UNIVERSIDADE
LUSÓFONA

Réplica da Bomba do CS:GO

Martim Monteiro: 22005557

Tiago Machado: 22306644

Arquiteturas Avançadas de Computadores | LEI | 10/01/2024

Índice

Lista de Figuras

Resumo

O projeto tem como objetivo criar uma réplica detalhada da bomba presente no jogo CS:GO (Counter-Strike: Global Offensive). Utilizará componentes como LCD, Keypad, Arduino, Switch de ligar/desligar, LED (verde e vermelho), Buzzer e possivelmente uma bateria para garantir a semelhança visual e funcional com a bomba do jogo.

Descrição do problema

A bomba no CS:GO pode ser armada através da introdução de um código, ao ser armada esta começa uma contagem decrescente com duração de 40 segundos. Durante este processo, a equipa adversaria é desafiada a desarmá-la, inserindo um código preciso antes que o tempo expire.

No jogo é utilizado um dispositivo que descobre o código da bomba para a desarmar. Devido à complexidade deste processo esta funcionalidade não será implementada, em vez disso, a reintrodução do código utilizado para a armar será suficiente para a desarmar. Este projeto propõe a criação de uma réplica em escala 1:1 da bomba do jogo. Além do aspeto técnico envolvido na reprodução dos componentes visuais, a integração de funcionalidades que repliquem o processo de desativação da bomba apresenta um desafio que complementa os conteúdos introduzidos ao longo do semestre.

Desenvolvimento

Material Utilizado

Na montagem desta réplica foram utilizados os seguintes componentes:

- 1 LCD (16x2)
- 1 Keypad
- 2 Arduino Uno R3
- 1 Switch
- 1 LED Verde
- 1 LED Vermelho
- 1 Piezo / Buzzer

Abordagem

No desenvolvimento da réplica foram utilizados dois Arduinos, desta maneira é possível conectar mais componentes ao nosso sistema, algo que, com apenas um Arduino, seria difícil devido ao número limitado de portas presentes em cada um.

Os Arduinos foram conectados entre si utilizado Serial.

O Arduino 1 é apenas reponsável por operar o Keypad e enviar as teclas premidas ao Arduino 2.

O Arduino 2 é responsável por operar todos os periféricos de output, neste caso, o LCD, o Piezo / Buzzer e os dois LED que indicam o estado atual do sistema. Ao receber um caracter do Arduino 1, o Arduino 2 trata de o mostar no LCD.

Durante toda a execução o Arduino 2 executa um timer que trata de fazer piscar os LED, neste caso o LED verde.

Após receber 7 caracteres do Arduino 1, estes são comparados com o código implementado (idêntico ao jogo, 7355608), o estado muda para arnado. Neste estado, o Arduino 2 pisca o led vermelho e utiliza o Buzzer para emitir “beeps” com maior frequência conforme a proximidade aos 40 segundos após o dispositivo ter sido armado. Após os 40 segundos, o dispositivo termina a execução para simular a detonação.

Implementação

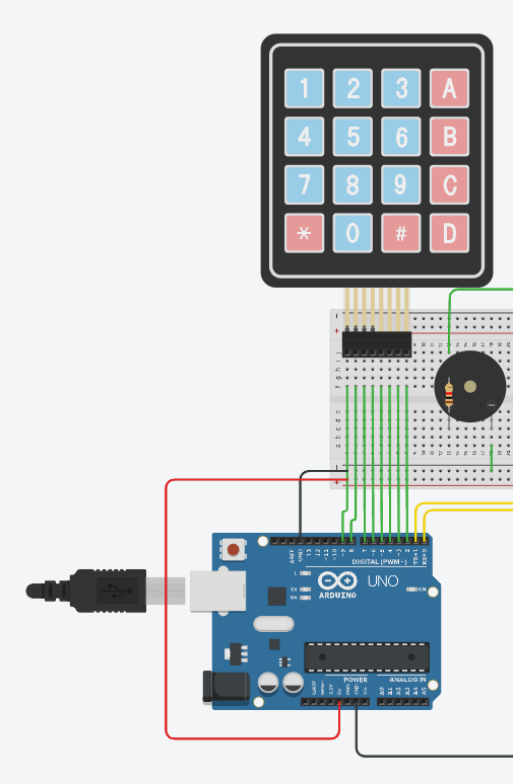

Figura 1 - Montagem do Arduino 1

Na **Figura 1 - Montagem do Arduino 1** encontra-se a instalação do Arduino 1 e a sua ligação ao Keypad.

Olhando para o código que irá ser executado no Arduino 1 conseguimos perceber melhor como é realizada a leitura dos numero pressionados:

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;

char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad kPad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
```

Neste excerto de código é possível reparar que foi utilizada a biblioteca “Keypad.h” do Arduino para auxiliar na comunicação com o Keypad. Foram declarados

o numero de linhas e de colunas a utilizar (neste caso 4x4, embora apenas 3x3 vão ser utilizadas), em seguida são definidos os carateres correspondentes a cada botão no Keypad, e por fim são definidos os pins a utilizar para comunicar com o Keypad e depois é inicializada a class Keypad e guardada na variável global kPad.

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  char key = kPad.getKey();

  if (key){
    Serial.write(key);
  }

}
```

No excerto acima temos a definição da função setup() e a definição da função loop().

Na função setup() é inicializado a comunicação Serial numa velocidade de 9600bps (bits per second). Tornando assim possível a comunicação através de Serial com outro Arduino (neste caso, com o Arduino 2).

Na função loop(), podemos ver a leitura do caracter a ser feita a partir do Keypad, e após verificar se o caracter lido não é vazio (caso nenhuma tecla tenha sido premida), o carater é enviado por Serial para o Arduino 2.

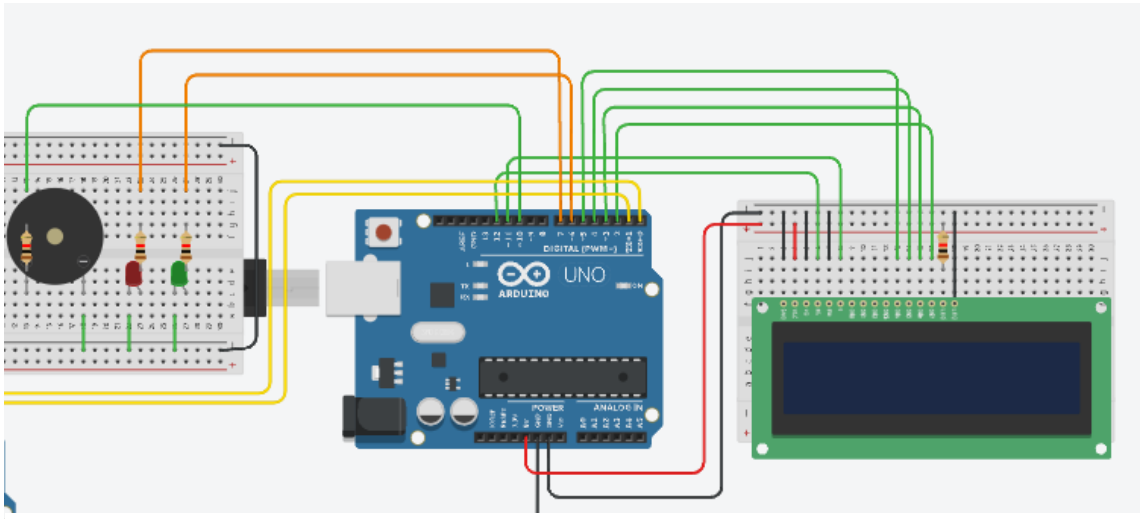

Figura 2 - Montagem do Arduino 2

Na **Figura 2** encontra-se a instalação completa do Arduino 2, sendo visível a sua ligação aos seus periféricos (Buzzer, LED Vermelho, LED Verde e LCD).

Para melhor entender o papel do Arduino 2 neste circuito é necessário olhar para o código do Arduino 2, sendo que este é onde a maioria do programa é executado.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

char inputCode[16];

bool activated = false;

int counter = 0;
int timerTicks = 0;
```

Devido á complexidade do código do Arduino 2, o código vai aparecer fragmentado para facilitar o seu esclarecimento.

No excerto acima temos a inclusão do header “LiquidCrystal.h” que facilitará a comunicação com o LCD. Logo de seguida inicializamos o nosso objeto LiquidCrystal e guardamo-lo na variável global lcd para simplificar o seu uso ao longo do programa.

Após a inicialização do LCD, é declarado um buffer de 16 carateres com o nome inputCode, este buffer é onde o código introduzido pelo o utilizador no Keypad, e transmitido para o Arduino 2, será armazenado.

A variável activated representa o estado do dispositivo, se este se encontra armado ou desarmado.

A variável timerTicks é incrementada com cada tick do timer1, que neste caso irá ser definido para gerar uma interrupção 20 vezes por segundo (20 hz, para facilitar o controlo de velocidade de alguns eventos com maior precisão). A variável counter está diretamente ligada à timerTicks, sendo que após o dispositivo ser armado, esta incrementará a cada segundo que passar para contar os 40 segundos de tempo de detonação.

```
void setup()
{
  lcd.begin(16,2);
  lcd.clear();

  WriteString("");
  Serial.begin(9600);

  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

  StartTimer();
}
```

A função setup() no Arduino 2, trata de ativar o LCD e limpar o ecrã do mesmo, em seguida chama a função WriteString() (descrita mais à frente neste documento), que escrever no LCD o seu parâmetro separado por espaços e preenche os espaços que não forem preenchidos com '*'. Neste caso escreve " * * * * * * *", para ilustrar que o código é composto por 7 dígitos.

Em seguida os pins 6 e 7 são colocados em modo de OUTPUT, estes pins correspondem aos LED Verde e Vermelho.

E por fim é chamada a função StartTimer() responsável por inicializar o timer1 com uma frequência de 20hz.

```
void loop()
{
  while (!Serial.available()) {}

  char value = Serial.read();


  AddToInputCode(value);

  WriteString(inputCode);

  if(strcmp(inputCode, " 7 3 5 5 6 0 8") == 0){
    delay(100);
    WriteString("");
    inputCode[0] = '\0';

    if(activated) {
      activated = false;
      counter = 0;
    }else {
      activated = true;

      StartTimer();
    }
  }
}
```

A função loop() apresentada acima não possui uma grande complexidade embora o seu aspeto.

A função começa por esperar pela receção do carater enviado pelo Arduino 1, que só o envia se este for válido. Depois de receber o carater este é guardado na variável value, adicionado ao código pela função AddToInputCode() e este é consequentemente escrito no LCD pela funçãp WriteString().

Após a inserção de um digito no código, o código é comparado com o código hardcoded “ 7 3 5 5 6 0 8”, e caso estes sejam iguais, esperamos 100ms para o código aparecer no LCD e após isto limpamos o código e imprimimos o texto vazio (“ * * * * * * *”) no LCD, para que alguém sem o código não possa ler do LCD e proceder a desarmar/armar o dispositivo.

Em seguida verificamos se o dispositivo se encontra armado ou desarmado e trocamos o seu estado. Caso este se encontre desarmado, mudamo-lo para armado e chamamos a função StartTimer() para recomeçar o timer1 para ter maior precisão no primeiro segundo de contagem. Caso este se encontre armado, colocamo-lo no estado desarmado e mudamos o counter para 0 (zero) para que se este for novamente armado a contagem começar do 0 e não do tempo anterior.

```
void WriteString(char* string) {
  lcd.clear();
  for(int i = 2; i < 16 - strlen(string); i+=2) {
    lcd.setCursor(i, 0);
    lcd.print(" *");
  }


  lcd.setCursor(16 - strlen(string), 0);
  lcd.print(string);


}

void AddToInputCode(char inputChar) {
  char temp[] = " ";
  char charToString[2] = {inputChar};

  strcat(temp, charToString);

  strcat(inputCode, temp);

}
```

Em cima temos finalmente as definições das funções WriteString() e AddToInputCode().

A função WriteString() começar por preencher da terceira casa do LCD (índice=2) para a frente com '*' separados por espaços até deixar no fim da linha espaço suficiente para preencher com a string passada como parâmetro, separando os seus carateres com espaços (' ');

A função AddToInputCode() apenas adiciona o carater passado como parâmetro ao fim da variável inputCode colocando um espaço antes do carater para melhor legibilidade no LCD.

```
ISR(TIMER1_COMPA_vect){
 timerTicks++;

 if(activated) {
    counter = timerTicks / 20;

    int beepThreshold = 20 * max(0.1 + 0.9 * ((40-counter)/40.0), 0.15);

    if(timerTicks % beepThreshold == 0) {
      sei();
      digitalWrite(6, LOW);

      tone(10, 2500);

      digitalWrite(7, HIGH);
      delay(125);
      noTone(10);
      digitalWrite(7, LOW);
      cli();

      if(counter == 40) {
        exit(0);
      }
    }


  }else {
    digitalWrite(7, LOW);
    digitalWrite(6, HIGH);
    delay(300);
    digitalWrite(6, LOW);
  }


}
```

E por fim temos a função correspondente à interrupção do timer1 no vetor de interrupções.

Começamos por incremente o timerTicks sendo que este descreve o numero de vez que esta rotina foi executada.

Em seguida verificamos se o dispositivo se encontra armado ou não. Caso este se encontre armado incrementamos o counter a cada 20 timerTicks, após isso calculamos a "beepThreshold" que trata de saber quantas tempo esperar entre "beeps" (sendo que estes aceleram quando o counter se aproxima de 40 segundos).

A fórmula para calcular a "beepThreshold" foi retirada do leak do source code do jogo que ocorreu em 2017 para melhor representar o comportamento do jogo.

Caso este tick do timer seja um em que ocorre o "beep" apagamos o led verde para prevenir o caso de ambos estarem acesos quando o estado passar para armado. Em seguida chamamos a função tone() no pin 10 (pin onde está ligado o terminal positivo do Buzzer) com uma frequência de 2500hz (frequência que melhor representa o som do "beep" no jogo), acendemos o led vermelho, e após 125ms (duração do "beep") chamamos a função noTone() no pin 10 para terminar o som e desligamos o LED vermelho, sendo que este é suposto apenas piscar a cada "beep".

Se o counter chegar a 40s, então terminamos a execução do programa, visto que num cenário real a excução terminaria por fatores externos.

Caso o estado seja desarmado, desligamos o LED vermelho por prevenção e piscamos o led verde por 300ms.

Conclusões

Todos as funcionalidades determinadas foram implementadas com sucesso e após algumas dificuldades os erros foram também resolvidos.

Em seguida encontra-se a montagem inteira do projeto e infelizmente não foi possível obter os componentes para montar o projeto em forma física. Sendo então, a montagem apresentada, realizada no Tinker Cad

É possível então consular o circuito na plataforma do **TinkerCad.**

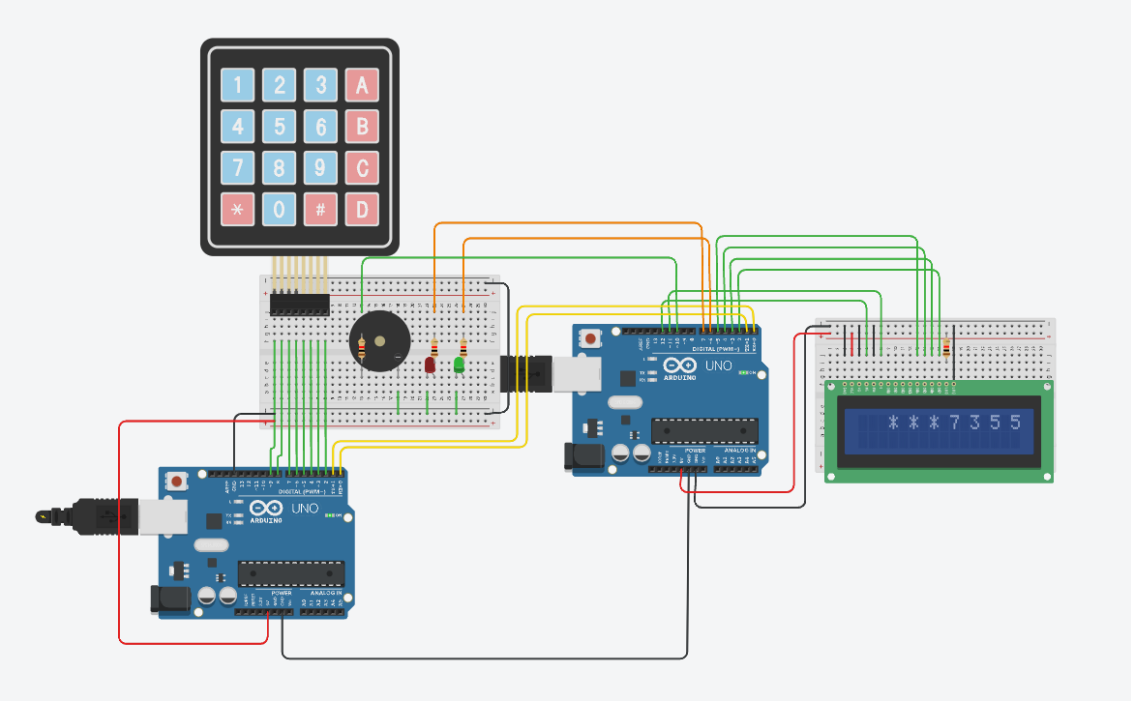

Figura 3- Montagem Completa

Caso o link acima não funcione:

https://www.tinkercad.com/things/9dtnJ1526gV-projeto-aac/editel?returnTo=%2Fdashboard%3Ftype%3Dcircuits%26collection%3Ddesigns&sharecode=G6vesrtRp21Yazg2uahzCaPuJH-iJuFt7-UxC10FzBw