ORGANIZAÇÃO DE COMPUTADORES

Nomes: Enzo Hahn Veroneze e Rafaela da Rosa Soares.

DOCUMENTAÇÃO JOGO DA VELHA EM ASSEMBLY MIPS.

O jogo da velha foi implementado através do *software* MARS, com a linguagem de máquina *Assembly MIPS.* O jogo funciona da forma tradicional, sendo o usuário contra a inteligência artificial desenvolvida.

Em cada partida, o usuário é o ‘X’ e a inteligência artificial o ‘O’. Os rounds sempre iniciam na vez do usuário alternando com a *I.A.*, até haver uma vitória ou empate.

É disponibilizado um tabuleiro (Figura 1) de nove posições liberadas no início do jogo, sendo alterada durante os rounds.

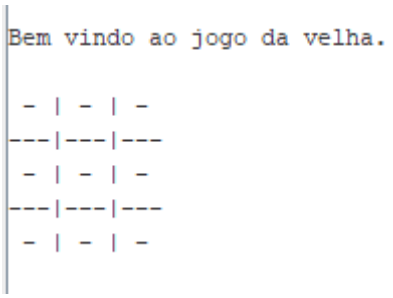

Figura 1: tabuleiro do jogo no terminal do software MARS.

É impressa uma mensagem para o usuário informar a linha e coluna respectiva à sua jogada, e caso a posição esteja desocupada, o hífen (“-”) é alterado para “X”. Caso não haja vitória, é passado para o próximo round, jogado pela inteligência artificial, que gera uma jogada em uma posição aleatória, caso não haja chance de vitória; ou no caso de haver, a posição escolhida é a que irá garantir a vitória da máquina.

Presente a situação de empate, o jogo para e informa o resultado, dando ao jogador a escolha de entrar em uma nova partida. Para a vitória, o sistema informa o vencedor e pede ao usuário se ele deseja jogar novamente.

A seguir consta a implementação das funções com exemplos.

1. SECTION

Na *section* do código foram determinadas informações sobre o jogo e as inserções necessárias de texto, assim como as máscaras utilizadas para determinar vitória. (Figura 2)

```
.data
char_X:          .byte   'X'
char_O:          .byte   'O'
char_dash:       .byte   '-'
char_vertical:   .byte   '|'
char_space:      .byte   ' '
str_separator:   .asciiz "\n---|---|---\n"
str_start:       .asciiz "\n\nBem vindo ao jogo da velha.\nDigite 1 para começar a jogar. \n"
str_moves:       .asciiz "\nInsira linha e coluna para jogada: "
str_fail:        .asciiz "\nNumeros invalidos, insira novamente."
str_tie:         .asciiz "\nPartida empatada."
str_win:         .asciiz "\nO ultimo jogador venceu a partida."
str_end:         .asciiz "\nDeseja jogar novamente? Digite 1 para sim, 0 para nao.\n\n"
str_exit:        .asciiz "\nJogo finalizado."

mask_1:          .byte   1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
mask_2:          .byte   0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0
mask_3:          .byte   0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0
mask_4:          .byte   1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0
mask_5:          .byte   0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0
mask_6:          .byte   0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0
mask_7:          .byte   1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0
mask_8:          .byte   0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0
```

Figura 2: section *.data* no código implementado.

2. FUNÇÕES

O jogo da velha possui 7 funções, incluindo a main. Seus retornos são utilizados para o funcionamento geral do jogo. São elas:

a) main:

A *main* é responsável por controlar o funcionamento geral do jogo, com chamada de procedimentos e funções.

Inicialmente ela indica o `stack pointer $sp`, apontando para o último espaço (24 bytes) a ser alocado na SRAM, de doze em doze bytes, para representar os 9 espaços/bytes de cada vetor (em X e O). Em seguida, ela chama a função *clear*, para limpar cada vetor.

O próximo funcionamento é dado com o início oficial do jogo, inicializando uma variável índice para acompanhar os rounds e uma variável de tamanho 9, para determinar o fim.

A função `draw_board` é chamada e inicia-se a função `loop` que controla o jogo inteiro. Na primeira linha, com a instrução "*beq*" compara-se o índice atual do jogo com a variável que guarda 9, e caso seja verdadeira ela vai para o procedimento/label *tie_round*, responsável por declarar empate e chamar a label *fim,* que define o recomeço ou o fim do jogo (Figura 4). Caso não seja caso de empate, o jogo continua com o movimento do usuário e depois da inteligência artificial, chamando a função `check_winner` e esperando retorno da variável `$v0`, para comparar o estado, determinando vitória (Figura 5).

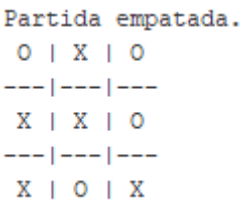

Figura 4: Partida empatada no terminal do *software* Mars.

No caso de não haver vitória, o índice é incrementado e passa para o próximo round com a chamada da função `loop` novamente, até que haja finalização da partida em algum dos casos.

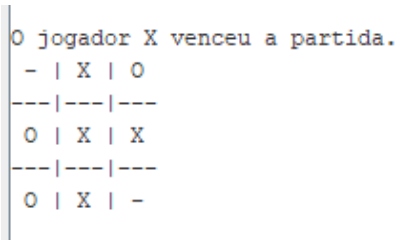

Figura 5: Vitória do jogador.

b) clear

Responsável por limpar um vetor de 12 posições, no caso do código, o vetor de X e O. São definidas duas variáveis de valor 0 e 12, respectivamente, e um *loop* responsável por ir zerando cada posição do vetor.

c) exit

Função que determina o fim do programa com código de retorno, através do `syscall` de saída.

d) draw_board

Responsável por desenhar o tabuleiro e atualizar a cada partida (Figura 6), utilizando das constantes de label `char` definidas na section *.data*. Ele desenha o caractere 'X', '0', ou hífen (“-”), no caso de um espaço vazio, seguido de uma barra vertical (“|”), e a cada três posições insere uma string divisória, separando as linhas.

```
Vez do computador...

 X | - | -
---|---|---
 O | X | -
---|---|---
 - | - | O

Insira a linha a para jogada (1 - 3): 1
Insira a coluna a para jogada (1 - 3): 2
□
 X | X | -
---|---|---
 O | X | -
---|---|---
 - | - | O

Vez do computador...

 X | X | O
---|---|---
 O | X | -
---|---|---
 - | - | O

Insira a linha a para jogada (1 - 3): 3
Insira a coluna a para jogada (1 - 3): 2
□
 X | X | O
---|---|---
 O | X | -
---|---|---
 - | X | O
```

Figura 6: print do tabuleiro atualizado.

e) check_winner

A função `check_winner` determina a vitória comparando o vetor atual com as máscaras determinadas na section *.data*. Ela retorna $v0 para a `main`. Caso o valor seja 1, há a vitória do jogador do *round*, senão o valor é colocado como 0.

f) move_player

A função define o movimento a ser realizado pelo jogador, recebendo os valores de linha e coluna. Enviando-os para outra label para checar se a posição do vetor já está ocupada e, caso não, o valor da posição é alterado para 1, colocando o X na posição do tabuleiro. (Figura 7)

```
Insira linha e coluna para jogada: 1
1
 X | - | -
---|---|---
 - | - | -
---|---|---
 - | - | -
```

Figura 7: atualização do tabuleiro, com o jogador X.

g) move_ai

A função define o movimento a ser realizado pela inteligência artificial, aleatorizando, quando não existe melhor jogada, valores de linha e coluna. Caso seja uma possível vitória, a *ia* preenche a lacuna necessária para vitória (Figura 8), alterando o valor do vetor para 1, vencendo sempre que possível. (Figura 9) Quando identifica que o jogador tem a possibilidade de vencer na próxima rodada, opta por uma jogada defensiva.

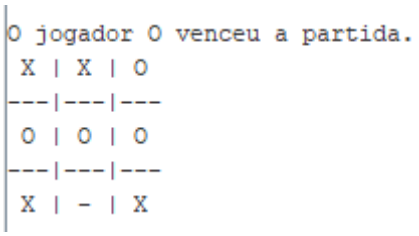

Figura 8: Vitória da inteligência artificial.

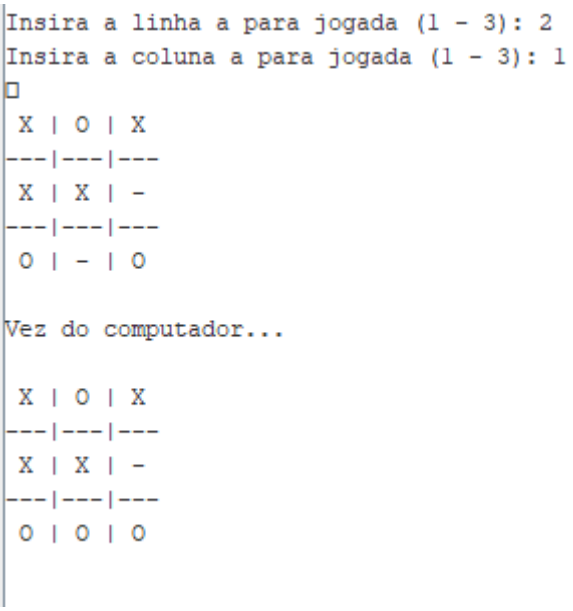

Figura 9: Rodada do jogador e do computador.

REFERÊNCIAS

Software MARS *(MIPS Assembler and Runtime Simulator)*.

Linguagem de montagem *Assembly*.

PATTERSON, David A.; HENNESSY, John L. **Organização e projeto de computadores : a interface hardware/software.** 5. ed. [S.l.]: Elsevier, 2017.