indicates key modifier status. Red means Alt is being held. Green means Control is being held. Blue means Shift is being held. If you are familiar with how RGB colors are mixed you can also keep track of multiple conditions. For example if the second pixel is cyan (a mixture of green and blue) you know that both control and shift are being pressed.

The arrow keys are used by both mouse and keyboard. Their function is obvious. In mouse mode we can also move the mouse diagonally using the "Set Up", "Stop/Mode", "Enter/Save", and "0 10+" buttons. In keyboard mode these correspond to Home, End, Page Up and Page Down functions. There are special buttons to change the speed of the mouse. You can scroll the mouse wheel up and down. You can click using left click, right-click or you can drag using left drag or right drag.

Determining Your Own Codes

If you're using a different remote then we are, you will have to determine the codes for your particular remote. In the IRLib "examples" folder is a sketch called "IRrecvDump". You can load that sketch, point your remote at the receiver, and get a dump of the codes zero serial monitor.

If I point my Sony DVD remote at the receiver and press the up arrow button I will get a dump of the raw data. I really only need to be concerned with the top line which reads:

`Decoded Sony(2): Value:42BCA (20 bits)`

This tells me that I'm using "Sony" protocol. It is the 20 bit variety. The code received was the hex value 0x42bca. You will have to repeat this for every button on your remote that you want to use. Edit the values into the table at the beginning of the IRmouse sketch. For example change the:

`#define CodeUp 0xfda05f //Up arrow`

to read

`#define CodeUp 0x42bca //Up arrow`

You also need to change the type of decoder you're using. For example at approximately line 63 change the line:

`IRdecodeNEC myDecoder;`

to read

`IRdecodeSony myDecoder;`

Once you have changed all of the codes to suit your remote, you can upload the IRmouse sketch to your device.

Source Code Listings

Code for Arduino Micro/Leonardo/Yun

Here is the source code for Arduino Micro and other ATmega32u4 based boards.

```
/* Example program for from IRLib 2 – an Arduino library for infrared encoding and decoding
 * Version 2.0  April 2017 by Chris Young http://tech.cyborg5.com
 * "IRmouse" Control a PCs mouse or arrow keys using an IR remote
 */
#include <Mouse.h>
#include <Keyboard.h>
#include <IRLibAll.h>
#include <Adafruit_NeoPixel.h>
//The IR codes shown below are for the Adafruit mini remote
//which uses an easy protocol. The comments after each code refer to the buttons
//on that remote and not to their actual function on this device.

//Codes used by both mouse and arrow modes
#define CodeAlt   0xfd00ff //Vol-
#define CodeCtrl 0xfd807f //Play/Pause
#define CodeShift 0xfd40bf //Vol+
#define CodeRls   0xfda857 //5
#define CodeMode 0xfd9867 //8

//Used by mouse and arrow but have different meanings
#define CodeUp 0xfda05f //Up arrow
#define CodeDown 0xfdb04f //Down arrow
#define CodeLeft 0xfd10ef //Left arrow
#define CodeRight 0xfd50af //Right arrow

//Used only in mouse mode
#define CodeUpLf 0xfd20df //Set up
#define CodeUpRt 0xfd609f //Stop/mode
#define CodeLfClk 0xfd906f //Enter/save
#define CodeDnLf 0xfd30cf //0 10+
#define CodeDnRt 0xfd708f //Repeat
#define CodeLfDrag 0xfd08f7 //1
#define CodeRtDrag 0xfd8877 //2
#define CodeRtClk 0xfd48b7 //3
#define CodeFaster  0xfd28d7 //4
#define CodeScrUp 0xfd6897 //6
#define CodeSlower 0xfd18e7 //7
#define CodeScrDn 0xfd58a7 //9

//Arrow mode only
#define CodeHome 0xfd20df //Set up
#define CodePgUp 0xfd609f //Stop/mode
#define CodeEnter 0xfd906f //Enter/save
#define CodeEnd 0xfd30cf //0 10+
#define CodePgDn 0xfd708f //Repeat
#define CodeGUI 0xfd08f7 //1
#define CodeInsert 0xfd8877 //2
#define CodeDelete 0xfd48b7 //3
#define CodeBkSp 0xfd28d7 //4
#define CodeSpace 0xfd6897 //6
#define CodeEsc 0xfd18e7 //7
#define CodeTab 0xfd58a7 //9
```

```
#define Bright  16  //brightness of pixels

#define MOUSE_MODE 0
#define ARROW_MODE 1
#define MAX_MODE (ARROW_MODE+1)

#define MOUSE_LED 0
#define SHIFT_LED 1

Adafruit_NeoPixel strip = Adafruit_NeoPixel(2, 6, NEO_GRB + NEO_KHZ800);//Output pin 6
IRrecv myReceiver(11);  //Input pin 11
IRdecodeNEC myDecoder;

int Speed;
int Current_Mode;
char Control_State;
char Shift_State;
char Alt_State;
long Previous;
uint32_t Shift_Status, Mouse_Status;

void UpdateStatus () {
  Mouse_Status=0;
  Shift_Status=0;
  strip.setPixelColor(MOUSE_LED,Bright*Mouse.isPressed(MOUSE_LEFT),Bright*Current_Mode,Bright*Mouse.isPre
  strip.setPixelColor(SHIFT_LED,Bright*Alt_State,Bright*Control_State,Bright*Shift_State);
  strip.show();
};
//This Change_Mode routine is overly complex but it's a holdover from earlier code
//where I had three modes instead of two. Note we do not want to repeat feature
//on this function or on the toggle functions so we set Previous=0
void Change_Mode () {
  Current_Mode=(++Current_Mode) % MAX_MODE;
  Previous=0;   UpdateStatus(); delay(500);
};

void setup() {
  strip.begin();    strip.show();
  Shift_Status=0;   Mouse_Status=0;
  myReceiver.enableIRIn(); // Start the receiver
  Current_Mode=0;
  Control_State=0; Shift_State=0; Alt_State=0;
  Speed=8;  Previous=0;   UpdateStatus();
  Mouse.begin();
}

void Toggle_Mouse(int Button) {
  if(Mouse.isPressed(Button)) Mouse.release(Button);
  else Mouse.press(Button);
  Previous=0; UpdateStatus();  delay(500);
}

void Mouse_Mode() {
  switch (myDecoder.value) {
  case CodeLeft:       Mouse.move(-Speed,0,0);     break;
  case CodeRight:      Mouse.move( Speed,0,0);     break;
  case CodeUp:         Mouse.move(0,-Speed,0);     break;
  case CodeDown:       Mouse.move(0, Speed,0);     break;
  case CodeUpRt:       Mouse.move( Speed,-Speed,0);     break;
  case CodeUpLf:       Mouse.move(-Speed,-Speed,0);     break;
  case CodeDnLf:       Mouse.move(-Speed, Speed,0);     break;
  case CodeDnRt:       Mouse.move( Speed, Speed,0);     break;
  case CodeLfClk:      Mouse.release(MOUSE_LEFT);  Mouse.click(MOUSE_LEFT); delay(400);break;
  case CodeRtClk:      Mouse.release(MOUSE_RIGHT);  Mouse.click(MOUSE_RIGHT); delay(400);break;
  case CodeLfDrag:     Toggle_Mouse(MOUSE_LEFT);     break;
  case CodeRtDrag:     Toggle_Mouse(MOUSE_RIGHT);     break;
  case CodeFaster:     Speed=min(30,++Speed); delay(100); break;
  case CodeSlower:     Speed=max(1,--Speed); delay(100); break;
  case CodeScrUp:      Mouse.move(0,0,1); delay(100);    break;
  case CodeScrDn:      Mouse.move(0,0,-1); delay(100);     break;
  };
};

void Toggle_Key(char *Toggle,char Key) {
  if(*Toggle){
    Keyboard.release(Key);  *Toggle=0;
  }
  else{
    Keyboard.press(Key);    *Toggle=1;
  };
  Previous=0; UpdateStatus(); delay(500);
};
//Releases all held mouse buttons and toggle keys
void Release_All() {
  Keyboard.releaseAll();
  Mouse.release(MOUSE_LEFT);   Mouse.release(MOUSE_RIGHT);   Mouse.release(MOUSE_MIDDLE);
  Alt_State=0;   Control_State=0;   Shift_State=0;
  UpdateStatus();
};
//In my experience some keys work better if you put a little extra delay.
void Key_Press (char Key,int D) {
  Keyboard.write(Key);   delay(150+D);
}

void Arrow_Mode() {
  switch (myDecoder.value) {
    case CodeLeft:   Key_Press(KEY_LEFT_ARROW,0); break;
    case CodeRight:  Key_Press(KEY_RIGHT_ARROW,0); break;
    case CodeUp:     Key_Press(KEY_UP_ARROW,0); break;
    case CodeDown:   Key_Press(KEY_DOWN_ARROW,0); break;
    case CodeInsert: Key_Press(KEY_INSERT,100); break;
    case CodeDelete: Key_Press(KEY_DELETE,100); break;
    case CodeEnter:  Key_Press(KEY_RETURN,100); break;
    case CodeBkSp:   Key_Press(KEY_BACKSPACE,100); break;
    case CodePgUp:   Key_Press(KEY_PAGE_UP,200); break;
    case CodePgDn:   Key_Press(KEY_PAGE_DOWN,200); break;
    case CodeHome:   Key_Press(KEY_HOME,0); break;
    case CodeEnd:    Key_Press(KEY_END,0); break;
    case CodeGUI:    Key_Press(KEY_LEFT_GUI, 100); break;
    case CodeSpace:  Key_Press(32, 100); break;//ASCII space
    case CodeEsc:    Key_Press(KEY_ESC, 100); break;
    case CodeTab:    Key_Press(KEY_TAB, 100); break;
  };
};
/*
//You can uncomment and use this routine to send control characters such as
//control-z for an undo button or control-c for copy etc.
void Send_Control(char Key,int D) {
 Release_All(); Keyboard.press(KEY_LEFT_CTRL);
 Keyboard.write(Key); Release_All();delay(150+D); Update ();
}
 */
void loop() {
  if (myReceiver.getResults()) {
    myDecoder.decode();
    //Adafruit remote uses NEC protocol which sends a special repeat code
    //if you are holding down the same button. We store each received code
    //in "Previous" so that we can properly handle repeat codes.
    if(myDecoder.value==0xffffffff)
       myDecoder.value=Previous;
    else
       Previous=myDecoder.value;
    switch(Current_Mode) {
      case MOUSE_MODE: Mouse_Mode(); break;
      case ARROW_MODE: Arrow_Mode(); break;
    }
    switch (myDecoder.value) {
      case CodeMode:  Change_Mode(); break;
      case CodeAlt:   Toggle_Key(&Alt_State,KEY_LEFT_ALT); break;
      case CodeShift: Toggle_Key(&Shift_State,KEY_LEFT_SHIFT); break;
      case CodeCtrl:  Toggle_Key(&Control_State,KEY_LEFT_CTRL); break;
      case CodeRls:   Release_All(); break;
    };
    myReceiver.enableIRIn(); // Re-Start the receiver
  }
}
```