Prova 3 – LAOC

Integrantes:

Eduardo Alves de Freitas
Matheus Dutra Cerbino

Introdução

O trabalho a seguir tem como objetivo a implementação de uma versão simplificada do jogo Pong na linguagem C e posteriormente a tradução desta implementação para a linguagem Assembly do MIPS, com o objetivo de testar os conhecimentos adquiridos durante o semestre na disciplina de Arquitetura e Organização de Computadores I.

Programa em C

O programa em C foi feito de maneira simplista, não implementando o controle pelo usuário apenas apresentando o funcionamento das lógicas de jogo

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define WINDOWS _WIN32 || _WIN64
#ifdef WINDOWS
#include <Windows.h>
#else
#include <unistd.h>
#endif

#define HEIGHT 480
#define WIDTH 640 // 16/9 resolution
#define MARGIN_HORIZONTAL 4
#define PADDLE_RADIUS 2 //Total paddle size of 5
#define DELAY_TIME 200
```

Acima temos a inclusão das bibliotecas, com destaque a inclusão das bibliotecas windows.h e unistd.h que foram incluídas através de um ifdef para permitir a execução multiplataforma do programa, onde ambas são utilizadas para a mesma função porém para sistemas operacionais distintos.

Além disso, temos os defines que são valores que são usados constantemente e tem valores fixos durante toda a execução do programa, são eles:

HEIGHT: Determina a altura do campo do jogo

WIDTH: Determina a largura do campo do jogo

MARGIN_HORIZONTAL: Determina a distância que as raquetes estão da parede traseira a esta.

PADDLE_RADIUS: Determina quantos pixels serão estendidos em cada extremidade da raquete, ou seja, esse valor usado como dois, temos o pixel central além de dois pixeis tanto para cima quanto para baixo do centro.

DELAY_TIME: Determina de quantos em quantos milissegundos a lógica do jogo executa.

```
char **screen;
int ballx = WIDTH / 2, bally = HEIGHT / 2;
int paddleOnex = MARGIN_HORIZONTAL,
    paddleOney = HEIGHT / 2,
    paddleOneSpeed = -1,
    paddleTwox = WIDTH - MARGIN_HORIZONTAL,
    paddleTwoy = HEIGHT / 2,
    paddleTwoSpeed = 1;
int speedBallx = 1, speedBally = 1;
```

Acima temos as declarações das variáveis globais, estas foram declaradas assim para facilitar a tradução para MIPS além facilitar o acesso a estas que são utilizadas em diversas partes do código. São elas:

ballx: Posição no eixo x da bola
bally: Posição no eixo y da bola
paddleOnex: Posição no eixo x do centro da raquete 1
paddleOney: Posição no eixo y do centro da raquete 1
paddleOneSpeed: Direção a qual a raquete 1 se move
paddleTwox: Posição no eixo x do centro da raquete 2
paddleTwoy: Posição no eixo y do centro da raquete 2
paddleTwoSpeed: Direção a qual a raquete 2 se move
speedBallx: Direção horizontal a qual a bola se move
speedBally: Direção vertical a qual a bola se move

```c
void printMatrix(char **matrix)
{
    int i = 0, j = 0;
    for (i = 0; i < HEIGHT + 1; i++)
    {
        for (j = 0; j < WIDTH; j++)
        {
            printf("%c", matrix[j][i]);
        }
        printf("\n");
    }
}

void drawMargin()
{
    int i;
    for (i = 0; i < HEIGHT; i++)
    {
        screen[0][i] = '*';
        screen[(int)WIDTH][i] = '*';
    }

    for (i = 0; i < WIDTH; i++)
    {
        screen[i][0] = '*';
        screen[i][HEIGHT] = '*';
    }
}
```

Essas são as funções de desenho, a printMatrix é responsável por exibir o campo juntamente da bolinha e as raquetes, apenas printando o conteúdo da matriz que guarda o estado do jogo. Enquanto a drawMargin é responsável apenas por desenhar um contorno delimitando a arena para facilitar a identificação do fim desta.

```c
void delay(int delayTime)
{
#ifdef WINDOWS
    Sleep(delayTime);
#else
    usleep(delayTime * 1000); /* sleep for 100 milliSeconds */
#endif
}

void drawPaddle(char **matrix, int paddlex, int paddley)
{
    int i = 0;
    for (i = 0; i <= PADDLE_RADIUS; i++)
    {
        matrix[paddlex][paddley + i] = '|';
        matrix[paddlex][paddley - i] = '|';
    }
}

void clearMatrix()
{
    int i, j;
    for (i = 0; i < WIDTH; i++)
    {
        for (j = 0; j < HEIGHT; j++)
        {
            screen[i][j] = ' ';
        }
    }
}

void clear()
{
    clearMatrix();
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}
```

A função delay é uma função multiplataforma que utiliza das funções das bibliotecas windows.h e unistd.h e é responsável por controlar o intervalo entre as execuções das lógicas do jogo.

A função drawPaddle daa a posição central da raquete desenha ela como um todo incluindo seu raio.

A função clearMatrix apenas reinicia a matriz de jogo, para evitar que eventos anteriores continuem registrados na tela, como ao movimentar um elemento este se manter onde antes estava além da nova posição.

A função clear também é multiplataforma e ela apenas limpa o console, para fornecer uma sensação de atualização da tela, em vez de como se fossem apenas prints sucessivos.

```c
void draw()
{
    clear();
    screen[ballx][bally] = 'O';
    drawPaddle(screen, paddleOnex, paddleOney);
    drawPaddle(screen, paddleTwox, paddleTwoy);
    drawMargin();
    printMatrix(screen);
}

void movePaddles(int *paddley, int *paddleSpeed)
{
    *paddley += *paddleSpeed;
    if ((*paddley + PADDLE_RADIUS) > HEIGHT)
    {
        *paddley--;
        *paddleSpeed *= -1;
    }
    if ((*paddley - PADDLE_RADIUS) < 0)
    {
        *paddley++;
        *paddleSpeed *= -1;
    }
}

void gameOver()
{
    ballx = WIDTH / 2;
    bally = HEIGHT / 2;
    paddleOnex = MARGIN_HORIZONTAL;
    paddleOney = HEIGHT / 2;
    paddleOneSpeed = -1;
    paddleTwox = WIDTH - MARGIN_HORIZONTAL,
    paddleTwoy = HEIGHT / 2;
    paddleTwoSpeed = 1;
    speedBallx = 1;
    speedBally = 1;
}
```

A função draw apenas agrega as funções de desenho, ela posiciona a bola no campo, além de invocar todas as outras funções de desenho.

MovePaddles é responsável por controlar a lógica das raquetes, decidindo se a próxima posição é valida, além de fornecer o movimento que esta apresenta (subindo e descendo).

GameOver reinicia as variáveis do jogo para seus estados iniciais, para simular um fim de jogo quando a bola sai da tela pelas laterais, ela é invocada por outra função.

```
void moveBall()
{
    ballx += speedBallx;
    if (ballx > WIDTH - MARGIN_HORIZONTAL)
    {
        if ((bally >= paddleTwoy - PADDLE_RADIUS) &&
            (bally <= paddleTwoy + PADDLE_RADIUS))
            speedBallx *= -1;
        else
            gameOver();
    }
    if (ballx < MARGIN_HORIZONTAL)
    {
        if ((bally >= paddleOney - PADDLE_RADIUS) &&
            (bally <= paddleOney + PADDLE_RADIUS))
            speedBallx *= -1;
        else
            gameOver();
    }

    bally += speedBally;
    if (bally > HEIGHT)
    {
        bally--;
        speedBally *= -1;
    }
    if (bally < 0)
    {
        bally++;
        speedBally *= -1;
    }
}

void loop()
{
    delay(DELAY_TIME);
    movePaddles(&paddleOney, &paddleOneSpeed);
    movePaddles(&paddleTwoy, &paddleTwoSpeed);
    moveBall();
    draw();
    loop(screen);
}
```

A função moveBall trata da lógica da bola, como colisão com as paredes e mudança de direção e avaliação de fim de jogo, a bola ganha velocidade oposta ao objeto o qual ela colidiu e é declarado fim de jogo quando está atinge um dos limites laterais do campo.

A função loop é quem executa todo o jogo, realizando o delay e chamando as funções de lógica e desenho na hora certa.

```c
int main()
{
    int i, j;
    screen = malloc(sizeof(char *) * WIDTH);
    for (i = 0; i < WIDTH; i++)
    {
        screen[i] = malloc(sizeof(char) * HEIGHT);
        for (j = 0; j < HEIGHT; j++)
        {
            screen[i][j] = ' ';
        }
    }

    loop();

    return 0;
}
```

A função main apenas inicializa a matriz de jogo com o tamanho fornecido nos defines e chama o loop o qual ficará responsável pelo jogo até o fim de sua execução.

Programa em MIPS Assembly

Para facilitar a tradução para o MIPS algumas das funções do código em C foram retiradas como as funções de desenho e delay, o que gerou um código intermediário o qual foi traduzido tentando manter a correspondência 1 para 1 das funções presentes no código em C para o código MIPS.

```
MAIN:

    #WIDTH = 640
    #HEIGHT = 480

    addi $s0 $zero 320 #ballx = WIDTH/2
    addi $s1 $zero 240 #bally = HEIGHT / 2
    addi $s2 $zero 240 #paddleOney = HEIGHT / 2
    addi $s3 $zero -1  #paddleOneSpeed = -1
    addi $s4 $zero 240 #paddleTwoy = HEIGHT / 2
    addi $s5 $zero 1   #paddleTwoSpeed = 1
    addi $s6 $zero 1   #speedBallx = 1
    addi $s7 $zero 1   #speedBally = 1

    addi $sp $sp -1228800 #Reservando espaco para o array de pixel WIDTHxHEIGHT

    add $t8 $zero $zero #int quit = 0
```

A main contempla a declaração das variáveis globais além da alocação da memória necessária para armazenar a matriz do jogo.

```
    LOOP:
    jal MOVEPADDLEONE #movePaddleOne()
    jal MOVEPADDLETWO #movePaddleTwo()
    jal MOVEBALL #moveBall()
    jal DRAW #draw()


    bne $t8 $zero END #if (quit != 1) return
    j LOOP

    END:
    li $v0, 10          # finaliza o programa
    syscall

DRAWPADDLE:
    add $t0 $zero $zero #int i = 0
    addi $t1 $zero 2 #PADDLE_RADIUS
    addi $t3 $zero 480 #$t3 = HEIGHT

    mult    $a0 $t3 #paddleX * Height, pora compor o endereco em forma de vetor unico
    mflo    $t2 #Endereco

    sll $t2 $t2 2 #Endereco * 4, tranformando para endereco de 32bits
    add $t2 $sp $t2 #paddlex = $t2

    FOR1_DRAWPADDLE:
        addi $t6 $zero 2 #Valor da barra '|' que compoe os paddles

        add $t4 $a1 $t0 #paddley + i
        add $t4 $t2 $t4 #Endereco em vetor unidimensional(paddley+1)
        sw $t6 0($t4) #screen[paddlex][paddley + i] = '|'

        sub $t5 $a1 $t0 #paddley - i
        add $t5 $t2 $t5 #Endereco em vetor unidimensional(paddley-1)
        sw $t6 0($t5) #screen[paddlex][paddley - i] = '|'

        addi $t0 $t0 1 #i++
    ble $t0 $t1 FOR1_DRAWPADDLE
    jr $ra
```

Temos as tags LOOP, END e DRAWPADDLE, as tags LOOP e DRAWPADDLE iniciam procedimentos que tentam corresponder às funções homônimas do código em C e a tag END é chamada para finalizar o programa quando necessário.

```
CLEARMATRIX:
    addi $t0 $zero 0 #variavel de iteração = i
    addi $t1 $zero 1228800 #valor maximo WIDTHxHEIGHT
    FOR1_DRAW:
        add $t2 $sp $t0 #Posicao de memoria a ser atualizada = i + memAdd
        sw $zero 0($t2)
        sll $t0 $t0 2
    blt $t0 $t1 FOR1_DRAW

    jr $ra

DRAW:
    add $t9 $ra $zero #Guarda o endereco de retorno

    addi $t2 $zero 1 #valor que representa a bola 'O'

    addi $t0 $zero 480 #Height
    mult $s0 $t0 #valor ballx em vetor unidimensional
    mflo  $t0
    add $t1 $t0 $s1 #[ballx][bally]
    sll $t1 $t1 2 #[ballx][bally] em palavras de 32bits
    add $t1 $sp $t1 #Endereco de [ballx][bally]
    sw $t2 0($t1) #screen[ballx][bally] = 'O'

    addi $a0 $zero 4 #arg0 = MARGIN_HORIZONTAL
    add $a1 $zero $s2 #arg1 = paddleOneY
    jal DRAWPADDLE #drawPaddle(MARGIN_HORIZONTAL, paddleOney)

    addi $a0 $zero 636 #arg0 = MARGIN_HORIZONTAL
    add $a1 $zero $s4 #arg1 = paddleOneY
    jal DRAWPADDLE #drawPaddle(MARGIN_HORIZONTAL, paddleOney)

    add $ra $t9 $zero #Resgata o endereco de retorno
    jr $ra
```

Como no caso anterior, ambas as tags são tentativas de emular as funções homônimas presentes no código C, tentando o mais fielmente possível reproduzir os passos de C em MIPS.

```
MOVEPADDLEONE:
    add $s2 $s2 $s3 #paddleOney += paddleOneSpeed

    addi $t0 $s2 2 #paddleOney + PADDLE_RADIUS
    addi $t1 $zero 480 #t1 = Height, Necessario para usar slt
    slt $t2 $t1 $t0 #(paddleOney + PADDLE_RADIUS) > HEIGHT)
    beq $t2 $zero END_IF1_MOVEPADDLEONE
        addi $s2 $s2 -1 #paddleOney--
        addi $t3 $zero -1 # $t3 = -1
        mult $s3 $t3 # $lo = paddleOneSpeed * -1
        mflo $s3 #paddleOneSpeed *= -1
    END_IF1_MOVEPADDLEONE:

    subi $t0 $s2 2 #paddleOney - PADDLE_RADIUS
    slti $t1 $t0 0 #(paddleOney - PADDLE_RADIUS) < 0)
    bne $t1 $zero END_IF2_MOVEPADDLEONE
        addi $s2 $s2 1 #paddleOney++
        addi $t3 $zero -1 # $t3 = -1
        mult $s3 $t3 # $lo = paddleOneSpeed * -1
        mflo $s3 #paddleOneSpeed *= -1
    END_IF2_MOVEPADDLEONE:

    jr $ra

MOVEPADDLETWO:
    add $s2 $s2 $s3 #paddleOney += paddleOneSpeed

    addi $t0 $s2 2 #paddleOney + PADDLE_RADIUS
    addi $t1 $zero 480 #t1 = Height, Necessario para usar slt
    slt $t2 $t1 $t0 #(paddleOney + PADDLE_RADIUS) > HEIGHT)
    beq $t2 $zero END_IF1_MOVEPADDLETWO
        addi $s2 $s2 -1 #paddleOney--
        addi $t3 $zero -1 # $t3 = -1
        mult $s3 $t3 # $lo = paddleOneSpeed * -1
        mflo $s3 #paddleOneSpeed *= -1
    END_IF1_MOVEPADDLETWO:

    subi $t0 $s2 2 #paddleOney - PADDLE_RADIUS
    slti $t1 $t0 0 #(paddleOney - PADDLE_RADIUS) < 0)
    bne $t1 $zero END_IF2_MOVEPADDLETWO
        addi $s2 $s2 1 #paddleOney++
        addi $t3 $zero -1 # $t3 = -1
        mult $s3 $t3 # $lo = paddleOneSpeed * -1
        mflo $s3 #paddleOneSpeed *= -1
    END_IF2_MOVEPADDLETWO:

    jr $ra
```

Temos agora as tags MOVEPADDLEONE e MOVEPADDLETWO, ambas tentam emular uma única função do código em C, como a passagem de diferentes parâmetros e complexa e muitas vezes confusa em MIPS, optamos por criar duas funções idênticas, cada uma para um caso distinto.

```
GAMEOVER:
    addi $s0 $zero 320 #ballx = WIDTH/2
    addi $s1 $zero 240 #bally = HEIGHT / 2
    addi $s2 $zero 240 #paddleOney = HEIGHT / 2
    addi $s3 $zero -1  #paddleOneSpeed = -1
    addi $s4 $zero 240 #paddleTwoy = HEIGHT / 2
    addi $s5 $zero 1   #paddleTwoSpeed = 1
    addi $s6 $zero 1   #speedBallx = 1
    addi $s7 $zero 1   #speedBally = 1

    jr $ra

MOVEBALL:
    add $s0 $s0 $s6 #ballx += speedBallx

    addi $t0 $zero 636 #WIDTH - MARGIN_HORIZONTAL
    slt $t1 $t0 $s0 #ballx > WIDTH - MARGIN_HORIZONTAL
    beq $t1 $zero END_IF1_MOVEBALL #if (ballx > WIDTH - MARGIN_HORIZONTAL)
        addi $t2 $s4 -4 #paddleTwoy - PADDLE_RADIUS
        addi $t3 $s4 4 #paddleTwoy + PADDLE_RADIUS
        and $t4 $t2 $t3 #(bally >= paddleTwoy - PADDLE_RADIUS) && (bally <= paddleTwoy + PADDLE_RADIUS)
        beq $t4 $zero ELSE_IF2_MOVEBALL #if ((bally >= paddleTwoy - PADDLE_RADIUS) && (bally <= paddleTwoy + PADDLE_RADIUS))
            addi $t5 $zero -1 # $t5 = -1
            mult $s6 $t5 #speedBallx * -1
            mflo $s6 #speedBallx *= -1
        j END_IF2_MOVEBALL

        ELSE_IF2_MOVEBALL: #else
            add $t9 $ra $zero
            jal GAMEOVER #gameOver()
            add $ra $t9 $zero

        END_IF2_MOVEBALL:
```

A tag GAMEOVER inicia o procedimento de GameOver que realiza exatamente a mesma função de sua função homônima em C, reiniciando as variáveis para um estado inicial.

A tag MOVEBALL inicia o procedimento que trata da lógica da bola, como a função homônima à esse procedimento em C possui diversos if's e else's alguns deles aninhados, a tradução para MIPS terminou sendo a mais longa, na imagem acima está presente apenas uma parte da função o resto está presente na imagem a seguir.

```
        END_IF2_MOVEBALL:

    END_IF1_MOVEBALL:

    slti $t0 $s0 4 #ballx < MARGIN_HORIZONTAL
    beq $t0 $zero END_IF3_MOVEBALL #if (ballx < MARGIN_HORIZONTAL)
        addi $t1 $s2 -4 #paddleOney - PADDLE_RADIUS
        addi $t2 $s4 4 #paddleOney + PADDLE_RADIUS
        and $t3 $t1 $t2 #(bally >= paddleOney - PADDLE_RADIUS) && (bally <= paddleOney + PADDLE_RADIUS)
        beq $t3 $zero ELSE_IF4_MOVEBALL #if ((bally >= paddleOney - PADDLE_RADIUS) && (bally <= paddleOney + PADDLE_RADIUS))
            addi $t4 $zero -1 # $t5 = -1
            mult $s6 $t4 #speedBallx * -1
            mflo $s6 #speedBallx *= -1
        j END_IF4_MOVEBALL

        ELSE_IF4_MOVEBALL: #else
            add $t9 $ra $zero
            jal GAMEOVER #gameOver()
            add $ra $t9 $zero

        END_IF4_MOVEBALL:

    END_IF3_MOVEBALL:

    add $s1 $s1 $s7 #bally += speedBally

    addi $t0 $zero 480 #$t0 = HEIGHT
    slt $t1 $t0 $s1 #bally > HEIGHT
    beq $t1 $zero END_IF5_MOVEBALL #if (bally > HEIGHT)
        addi $s1 $s1 -1 #bally--
        addi $t2 $zero -1# -1
        mult $s7 $t2 #speedBally * -1
        mflo $s7 #speedBally *= -1
    END_IF5_MOVEBALL:

    slt $t3 $s0 $zero #bally < 0
    beq $t3 $zer0 END_IF6_MOVEBALL #if (bally < 0)
        addi $s1 $s1 1 #bally++
        addi $t4 $zero -1 # -1
        mult $s7 $t4 #speedBally * -1
        mflo $s7 #speedBally *= -1
    END_IF6_MOVEBALL:

    jr $ra
```

Conclusão

Umas das dificuldades encontradas foi a presença poucos registradores, por isso, limitamos o uso de variáveis, pois a adição de variáveis traria grande trabalho ao necessitar de constantemente fazer o spill dos registradores e buscar conteúdos. Além de aumentar consideravelmente o código.

Como a implementação foi feita com conhecimento básico da linguagem MIPS o que acarretou um código longo e pouco otimizado, o que provavelmente acarretará em uma queda de performance em relação ao código em C devido às otimizações feitas pelo compilador