# The Hunger Games

- **By Group 16**

A **real time shooting based survival game** in which each player is for himself. Each player is equipped with a laser gun, a magazine full of ten bullets and a task to reach the centralized server that is continuously monitoring the entire game. The server may reward or penalize the player depending upon her/his performance and creates bombs to make the game more difficult and tricky.


Group 16 consists of:

a.) Mudit Agrawal – 140101042

b.) Paritosh Mittal – 140101048

c.) Manoj Ghuhan A – 140101082

d.) Kevin Pandya – 140101047

## Salient Features

- Centralized System : The game revolves around Raspberry Pi2 and which at any point of time, knows every attribute of each player and is the ultimate target to reach.
- Wireless Communication : Use of Xbee and Radio Frequencies to trigger Wireless communication and exchange of critical information is a key aspect of the game.
- Real-time Display : All critical information like the health, bullet count etc. are displayed using a LCD monitor, given to each player.
- Innovative Bullet Count : A reflection based shooting mechanism is incorporated to keep an accurate track of bullets of each player.
- Unpredictable gameplay : Pi controls the game, and randomly generates Mines, Health and Armour packets, thereby making it different every time.

## Components Used :

- Raspberry Pi2
- Arduino Mega
- Arduino Base Shield
- Xbee Shield
- Xbee S1
- Xbee Dongle
- BLE Beacon
- LCD Display
- Light Dependent Resistors
- Grove LED Bars
- Buzzer
- Laser
- LED

## Softwares Used :

- Arduino IDE

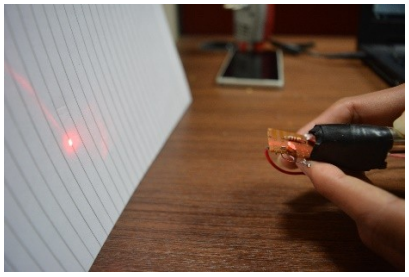
  https://www.arduino.cc/en/Main/Software
- XCTU for Xbee Configuration

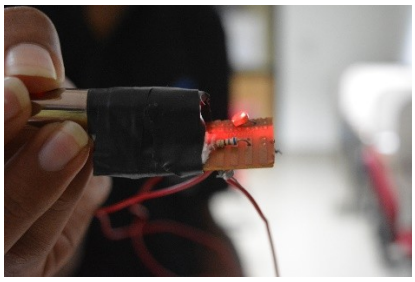
  https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu
- Python - for the script on the Raspberry Pi

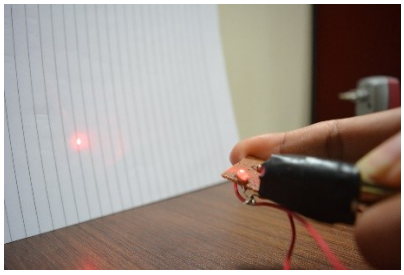
# The Shooting Module:

Each player was given a laser gun and a circuit to keep count of the shots fired from the laser.  The Circuit consisted of a **Light Dependent Resistor (LDR)** in series with a normal ohmic resistor. Entire circuit was given 5V and potential across LDR was measured.

Since Laser is a very high intensity source, potential across LDR saw a quick increase from its previous value. This difference was measured and used to keep track of the bullets consumed.







Hardware Used:
a.) Laser

b.) Light Dependent Resistor

c.) Led Bar to show player available bullets

# The Health Module:

Multiple LDRs were soldered in parallel to create a grid of light sensitive sensors such that, whenever laser fell on any one of them a spike in potential drop was observed detecting a **hit**.

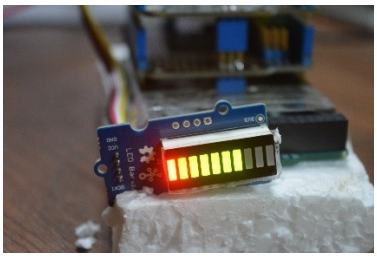
An LED bar was used to show player (in real-time) her/his health. Alongside this was a buzzer that rang whenever hit. A long buzzer indicated that player is out of health.

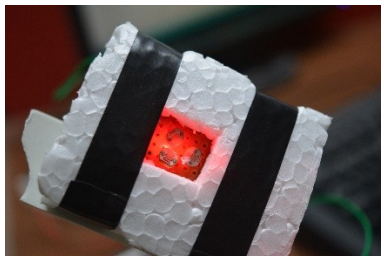
Health Reduced due to:

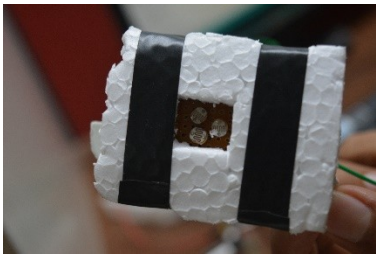
a.) **Bomb –** Health was reduced by a 20 whenever player came close to a mine planted by the centralized server to prevent player from reaching it.

b.) **Head-Shot –** A major damage of 50 points is incurred whenever an opponent is able to hit a shot directly on the head. Since head is the most sensitive spot, a shot there results in maximum damage.

c.) **Chest Shot –** A normal damage of 10 points whenever any other sensor is hit if Armour is 0.



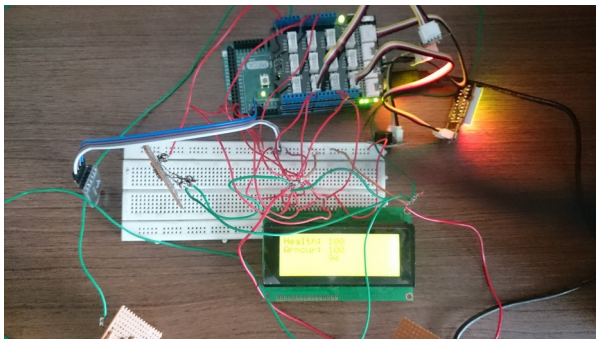




Library Link: https://github.com/Seeed-Studio/Grove_LED_Bar

# The Armour Module

- An LCD Bar was also used to display Health and Armour.
- The Armour will decrease whenever there is a chest shot but not when there is head shot.
- Also every 100 sec you will be rewarded with full Armour if you are alive till then while your health will remain same.
- There is Timer in the LCD Display itself that will count the time. Your health will not be affected if you have greater than 0 Armour unless it is a head shot.

# LCD Display

It uses I2C as communication method with microcontroller for the display.

Library used: LiquidCrystal.h

//Initialize the library with number of interface pins
LiquidCrystal lcd(12, 7, 6, 25, 24, 23, 22)

Some of the LiquidCrystal library function used were
LiquidCrystal.**begin**(x,y); *//set up the LCD's number of x columns and y rows:*

LiquidCrystal.**setCursor**(x,y); *//set the cursor to column x, line y*

LiquidCrystal.**print**(message); *//prints the message on lcd display*

LiquidCrystal.**clear**(); // Clear screen

Pin connections

---

|VSS| -> Arduino GND

|VDD| -> Arduino +5v

|VO | -> Arduino GND pin

|RS | -> Arduino pin 12

|RW | -> Arduino 7

|E | -> Arduino pin 6

|D0 | -> Arduino - Not Connected

|D1 | -> Arduino - Not Connected

|D2 | -> Arduino - Not Connected

|D3 | -> Arduino - Not Connected

|D4 | -> Arduino pin 25

|D5 | -> Arduino pin 24

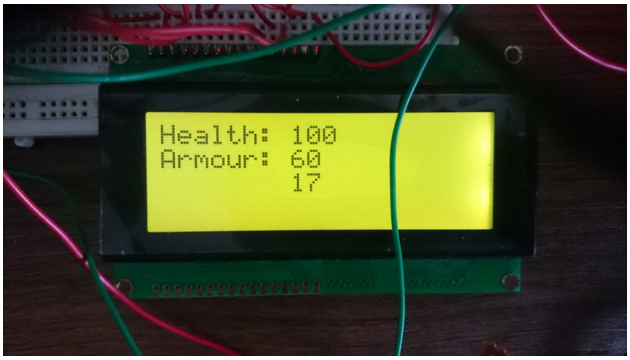
|D6 | -> Arduino pin 23

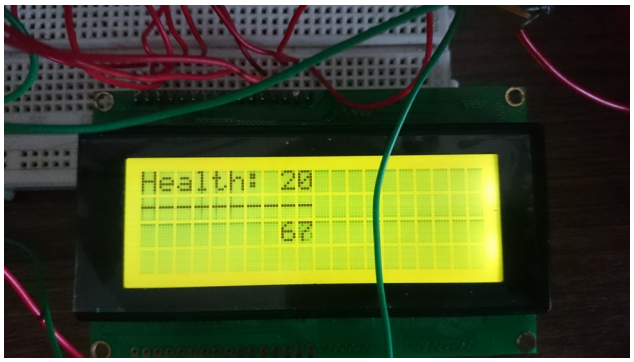
|D7 | -> Arduino pin 22

|A | -> Arduino Pin 13

|K | -> Arduino GND

**Note:** If you're refering to the data sheet of LCD display you will find that VO is connected through 10k potentiometer. But in our case we're connecting arduino to pc through USB so current is small enough so poterntiometer is not required.




Useful Link: https://www.arduino.cc/en/Tutorial/HelloWorld

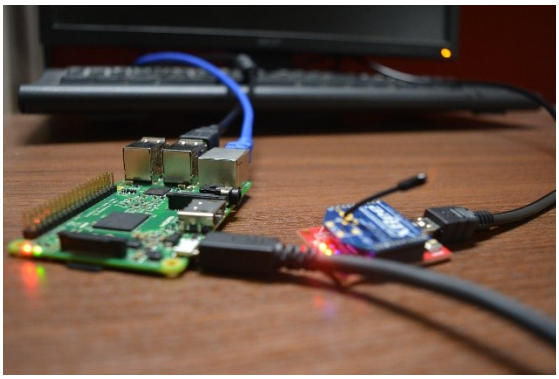
Library Used: https://www.arduino.cc/en/Reference/LiquidCrystal

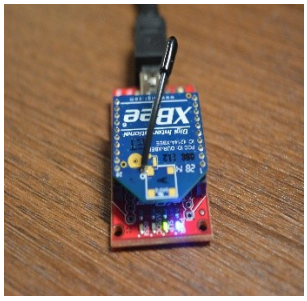
# The Communication Module

A **network of XBees** was created to ensure proper data transfer and game synchronization between each player and the centralized server.

Each Arduino had an **Xbee shield** and one Xbee that, was capable of sending and receiving data to the raspberry pi2. Raspberry pi2 had an **Xbee dongle** that enabled it to communicate using Serial libraries of python.

All Xbee were configured to communicate on Channel ‘C’ and were given a PAN ID of 1111. Each Xbee on Arduino had their DL as the address of the Xbee on pi. Xbee pi on the other had was able to send data to every other Xbee using its python library.





Pi used to receive data in the form of a Json Object that contains source info and data, making it easy for pi to classify data based on source. Strings contained particular tokens that helped pi decide necessary actions required. Script was written in such a way that pi knew every small thing that was happening in the game and hence updates were made in real-time.

# Distance Approximation

Hm 10 BLE beacons were used to approximate the distance of each player from the server. The beacons were Bluetooth 4.0 beacons and RSSI value of signals was used to approximate the distance.

The beacon on pi was set to iBeacon mode using AT commands. Below are the AT commands used for the project:

**AT**
**AT+RESET**
**AT+IMME1**
**AT+ROLE1**
**AT+IBEA1**
**AT+DISC?**
**AT+DISI?**

Every Arduino had a Beacon that, using DISI command and the address of that particular beacon finds the RSSI value or the percentage signal strength value, that is used to approximate the distance.

The knowledge of inter player distance allowed Pi to plot bombs and random health packages, making the game a more interesting and tricky!

Datasheet Link:
http://fab.cba.mit.edu/classes/863.15/doc/tutorials/programming/bluetooth/bluetooth40_en.pdf

# Raspberry Pi2

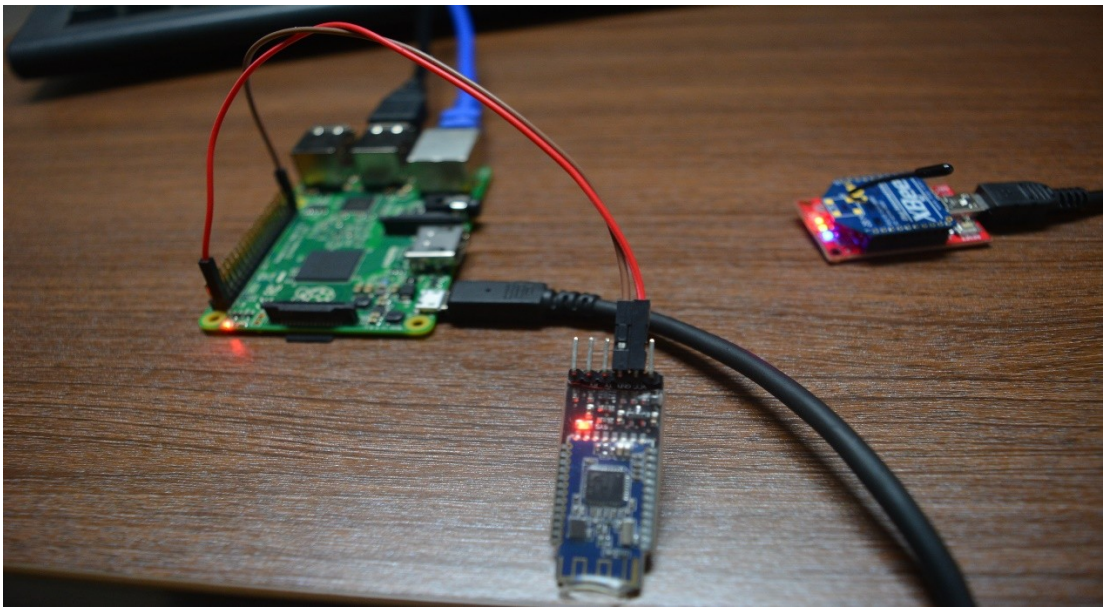
The centralized server we have been saying so far is nothing but raspberry pi. A python Script is run in Pi that contains a Class called Player. Each player has bullets, health and distance from server as parameters.

Pi has a Xbee dongle that receives the data from players and updates the parameters accordingly. Alongside this, pi also has a Beacon set in iBeacon mode to allow other beacons (on player arduino) to calculate approximate distance, allowing it to plant bombs in the game.

The bombs appear in circles as mine fields, they randomly appear for a particular time and inflicts a damage of 20 units to the person close to it.

Whenever a shot if fired of a player is hit, pi knows and accordingly makes the necessary updates, thereby being up-to-date all through the game.

A player wins the game if he is the first one to reach the pi.



Useful Link: https://github.com/nioinnovation/python-xbee

# Hardware Assembly Guide

Follow the following points one by one to assemble all the hardware properly as per the declaration on the code.

1.) Each Arduino has two shields, placed one above the other:

a.) A Xbee Shield with a Xbee on it.

b.) A base Shield for other Sensors

2.) Put Health LED bar on D4 of the base Shield

3.) Put Bullet LED bar on D8 of the base Shield

4.) Put Buzzer on A0 of the base Shield

5.) Laser gun and Shooting targets ground will go to the ground or Arduino

6.) Power Laser Gun, and Shooting targets with 5V Vcc

7.) Put the third end of Laser Gun to A4

8.) Put the third end of Targets to A5 and for head-shot A3.

9.) Beacon on each Arduino will be powered with 3.3 V and ground will be same as Arduino ground.

10.) Tx of Beacon will go to slot 19 (Rx1) and Rx of Beacon will go to slot 18(Tx1).

**This Being a **Real-time outdoor game**, it can be played anywhere you want. The world is your stage here. Although care should be taken as external power needs to be given to drive each Arduino board and corresponding sensors.

# Problems Faced

### Using Xbee!

Xbees need to be configured first using XCTU software, by default, Xbee shield uses the same Serial port as used by the Arduino and hence won't work. Using Software Serial Library we change the serial ports and Jumpers are put to the corresponding slots to ensure proper Communication

### Using Xbee and Bluetooth Simultaneously!

Both Xbee and Bluetooth use serial ports for communication and hence by default, only one can work. Putting Bluetooth on a different Hardware Serial port solves the problem.

### Bluetooth RSSI value for Distance!

The RSSI value is known to jump abruptly and hence we need to average it over a particular time duration to normalize the anomalies.

### Bullet Count!

An innovative way of Reflection based shot firing technique helps keeping in track of the bullets being fired.

### Normalize Varying Light:

Game when played in dark or in Direct Sunlight, affects the value of LDRs and hence to avoid always changing code, we used difference of intensities or measuring the spike in voltage drop to signify a hit.

# Conclusion

Working with the Raspberry Pi and Arduino was an exciting experience. We were able to appreciate the power and potential of the Arduino and the Pi and their ability to work with multiple sensors at a time. The project also helped us better understand how to work with hardware and low level software. As the world starts to realize the potential of IoT and Real World Games. This project has provided a glimpse of that future, and we hope it is useful for us in the future.

# Future Works

### Map

Additional feature of map can be added in the LCD Display itself. The distance of player can be measured from the centralized server and can be reflected on LCD Display. Along with this various positions of Armour and Allies can be shown on the Display.

### Virtual Reality

The game can be taken to VR level with suitable equipments.

### Weapons

The concept of different weapons can be introduced if we can have lasers who can emit rays at different wavelength acoording to the provided code.

### Training

This project with full-fledged equipments can be used to train in Army and other combat scenarios. As we all know evolution of warfare games is closely related to real world combat scenarios.

## Link to the Code of project