Engduino Duels Tutorial

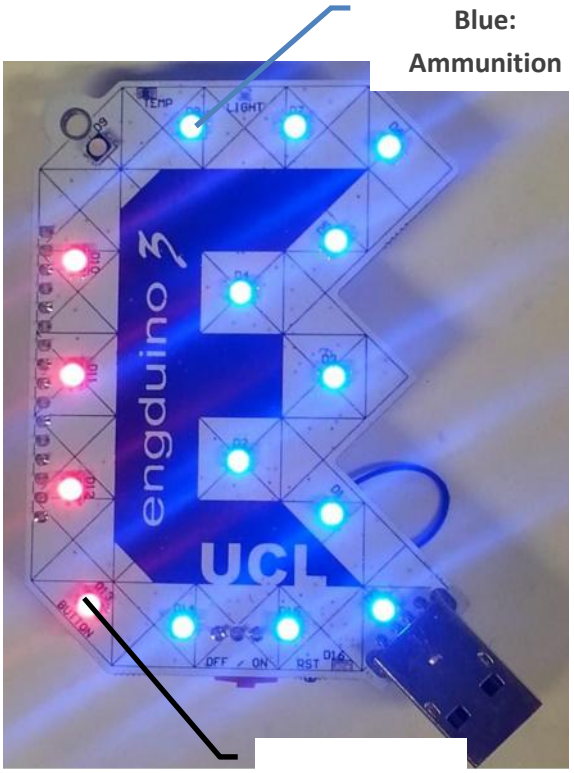
By Aaron Cuthbertson, Al-Yasa Ismail and Jake Chin

In this tutorial you will program a simple two-player dueling game using the infrared transceivers attached to the Engduino. As an expansion, you will be adding health and ammo to the game.

In this tutorial you will learn how to program a simple multiple-player dueling game using the infrared transceivers, buttons and all of the LEDs attached to the Engduino.

The finished code will allow the Engduino to transmit the infrared signal on the press of the button and also flash all of the LEDs blue to indicate it has done so. In addition, another Engduino will be able to receive the signal -if it is close enough to the transmitter- and flash all of the LEDs red to show that the signal has been received.

As an expansion, you will be adding health and ammo to the game, in the form of red and blue LEDs respectively, ultimately making the game more detailed and interesting.



The Game

This game uses the Engduino button for shooting your opponents; and the infrared on the Engduino board to shoot. At the start of the game, each player has 11 ammunitions (ammo) represented by blue lights and 4 heaths represented by red lights. Each time the player shoots by pressing the button, ammunition is used and the number of blue lights goes down by one. If the other player is shot (received an infrared signal from the other player), he/she lost a life and a red light goes out. The winner is the player who has any red light(s) remains when other players have them all out.

Programming

Step 1: Setting up

```
#include <EngduinoLEDs.h>
#include <EngduinoButton.h>
#include <EngduinoIR.h>
```

First of all, like with any Engduino program, you need to tell the program which parts of the board you will be using before you begin with the rest of your code.

This includes the Engduino libraries that are necessary for the code, and lets the program know you will be using them.

```
void setup()
{
  EngduinoLEDs.begin();
  EngduinoButton.begin();
  EngduinoIR.begin();
}
```

This starts the Engduino libraries so that you can use functions relating to them.

```
void loop() {
  int      len = 0;
  uint8_t buf[IRBUFSZ];
}
```

This declares an integer named **len** equal to 0 within a loop. ...once the loop is repeated, `len` returns to its original value of 0.

Step 2: Button and Infrared (IR)

It also declares a `uint8_t` variable **buf** as a buffer. The variable type **`uint8_t`** is the same as a byte in that it stores an 8-bit value in Arduino C. It symbolised unsigned integer type 8 bit and determines how this variable is represented in the computer binary memory.

```
void loop() {
  int      len = 0;
  uint8_t buf[IRBUFSZ];

  if (EngduinoButton.wasPressed()) {

    EngduinoIR.send(len, +1);

    EngduinoLEDs.setAll(BLUE, 1);
    delay(100);
    EngduinoLEDs.setAll(OFF);
  }
```

The first two lines within the loop declare an integer and a byte, respectively. `len` is the variable that will be sent with the infrared signal (to be received). The byte is used as a buffer in which to receive a message or signal if one has been sent by another user.

The if-statement checks if the user has pressed the button on the Engduino. If so, the Engduino sends an infrared signal with the value of **`len + 1`**: **`len`** equals 0, so it sends the value of 1. It then flashes all the LEDs blue to tell the user the signal has been sent.

```
len = EngduinoIR.recv(buf, 1000);
if (len > 0) {
    EngduinoLEDs.setAll(RED, 1);
    delay(150);
    EngduinoLEDs.setAll(OFF);
  }
```

`len` is altered to equal the received signal. The if-statement is there to check if **`len`** is more than zero, meaning that the signal has been received from another Engduino. If it has, it will flash all the LEDs red once to show that the player has been hit.

```
#include <EngduinoLEDs.h>
#include <EngduinoButton.h>
#include <EngduinoIR.h>

void setup()
{
  EngduinoLEDs.begin();
  EngduinoButton.begin();
  EngduinoIR.begin();
}

void loop() {
  int      len = 0;
  uint8_t buf[IRBUFSZ];

  if (EngduinoButton.wasPressed()) {

    EngduinoIR.send(len, +1);

    EngduinoLEDs.setAll(BLUE, 1);
    delay(100);
    EngduinoLEDs.setAll(OFF);
  }

  len = EngduinoIR.recv(buf, 1000);
  if (len > 0) {
      EngduinoLEDs.setAll(RED, 1);
      delay(150);
      EngduinoLEDs.setAll(OFF);
    }
}
```

And that's your code! This will send and receive infrared signals with other Engduinos.

Step 3: Health and Ammo

Now let's add health and ammo to it to make the game more exciting.

```
int b=9;
int r=8;

int ammo = 11;
int health = 5;
```

First, declare some variables just above setup. The variables **b** records the blue LEDs that represents the current ammo; and **r** records the red LEDs that is used to represent health. Variables **ammo** and **health** are two variables used to store the maximum **health** and **ammo** values.

```
void setup()
{
  EngduinoLEDs.begin();
  EngduinoButton.begin();
  EngduinoIR.begin();
  EngduinoLEDs.setLED(0, BLUE);
  EngduinoLEDs.setLED(1, BLUE);
  EngduinoLEDs.setLED(2, BLUE);
  EngduinoLEDs.setLED(3, BLUE);
  EngduinoLEDs.setLED(4, BLUE);
  EngduinoLEDs.setLED(5, BLUE);
  EngduinoLEDs.setLED(6, BLUE);
  EngduinoLEDs.setLED(7, BLUE);
  EngduinoLEDs.setLED(8, BLUE);
  EngduinoLEDs.setLED(9, RED);
  EngduinoLEDs.setLED(10, RED);
  EngduinoLEDs.setLED(11, RED);
  EngduinoLEDs.setLED(12, RED);
  EngduinoLEDs.setLED(13, RED);
  EngduinoLEDs.setLED(14, BLUE);
  EngduinoLEDs.setLED(15, BLUE);
}
```

Next, add this code to setup. This will initialise the LEDs so they display the starting **health** and **ammo** values.

Step 4: The Duel-logic and Communication

```
void loop() {
  int     len = 0;
  uint8_t buf[IRBUFSZ];

  if (EngduinoButton.wasPressed()) {
    EngduinoIR.send(len, +1);
    ammo = ammo -1;
    b = (b-1);
    EngduinoLEDs.setLED(b, OFF);
    delay(200);
  }
```

Now add this to the if-statement that checks if the player has pressed the button. This will reduce the values of both variable **ammo** and **b by 1** and turn whichever LED is equal to **b** on the Engduino board to off. It will also send an infrared communication signal to your opponent's Engduino board – or shoot your opponent per say.

```
len = EngduinoIR.recv(buf, 1000);
if (len > 0) {
  health = health - 1;
  r = (r+1);
  EngduinoLEDs.setLED(r, OFF);
}
```

After that, change the second if-statement in the loop to the code above. Similar to the first if-statement, the values of both **health** and **r** will be decreased by one if the Engduino is "hit" by the opponent's Engduino. In real term it means that we have received an infrared message from your opponent. The **r^{th} LED** on the Engduino board will be turned off.

```
  if(ammo < 1) {
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
  }
```

If the player's **ammo** is lower than 1, i.e., they run out of admonition. We flash all the LEDs blue to show that the player has lost.

```
if(health < 1) {
  EngduinoLEDs.setAll(RED);
  delay(500);
  EngduinoLEDs.setAll(OFF);
  delay(500);
  EngduinoLEDs.setAll(RED);
  delay(500);
  EngduinoLEDs.setAll(OFF);
  delay(500);
  EngduinoLEDs.setAll(RED);
  delay(500);
  EngduinoLEDs.setAll(OFF);
  delay(500);
}
```

This will check if the player's **`health`** is lower than 1: i.e. if they are out of **`health`**. If so, it will flash all the LEDs red to show that the player has lost.

And that's everything! The complete program is shown on the next page. This will send and receive signals with other Engduinos, but also store variables for health and ammo to make it into a game. Try changing the values of the health and ammo variables: what effect does it have on the game? And what would happen if ammo was less than health?

Further challenges

- In Setup, can you make the code shorter by using a for-loop?
- Can you set up a separate function noAmmo() and noHealth() for the codes to display the results when we run out of ammunition and health? This is to clarify the logic flow and also to encourage code reuse.
- Can you form teams with this game? Write out your idea and logic and try implementing it.
- What else would you like to do to improve this game?

```
#include <EngduinoLEDs.h>
#include <EngduinoButton.h>
#include <EngduinoIR.h>

int b=9;
int r=8;

int ammo = 11;
int health = 5;

void setup()
{
  EngduinoLEDs.begin();
  EngduinoButton.begin();
  EngduinoIR.begin();
  EngduinoLEDs.setLED(0, BLUE);
  EngduinoLEDs.setLED(1, BLUE);
  EngduinoLEDs.setLED(2, BLUE);
  EngduinoLEDs.setLED(3, BLUE);
  EngduinoLEDs.setLED(4, BLUE);
  EngduinoLEDs.setLED(5, BLUE);
  EngduinoLEDs.setLED(6, BLUE);
  EngduinoLEDs.setLED(7, BLUE);
  EngduinoLEDs.setLED(8, BLUE);
  EngduinoLEDs.setLED(9, RED);
  EngduinoLEDs.setLED(10, RED);
  EngduinoLEDs.setLED(11, RED);
  EngduinoLEDs.setLED(12, RED);
  EngduinoLEDs.setLED(13, RED);
  EngduinoLEDs.setLED(14, BLUE);
  EngduinoLEDs.setLED(15, BLUE);
}

void loop() {
  int     len = 0;
  uint8_t buf[IRBUFSZ];

    EngduinoIR.send(len, +1);
    ammo = ammo -1;
    b = (b-1);
    EngduinoLEDs.setLED(b, OFF);
    delay(200);
  }
  len = EngduinoIR.recv(buf, 1000);
  if (len > 0) {
    health = health - 1;
    r = (r+1);
    EngduinoLEDs.setLED(r, OFF);
  }

  if(ammo < 1) {
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(BLUE);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
  }

  if(health < 1) {
    EngduinoLEDs.setAll(RED);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(RED);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
    EngduinoLEDs.setAll(RED);
    delay(500);
    EngduinoLEDs.setAll(OFF);
    delay(500);
  }
}
```