Carthafind

Titolo del progetto: Carthafind
Alunni: Nadir Barlozzo
Classe: Info 3
Anno scolastico: 2017/2018
Docente responsabile: Luca Muggiasca

1 Introduzione

1.1 Informazioni sul progetto

Docente Responsabile: Luca Muggiasca
Scuola: Scuola d'Arti Mestieri Trevano
Sezione: Informatica
Materia: Modulo 306
Data inizio progetto: 16.03.2017
Data consegna progetto: 18.05.2018

1.2 Abstract

As the use of machines and robots to increase physical performances in sport competitions increases, more people start using these reaction, strength and speed machines to boost their performances. We were told to recreate a Batak machine and in this documentation will go through the development of it.

1.3 Scopo

Lo scopo del progetto consiste nel creare un applicativo utilizzato per la ricerca avanzata tra i progetti già conclusi alla Scuola Arti e Mestieri di Trevano, tramite parole chiave, descrizioni, titoli o docenti responsabili.

Analisi

1.4 Analisi del dominio

Molti dei progetti conclusi nel nostro istituto dopo la loro conclusione vengono dimenticati, messi in un armadio e lasciati a prendere polvere. Lo scopo di questo progetto è schedare e classificare ogni progetto e metterli alla portata di tutti, sia persone interne che esterne alla scuola, tramite un applicativo di ricerca così da avere un punto di ricerca ed interesse per evitarsi di intralciare in idee già definite.

1.5 Analisi dei costi e benefici

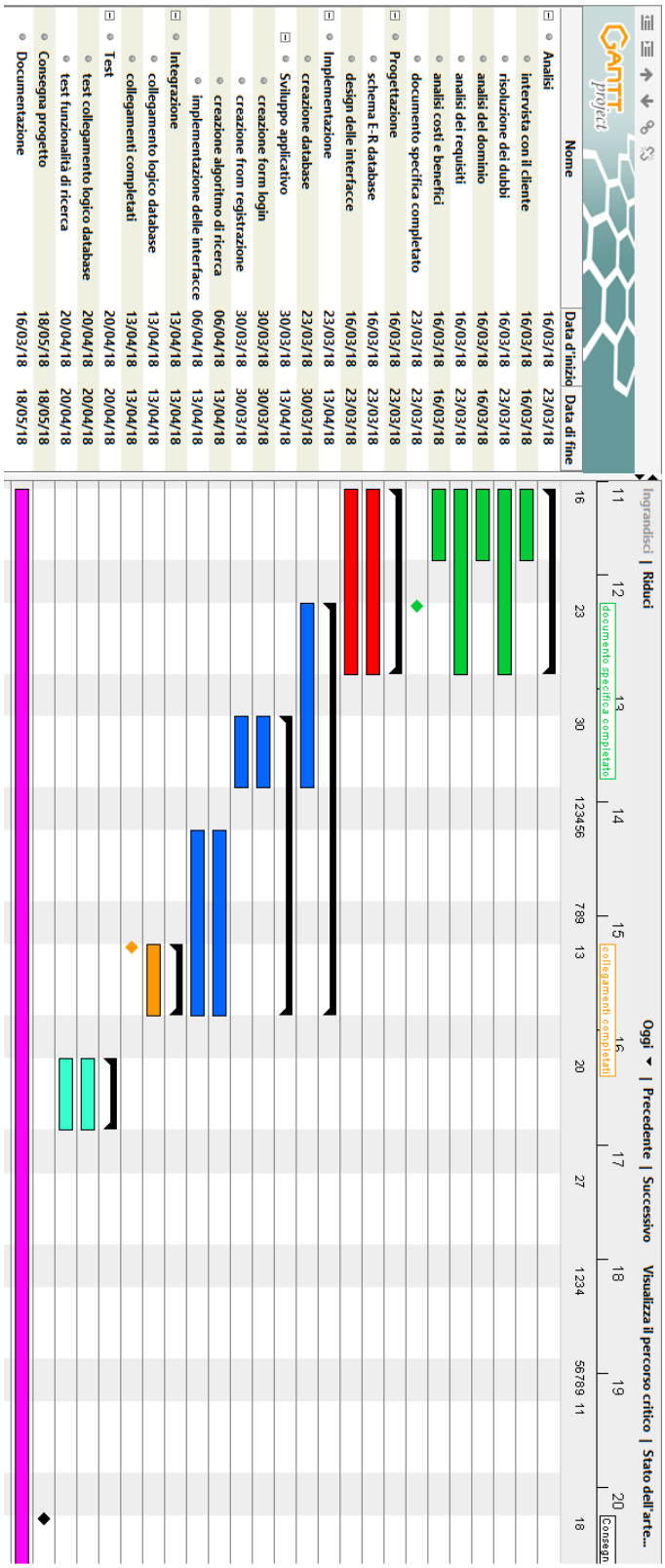
Categoria	**Costo**
Spazio web	x
Personale	x ore * 1 persona * 50 CHF = x
TOT	**x**

1.6 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Gestione registrazione
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Deve esserci un form per la registrazione di un nuovo utente
002	Deve esserci un campo per il nome di tipo testo
003	Deve esserci un campo per il cognome di tipo testo
004	Deve esserci un campo per il nickname di tipo testo
005	Deve esserci un campo per la password di tipo testo codificato
006	Deve esserci un campo per l'email di tipo testo e domini accettati sono solo *samtrevano* ed *edu*
007	Deve essere mandato un link per confermare la registrazione sull'email data

ID: REQ-02	
Nome	Gestione login
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Deve esserci un form per il login di un utente
002	Deve esserci un campo per il nickname di tipo testo
003	Deve esserci un campo per la password di tipo testo codificato

1.7 Pianificazione



1.8 Analisi dei mezzi

1.8.1 Software

Microsoft Word 2013
Gantt Project 2.8.5
PowerPoint 2013
HeidiSQL
Notepad++ 7.5
Google Chrome 64.0
TortoiseGit 2.5.0.0

1.8.2 Hardware

PC portatile

2 Progettazione

2.1 Design dei dati e database

Abbiamo creato uno schema ER che rappresenta la struttura che dovrà avere il database.
Il database verrà utilizzato per salvare i dati inerenti ai giocatori e ai loro punteggi, in modo da poter creare una classifica per ogni modalità.

Ogni volta che si avvierà una partita, verrà creata una riga nella tabella ‘Giocatore’ con il nuovo ID incrementale e con il nome corrispondente.

Nella tabella ‘Modalità’ sono contenute tutte e 23 le modalità di gioco.

La tabella ‘Utilizza’ invece contiene l’ID del giocatore e l’ID della modalità in combinazione come chiave primaria e il punteggio effettuato dal giocatore nella rispettiva modalità.

2.2 Design delle interfacce

Per descrivere il design delle interfacce abbiamo realizzato 23 schemi rappresentanti ogni modalità.
VEDI ALLEGATO A.

3 Implementazione

È stato definito come punto di partenza un piccolo codice trovato su internet che calcolava il tempo di reazione per premere un pulsante. Una volta capito il codice, è stato possibile adattarlo in modo da cominciare a lavorare sulle modalità, a partire dalla prima. Si sono verificati innumerevoli problemi iniziali dovuti al capire la logica migliore da utilizzare per definire al meglio le modalità, ma una volta completata la prima, è stato possibile affrettare i tempi e creare quasi la metà delle modalità modificando relativamente poco codice.
Non avendo specifiche precise, abbiamo proceduto ad effettuare la scelta delle modalità tramite un telecomando infrarossi, cosa che si è rivelata purtroppo una perdita di tempo, poiché non teneva in conto il fatto che il lavoro sarebbe stato presentato ad una fiera e che quindi oltre alla possibilità di interferenze o di una mira errata, era molto probabile che il telecomando sparisse una volta dato a passanti. È stato quindi risolto relativamente in fretta modificando il codice e facendo fare la scelta direttamente tramite la pressione dei bottoni.
Nel corso del progetto ci si è resi conto della difficoltà in implementare alcune di queste modalità (5, 6, 10, 20) ed è stato deciso di escluderle dal progetto. Le modalità rimanenti sono state divise in funzioni che raggruppano le modalità simili tra loro, in modo da utilizzare un unico programma '.ino', semplice da inserire e gestire su Arduino. Per rendere più leggibile il codice, è stato fatto tramite la possibilità di implementare più file in un unico sketch Arduino.

Codice basilare spiegato nel dettaglio

```
150  do {
151    for (int i = 0; i < sizeof(buttonPins) / sizeof(buttonPins[0]); i++) {
152      currentButtonsState[i] = debounce(i);
153      if (currentButtonsState[i] == true && lastButtonsState[i] == false && i < 10) {
154        arraySelected[0] = arraySelected[1];
155        arraySelected[1] = i;
156        modeSelected = arraySelected[1] + arraySelected[0] * 10;
157        lcd.clear();
158        lcd.setCursor(0, 0);
159        lcd.print("Mod disponibili:");
160        lcd.setCursor(0, 1);
161        lcd.print("1 - 15");
162        lcd.setCursor(0, 2);
163        lcd.print("modalita");
164        lcd.setCursor(9, 2);
165        lcd.print(modeSelected);
166        if  (modeSelected < 1 || modeSelected > nMod) {
167          lcd.setCursor(0, 4);
168          lcd.print("mod non valida");
169        }
170      }
171      lastButtonsState[i] = currentButtonsState[i];
172    }
173  } while (!currentButtonsState[11] || !(modeSelected > 0 && modeSelected <= nMod));
```

Questa parte di codice serve a selezionare la modalità. Non è possibile proseguire fino a quando il tasto @ (il pulsante 11) sia stato premuto e contemporaneamente la modalità selezionata sia maggiore di 0 e minore o uguale al numero di modalità implementate (15, nel nostro caso). Il for che si apre a riga 151 controlla tutti i bottoni e ne salva lo stato a riga 152 tramite la funzione debounce (spiegata in seguito). A riga 152 controlliamo che il bottone sia stato premuto e per evitare che rientri dentro l'if anche in seguito mentre sta venendo premuto, controlliamo che lo stato precedente (salvato alla fine del for alla riga 171) sia a false e che quindi non stava venendo premuto. Viene inoltre controllato che il numero del pulsante premuto non sia superiore a 9, poiché altrimenti non sarebbe stato premuto un pulsante inerente al cambio di modalità.
Alle righe 154 e 155 salviamo il valore della modalità in un array, utilizzando un sistema a shift, quindi il primo numero (unità) una volta premuto il secondo numero fa diventare il primo numero rappresentante delle

decine mentre il secondo diventa l'identificativo per le unità. Inserito il terzo numero, esso diventa unità, il secondo numero diventa decina e il primo numero viene eliminato.
Alla riga 156 definisco effettivamente la modalità selezionata e in seguito la mostro all'utente tramite LCD.

```
2  boolean debounce(int n)
3  {
4    boolean current = (!digitalRead(buttonPins[n]));
5    if (lastButtonsState[n] != current)
6    {
7      delay(1);
8      current = !digitalRead(buttonPins[n]);
9    }
10   return current;
11 }
```

Questa funzione è fondamentale per tutta la parte di codice, poiché ritorna il valore del bottone corrispondente al numero passato come parametro. Se il valore letto in current è diverso dal valore precedente, viene fatto passare un millisecondo e riregistrato il valore, in modo che esso sia il più corretto possibile. Il digitalRead() è negato in entrambi i casi poiché per i bottoni è stata usata una resistenza di pull-up e di conseguenza il valore letto normalmente corrisponde ad 1, mentre quello quando viene premuto corrisponde a 0.

```
9   while (true) {
10    for (int i = 0; i < dimensions; i++)
11    {
12      currentButtonsState[i] = debounce(i);
13      if (currentButtonsState[i] == true && currentButtonsState[i] != lastButtonsState[i] && i == currentNumber)
14      {
15        score++;
16        stampLCD();
17        digitalWrite(buzzerPin, HIGH);
18        digitalWrite(buzzerPin2, HIGH);
19        delay(delayValue);
20        digitalWrite(buzzerPin, LOW);
21        digitalWrite(buzzerPin2, LOW);
22        setNextButton();
23      }
24
25      lastButtonsState[i] = currentButtonsState[i];
26      delay(1);
27      timerGame = millis() - timer;
28      if (timerGame >= countdown)
29      {
30        for (int i = 0; i < dimensions; i++)
31        {
32          digitalWrite(ledPins[i], LOW);
33        }
61          scores[0] = score;
62          scores[1] = timerGame;
63          clearVariables();
64          return;
```

Questo è il corpo primario di quella che si può definire la modalità più classica e semplice, ovvero quella in cui si cercano di totalizzare il maggior numero di punti nel tempo stabilito (categoria 1, vedi sotto).
Il ciclo while(true) serve ad imitare il comportamento del corpo loop principale e permette di ripetere il codice fino a quando la condizione a riga 28 si verificherà (se il tempo di gioco diventa superiore al tempo concesso al gioco, ovvero la variabile countdown), andando così dapprima a spegnere tutti i led dei bottoni, salvare i punteggi (punteggio e tempo) e risettare ai propri valori di default tutte le variabili (grazie alla riga 63 con la funzione clearVariables), andando infine a fare un return, uscendo completamente dalla funzione della modalità. Il ciclo for controlla i pulsanti che sono da controllare (in base a dimensions, che indica il numero di pulsanti utilizzati da 0 al numero), salvandone i valori a riga 12 tramite la funzione debounce.

Se il pulsante è premuto e al "giro del ciclo" precedente non lo era, significa che lo stato del bottone è appena cambiato e nel caso il bottone in questione sia anche il numero acceso (numero presente in currentNumber), il punteggio del giocatore viene aumentato, viene stampato il nuovo punteggio sull'LCD tramite la funzione stampLCD() e dopo aver fatto un rumore con i cicalini (della durata della variabile delayValue, che di default abbiamo deciso di lasciare a 20 millisecondi.
Alla riga 22 viene quindi chiamata la funzione setNextButton (spiegata in seguito).
A riga 25 viene salvato il valore attuale come precedente. Dopo aver calcolato il tempo passato e salvato nella variabile timerGame a riga 27, controlliamo che il tempo passato non sia superiore a il tempo a disposizione per il gioco. Nel caso così non fosse, vengono eseguite gli ultimi comandi descritti precedentemente prima che la funzione si chiuda.

```
13 void setNextButton()
14 {
15   while (true)
16   {
17     currentNumber = random(0, dimensions);
18
19     if (currentNumber != lastNumber)
20     {
21       break;
22     }
23   }
24   lastNumber = currentNumber;
25
26   for (int i = 0; i < dimensions; i++)
27   {
28     digitalWrite(ledPins[i], LOW);
29   }
30   digitalWrite(ledPins[currentNumber], HIGH);
31   return;
32 }
```

Questa funzione si occupa di definire il prossimo numero da premere nella sequenza. Esso deve essere diverso da quello attuale e deve essere casuale (da 0 fino al massimo consentito dai bottoni utilizzati, come mostrato nella riga 17). Se il numero risulta uguale al precedente, non esce dal ciclo while e crea un nuovo valore casuale, continuando così fino a quando finalmente crea un numero diverso dall'ultimo numero. Vengono spenti tutti i led in modo da essere sicuri che non ci siano led inutili accesi e per finire accendiamo quello che siamo sicuri che l'utente dovrà andare a premere.

LCD

Per la stampa di alcuni dati abbiamo utilizzato un lcd. Qui di seguito sarà mostrato un piccolo esempio.

In queste poche righe viene mostrato come creare un oggetto "LiquidCrystal" (link libreria **sitografia**).

```
2 #include <Wire.h>
3 #include <LiquidCrystal_I2C.h>
4
5 //primi dati sono inizializzazione delle porte A4 e A5
6 //gli altri si riferiscono alle righe e alle colonne
7 //altri codici per lcd 0x20, 0x27
8 LiquidCrystal_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3,POSITIVE);
```

Mentre qui come inzializzare l'oggetto e farlo funzionare facendo sì che non stampi stringhe casuali appena fa l'accensione.

```
12 void setup() {
13   Wire.begin();
14   // inizializzazione dell'oggetto lcd_I2C
15   lcd.begin(20,4);
16   lcd.clear();
17   //accensione retro illuminazione
18   lcd.backlight();
19 }
```

All'inizio del loop si pulisce sempre lo schermo per poi stampare sempre il nuovo testo senza sovrapporvi qualcosa.

```
21 void loop(){
22   lcd.clear();
23
24   testoBase();
```

In questo metodo viene stampata una semplice tempo e poi punti.

```
39 void testoBase(){
40   lcd.setCursor(0,0);
41   lcd.print("Tempo: ");
42
43   lcd.setCursor(0,2);
44   lcd.print("Punti: ");
45 }
```

Categoria 1:

In questa prima categoria vengono raggruppate tutte le modalità che richiedono che i pulsanti vengano accesi sequenzialmente e che rimangano accesi fino a quando non vengono premuti. Lo scopo è ottenere il maggior numero di punti nel tempo definito dalla modalità. Questa funzione necessita 3 parametri, il primo è un intero e riguarda il numero di pulsanti utilizzati (utilizzato per scegliere con facilità tra senior e junior, non escludendo la possibilità di utilizzare per esempio ancora meno pulsanti per una eventuale modalità baby). Il secondo parametro è un long, vista la grandezza che si necessita, e riguarda il tempo in millisecondi a disposizione per premere i pulsanti. L'ultimo parametro è un booleano e serve semplicemente per indicare (tramite il valore passato 'true') se la modalità è a staffetta (unica per ora implementata in questo modo: modalità 8), dove il tempo a disposizione è per giocatore e 4 giocatori si intercambiano tra loro per arrivare a fare più punti possibili.
Modalità 1, Modalità 2, Modalità 8, Modalità 15, Modalità 16, Modalità 19

Categoria 2:

In questa seconda categoria sono inserite tutte le modalità dove il numero di pulsanti è definito (dalla modalità) e bisogna cercare di premere i pulsanti che si accendono sequenzialmente nel minor tempo possibile. Utilizza 2 parametri, il primo è un intero e indica il numero di pulsanti utilizzati, il secondo è il numero di pulsanti che bisogna premere correttamente ed è anch'esso un intero.
Modalità 3, Modalità 17, Modalità 18

Categoria 3:

La terza categoria raggruppa le modalità dove viene richiesto che i pulsanti si accendano per un secondo per poi spegnersi. Premere i pulsanti sbagliati o non premerli in tempo comporta una diminuzione del tempo a disposizione per reagire e premere i pulsanti. Dispone di 3 parametri. Il primo è un intero ed è il numero di pulsanti utilizzati, il secondo (intero) il numero di pulsanti che bisogna premere per terminare la modalità, mentre l'ultimo è un booleano che indica una specializzazione della categoria: i pulsanti utilizzati sono solo i 4 angoli (pulsanti 0, 1, 8, 9) e in caso di tempo di reazione superato il tempo a disposizione per reagire non diminuisce.
Modalità 4, Modalità 7, Modalità 11, Modalità 12, Modalità 13, Modalità 14.

Categorie non implementate

Queste categorie non sono state implementate poiché nonostante sembrasse funzionassero da sole, una volte integrate con l'hub sono spuntati vari problemi che non siamo riusciti a risolvere in tempo, costringendoci a rimuoverle per non intaccare il prodotto.

Categoria 4:

Modalità 21, Modalità 22

Categoria 5:

Modalità 23

Categoria 6:

Modalità 9

Collegamento Arduino → PHP → DB

Per collegare Arduino con il DB abbiamo sfruttato PHP creando i due file "update_modality.php" e "update_score.php" che salvano i dati (modalità e punteggio) nel database.
I valori dei dati vengono presi tramite l'array superglobale $_GET:

```
3   $modality = $_GET['modality'];
3   $score = $_GET['score'];
```

Dopodiché vengono inviati al database tramite delle query:

```
21  $sql = "UPDATE `utilizza` SET `Mod_IDModalita` = ".$modality." WHERE `nr_partita` = ".$row['nr_partita'];
21  $sql = "UPDATE `utilizza` SET `punteggio` = ".$score." WHERE `nr_partita` = ".$row['nr_partita'];
```

Per quanto riguarda il form e quindi anche l'accesso del giocatore con i propri dati abbiamo usato PHP per creare una pagina web (index.php) che prende questi dati e li inserisce nel database:

```
36      if (!$error) {
37
38          $query = "SELECT `Gio_ID`, `Gio_Email` FROM `giocatore` WHERE `Gio_Email`='$email'";
39          $result = $conn->query($query);
40          $row = $result->fetch_assoc();
41          $count = $result->num_rows;
42
43          if($count == 1) {
44              $_SESSION['user'] = $row['Gio_ID'];
45
46              $query = "INSERT INTO `utilizza` (`punteggio`, `Gio_IDGiocatore`, `Mod_IDModalita`) VALUES (0,".($row['Gio_ID'])."), 24)";
47              $conn->query($query);
48
49              $query = "SELECT `nr_partita` FROM `utilizza` ORDER BY `nr_partita` DESC LIMIT 1";
50              $result = $conn->query($query);
51              $row = $result->fetch_assoc();
52
53              $_SESSION['game'] = $row['nr_partita'];
```

In questo codice viene verificato che il tentato login non contenga degli errori (riga 36), e seleziona nel database il giocatore che corrisponde alla data email (riga 38).
Se il giocatore esiste (riga 43) viene creata una riga nella tabella 'utilizza' del database che corrisponde alla partita attuale assegnando, temporaneamente dei valori gestibili (Punteggio=0, Modalita=24).
Una volta effettuato il login, viene visualizzata la pagina 'Home.php' il cui CSS viene aggiornato ogni 5 secondi tramite una funzione JavaScript richiamante il file 'Update.php' che aggiorna il punteggio e la modalità che l'utente sta giocando:

```
63              setInterval(function() {
64                  xmlhttp = new XMLHttpRequest();
65                  xmlhttp.onreadystatechange = function() {
66                      if (this.readyState == 4 && this.status == 200) {
67                          document.getElementById("content").innerHTML = this.responseText;
68                      }
69                  };
70                  xmlhttp.open("GET","update.php");
71                  xmlhttp.send();
72
73              }, 5000)
```

Il metodo setInterval() permette il chiamo della data funzione ogni tot tempo (in questo caso 5000 millisecondi).
Bisogna comunque collegare il tutto dal lato Arduino:
Questo viene effettuato sfruttando una libreria trovata online chiamata 'Ethernet2' (https://github.com/adafruit/Ethernet2).
Le seguenti variabili permettono la connessione:

```
110 //variabili per collegare Arduino a DB SQL
111 byte mac[] = {0x90, 0xA2, 0xDA, 0x11, 0x1D, 0x55};
112 IPAddress ip(192, 168, 5, 16);
113 IPAddress server(192, 168, 5, 17);
114 EthernetClient client;
115 int conn;
```

L'array mac contiene il MAC Address dell'Arduino Mega mentre la variabile ip contiene l'indirizzo IPv4 da assegnare all'Arduino.
La variabile server contiene l'indirizzo statico da assegnare alla macchina collegata e la variabile conn è la connessione che viene creata.
Tramite questi comandi, Arduino prova a connettersi al server Apache tramite la sua porta che nel nostro caso è la 3306.

```
141   Ethernet.begin(mac, ip);
142   conn = client.connect(server, 3306);
```

Infine, se la connessione è avvenuta con successo, viene passato al file 'update_score.php' il punteggio effettuato dal giocatore tramite una richiesta HTTP:

```
250     if (conn) {
251       Serial.println("connesso");
252       client.println("GET /reactiongame/update_score.php?score=" + String(scores[1]) + " HTTP/1.1");
253       client.println("Host: 192.168.5.17");
254       client.println("Connection: close");
255       client.println();
256     }
```

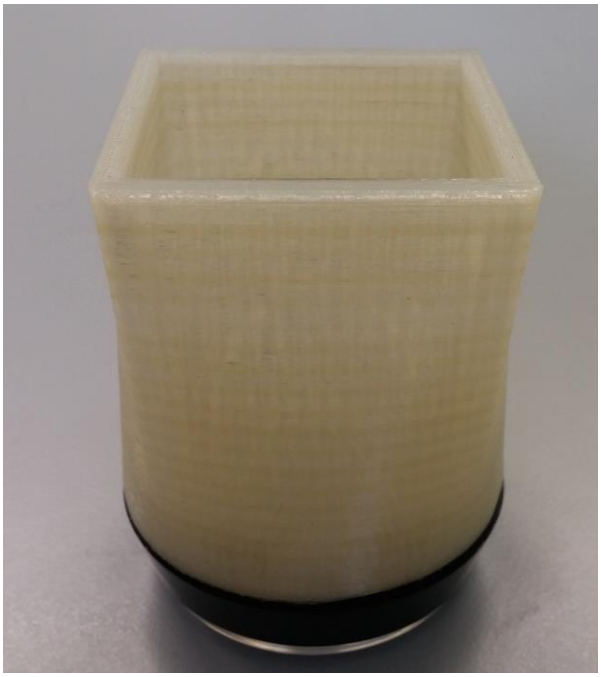
Parte fisica

Prima di cominciare a fare del codice abbiamo definito quali pin dovessero essere attribuiti ai LED e ai bottoni. Per i bottoni abbiamo deciso di utilizzare i pin pari, che partono dal numero 22 fino al numero 44, mentre per i LED i pin dispari che partono dal 23 e finiscono al 45. Poi sono stati usati i pin 20 (SDA), 21 (SCL) e i pin per il 5[V] e la messa terra (GDN), in aggiunta è stato utilizzato il pin 6 per i cicalini.
Quando ci sono arrivati i pezzi per la batak grande il professore barchi ha unito i pezzi fisici per poi lasciare a noi ill posto. Erik e Dymuan hanno ideato e preparato un foglio con le misure per avere una piccola idea della lunghezza totale del filo necessario per la batak perché non c'era a scuola. Dopo aver ordinato circa 90 [m] di filo traccia con diversi colori. In aggiunta abbiamo ordinato altre cose, tipo dei piedini con una grandezza differente da quella pensata inzialmente per il collegamento dei LED e shield ethetnet di arduino.

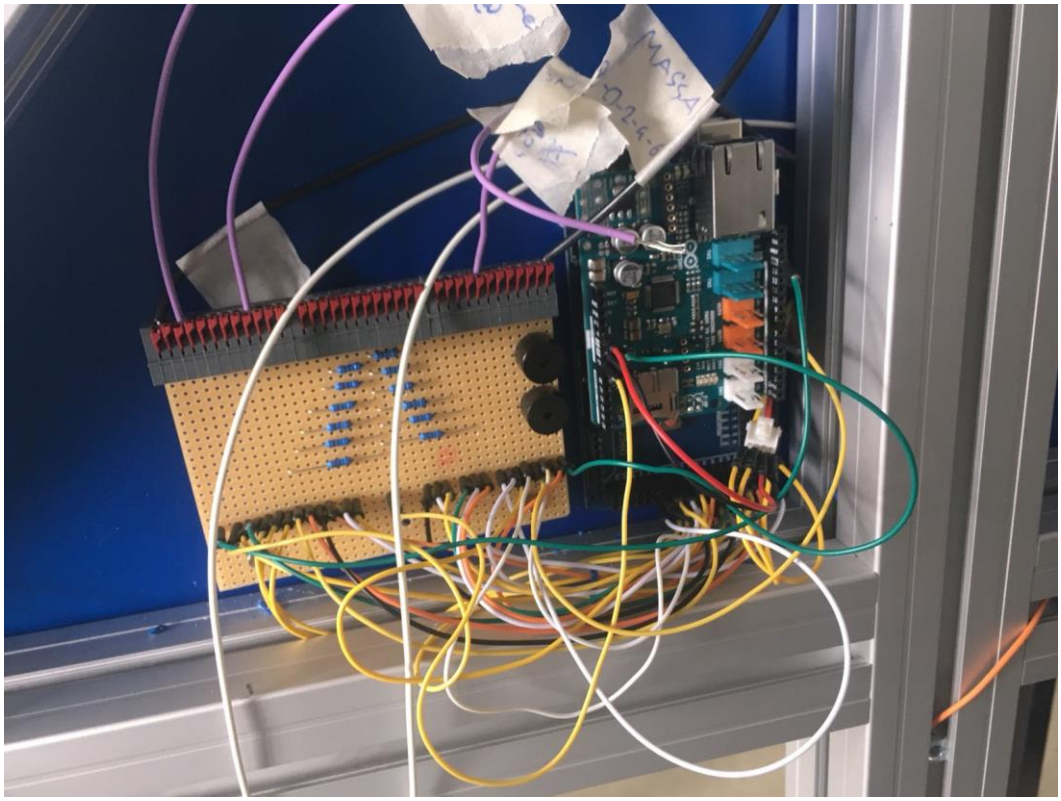
Per la costruzione Nadir, Luca e Dyuman hanno fatto passare i cavi in fori fatti dal professore Barchi, poi hanno attaccato la maggior parte dei bottoni attaccatti alla batak con dei piedini costruiti con la stampante 3D, sono stati ideati dal signor Barchi.

Per una differente idea la batak grande non possiede le resistenze collegate in pull-down ma in pull-up perché i collegamenti a massa sono tutti comuni per cui abbiamo dovuto mettere i collegamenti in pull-up.

Fisico del Bottone per attacarlo alla batak.

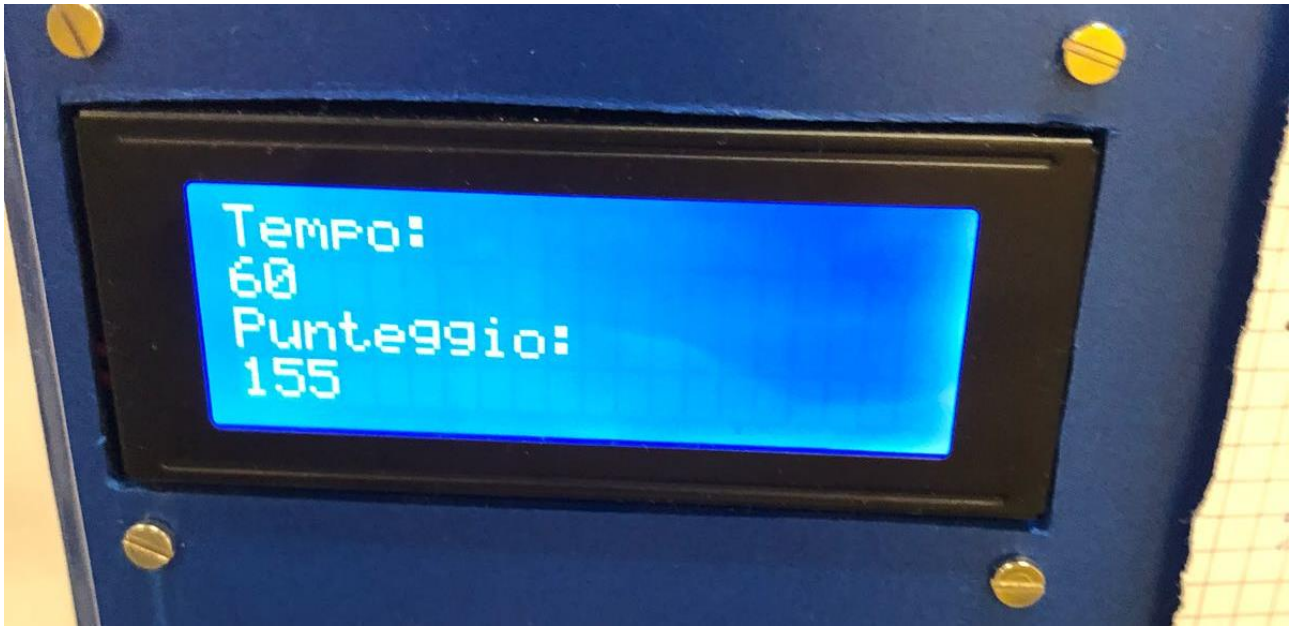


Mentre Erik ideava il circuito da saldare.
Saldatura e completamento dei collegamenti tra bottoni, led al circuito sono stati completati da Dyuman, Erik e Luca durante il fine settimana.
In questa foto viene mostrato il circuito costruito da erik con di fianco l'ArduinoMega 2560 con sopra lo shield.



Il risultato finale è così:

In questa foto si vede come funziona ldc sulla batak



Questo è uno schema elettrico di un bottone-LED

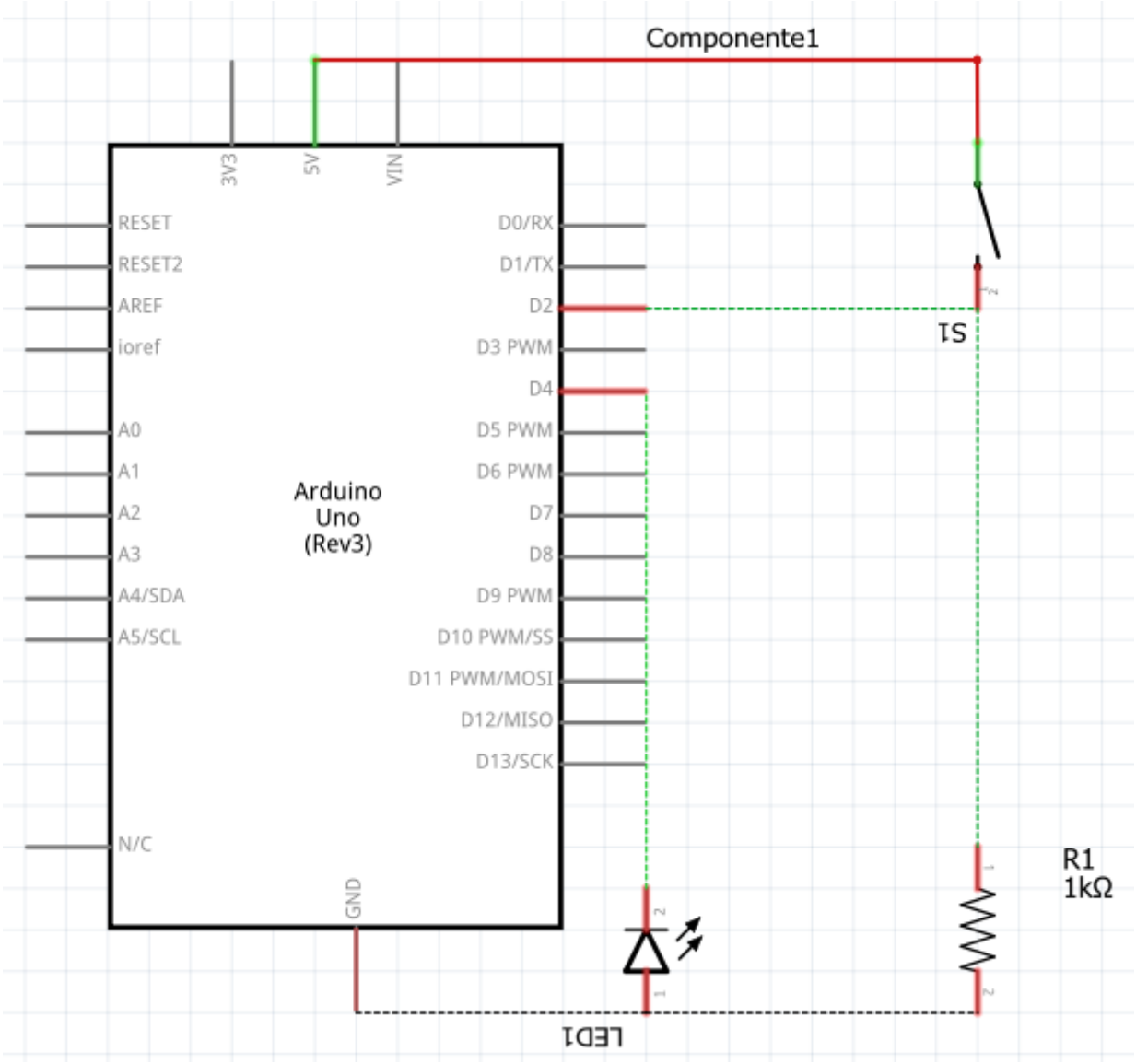


Figura 1: Schema elettrico fatto in fritzing per il collegamento di un singolo bottone-LED

Questo è lo schema elettrico dei bottoni con le rispettive resistenze.

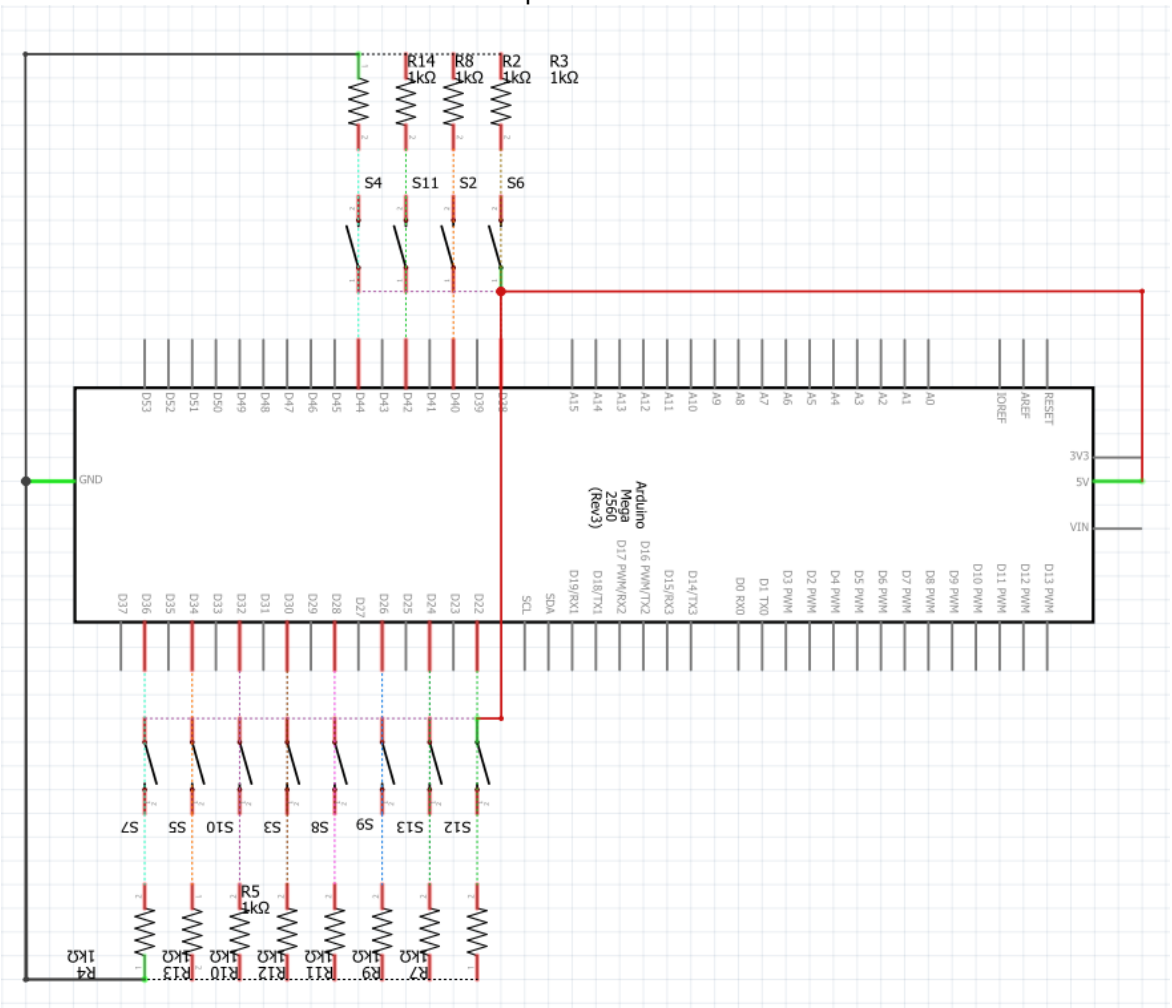


Figura 2: Schema elettrico dei bottoni

Questo schema invece è per i bottoni, sono stati separati per avere una chiarezza migliore e per un fattore di collegamenti.

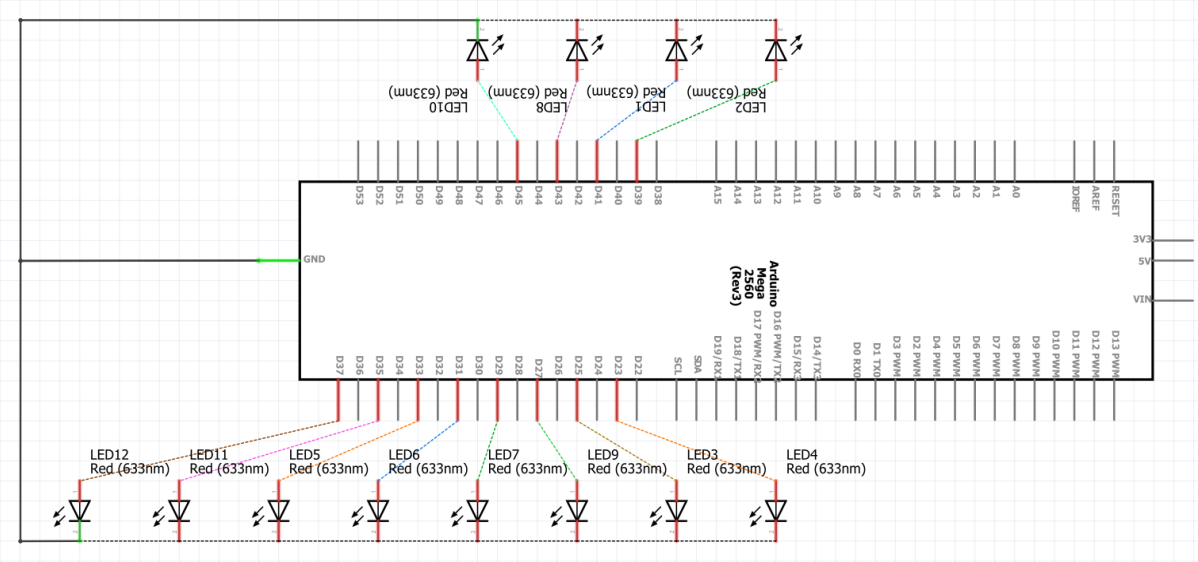


Figura 3: Schema elettrico LED

4 Test

Test Case: **Riferimento**:	TC-001 REQ-02	**Nome:**	Controllo funzionamento bottone
Descrizione:	Ottenimento dato da parte del bottone per vedere il suo funzionamento.		
Prerequisiti:	Possedere I bottoni-LED necessari per la batak ed un paio di riserva, piccolo progamma che mostra i dati che fa passare il bottone.		

Procedura:	1. Prendere il bottone-LED e distinguere quali sono i pin del LED e quali del bottone.
	2. Fatto questo prendere dei morsetti a coccodrillo (uno rosso ed uno nero se possibile) ed attacarli ai piedini del bottone.
	3. Attaccare al coccodrillo del GND (della massa) una resistenza da 1 [kΩ] e tra loro mettere un filo che si collega ad un pin (digitale) di Arduino.
	4. Poi mettere la resistenza nel pin della massa dell'Arduino.
	5. Mentre per il coccodrillo del voltaggio bisogna collegargli un filo che si mette nel pin del +5[V] di Arduino.
	6. Aprire il programma di arduino e segeguire I seguenti comandi:
	`void setup()`
	`{`
	`  //Funzionamento dei pin`
	`  pinMode(13, INPUT);`
	
	`  Serial.begin(9600);`
	`}`
	
	`void loop()`
	`{`
	`  Serial.println(digitalRead(13));`
	`}`		

Risultati attesi:	Nel pannello del Serial si dovrebbe vedere 1 quando viene premuto il bottone mentre 0 quando non viene premuto.		

Test Case:	TC-002	**Nome:**	Controllo funzionamento led
Riferimento:	REQ-02		
Descrizione:	LED si accende		
Prerequisiti:	Possedere I bottoni-LED necessari per la batak ed un paio di riserva, piccolo progamma che fa accendere il LED.		

Procedura:	1. Prendere il bottone-LED e distinguere quali sono i pin del LED e quali del bottone.
	2. Fatto questo prendere dei morsetti a coccodrillo (uno rosso ed uno nero se possibile) ed attacarli ai piedini del LED (rosso per il più del LED).
	3. Attaccare al coccodrillo del GND (della massa) attacarlo ad un filo che si collega al pin di Arduino che e collegato a massa.
	4. Mentreal coccodrillo del voltaggio collegargli un filo che si mette in un pin (digitale) di Arduino.
	5. Aprire il programma di arduino e segeguire I seguenti comandi:
	`void setup() {`
	`  pinMode(9, OUTPUT);`
	`}`
	`void loop() {`
	`  digitalWrite(9, HIGH);`
	`}`		

Risultati attesi:	Il LED del bottone si illumina.		

Test Case:	TC-003	**Nome:**	Categoria uno
Riferimento:	REQ-03		
Descrizione:	Primo gruppo di modalità (tutte simili tra loro), i pulsanti si accendono sequenzialmente e rimangono accesi fintanto che l'utente/giocatore non preme il bottone che si illumina. Lo scopo è eseguire il maggior numero di punti nel tempo prestabilito.		
Prerequisiti:	Funzoinamento della batak machine e delle rispettive modalità: 1, 2, 3, 4, 5, 6.		
Procedura:	1. Premere il bottone "e/@" nella batak machine e selezionare una delle seguenti modalità: 1, 2, 3, 4, 5, 6. 2. Appena il gioco parte aspettare che un bottone si accende e che rimanga accesso finché il giocatore non lo preme, mentre il tempo continua a scorrere finché non scade. Nel caso venisse premuto un bottone che non si illumina non viene cambiato il bottone e non aumenta il punteggio.		
Risultati attesi:	Finito il tempo stabilitò per la modalità selezionata viene mostrato il tempo passato e il punteggio eseguito.		

Test Case: **Riferimento**:	TC-004 REQ-03	**Nome:**	Categoria due
Descrizione:	Secondo gruppo di modalità (tutte simili tra loro), premere un certo numero di pulsanti nel minor tempo possibile.		
Prerequisiti:	Funzoinamento della batak machine e delle rispettive modalità: 7, 8, 9		
Procedura:	1. Premere il bottone "e/@" nella batak machine e selezionare una delle seguenti modalità: 7, 8, 9. 2. Appena il gioco parte un bottone si accende e rimane accesso finché il giocatore non lo preme, mentre il tempo avanza all'infinito. Il gioco termina quando il giocatore preme il numero necessario di bottoni. Se l'utente sbaglia pulsante il gioco non avanza.		
Risultati attesi:	Dopo aver premuto I pulsanti stabiliti dalla modalità selezionata viene mostrato il tempo passato e il punteggio ottenuto.		

4.1 Mancanze/limitazioni conosciute

Per mancanza di tempo, non siamo riusciti ad implementare delle modalità di gioco e a risolvere correttamente alcuni problemi scaturiti da modalità già create. Inoltre, la parte di collegamento arduino → php → database, nonostante il tempo dedicatogli, risulta non utilizzabile, poiché funziona correttamente ma non siamo riusciti a risolvere il problema della connessione arduino → php, che funzionava solamente la prima volta mentre doveva essere usata almeno 2 volte per gioco.
La maggior parte dei problemi erano causati dai cavi dei bottoni che essendo dei "cavi treccia", quindi non tanti fili interni intrecciati e necessitano delle particolari scarpette per effettuare dei contatti con il piedino del bottone. I suddetti cavi appunto uscivano di continuo dalla scarpetta per alcuni errori da parte della manodopera. Mentre per la Batak finale abbiamo avuto dei problemi causati dall'arrivo dei pezzi in ritardo per cui abbiamo dovuto correre per la costruzione e fermarci il fine settimana (i giorni: 4.03.2018 e il 5.03.2018) per completare adeguatamente le saldature e i collegamenti dei cavi al circuito costruito, con un paio di test.

5 Consuntivo

6 Conclusioni

Il prodotto esiste già in molte forme, ma oltre che costare parecchio ha il grosso limite di avere poche modalità di gioco. Il nostro prodotto invece permette di tenere i costi al minimo e soprattutto di avere la possibilità di crescere e adattarsi alle richieste dei clienti. L'unico limite è la fantasia per quanto riguarda i giochi che è possibile creare all'interno della nostra BATAK machine. Inoltre è perfetta sia per fiere, che grazie alla classifica farà risaltare il lato competitivo del gioco, spingendo sempre più gente a giocare.

6.1 Sviluppi futuri

Nuove modalità, risoluzione dei problemi e collegamento sono tutte cose risolvibili con un po' più di tempo a disposizione per sbatterci la testa. Per migliorare il progetto (dal lato fisico/meccanico) è cambiare i bottoni, cioè non utilizzare dei bottoni meccanici ma magnatici per una fattore di logoramento (si rivinano più lentamente, è difficile spaccarli a differenza degli altri) ed utilizzo in oltre per i LED ci è stato consigliato di utilizzare delle particolari strisce di LED in modo che al suo interno illuminano bene la cassa del bottone.

6.2 Considerazioni personali

Questo progetto è stato la nostra prima esperienza come lavoro a 4 e di conseguenza la nostra organizzazione non è partita nei migliori dei modi. Alla fine, fino all'ultimo mese più che lavorare in 4 abbiamo lavorato come 2 coppie (Bulloni-Rausa, Barlozzo-Stalliviere) che si aggiornavano sul loro procedimento e che nell'ultimo mese hanno messo insieme tutti i pezzi. Una volta completata, la situazione invece è diventata più amalgamata

7 Bibliografia

7.1 Sitografia

https://www.arduino.cc/en/Guide/HomePage, Arduino.
https://arduino-info.wikispaces.com/LCD-Blue-I2C, Funzionamento lcd su Arduino.
https://learn.adafruit.com/adafruit-led-backpack/0-dot-56-seven-segment-backpack, Ricerca funzionamento back pack per i sette segmenti.
https://www.arduino.cc/en/Reference/Wire, Collegamento lcd ad Arduino con i rispettivi pin.
https://www.adafruit.com/product/1192, Ricerca immagini per bottone Batak grande.
https://www.adafruit.com/product/3429, Ricerca immagini per bottone Batak prova.

8 Allegati

Allegato A: I3_Diari_ReactionGame.pdf
Allegato B: Interfacce (consegnate separatamente).