CSGO-Inspired Airsoft Bomb

Introducere

De câțiva ani, airsoft-ul și paintball-ul s-au dezvoltat tehnologic prin adăugarea unor dotări speciale echipamentelor și terenurilor de joc, aducând o notă de realism în experiența de joc. Inspirându-se din strategiile și scenariile din jocul CS:GO, comunitatea a adoptat diverse tactici pentru a aduce o notă de realism în meciuri.

În acest context, proiectul meu se axează pe simularea unei bombe asemănătoare celei din CS:GO, cu scopul de a recrea scenariul de joc “Bomb Defusal”.

Spre deosebire de modelul de bombă clasic, în cadrul proiectului îmi propun ca aceasta să conțină mai multe scenarii de joc pentru a oferi o experiență mai captivantă și diversificată.

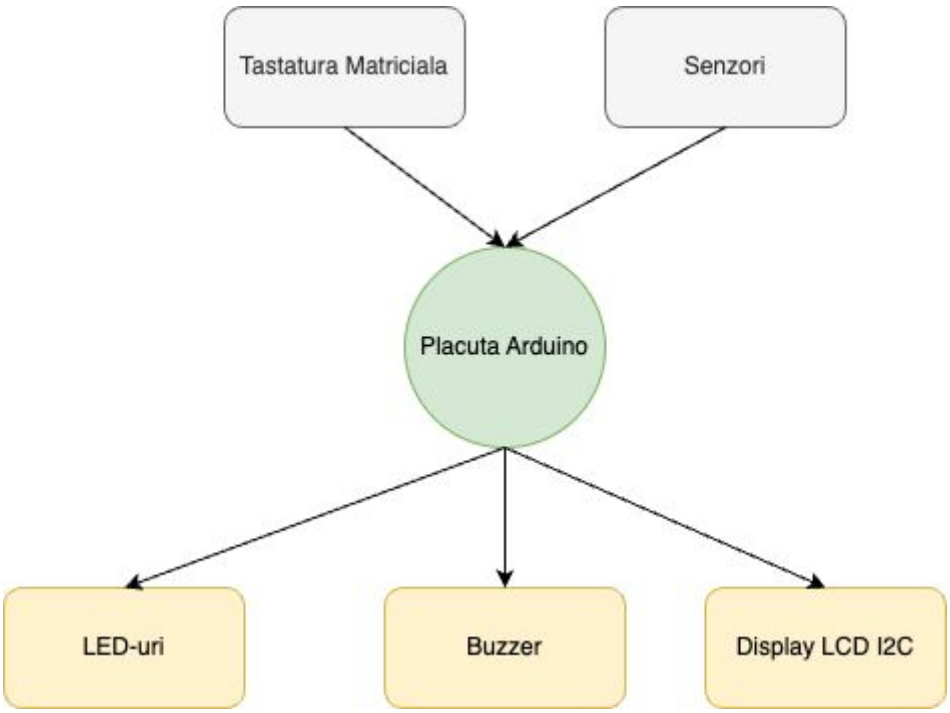
Descriere generală

Inițial, pe display-ul afișat sunt prezentate cele două scenarii de joc, iar alegerea se face prin intermediul tastaturii matriciale.

În primul scenariu de joc, cel clasic, utilizatorii trebuie să introducă codul de amorsare și timpul folosind tastatura. Pe măsură ce timer-ul scade (afișat pe display-ul I2C), buzzer-ul și LED-urile indică fiecare secundă care trece. Atunci când codul de dezamorsare este introdus greșit, timer-ul scade mai rapid, iar sunetul și lumina emise de buzzer și LED-uri devin și ele mai intense. Cu cât sunt introduse mai multe coduri greșite, cu atât crește ritmul. În final, dacă timpul se scurge complet, bomba explodează, semnalizată printr-un sunet specific din buzzer.

Al doilea scenariu este similar din punct de vedere tehnic. Diferența majoră constă în dezamorsarea bombei, care este realizată prin introducerea unor chei specifice. Pe măsură ce sunt introduse cheile, se captează prezența lor prin intermediul unor senzori, iar timer-ul scade din ce în ce mai rapid. Dezamorsarea are loc doar atunci când toate cheile sunt introduse corect în bombă.

Schema high-level:



Hardware Design

Lista de piese

- arduino Mega 2560;
- 2 senzori de culoare TCS230;
- 4×4 Matrix Keypad Adafruit;
- Display LCD I2C;
- led-uri;
- buzzer;
- rezistente;
- breadboard;
- switch

Elemente aditionale

- geanta din plastic pentru unelte
- chei
- carcasa din plexiglas
- elemente de decor

Schema electrica

Asamblarea componentelor

Mai jos au fost legate componentele ce tin de timer, buzzer si led-uri, care alcatuiesc modul de joc 1. Led-urile au fost legate la pinii digitali 23, 25 si buzzer-ul la pinul 27, pentru a fi mai aproape de breadboard. S-au adaugat niste rezistente leduirlor pentru a ma asigura ca nu se ard. Display-ul LCD I2C este conectat la pinii SCL, SDA, 5V de pe placuta.

Senzorii de culoare au fiecare cate 4 pini care trebuie conectati la pini PWM, un pin pentru output si sunt alimentati cu 5V. Am ales pinii 5 si 11 pentru output.

Componentele asamblate:

Software Design

Repo github: https://github.com/dbianca0907/CSGO-Inspired-Airsoft-Bomb.git
Laboratoare folosite: Lab0.GPIO, Lab3.Timere.PWM, Lab6.I2C

Organizarea codului

Asa cum se poate observa si in repo-ul de github, codul este organizat intr-un fisier care contine toate functiile programului si un utils.h care contine toate librariile, pinii, si valorile pentru calibrare folosite.

In functia de setup(), setez modul pinilor pentru led-uri, buzzer si senzori. De asemenea, initializez display-ul LCD si setez afisajul de inceput:

```
void setup() {
  Serial.begin(9600);
  // setup led-uri si buzzer
```

```
  lcd.init();
  lcd.backlight();
  pinMode(led_pin1, OUTPUT);
  pinMode(led_pin2, OUTPUT);
  pinMode(buzzer, OUTPUT);
  //setup senzori
  pinMode(S01, OUTPUT);
  pinMode(S11, OUTPUT);
  pinMode(S21, OUTPUT);
  pinMode(S31, OUTPUT);
  pinMode(output1, INPUT);
  pinMode(S02, OUTPUT);
  pinMode(S12, OUTPUT);
  pinMode(S22, OUTPUT);
  pinMode(S32, OUTPUT);
  pinMode(output2, INPUT);
  digitalWrite(led_pin2, HIGH);
  digitalWrite(S01, HIGH);
  digitalWrite(S11, LOW);
  digitalWrite(S02, HIGH);
  digitalWrite(S12, HIGH);

  displayInitialScreen();
}
```

In loop(), initial se slecteaza modul de joaca, iar in cazul in care se apasa o tasta diferita de 1 sua 2 se afiseaza un mesaj de eroare, iar buzzerul semnaleaza printr-un sunet specific.

```
void loop() {
  char key = customKeypad.getKey();
  if (key) {
    beep(500);
    blink_led_green();
    if (key == '1') {
      game_mode_1();
    } else if (key == '2') {
      game_mode_2();
    } else {
      displayErrorMessage();
      lcd.setCursor(0, 1);
      lcd.print("PRESS 1 OR 2");
      delay(2000);
      displayInitialScreen();
    }
  }
}
```

Programarea functionalitatilor de baza ale bombei

Asa cum s-a explicat anterior, dupa ce utilizatorul selecteaza modul de joc dorit, se apeleaza functiile specifice acestui mod. Indiferent de modul de joc ales (fie ca este vorba despre un mod de joc clasic sau unul bazat pe chei), utilizatorul are responsabilitatea de a seta un interval de timp dupa care bomba va fi detonata.

In cazul modului de joc 1, in care jucatorul interactioneaza doar folosind tastatura, acesta trebuie sa introduca codul de amorsare al bombei. Dupa setarea timpului si introducerea codului, utilizatorul are optiunea de a rescrie codul sau timpul, oferind astfel flexibilitate in configurarea bombei.

Biblioteci Folosite

Pentru a implementa aceasta functionalitate, m-am folosit de functiile de baza ale urmatoarelor librarii:

- “LiquidCrystal_I2C.h” pentru gestionarea afisajului LCD.

- “Keypad.h” pentru gestionarea tastaturii matriciale.

Contorizarea Timpului

Pentru a implementa functionalitatea de contorizare a timpului ramas pana la detonare, am utilizat functia millis(). Aceasta permite verificarea continua a timpului ramas si, concomitent, verificarea corectitudinii codului introdus de utilizator.

Implementarea Specifica Modulului de Joc 1

Dupa amorsarea bombei, incepe decrementarea timpului si verificarea codului introdus de jucator. Daca codul introdus este corect, se apeleaza functia de dezamorsare. In caz contrar, timpul de decrementare se reduce progresiv cu 300 ms, ajungand la un minim de 150 ms, pornind de la 1000 ms. Cand timpul expira complet, bomba declanseaza un sunet specific exploziei.

```
while (true) {
    // Verificăm și gestionăm decrementarea timpului
    decrementTime();
    if (minutes == 0 && seconds == 0) {
      // BOOM
      displayBombExplode();
    }
    char key = customKeypad.getKey();
    // Verificăm dacă s-au introdus date de la tastatură
    if (key) {
      beep(1000);
      blink_led_green_code();
      lcd.setCursor(6 + pos_code, 0);
      lcd.print("*");
      // Citim codul introdus
      input_code[pos_code] = key;
      if (pos_code == 3) {
        for (int i = 0 ; i < 4; i++) {
```

```
          if (input_code[i] != code[i]) {
            correct_code = false;
          }
        }
         // Dacă codul este corect, oprim timerul
        if (correct_code) {
          displayBombDefused();
          break; // Ieșim din buclă pentru a termina jocul
        } else {
          error_code_sound();
          // Dacă codul este incorect, scădem timpul mai rapid
          if (interval > 300) {
            interval -= 300;
          }
          // Daca timpul atinge un anumit prag, scade si mai rapid
          if (interval == 300) {
              interval = 150;
          }
          for (int i = 0; i < 4; i ++) {
            input_code[i] = '\0';
            lcd.setCursor(i + 6, 0);
            lcd.print(' ');
          }
          pos_code = 0;
        }
        correct_code = true;
      } else {
        pos_code++;
      }
    }
```

Acest cod exemplifica modul in care se gestioneaza decrementarea timpului si verificarea codului introdus de jucator. Implementarea este robusta, asigurandu-se ca timpul se scurge corespunzator si ca se iau masuri adecvate in functie de corectitudinea codului introdus.

Calibrarea Senzorilor

1. **Determinarea valorilor minime si maxime pentru RGB**

Pentru a calibra senzorii de culoare TCS230, trebuie să determinăm valorile minime și maxime pentru fiecare culoare (roșu, verde, albastru). Aceasta se realizează prin măsurarea răspunsului senzorului atunci când are în față un obiect alb (pentru valori minime) și un obiect negru (pentru valori maxime).

In funcția setup(), am configurat pinii senzorului pentru a stabili scala de frecvență:

```
digitalWrite(S0, HIGH);
digitalWrite(S1, LOW);
```

Pentru a citi valorile fiecarei culori, am setat pinii S2 și S3 pentru a selecta filtrul corespunzător:

Rosu:

```
digitalWrite(S2, LOW);
digitalWrite(S3, LOW);
pw = pulseIn(output, LOW);
```

Verde:

```
digitalWrite(S2, HIGH);
digitalWrite(S3, HIGH);
pw = pulseIn(output, LOW);
```

Albastru:

```
digitalWrite(S2, LOW);
digitalWrite(S3, HIGH);
pw = pulseIn(output, LOW);
```

Pentru fiecare culoare, am inregistrat valorile minime si maxime obtinute:

```
int redMin = 54;
int redMax = 247;
int blueMin = 58;
int blueMax = 318;
int greenMin = 69;
int greenMax = 353;
```

2. **Citirea valorilor de la Senzori si Maparea Lor**

Pentru fiecare valoare citita de la senzor, am utilizat functia map() pentru a converti valorile de frecventa intr-un interval de la 0 la 255, unde 255 reprezinta valoarea minima (corespunzand culorii albe), iar 0 reprezinta valoarea maxima (corespunzând culorii negre).

```
red = getRedPW(1);
redValue = map(red, redMin, redMax, 255, 0);
```

3. **Determinarea culorilor si validarea cheii introduse**

Mediul in care se afla senzorii este unul inchis, asa cum se poate observa si la reprezentarea hardware, deci nu este nevoie ca senzorii sa fie recalibrati.

In funcție de valorile redValue, greenValue și blueValue, senzorii pot detecta culoarea dominanta pentru a verifica dacă cheia corectă a fost introdusă.

```
int verifyKey1() {
  // Verificare cheie introdusa corect
  if (redValue > blueValue && redValue > greenValue
      && (redValue - blueValue) >= offsetSensor && (redValue - greenValue)
```

```
>= offsetSensor) {
    return 1;
  }
  // verificare cheie introdusa incorect
  if (blueValue > redValue && blueValue > greenValue
      && (blueValue - redValue) >= offsetSensor && (blueValue - greenValue)
>= offsetSensor) {
    return 2;
  }
  return 0;
}
```

Rezultate Obţinute

DEMO

Concluzii

Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.

Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

Export to PDF