Jogo Pong controlado pelo ESP32

Igor Da Silva Zagonel
IFMT - MT
igorzagone@gmail.com

***Resumo* – Nesse Relatório é descrito o primeiro projeto da matéria Micro controladores do curso de engenharia da computação do IFMT campus Cuiabá. Esse projeto consiste no desenvolvimento do primeiro jogo eletrônico criado chamado Pong na plataforma Unity com o objetivo de controlar o jogador através de um modulo de Joystick controlado pelo microcontrolador ESP32 na porta serial do computador.**

1. INTRODUÇÃO

No atual estágio de desenvolvimento tecnológico da microeletrônica é possível adaptar e traduzir problemas complexos do cotidiano para um meio eletrônico a partir de circuitos integrados, pois a cada dia esses componentes estão sendo aprimorados de modo que sua capacidade de processamento se tornou extremamente alta e o manuseio e a sua aplicação se tornaram mais simples. [1]
E com todo esse desenvolvimento os centros industriais se aproveitam desse benefício tecnológico para constantemente procurar métodos e soluções para aumentar a produtividade e o lucro da empresa, sendo que o meio da microeletrônica e da automação de maquinas que esses feitos são possíveis. [2].
Neste relatório será abordado o desenvolvimento do jogo Pong utilizando a plataforma Unity controlado por um modulo joystick através do microcontrolador Esp32.

2. A TEORIA

O desenvolvimento de jogos 3D ou 2D pode ser uma tarefa extremamente difícil dependendo das ferramentas que são escolhidas para dar continuidade na construção do jogo. Para evitar essa dificuldade inicial e ao decorrer do desenvolvimento foram criadas as Game Engine, como o Unity3D, que são ferramentas que auxiliam no desenvolvimento de jogos ao reunir diversas funcionalidades necessárias para a realização desde a programação à construção da dinâmica de jogo e aparência. [4]

O Unity 3D é uma plataforma que possui uma interface gráfica bastante simples e amigável que facilita na hora do desenvolvimento de jogos de diversos gêneros. Essa Game Engine é baseada no modelo mais moderno de arquitetura de games baseado em composição, nesse tipo de arquitetura os objetos no jogo são criados a partir de diversas funcionalidades que podem ser agregadas ou removidas de acordo com a vontade do programador.

O jogo Pong que foi proposto a ser desenvolvido consiste num jogo na onde o jogador controla uma raquete para cima e para baixo a fim de acertar uma bola que se movimenta para todos os lados, isso é, basicamente um jogo de ping Pong eletrônico. Os principais objetos utilizados para a construção do jogo Pong são: RigidBody2D e BoxCollider2D que são os responsáveis a dar vida e propriedades físicas do mundo real para o mundo virtual.
O RigidBody2D é o responsável por adicionar características físicas ao corpo criado, no caso do jogo criado foi adicionado esse objeto no jogador, no jogador do computador e na bola. Com essa característica RigidBody2D é possível adicionar a massa do objeto, o efeito que a gravidade terá nele e é possível dizer como ele se comportará ao acrescentar uma nova física a ele.

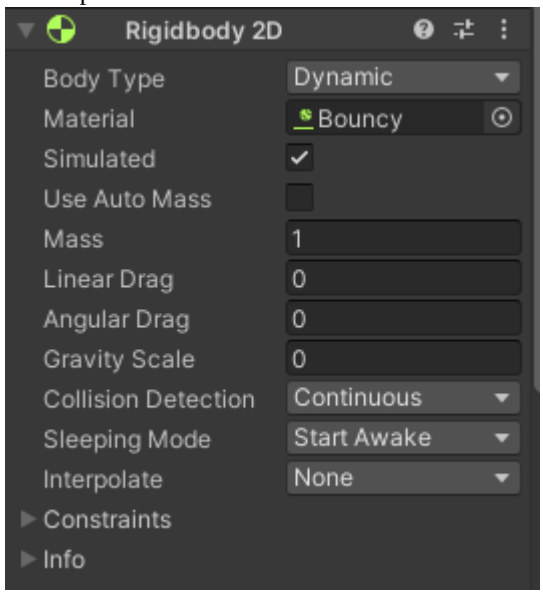

Fig. 1 – RigidBody2D na interface do Unity3D.

O BoxCollider2D é o responsável por dizer a área do objeto criado que será considerado como o corpo dele, isso é, quais partes do RigidBody2D será possível outro RigidBody2D colidir.

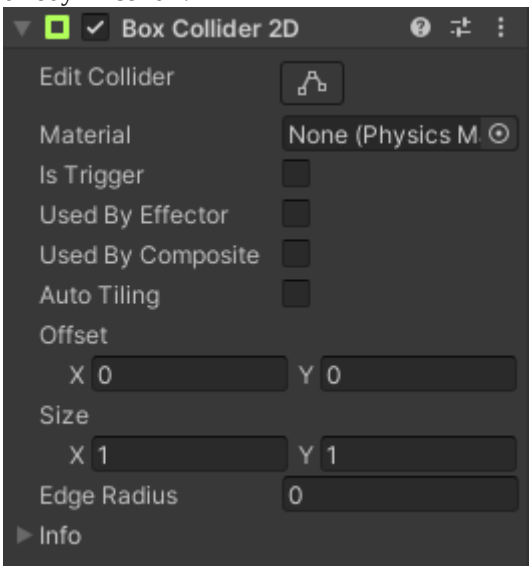

Fig. 2 – BoxCollider2D na interface do Unity3D.

O microcontrolador utilizado para controlar o jogador na aplicação desenvolvida foi o ESP32. Esse micro controlador é desenvolvido pela empresa Espressif e é um meio inovador para o desenvolvimento de aplicações de automações devido a sua versatilidade ao integrar bluetooth, Wifi, processador dual core e diversos outros sensores que vem embutido na placa. [3]

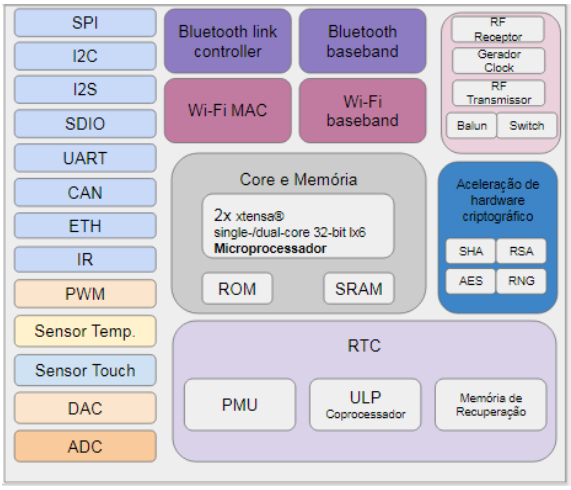

Fig. 3 – Diagrama de componente ESP32.

Em relação a Pinagem do ESP32 podemos encontrar 34 portas GPIO, 3 portas SPI, 2 portas I2S, 18 Canais ADC, 3 portas UART e 10 pinos de leitura PWM.

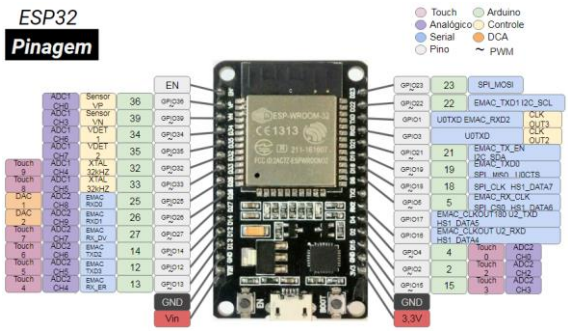

Fig. 4 – Pinagem do microcontrolador ESP32.

3. TABELAS E CIRCUITOS

As montagens de circuito e cenas do Unity nesse projeto consistem em implementações extremamente simples. Em relação ao microcontrolador utilizado e o modulo joystick foi usado a IDE do Arduino para realizar a programação necessária.

O programa desenvolvido consiste em reconhecer a entrada no conversor AD do ESP32 em que está ligado o modulo Joystick, pois esse modulo consiste internamente de 2 potenciômetros que são encarregados de variar o eixo X e o eixo Y.

Devido a especificidade do jogo Pong somente o eixo X desse modulo foi utilizado. Portanto, o modulo está ligado somente em 3 portas do microcontrolador, a porta 36 que é uma das portas do conversor AD e as outras 2 portas são o GND e o 5V do ESP32.

Fig. 5 – Modulo de Joystick.

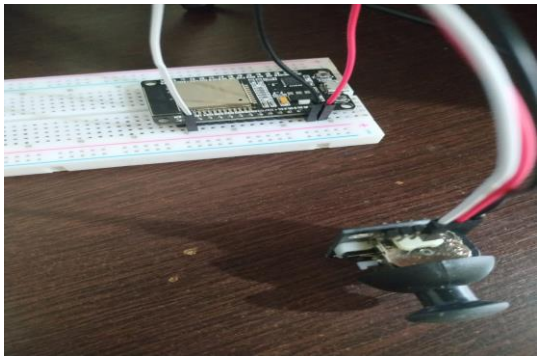

Fig. 6 – Ligação feita no ESP32.

Na IDE do Arduino é declarado uma variável que é a responsável por receber os valores lidos do conversor AD e a partir desse valor obtido é possível realizar uma lógica de programação para saber se o Joystick está sendo empurrado para cima ou para baixo.

```
const int PinVrx = 36;
int valorVrx = 0;
void setup() {
  pinMode(PinVrx, INPUT);
  Serial.begin(9600);
}
void loop() {
valorVrx = analogRead(PinVrx);
if (valorVrx < 2700) {
Serial.println("ui_up");}
if (valorVrx > 3100) {
Serial.println("ui_down");}
if(valorVrx > 2700 && valorVrx < 3100){
    Serial.println("Parado");}
}
```

Fig. 6 – Código do ESP32 para reconhecer o Modulo Joystick.

No código da figura 6 na função setup é declarado que o pino 36 será um input e é iniciado a comunicação serial entre o ESP32 e o computador. Na função loop é feito a leitura da porta analógica declarada e é armazenada na variável valorVrx, e a partir desse valor é feito 3 condicionais para checar se o analógico está para cima, para baixo ou parado dependendo do valor recebido do conversor AD.

A parte de programação e ligação dos componentes do ESP32 se resumem totalmente na programação da IDE do Arduino e na ligação do modulo Joystick.
Já em relação a programação e estruturação do jogo no Unity foi necessário diversos componentes e linhas de código em C# para que o jogo pudesse funcionar corretamente.
Para estruturar o jogo foi criado 12 cenas que possuem diversas funções, e cada uma delas está descrita na lista a baixo:

- **GameManager**: Essa cena é responsável por controlar todo o funcionar do jogo, desde reiniciar a cada ponto feito a marcar cada ponto na tela principal do jogo

- **MainCamera**: Essa é a câmera principal do jogo, isso é, é o ponto de vista que será mostrado ao jogador.

- **Ball**: Essa é a cena da bola, nela está inserido os códigos necessários para o funcionamento da física da bola.

- **PlayerPaddle**: Esse é o objeto controlado pelo jogador, nele está o código necessário para fazer a movimentação do player além de fazer a comunicação com o ESP32 via serial.

- **ComputerPaddle**: Esse objeto que é bem parecido com o PlayerPaddle é o objeto que representa o oponente nesse jogo, nele é possível encontrar o código que controla a posição do padle a partir da posição da bola.

- **TopWall**: Responsável por delimitar o campo de jogo na parte superior, assim como responsável por ter a física de refletir objetos que se choquem com ela.

- **BottomWall**: Responsável por delimitar o campo de jogo na parte inferior, assim como responsável por ter a física de refletir objetos que se choquem com ela.

- **LeftWall**: Responsável por delimitar o campo de jogo na parte esquerda, assim como responsável por marcar os pontos do computador.

- **RightWall**: Responsável por delimitar o campo de jogo na parte direita, assim como responsável por marcar os pontos do jogador.

- **Line**: É a cena da linha branca que separa os dois campos.

- **Canvas**: É a cena na qual será mostrado os pontos de cada jogador.

- **EventSystem**: Faz parte da estrutura de eventos do jogo.

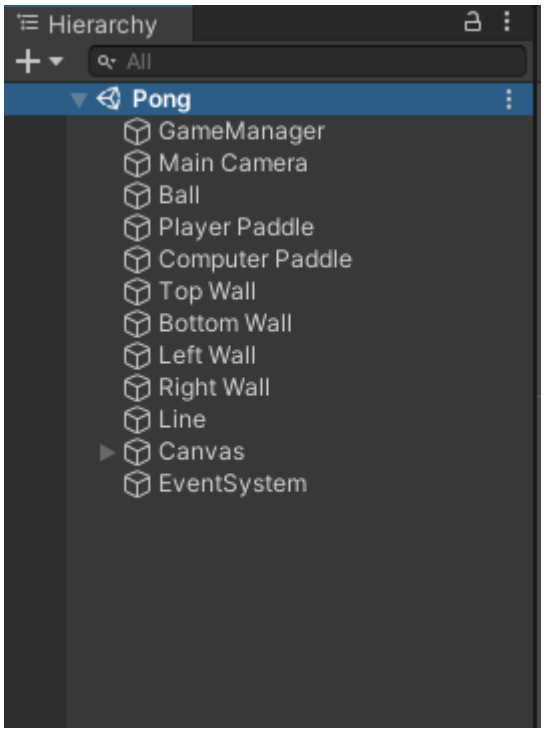

Fig. 7 – Hierarquia das cenas criadas.

E dentre essas cenas temos algumas codificações feitas para que o funcionamento do jogo aconteça, dentre esses códigos estão a codificação da movimentação do jogador, a movimentação do computador e da física relacionada a bola e as paredes.

No código dentro na da cena da bola é possível encontrar algumas funções especificas:

No começo do código da cena ball é possível ver as declarações do necessárias para iniciar o código, primeiramente é instanciado um novo Rigidbody2D e na função awake é iniciado esse elemento instanciado com as informações da própria cena, no caso o Rigidbody2D da bola.

```
public float speed = 200f;
public new Rigidbody2D rigidbody { get; private set; }

private void Awake(){
    this.rigidbody = GetComponent<Rigidbody2D>();
}
```
Fig. 8 – Função awake na cena ball.

Nessa função é reiniciado a posição da bola na tela.

```
public void ResetPosition(){
    this.rigidbody.velocity = Vector2.zero;
    this.rigidbody.position = Vector2.zero;
}
```
Fig. 9 – Função de resetar a posição da bola.

A função da figura 10 é a responsável por acrescentar uma força em um determinado ângulo para a saída da bola no meio da plataforma.

```
public void AddStartingForce() {
    float x = Random.value < 0.5f ? -1f : 1f;
    float y = Random.value < 0.5f ? Random.Range(-1f, -0.5f)
                                  : Random.Range(0.5f, 1f);
    Vector2 direction = new Vector2(x, y);
    this.rigidbody.AddForce(direction * this.speed);
}
```

Fig. 10 – Função para adicionar um ângulo de saída para a bola.

A função da figura 11 é a responsável por adicionar uma força no objeto rigidbody.

```
public void AddForce(Vector2 force){
    this.rigidbody.AddForce(force);
}
```

Fig. 11 – Função para adicionar uma força no objeto rigidbody.

Em relação as paredes superior e inferior a única codificação que está associada a elas é o código da física de refletir a bola quando ela colide com as paredes. Essa função é a responsável por adquirir o vetor normal do objeto que colidiu com a parede, ou seja, qualquer objeto que colidir será refletido para direção de sua normal.

```
[RequireComponent(typeof(BoxCollider2D))]
public class BouncySurface : MonoBehaviour{
    public float bounceStrength = 1f;
    private void OnCollisionEnter2D(Collision2D collision){
        Ball ball = collision.gameObject.GetComponent<Ball>();
        if (ball != null){
            Vector2 normal = collision.GetContact(0).normal;
            ball.AddForce(-normal * this.bounceStrength);
        }
    }
}
```

Fig. 12 – Função superfície refletora.

Em relação a raquete do computador é feito a movimentação automática da raquete calculando a posição da bola e se movendo de acordo com ela, e quando a bola está no campo do inimigo é comandado a raquete do computador ir para o meio da sua posição.

```
public Rigidbody2D ball;
private void FixedUpdate(){
    if (this.ball.velocity.x > 0f){
        if (this.ball.position.y > this.rigidbody.position.y) {
            this.rigidbody.AddForce(Vector2.up * this.speed);
        } else if (this.ball.position.y < this.rigidbody.position.y) {
            this.rigidbody.AddForce(Vector2.down * this.speed);
        }
    }else{
        if (this.rigidbody.position.y > 0f) {
            this.rigidbody.AddForce(Vector2.down * this.speed);
        } else if (this.rigidbody.position.y < 0f) {
            this.rigidbody.AddForce(Vector2.up * this.speed);
        }
    }
}
```

Fig. 13 – Função de movimentação do computador.

Em relação a raquete do player é feito a movimentação a partir do resultado lido na porta serial que é gerado no analógico conectado ao ESP32.

Nesse código é criado uma nova porta serial e ela é iniciada na função Start. Após isso é feito a leitura dos resultados enviados pelo ESP32 e é então executado a movimentação se baseando neles.

```
SerialPort porta = new SerialPort("COM7", 9600);
public Vector2 direction { get; private set; }
void Start(){
    porta.Open();
    porta.ReadTimeout = 10;
}
private void Update(){
    if(porta.IsOpen){
    string value = porta.ReadLine();

    if (value.Equals("ui_up")) {
        this.direction = Vector2.up;
        value = "";
    }
    if (value.Equals("ui_down")) {
        this.direction = Vector2.down;
        value = "";
    }
    if(value.Equals("Parado")){
        this.direction = Vector2.zero;
        value = "";
    }
}

private void FixedUpdate(){
    if (this.direction.sqrMagnitude != 0) {
        this.rigidbody.AddForce(this.direction * this.speed);
    }
}
```

Fig. 14 – Função de movimentação do jogador.

4. COMENTÁRIOS E CONCLUSÕES

Mesmo que essa ideia de projeto seja extremamente simples desde a parte de controle pelo microcontrolador ESP32 e o modulo Joystick até no desenvolvimento completo do jogo Pong utilizando a gameEngine Unity3D é possível entender e aplicar o básico de leitura de portas analógicas e comunicação serial por parte do ESP32 e por parte do Unity 3D é possível entender a dinâmica de funcionamento da plataforma assim como as principais funções de seus objetos de cena.

Logo, esse trabalho mesmo com sua simplicidade é possível entender diversos e aplicar diversos conceitos de micro controladores e programação de jogos em uma gameEngine que vão desde a comunicação serial no código do Unity3D até na criação da aparência do próprio jogo.

5. REFERÊNCIAS

[1] Edson Eleutério, “Vantagens na Elaboração de Projetos Digitais Utilizando Dispositivos Lógicos Programáveis”, Natal, 1998, p. 3.

[2] João Inácio, “Implementação de circuitos lógicos fundamentados em uma classe logicas paraconsistentes anotadas”, São Paulo, 1997, p. 140.

[3] Curto Circuito, “Conhecendo o ESP 32”, São Paulo, 2018. Disponível em <https://www.embarcados.com.br/portas-logicas/>
Acesso em 19 de outubro de 2021.

[4] Erik Passos, José Ricardo, Fernando Ribeiro, Pedro Morão, “Tutorial: Desenvolvimento de jogos com o Unity3D”, Rio de janeiro, 2009.