Implementação do jogo Batalha Naval na linguagem Assembly

Allan Felipe Lemes

[1] Departamento de Informática, Centro de Ciências Exatas e Tecnologia, UCS
Rua Francisco Getúlio Vargas, 1130, 95001-970, Caxias do Sul, RS, Brasil

{aflemes}@ucs.br

Resumo. *Neste artigo é descrito toda a implementação do jogo Batalha Naval desenvolvido na linguagem Assembly 8086.*

1. Problema

O problema proposto é o desenvolvimento do jogo Batalha Naval na linguagem Assembly 8086. O jogo foi desenvolvido para ser jogado entre dois jogadores, onde um jogador é uma pessoa e o outro jogador é o próprio programa. Cada jogador poderá informar a posição de 5 embarcações de tamanhos diferentes em uma matriz 10x10, o jogo é divido em turnos, ou seja, a cada novo turno o jogador poderá informar a posição que deseja efetuar o tiro, informando a coluna e a linha, após isso, será a vez do outro jogador fazer o mesmo processo. Vence o primeiro jogador que afundar todas as embarcações adversárias.

2. Solução

Nesta seção será descrito todos os detalhes do código do programa, tais como ele foi desenvolvido, os métodos que compõe o programa, as suas variáveis entre outras coisas.

2.1. Algoritmo

Ao iniciar o desenvolvimento deste projeto, me deparei com que eu acredito ser o grande desafio da implementação (além da adaptação a esse tipo de linguagem), pensar na melhor forma de armazenar as embarcações, tanto do jogador, quanto do computador. Poderia armazenar as embarcações em um vetor simples, onde cada posição a embarcação esteja identificada na parte baixa e parte alta (exemplo: registrador AX, parte baixa (AL), armazena a coluna e na parta alta (AH) a linha), porém para atualizar a informação seria muito complexo e suscetível a falhas. Porém entendo que a melhor forma de armazenar essas informações é em uma matriz, tal qual está previsto no problema. Porém em Assembly não temos o tipo de dado 'matriz', o que podemos fazer é reservar um espaço pré determinado de memória de um determinado tipo, no qual o denominamos de vetor, e é isso que foi feito. Como o problema apresentado requer uma matriz 10x10, o nosso vetor possui 100 posições, onde a cada 10 posições é atribuído como sendo uma nova linha.

Para inicializar este vetor, solicitamos as posições da embarcação para o jogador, primeiro solicitando a coluna, depois a linha e por fim a sua direção (V para vertical e H para horizontal). Depois, com base nessas informações, conseguimos encontrar a primeira posição em que a embarcação ficará armazenada, para isso multiplicamos a linha informada por 10 (por ser uma matriz 10x10), e após, adicionamos o valor referente a coluna, por fim, temos o valor que corresponde ao deslocamento dentro deste vetor. Agora que já sabemos a posição inicial da embarcação, precisamos saber a sua direção, caso

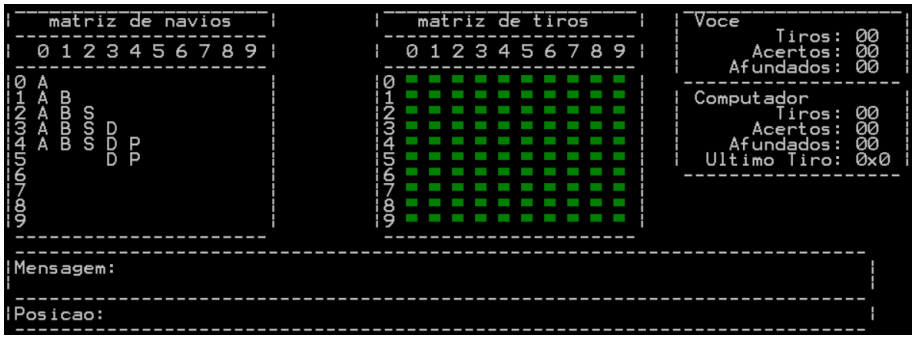

Figure 1. Imagem ilustrando a tela principal contendo as embarcações do jogador

for horizontal, apenas será necessário armazenar o respectivo caractere que identifica essa embarcação (ver Tabela 1) sequencialmente, exemplo: Embarcação ”Destroyer”, tamanho 3, posição inicial 45. Dentro do vetor ele ficará dessa forma, [45] = ”D”, [46] = ”D”, [47] = ”D”. O mesmo acontece caso a embarcação fique na vertical, porém neste caso, as posições são multiplicadas por 10, tomemos o mesmo exemplo de cima, o resultado ficará dessa forma: [45] = ”D”, [55] = ”D”, [65] = ”D”.

Porém antes de armazenar essa embarcação é feita uma consistência dentro do vetor para verificar se a coordenada informada pelo jogador não sobrepõe alguma outra embarcação previamente armazenada. Para isso, o programa acessa todas as posições em que a embarcação deverá ficar e as compara com o valor da posição, caso este valor for 0, a posição está livre, caso contrário, ali existe uma embarcação e o processo é cancelado. Ao encontrar uma embarcação na coordenada informada, o programa retorna ao cliente informando-o e solicitando novas coordenadas. O mesmo acontece para a validação dos limites da matriz, caso a coordenada informada excede o limite da matriz, tanto verticalmente, quanto horizontalmente, o programa solicita novas coordenadas ao jogador.

O mesmo processo descrito acima acontece no caso do posicionamento das embarcações do computador, com o diferencial que não é listado nenhuma mensagem para o computador. No caso do computador, essas posições são geradas aleatoriamente conforme o relógio do computador. Utilizando a interrupção 21h código de função 2C conseguimos obter um valor entre 0 e 99 referente aos milissegundos do relógio do computador e assim dividimos esse valor por 10 e obtemos um valor entre 0 e 9 que primeiramente atribuímos como sendo da coluna e após como sendo da linha. Por fim, a direção horizontal é atribuída caso este valor seja menor que 50, caso contrário, a direção acaba sendo a vertical.

Depois de todo este processo, o resultado obtido é todas as 5 embarcações posicionadas em seus respectivos lugares (no caso do computador essa informação não é impressa em tela) conforme ilustrado na Figura 1 e assim, o jogo pode ser iniciado.

O primeiro a jogar será sempre o jogador humano, o jogo irá solicitar a coluna e a linha do local onde ele deseja efetuar o disparo. Caso o jogador tenha atingido alguma embarcação, na matriz de tiro será impresso o caractere ”o” em verde conforme ilustrado na Figura 2, caso o jogado tenha errado, irá aparecer o caractere ”x” também ilustrado na Figura 2. O mesmo processo ocorre no turno do computador.

Embarcação	Símbolo	Tamanho
Porta Aviões	A	5
Navio Guerra	B	4
Submarinos	S	3
Destroyer	D	3
Barco Patrulha	P	2

Table 1. Tipos de Embarcações

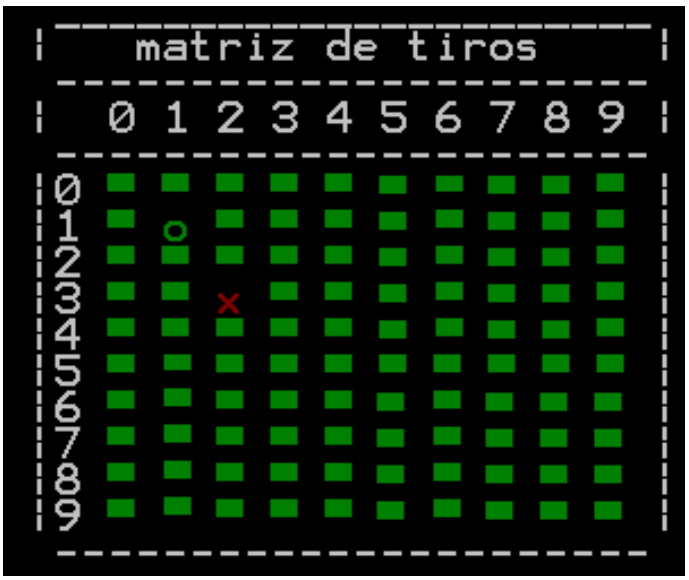

Figure 2. Imagem ilustrando a matriz de tiros com erros e acertos

Para mensurar a quantidade de danos a embarcação foi utilizado um vetor de 5 posições denominado *vet_afundados* ou *vet_afundados_bot* para o computador. Ela é inicializada com o respectivo tamanho da embarcação, conforme pode ser verificado na tabela 1. Onde a posição ”0” se refere a primeira embarcação da tabela e assim sucessivamente. Sempre que alguma embarcação for atingida, esse vetor é decrementado (conforme a embarcação atingida) e quando esse valor chega a 0, essa embarcação passa a ser caracterizada como ”afundada” e essa informação é atualizada no vetor ”afundados” e ”afundados_bot” (no caso do computador). O primeiro a vencer o jogo é aquele que afundar todas as 5 embarcações do adversário primeiro.

2.2. Procedimentos

Nessa subseção será descrito todos os procedimentos do programa.

POS_CURSOR

Este procedimento é responsável por setar o cursor na respectiva posição, utilizando a interrupção 10h código de função 02. Recebe por entrada os registradores BX e DX, onde BH indica a página, DL indica a coluna e DH indica a linha.

LER_CHAR_VIDEO

Procedimento que lê o caractere digitado pelo usuário em determinada posição previamente setada, utilizando a interrupção 10h código de função 08. Não recebe nen-

hum parâmetro de entrada e retorna em AL o caractere digitado.

LER_CHAR

Lê o caractere digitado pelo usuário sem listar em tela, utilizando a interrupção 21h código de função 7, ao fim, retorna em AL o caractere digitado.

LER_KEY

Procedimento responsável por ler a tecla digitado pelo usuário, utilizado para ler a seta direcional nas telas de Menu e Fim de jogo, procedimento utiliza a interrupção 16h código de função 0. Retorna em AL a tecla pressionada.

ESC_SINGLE_CHAR_VIDEO

Neste procedimento é feito o processo de escrever o respectivo caractere em tela, recebe por entrada os registradores AX, BX, onde no registrador AX na sua parte baixa (AL) encontra-se o caractere a ser impresso e na parte alta de BX (BH) a página onde será listada. Utiliza a interrupção 10h código de função 09 e ao fim do processo é impresso o caractere em tela.

ESC_STRING_VIDEO

Escreve determinado caractere em tela recebendo por parâmetro os registradores, BX, CX, DX e BP, onde em BX é utilizado a parte alta (BH) para indicar a página e a parte baixa (BL) para indicar a cor do caractere a ser impresso, no registrador CX é utilizado para indicador o tamanho da String a ser impressa, o registrador DX, é utilizado a parte alta (DH) para indicar a linha e a parta baixa (DL) para indicar a coluna onde será impresso o caractere. Por fim, o registrador BP é onde está localizado a 'String' (vetor de caracteres) que será impresso. Neste método é utilizado a interrupção 10h código da função 13h.

ESC_CHAR

Neste procedimento é feito a escrita do caractere em tela, recebe por parâmetro o registrador DX, e utiliza a parte baixa (DL) para indicar o caractere a ser impresso. Utiliza a interrupção 21h código de função 2.

TROCA_PAGINA

Efetua a troca de página do console, recebe por parâmetro o registrador AX onde na parte baixa (AL) é indicado a página para qual feita a troca. É utilizado a interrupção 10h código da função 5.

LIMPAR

Este procedimento é utilizado para 'limpar' o caractere em tela, não recebe nenhum registrador por parâmetro e utiliza o procedimento ***ESC_CHAR*** para escrever o caractere " " na tela.

LER_ALFABETO

Neste procedimento é feito a leitura de um caractere do alfabeto (entre A...Z) e o escreve na tela (escreve onde está posicionado o cursor), porém permite apenas informar os caracteres: V ou H. Depois de informado o caractere é necessário que seja pressionado "ENTER" para sair do procedimento. Retorna no registrador AX na parte baixa (AL) o

caractere digitado, caso o usuário tenha informado o caractere em maiúsculo, o programa o converte para minúsculo.

LER_NUMERO

Procedimento responsável por ler um número informado pelo usuário e o escrever em tela, permite apenas um número (entre 0..9), para sair do método e finalizar a inserção, o usuário deve pressionar ”ENTER”. Por fim, o número informado fica armazenado no registrador CX, na parte baixa (CL).

SIGLA_EMBARCA

Procedimento responsável por retornar a sigla de determinada embarcação, recebe por parâmetro o registrador AX na parte baixa (AL) o indicador do barco, podendo ser entre 0 e 4. Retorna ao procedimento chamador o registrador DX, contendo na parte alta (DH) o caractere referente a embarcação.

TAMANHO_EMBARCA

Retorna ao programa chamador o tamanho de determinada embarcação, recebe por parâmetro o registrador AX, na parte baixa (AL) a informação referente a embarcação, podendo ser entre 0 e 4. Ao fim, retorna no registrador CX, na parte baixa (CL) o tamanho da embarcação, podendo ser entre 2 e 5.

SOBREPOSICAO

Neste método é feito toda a lógica para verificar se determinada posição informada para a embarcação (ou gerada aleatoriamente quando for no caso do programa) não sobrepõe alguma outra embarcação previamente informada. Também verifica se determinada posição não excede os limites da matriz. Recebe por parâmetro de entrada os registradores, AX, BX, CX, e DX, onde no registrador AX é utilizado a parte baixa (AL) para indicar a embarcação, no registrador BX é utilizado tanto a parte alta (BH) para indicar a coluna quanto a parte baixa (BL) para indicar a coluna. No registrador CX a parte alta (CH) indica a direção da embarcação e na parte baixa (CL) indica o tamanho da embarcação, por fim no registrador DX a parte alta (DH) indica se o procedimento foi executado pelo jogador (00) ou pelo programa (01). Retorna no registrador DX (0) caso não tenha ocorrido erro e 01 quando ocorreu algum erro de sobreposição.

VALIDA_LIMITES

Procedimento responsável por verificar se as coordenadas informadas (ou gerados no caso do programa) não evadem a matriz 10x10. Recebe por parâmetro de entrada os registradores AX, BX, CX e DX, onde no registrador AX a parte baixa (AL) indica qual é a embarcação, no registrador BX a parte alta (BH) indica a linha e a parte baixa (BL) indica a coluna, no registrador CX, a parte baixa (DL) indica a direção da embarcação. Por fim no registrador DX a parte alta (DH) indica se o procedimento foi executado pelo jogador (00) ou pelo programa (01). Retorna no registrador DX (0) caso não tenha ocorrido erro e 01 quando ocorreu algum erro de sobreposição.

POSCIONA_BARCO

Neste método é feito o posicionamento da embarcação na tela do jogador, recebe por entrada os registradores BX, SI, no caso do registrador BX a parte alta (BH) indica a

Figure 3. Imagem ilustrando a console do programa

linha, na parte baixa (BL) indica a coluna, o registrador SI é utilizado para armazenar a direção da embarcação. Retorna no registrador DX a nova posição a ser impressa em tela.

DESENHA_BARCOS

Procedimento responsável por listar em tela a respectiva embarcação, tanto na matriz do jogador, quando (eventualmente) na matriz do tiros. Recebe por parâmetro de entrada os registradores AX, BX e CX, onde no registrador AX, a parte alta (AH) se o procedimento foi executado pelo jogador (00) ou pelo programa (01), a parte baixa (AL) indica qual a embarcação. O registrador BX, na parta alta (BH) indica a coluna e na parta baixa (BL) indica a linha da embarcação, por fim, o registrador CX indica a direção da embarcação. Ao fim do processo é listado na matriz do jogador a respectiva embarcação.

MSG_CONSOLE

Neste procedimento é feito a impressão de uma string na parte denominada ”console”, essa parte fica na parte inferior do programa. Recebe por entrada os registadores BP e CX, onde BP armazena a posição inicial da string a ser impressa e em CX o tamanho dessa string.

LISTA_ERRO_SOBREPOSICAO

Procedimento responsável por listar a mensagem informando a sobreposição das embarcações para o jogador. Não recebe nenhum parâmetro de entrada.

BUSCA_NOME_BARCO

Procedimento responsável por buscar o nome da embarcação, onde os nomes podem ser: ”Porta Avião”, ”Submarino”, ”Destroyer”, ”Navio de Guerra” ou ”Barco Patrulha”. Recebe o registrador AX, como parâmetro de entrada onde a parte baixa (AL) indica qual é a embarcação. Por fim retorna no registrador BP, a posição inicial da string referente a embarcação.

LISTA_CONSOLE

Lista o ’console’ para o jogador. Console é o local onde serão listados as mensagens bem como requisitado as informações ao jogador, na Figura 3 é ilustrado este console. Não recebe nenhum parâmetro de entrada.

LISTA_MOLDURA_PRINCIPAL

Neste procedimento é feito a listagem de toda a moldura da tela principal do programa. Na figura 4 é ilustrado essa moldura.

PEDE_INFORMACOES

Neste procedimento é feito a listagem de mensagens em tela e também é solicitado a informação. Recebe por entrada o registrador AX onde a parte baixa (AL) indica qual é a mensagem, por fim, no registador CX é retornado o caracter digitado.

Figure 4. Imagem ilustrando a moldura principal

ARMAZENA_BARCOS

Procedimento responsável por armazenar as embarcações na matriz do jogador, recebe por entrada os registradores AX, BX, CX e DI, onde no registrador AX a parte baixa (AL) indica qual é a embarcação, o registrador BX indica a posição da embarcação onde a parte alta (BH) indica a linha e a parte baixa (BL) indica a coluna, O registrador CX indica a direção da embarcação e o registrador DI indica a posição inicial do vetor que será armazenado.

INIT_BARCOS_PLAYER

Procedimento responsável por inicializar as embarcações do jogador, nela são solicitados as informações da embarcação bem como a sua validação respeitando os limites da matriz e se não houve sobreposição e também os armazena no respectivo vetor. Ao fim deste procedimento, o resultado esperado é que todas as 5 embarcações tenham sido devidamente validadas e armazenadas.

RANDOM_NOVO

Procedimento responsável por gerar um valor aleatório com base no relógio do computador. Neste procedimento é utilizado a interrupção 21h código de função 2C. Retorna no registrador DX, na parte baixa, o valor ”aleatório” entre 1 e 100.

GERA_POSICOES_ALEAT

Neste procedimento é feito toda as tratativas para retornar as devidas posições aleatórios para que o computador possa armazenar as suas respectivas embarcações. Retorna nos registrador BX e CX, onde no registrador BX a parte alta (BH) indica a coluna, na parta baixa (BL) indica a linha e no registrador CX indica a coluna.

ARMAZENA_SEC

Procedimento secundário para armazenar em um vetor as posições gerados aleatórias gerada pelo computador. Recebe por parâmetro de entrada os registradores AX, BX e CX onde em AX, a parte baixa (AL) indica qual é a embarcação, no registrador BX, a parte baixa (BL) indica a coluna e na parte alta (BH) indica qual a linha, por fim, no registrador CX, indica qual é a direção da embarcação.

INICIALIZA_BOT

Neste procedimento é feito todos os passos necessários para inicializar as embarcações do computador, desde gerar os valores aleatórios, validar as embarcações e as armazenar. Não recebe nenhum registrador de entrada. Ao fim do processo é esper-

Figure 5. Imagem ilustrando a tela principal

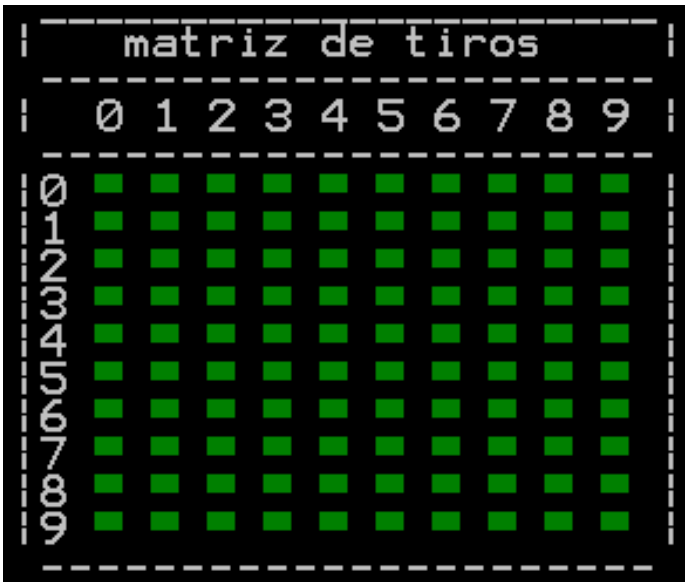

Figure 6. Imagem ilustrando a matriz de tiros

ado que todas as 5 embarcações tenham sido devidamente inicializadas.

TELA_INICIAL

Método destinado a listar a tela inicial da aplicação. Na figura 5 é possível visualizar a tela inicial da aplicação.

MAT_TIROS

Neste procedimento é inicializado a matriz de tiros da aplicação, onde cada posição da matriz deve ser inicializa com um quadrado verde. Na figura 6 é possível visualizar como é o resultado deste método.

SOLICITA_TIRO_PLAYER

Procedimento responsável por solicitar ao jogador as coordenadas para efetuar o disparo, Não recebe nenhum parâmetro de entrada, porém retorna o registrador BX onde a parte alta (BH) retorna a coluna (informada pelo jogador) e a parte baixa (BL) retorna a linha, essa também informada pelo jogador.

DESENHA_TIRO

Método responsável por imprimir o 'tiro' do jogador na matriz de tiros (no caso onde estão armazenados as embarcações do computador). Este método recebe os registradores AX, BX, CX e DX de entrada, onde no registrador AX, a parte baixa é onde está armazenado o caractere que será listado, podendo ser "o" caso encontrou uma embarcação ou "x" caso não tenha acertado a embarcação. O registrador BX, na parte baixa (BL) é armazenado a cor do caractere que será impresso, podendo ser vermelho caso tenha errado e verde, caso tenha acerto a embarcação. NO registrador CX, a parte alta (CH) armazena a posição da linha, e na parte baixa (CL) é armazenado a posição da coluna, por fim, no registrador DX, a parte alta (DH) indica a origem de quem chamou o método, podendo ser "0" caso tenha sido o jogador, o "1" caso tenha sido o computador. O resultado esperado deste método é que o respectivo caractere tenha sido impresso na matriz de tiros e/ou na matriz de embarcação do jogador.

ATUALIZA_TIRO

Procedimento responsável por atualizar o vetor de tiros. Esse vetor de tiros é utilizado para armazenar quantos disparos foram feitos pelos jogadores. Recebe por entrada o registrador DX, onde a parte alta (DH) indica a origem de quem chamou o método, podendo ser "0" caso tenha sido o jogador, o "1" caso tenha sido o computador.

ATUALIZA_ACERTO

Procedimento responsável por atualizar o vetor de acertos. Esse vetor de acertos é utilizado para armazenar quantas partes da embarcação foram acertadas pelos jogadores. Recebe por entrada o registrador DX, onde a parte alta (DH) indica a origem de quem chamou o método, podendo ser "0" caso tenha sido o jogador, o "1" caso tenha sido o computador.

ATT_PLACAR_TIROS

Procedimento responsável por atualizar o placar dos tiros, tanto do jogador quanto do computador. O procedimento recebe por entrada os registradores DX e SI, onde o registrador DX armazena a coluna e a linha de qual elementos será atualizado em tela, já o registrador SI armazena o endereço do vetor onde está localizado as informações. Ao fim do procedimento, o placar de tiros é atualizado.

ATT_PLACAR_TIROS

Responsável por atualizar todos as informações a cerca do jogo da parte destinada aos placares, conforme ilustrado na 7. Dentro do procedimento é atualizado os números referentes aos tiros efetuados pelo jogador, bem como seus acertos e os navios adversários afundados. Por parte do computador são atualizados também os tiros efetuados, os acertos e as embarcações adversárias afundadas e também onde foi efetuado o último tiro, este divido por coluna x linha. Ao fim do processo todos os números do placar serão atualizados.

MSG_REPETIDO

Procedimento destinado a informar ao jogador que ele informou uma coordenada anteriormente informada.

ATT_AFUNDADOS

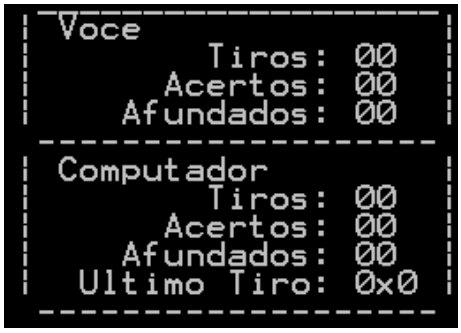

Figure 7. Imagem ilustrando o placar do jogo

Procedimento responsável por atualizar o vetor referente a quantidade de embarcações afundadas, tanto por parte do jogador, quanto por parte do computador. Recebe por parâmetro de entrada o registrador DX, onde na parte alta (DH) indica a origem de quem chamou o método, podendo ser ”0” caso tenha sido o jogador, o ”1” caso tenha sido o computador.

ATT_VET_AFUNDADOS

Neste procedimento é atualizado as informações referente a quantidade de ’vidas’ restantes da embarcação, tanto por parte do jogador quanto do computador. Quando ocorre um acerto, esse procedimento é chamado para verificar se a determinada embarcação foi de fato afundada ou ainda possui vidas restantes. Caso não tenha, o procedimento mencionado acima *ATT_AFUNDADOS* é chamado e atualizado. Este procedimento recebe por parâmetro de entrada o registrador BX e o DX, onde o BX é o deslocamento dentro do vetor referente a matriz do jogador ou do computador. O registrador DX, onde na parte alta (DH) indica a origem de quem chamou o método, podendo ser ”0” caso tenha sido o jogador, o ”1” caso tenha sido o computador.

VERIFICA_TIRO

Procedimento destinado a verificar se as coordenadas informadas pelo jogador (ou pelo computador), acertaram alguma embarcação ou erraram a embarcação. Caso tenha acertado o processo chama todas as rotinas de atualização, desenha o respectivo caractere em tela ”o” e também atualiza o placar. Caso tenha errado, faz o mesmo processo, porém desenha o caractere ”x”. Recebe por parâmetro de entrada os registradores BX, DX e SI, onde no registrador BX a parte alta (BH) indica a coluna informada, já na parte baixa (BL) fica armazenado a linha informada. O registrador DX armazena na parte alta (DH) a origem de quem chamou o método, podendo ser ”0” caso tenha sido o jogador, o ”1” caso tenha sido o computador. Por fim, no registrador SI fica armazenado a posição inicial do vetor de tiros do adversário. Ao fim do processo, o disparo efetuado é computado, impresso e atualizado.

INIT_AFUNDADOS

Responsável por inicializar com 0 o vetor referente as embarcações afundadas, atualiza tanto o vetor destinado ao jogador quanto o vetor destinado ao computador.

ÚLTIMO_TIRO

Neste procedimento é feito a atualização do último tiro efetuado pelo computador,

essa informações é atualizado no Placar. Recebe por parâmetro de entrada o registrador BX, onde na parta alta (BH) está armazenado a linha gerada pelo computador e na parta baixa (BL) a coluna gerada pelo computador. Ao fim do processo o campo ”Último tiro” no placar é atualizado conforme disparo efetuado pelo computador.

TIRO_BOT

Procedimento destinado a efetuar o disparo por parte do computador. Dentro do procedimento é chamado o procedimento de geração de número aleatórios para atribuir o disparo ao computador, após é verificado se o disparo já não ocorreu naquela coordenada e caso não, efetua o disparo atualizando o placar de último tiro.

TIRO_PLAYER

Neste procedimento é solicitado ao jogador as coordenadas para efetuar o disparo, após informado, é verificado se o disparo já não ocorreu anteriormente. Caso não é feito o disparo.

RESET_MATRIZ_TIRO

Procedimento destinado a inicializar o vetor de tiros com 0. É atualizado tanto a vetor do jogador quanto o vetor do computador. Ao fim do processo, ambas os vetores serão inicializados com 0.

RESET_BARCO_PLAYER

Este procedimento é chamado exclusivamente quando na opção de fim de jogo o jogador selecionar a opção ”Reiniciar Jogo”. Esse procedimento coloca as embarcações do jogador tal qual ele informou anteriormente.

ARMAZENA_BARCO_VET

Procedimento destinado a armazenar as posições das embarcações em um vetor. Isso é feito para que futuramente essas informações sejam facilmente acessadas. Recebe por parâmetro de entrada os registradores AX, BX e CX onde no registrador AX na parte baixa (AL) é informado qual a embarcação, no registrador BX, na parte alta (BH) é informado a coluna da embarcação e na parta baixa (BL) a sua respectiva linha. Já o registrador CX armazena a direção da embarcação. Ao fim do processo, o vetor ”vet_matriz_player” é atualizado com as coordenadas da embarcação.

TELA_FIM_JOGO

Este procedimento é chamado quando o jogo acaba, tanto para o jogador quanto para o computador. Nele é listado as opções de ”Novo Jogo” ou ”Reiniciar Jogo”, na 8 podemos ver uma ilustração da tela de fim de jogo no caso do jogador ser o vitorioso.

SOLICITAR_OPCAO

Procedimento destinado a solicitar as opções de fim de jogo ao usuário. Nele é feito toda a ação de movimentação do cursor, para cima e/ou para baixo e também chama o respectivo procedimento conforme a opção selecionada pelo usuário.

INDICADOR_OPCAO

Neste procedimento é feito toda a a ação do indicador ”-¿” seja as ações para cima e/ou para baixo, apagando a seta anterior e colocando na nova posição.

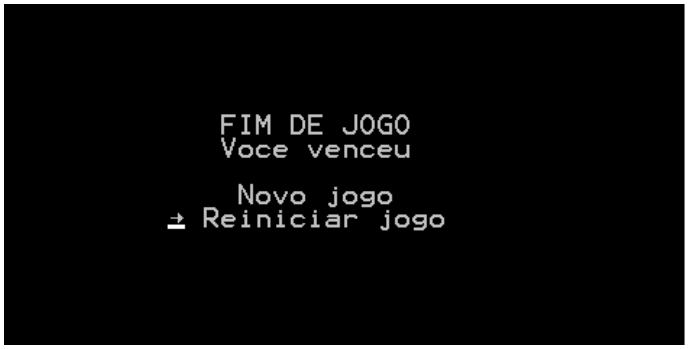

Figure 8. Imagem ilustrando a tela de fim de jogo no caso do jogador for vitorioso

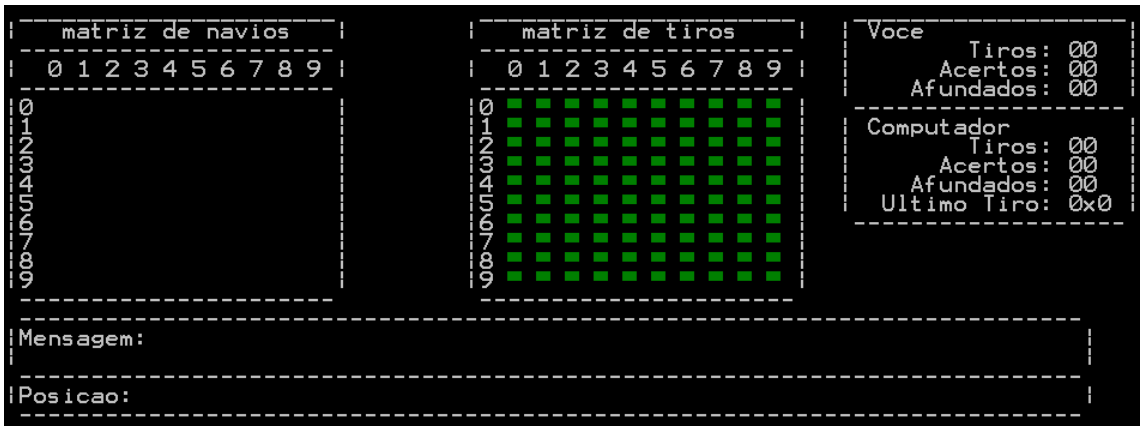

Figure 9. Imagem ilustrando a principal do jogo

INIT_TIROS

Procedimento responsável por inicializar o vetor referente aos disparos efetuados pelo jogador, o procedimento irá setar a quantidade para 0 neste vetor.

INICIO_JOGO

Neste procedimento é feito toda a inicialização do jogo. Ele irá chamar os métodos de inicialização do vetor de tiros, do vetor de afundados e das variáveis auxiliares. Após, irá chamar o procedimento ”tiro_player” que fará todo o processo de solicitar a posição ao jogador, validar e atualizar as informações. Depois disso o procedimento irá verificar se todas as embarcações foram afundadas, tanto por parte do jogador quanto por parte do computador, caso sim, irá chamar o procedimento responsável pelo fim de jogo.

INIT_TELA_PRINCIPAL

Procedimento responsável por listar todos os elementos que compõe a tela principal do jogo, a tela da moldura principal, da console e matriz de tiros, ao fim do processo, todos os elementos estão listados em tela, conforme é ilustrado na 9.

NOVO_JOGO

Procedimento destinado a agregar todos os procedimentos necessário para efetuar a chamada de um novo jogo, chamando os procedimentos de inicialização dos vetores e ou variáveis, listando os elementos na tela e reinicializando as embarcações do computador.

REINICIAR_JOGO

Procedimento destinado a agregar todos os procedimentos necessário para efetuar a chamada de reiniciar o jogo, chamando os procedimentos de inicialização dos vetores e ou variáveis, listando os elementos na tela e reinicializando as embarcações do computador.

2.3. Principais variáveis

Nesta subseção será descrito todas as principais variáveis do programa.

MATRIZ_PLAYER

Essa 'variável' é utilizada como um vetor onde é armazenado as embarcações do jogador, o tamanho desse vetor é 100, ou seja 10x10, onde a cada 10 posições é considerado uma nova linha, por exemplo: A posição 15 deste vetor se refere a posição 1x6 da matriz, linha 1 coluna 6. Nela também são armazenados os disparos efetuados pelo computador.

VET_MATRIZ_PLAYER

Aqui é onde é armazenado a posição das embarcações do jogador. Essa variável é utilizada na opção "REINICIAR_JOGO" para facilitar o posicionamento das embarcações em tela.

MATRIZ_BOT

Essa 'variável' é utilizada como um vetor onde é armazenado as embarcações do computador, o tamanho desse vetor é 100, ou seja 10x10, e funciona da mesma forma que a variável "MATRIZ_PLAYER".

VET_AFUNDADOS

Variável utilizada para armazenar as 'vidas' de cada embarcação do jogador. Por exemplo, ela é inicializada com o tamanho de cada embarcação, a embarcação 0 referente ao Porta aviões começa com 5. A cada vez que o Porta aviões é alvejado, essa variável é decrementada, quando ela chega a 0, a embarcação é declarada como afundada.

VET_AFUNDADOS_BOT

Variável utilizada para armazenar as 'vidas' de cada embarcação do computador. Ela funciona da mesma forma que a variável VET_AFUNDADOS.

TIROS

Essa 'variável' é utilizada para armazenar a quantidade de disparos efetuados pelo jogador.

ACERTOS

Variável utilizada para armazenar a quantidade de embarcações que foram alvejadas pelo jogador.

AFUNDADOS

Variável utilizada para armazenar a quantidade de embarcações afundadas pelo jogador.

TIROS_BOT

Essa 'variável' é utilizada para armazenar a quantidade de disparos efetuados pelo computador.

ACERTOS_BOT

Variável utilizada para armazenar a quantidade de embarcações que foram alvejadas pelo computador.

AFUNDADOS_BOT

Variável utilizada para armazenar a quantidade de embarcações afundadas pelo computador.

3. Conclusão

De fato, programar qualquer coisa em Assembly é muito mais complexo que nas outras linguagens de auto nível, isso se deve ao fato de que todas as facilidades presentes em outras linguagens não estão presentes no Assembly, tudo deve ser feito pelo próprio programador.

Entendo que uma das maiores dificuldades apresentadas para um novo programador em Assembly é se familiarizar com o uso dos registradores (os poucos), saber usar utilizar a pilha adequadamente, colocar e depois retirar na ordem certa. Diversas vezes no decorrer do projeto me vi precisando utilizar mais registradores do que a linguagem fornece, com isso tive de buscar alternativas para resolver o meu problema. Apesar de desafiadora, ela instiga o programador a pensar numa forma melhor de realizar determinada operação, isso, no meu entendimento acarreta que o programa tende a ter menos bugs, pois obriga o programador a entender um pouco mais de linguagem.

Por fim, apesar de todos as dificuldades que eu passei no desenvolvimento, entendo que programar em Assembly não é tão complexo quanto aparenta ser, depois do (longo) período de adaptação a linguagem, ela acaba se tornando uma outra linguagem de programação, com as suas particularidades e sendo um pouco mais complexa que as outras.