```
/************************************************
LAOC1 - PROVA 3
Aluno: Thiago Lima Bahia Santos
Matrícula: 20183000302
Descrição do programa: Pong
Data: 29/11/19
************************************************/
```

1 INTRODUÇÃO

Pong é um dos primeiros videogames arcade. O jogo consiste em uma quadra de tênis em 2D, com duas barras verticais em lados opostos da tela, representando os jogadores, e uma bola. Foi originalmente publicado pela Atari em 1972, ajudando a estabelecer a indústria de videogames como um novo formato de jogos, juntamente com o primeiro console doméstico, o Magnavox Odyssey. Pong foi o primeiro jogo comercialmente bem-sucedido da história, sendo refeito em inúmeras plataformas domésticas e portáteis após seu lançamento.

O objetivo desse trabalho é reinterpretar o jogo em uma forma mais rudimentar, com foco na tradução, implementação e estudo de seu Assembly MIPS (*Microprocessor without interlocked pipeline stages*). O código foi originalmente escrito em Linguagem C e posteriormente testado com o simulador MARS.

2 DESCRIÇÃO DO PROGRAMA EM C

2.1 Variáveis, procedimentos e funções

```
typedef struct{
   int x, y;                                    //Define o centro do objeto em x e y
   int size;                                    //Define o tamanho do objeto
   char c;                                 //Define o caractere do objeto
} Objeto;

char Pong[SCREEN_I][SCREEN_J];

void InicializaJogador(Objeto *j, int x, char c) {

        (*j).x = x;
        (*j).y = SCREEN_I / 2; //Inicia jogador ao centro
        (*j).size = BAR_SIZE;
        (*j).c = c;

}

void PosicionaJogador(Objeto j) {

        int i;

        for (i = j.y; i <= j.y + (BAR_SIZE / 2); i++)
                Pong[i][j.x] = j.c;

        for (i = j.y; i >= j.y - (BAR_SIZE / 2); i--)
                Pong[i][j.x] = j.c;

}
```

```
int EscolheMovimentoJogador(Objeto j, int *mov) {

	//Verifica se o Jogador colidiu com o limite superior
	if ( j.y - ( j.size / 2 ) + 1 <= 2 )
		*mov = 0; //Se mov = 0, movimenta o Jogador para baixo

	//Verifica se o Jogador colidiu com o limite inferior
	if ( j.y + ( j.size / 2 ) - 1 >= SCREEN_I - 3 )
		*mov = 1; //Se mov = 1, movimenta o Jogador para cima

}

void MovimentaJogador(Objeto *j, int mov) {

	if (mov == 1)
		(*j).y--; //Movimenta o Jogador selecionado para cima
	else
		(*j).y++; //Movimenta o Jogador selecionado para baixo

}

void RandomizaInicioBola(Objeto *b) {

	srand(time(NULL));
	(*b).y = ( (SCREEN_I / 2) + (rand() % BAR_SIZE) * (rand() % 2) * (-1) );

}

void InicializaBola(Objeto *b, int x, char c) {

	(*b).x = x;
	RandomizaInicioBola(b);
	(*b).size = BALL_SIZE;
	(*b).c = c;

}

void PosicionaBola(Objeto *b) {

	Pong[(*b).y][(*b).x] = (*b).c;

}

void MovimentaBola(Objeto *b, int *x, int *y) {

	//Verifica se a Bola colidiu com o limite superior
	if((*b).y <= 1)
		*y *= -1;

	//Verifica se a Bola colidiu com o limite inferior
	if ((*b).y >= SCREEN_I - 2)
		*y *= -1;

	(*b).x += *x; //Movimenta a bola em x
	(*b).y += *y; //Movimenta a bola em y

}
```

```
void Rebate(Objeto *j, Objeto *b, int *x) {

	//Verifica se a Bola colidiu com o Jogador A
	if ( ( (*b).x - 1 == (*j).x ) &&
		((*b).y <= (*j).y + ((*j).size / 2)) &&
		((*b).y >= (*j).y - ((*j).size / 2)) )
			*x *= -1;

	//Verifica se a Bola colidiu com o Jogador B
	if ( ( (*b).x + 1 == (*j).x ) &&
		((*b).y <= (*j).y + ((*j).size / 2)) &&
		((*b).y >= (*j).y - ((*j).size / 2)) )
			*x *= -1;

}

int VerificaPontuacao(Objeto b, int *s1, int *s2) {

	if (b.x >= SCREEN_J - 1) {
		(*s1)++;
		return 1;
	}

	if (b.x <= 0) {
		(*s2)++;
		return 1;
	}

	return 0;

}

void InicializaPong() {

	int i, j;

   for (i = 0; i < SCREEN_I; i++) {
	Pong[i][0] = '|';
	Pong[i][SCREEN_J-1] = '|';
   }

   for (i = 0; i < SCREEN_J; i++) {
	Pong[0][i] = '=';
	Pong[SCREEN_I-1][i] = '=';
   }

   for (i = 1; i < SCREEN_I-1; i++)
      for (j = 1; j < SCREEN_J-1; j++)
         Pong[i][j] = ' ';

}

void AtualizaPong(Objeto j1, Objeto j2, Objeto b) {

	InicializaPong();
	PosicionaJogador(j1);
	PosicionaJogador(j2);
	PosicionaBola(&b);

}
```

2.2 Notas de Implementação

O primeiro passo para a implementação do código em C foi a abstração de como seriam representadas as entidades (1) barra (jogadores) e (2) bola. Para isso foi definido um novo tipo chamado Objeto, consistindo em uma estrutura que carrega as informações que definem o centro da entidade nos eixos x e y, o seu tamanho e o seu caractere. Instâncias desses objetos foram então aplicadas em uma matriz de 48 por 64 pixels[1], responsável por definir o que efetivamente está acontecendo no jogo.

Os jogadores basicamente possuem uma posição de início, uma movimentação para cima e para baixo limitada ao campo definido pela matriz e a capacidade de rebater. Para isso foram criados os procedimentos: InicializaJogador(), que constrói suas informações e PosicionaJogador() e MovimentaJogador(), responsável por posicionar e movimentar seus centros em campo. Além disso, foi necessário criar uma função EscolheMovimentoJogador(), para verificar se o jogador colidiu com o limite inferior ou superior e, então, poder escolher a movimentação correta. A capacidade de rebater é controlada pelo procedimento Rebate(), que verifica se o corpo do jogador colidiu com o corpo do bola, alternando sua direção.

A bola segue basicamente o mesmo princípio, exceto pelo fato de que sua movimentação é aleatória. Seus procedimentos são: SorteiaInicioBola(), que define em qual direção do eixo y a bola deve sair; InicializaBola(), que constrói suas informações e PosicionaBola() e MovimentaBola(), responsáveis por sua movimentação dentro dos limites do campo.

Quando a bola encontra o limite mais à direita ou o limite mais à esquerda da matriz, representado pela informação que carrega em seu eixo x, e não houve colisão com qualquer um dos jogadores, então o procedimento VerificaPontuacao() confere qual o limite tocado e adiciona um ponto para o jogador posicionado no lado oposto.

Os procedimentos InicializaPong(), AtualizaPong() e ImprimePong() são os responsáveis por controlar as interações entre as entidades e por imprimir o campo, os jogadores e a bola a cada interação. Dentre esses procedimentos de fechamento, cabe ressaltar principalmente o AtualizaPong(), que a todo momento reseta a matriz e define um novo posicionamento para todos os objetos.

A função main() basicamente cria as instâncias dos Jogadores 1 e 2 e da bola e executa todos os procedimentos e funções necessários respeitando uma ordem lógica, dentro de um loop infinito de while (1). Nesse loop foi necessário o uso da função usleep() para diminuir a velocidade de execução e tornar a visualização do Pong mais amigável durante sua execução no Terminal.

[1] Originalmente foi tentado implementar uma área de jogo de 480 por 640 pixels. No entanto, devido as limitações de exibição do Terminal, esse tamanho precisou ser redimensionado proporcionalmente para 48 por 64 pixels.

3 APRESENTAÇÃO DA EXECUÇÃO DO PROGRAMA

A implementação e os testes do programa foram realizados no ambiente de uma máquina virtual emulada a partir do software *Oracle VM VirtualBox*. Para a demonstração de seu funcionamento foram selecionadas imagens de eventos específicos que evidenciam as diversas partes do código.

3.1 Configurações da máquina utilizada

Sistema:

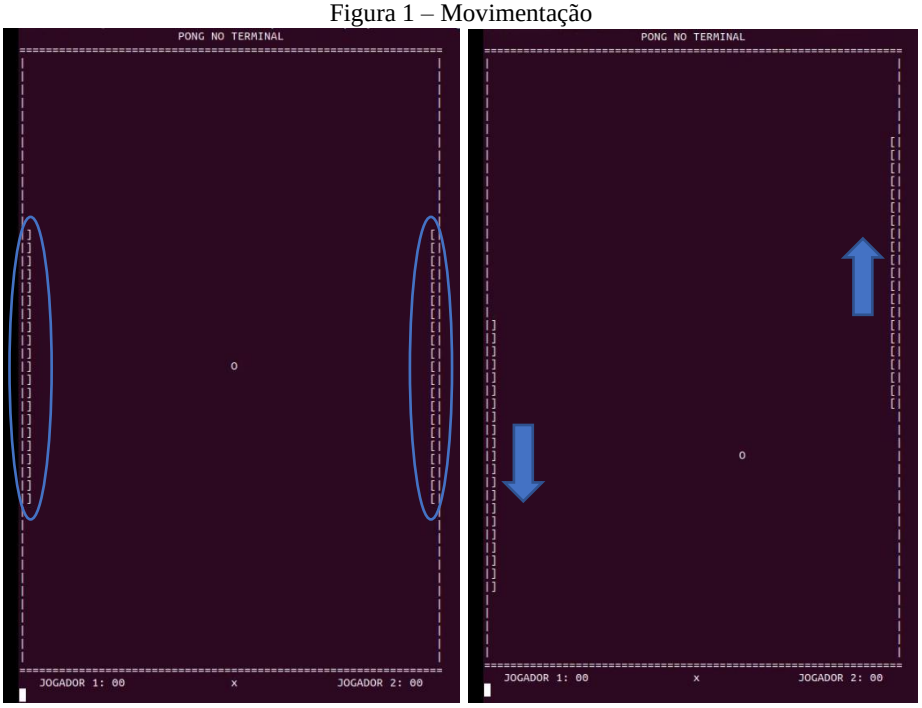
- Windows 10 Home
- Processador: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
- Memória instalada (RAM): 8,00 GB (utilizável 7,86 GB)

Máquina Virtual:

- Ubuntu 18.04 Desktop
- Processador(es) utilizados: 2 de 8 threads
- Memória base: 4,00 GB

3.2 Comportamento dos Jogadores

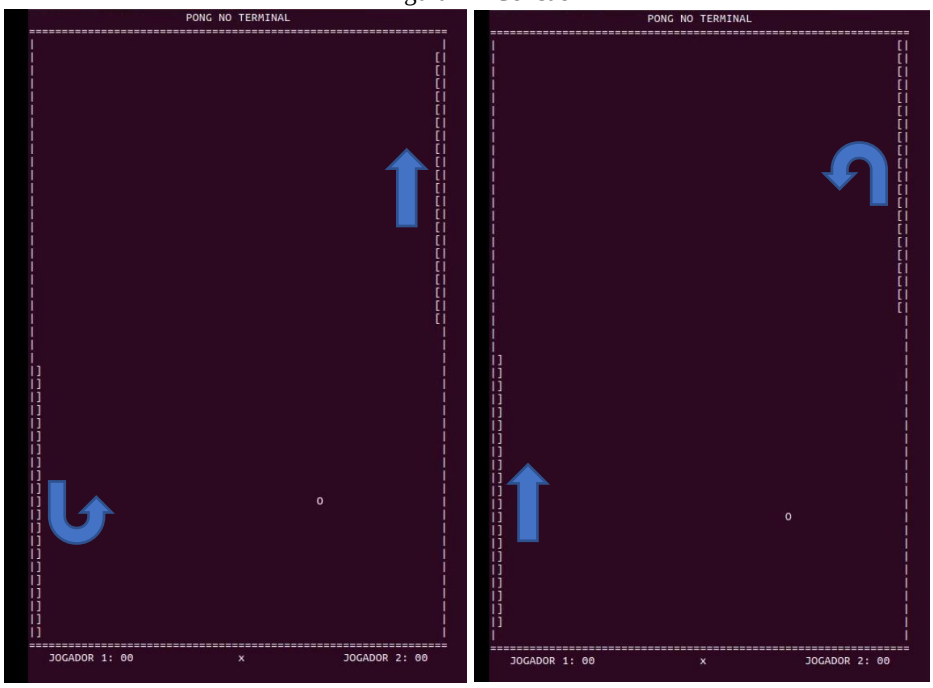
Figura 1 – Movimentação



Fonte: Produzido pelo autor

Os Jogadores 1 e 2 inicialmente começam centralizados e se movimentam em sentidos alternados (contrários). A decisão de movimentação inicial é aleatória, se “0” o Jogador desce, se “1” o jogador sobe.

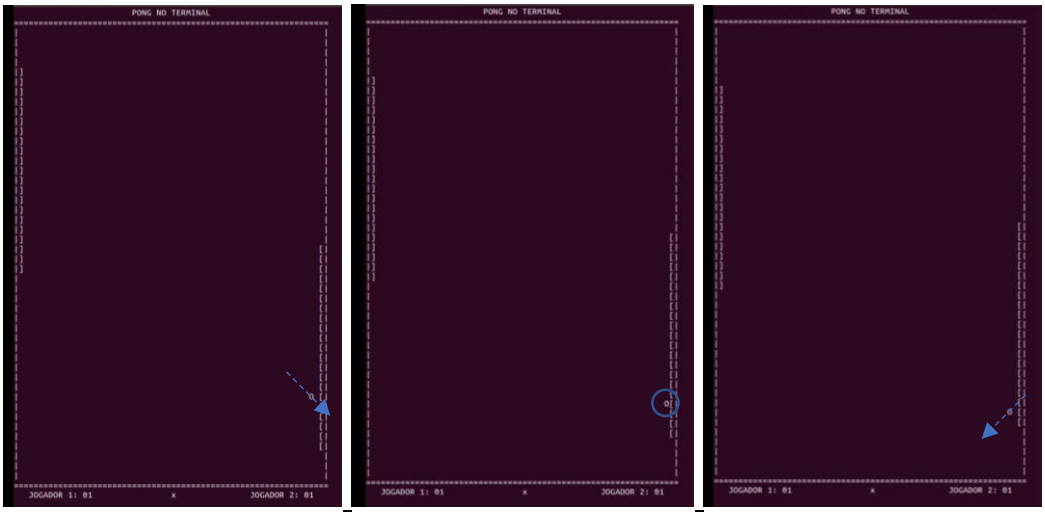
Figura 2 – Colisão



Fonte: Produzido pelo autor

Ao colidir com uma parede o Jogador precisa entender que chegou ao limite da tela e que sua movimentação precisa ser invertida.

Figura 3 – Rebatida

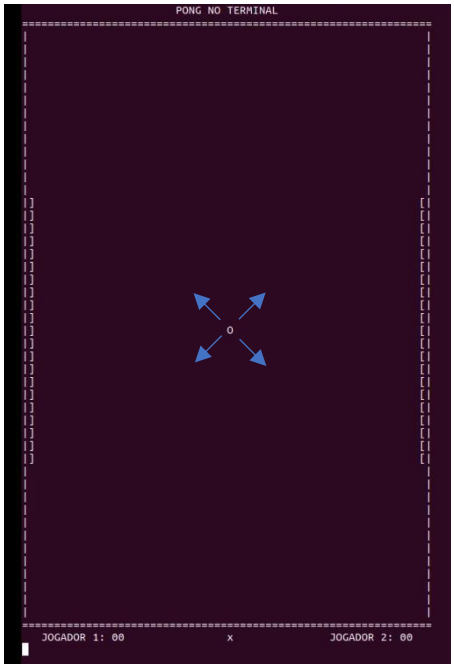


Fonte: Produzido pelo autor

A rebatida foi incluída como uma interação do Jogador, pois intuitivamente é mais fácil percebê-la dessa forma. No entanto, esse evento também pode ser entendido como um comportamento de movimentação da bola. A bola tem como limite a área superior e inferior do mapa e o que pode ser entendido como o corpo dos Jogadores. Ao colidir com qualquer um desses elementos a sua direção precisa necessariamente ser alterada.

3.3 Comportamento da Bola

Figura 4 – Caminho

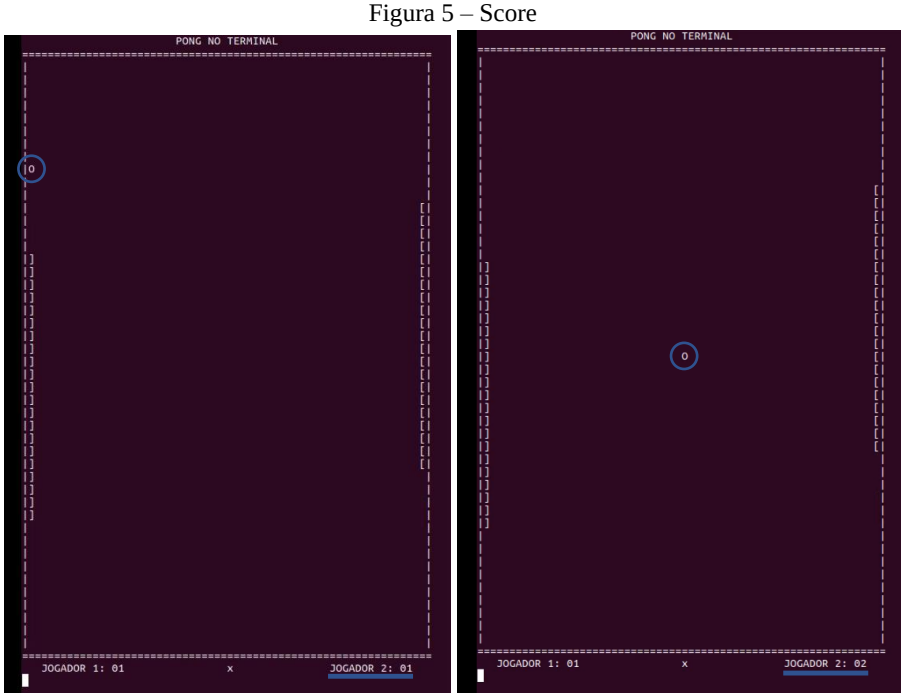


Fonte: Produzido pelo autor

O comportamento da bola é completamente aleatório. Tanto sua posição inicial, quanto o caminho que vai percorrer são gerados aleatoriamente sempre que uma nova inicialização é chamada. O tamanho dos Jogadores também influencia na randomização, quanto maior as barras, maior é o range de números que a função rand() poderá gerar.

3.4 Pontuação

Figura 5 – Score



Fonte: Produzido pelo autor

Quando a bola ultrapassa os limites mais à esquerda ou mais à direita do campo e não há colisão com nenhum dos Jogadores, então configurou-se um ponto para o Jogador que se encontra posicionado no lado oposto ao ultrapassado. Em sequência, a bola é reiniciada em uma posição aleatória tendo como base o centro do mapa.

4 DESCRIÇÃO DO PROGRAMA EM ASSEMBLY DO MIPS

Assembly é uma linguagem de programação de baixo nível projetada para um tipo específico de processador. Esse tipo de linguagem pode ser produzido compilando o código-fonte de uma linguagem de programação de alto nível, mas também pode ser escrito do zero. No caso do Projeto Pong, como o código em C já estava finalizado e programar em baixo-nível não é algo comumente praticado (exceto em situações muito específicas), foi utilizado a ferramenta *Compiler Explorer*[2], juntamente com o compilador MIPS gcc 5.4 para a conversão de C em MIPS.

4.1 Assembly do MIPS

Uma vez que o algoritmo em C necessitou de 246 linhas para ser finalizado, é evidente que a tradução para baixo-nível (Assembly do MIPS) ficaria muito mais extenso. Por esse motivo, o código foi colocado e comentado em um arquivo à parte. Ao todo foram necessárias 1012 linhas para converter o Pong. Em um comparativo, é como se houvesse aproximadamente 4 instruções MIPS para cada linha de código em C.

De modo geral, é possível perceber que alguns comandos da Linguagem MIPS foram evidentemente mais utilizadas que outras. A titulo de exemplo, pode-se destacar as instruções lw (*Load Word*), sw (*Store Word*), add (*Add*), addiu (*Add Immediate Unsigned*), j (*Jump*), jal (*Jump and Link*) e, também, a função move (*Move*). Curiosamente essa última, apesar de pouco abordada em sala de aula, apareceu de forma recorrente no código traduzido.

O código em MIPS pode ser compilado com sucesso mediante o uso da ferramenta MARS. No entanto, sua execução ficou comprovada apenas pelo Terminal, uma vez que o simulador utilizado possui suas limitações de uso.

[2] Ferramenta disponível em https://godbolt.org/

5 CONSIDERAÇÕES FINAIS

O objetivo inicialmente proposto por esse projeto, apesar de trabalhoso, foi concluído com sucesso. Esse trabalho foi de extrema importância para o entendimento de como uma linguagem de alto-nível e uma linguagem de baixo-nível se relacionam e suas respectivas importâncias. Durante a execução das tarefas propostas ficou visível como o processador prefere executar tarefas mais simples, não sendo tão importante o volume de trabalho a ser realizado.

A maior limitação encontrada foi realmente trabalhar sozinho, dada a quantidade de atividades necessárias ao Projeto. No entanto, foi gratificante poder concluí-lo. O aprendizado adquirido durante a sua realização são cruciais para a formação de todo e qualquer estudante de Engenharia de Computação. Sobretudo para aqueles que desejam seguir carreira com enfoque na construção de Hardware.

6 REFERÊNCIAS BIBLIOGRÁFICAS

PATTERSON, D.; HENESSY, J. L.; Organização e projeto de computadores: a interface hardware/software. 4ª Edição. São Paulo: Elsevier, 2009.