UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ

Ciência da computação

Alex Sandro Zarpelon
Bruna Gabriela Disner

Disciplina: Organização de Computadores

Trabalho: Jogo de batalha naval

Chapecó - SC
2021

1. Descrição do problema

O problema consiste em implementar o jogo de batalha naval em uma matriz 10x10, que permite ao jogador inserir embarcações, atirar e verificar a sua pontuação conforme o resultado do seu tiro. Para isso precisamos implementar um algoritmo utilizando funções previamente definidas na descrição do trabalho, criar funções de acordo com a necessidade e desenvolver uma lógica que atenda aos requisitos do jogo.

A função insere embarcações é responsável por inserir os barcos na matriz, é a mais complexa a ser implementada pois também verifica se o comprimento, o valor da linha e coluna estão de acordo com o tamanho da matriz e se já há uma embarcação inserida na posição. Essa função também calcula o valor do deslocamento da memória para a posição que o barco será inserido.

Após a inserção dos barcos, o jogo permite ao jogador digitar as posições em que deseja atirar, o jogo precisa analisar o tiro e mostrar a pontuação que o jogador alcançou e se acertou o tiro, e mostrar na matriz a posição do tiro. Outro problema é analisar a posição do tiro efetuado pelo jogador com a posição que está o barco e mostrar o resultado para o jogador.

2. Solução

Para solucionar o problema utilizamos as seguintes funções:

- insere_embarcacoes: responsável por inserir as embarcações no jogo;
- jogo: responsável pela inicialização do jogo e registrar a pontuação;
- jogar: recebe a posição do tiro e executa as mesma;
- printa_situacao: mostra a pontuação do jogo depois de cada jogada;
- zera_controledebarcos: zera o controle_barcos responsável pelo controle de barcos afundados;
- zera_voce: zera as pontuações do jogo quando vai reiniciar uma nova partida;
- zera_matriz: remove todos os barcos já inseridos na matriz;
- printa_matriz_espiao: mostra para o jogador tudo o que está na matriz;
- printa_matriz_padrao: mostra para o jogador os barcos atingidos e os tiros que não acertaram embarcações;

Explicação, protótipo e algoritmo de cada função:

Função de inserção dos barcos.

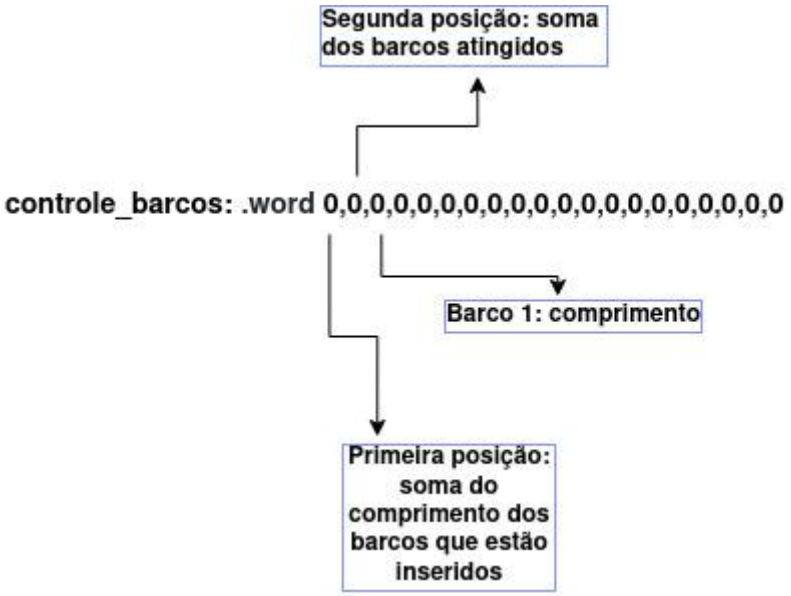
- Prototipo: int insere_embarcacao (int matriz[10][10], int controle_barcos[4], char navios1, char navios2, char navios3)
- Explicação detalhada do que a função faz:
    - A função inicia com a escolha do barco que o jogador deve fazer, para realizar essa tarefa implementamos um switch que através do beq compara a entrada do jogador com o barco escolhido e direciona para a função que irá carregar o barco escolhido para a memória e realizar a inserção da embarcação no jogo.
    - Na função continua_ins carregamos a matriz que vai receber os barcos e também a string com a descrição dos navios na seguinte ordem: primeiro valor é o número de embarcações, após é a disposição 0(horizontal) e 1(vertical), em seguida o comprimento e a linha e a coluna inicial. Podemos observar isso no seguinte exemplo:

```
"2\n0 4 1 1 \n0 4 3 0 "
```

- 2 é o número de embarcações, 0 quer dizer que é na horizontal, 4 é o comprimento, 1 1 significa que inicia na linha e coluna 1.
- Fazemos a contagem de embarcações da string e utilizamos o **addi** para andar duas posições na string e chegar na posição da disposição da embarcação.
- Para podermos percorrer toda a string e inserir os barcos na matriz, implementamos o **FOR (estrutura de repetição)** que vai repetir conforme o número de embarcações. Dentro da estrutura de repetição o primeiro valor que salvamos é a disposição do navio para obter o valor absoluto lido na string, fazemos o valor obtido -48, guardamos o valor em a5 e andamos duas posições na string assim alcançamos o comprimento do navio.
- Novamente avançamos duas posições e guardamos em t4 a linha inicial do navio e utilizando o **bne(condição)** verificamos se a linha é válida utilizando a seguinte lógica:
  - em s9 salvamos o valor 32 que é referente ao espaço na tabela ascii, andamos um byte na string e então verificamos se a posição da string é diferente de s9 que é o espaço. Se for diferente de espaço significa que temos um algarismo consequentemente é maior que 9 então ultrapassa o tamanho da matriz, assim chamamos a função pos_invalida que irá imprimir na tela que a posição é inválida.
  - Exemplo: utilizando esse string podemos observar que o valor da linha é 12 logo o algoritmo vai reconhecer que após o algarismo 1 tem o algarismo 2 que é diferente de espaço e então irá chamar a função pos_invalida.

    Exemplo: navios2: .string "2\n0 4 12 1 \n0 4 3 0
- Novamente pulamos um byte e chegamos no valor da coluna que salvamos em t5, pulamos um byte e fazemos a mesma verificação que descrevemos acima, se é diferente de espaço então a posição é inválida.
- Para sabermos se o jogador já afundou todas as embarcações criamos o vetor controle_barcos na seguinte disposição: primeira posição soma do comprimento de todos os barcos, quando a

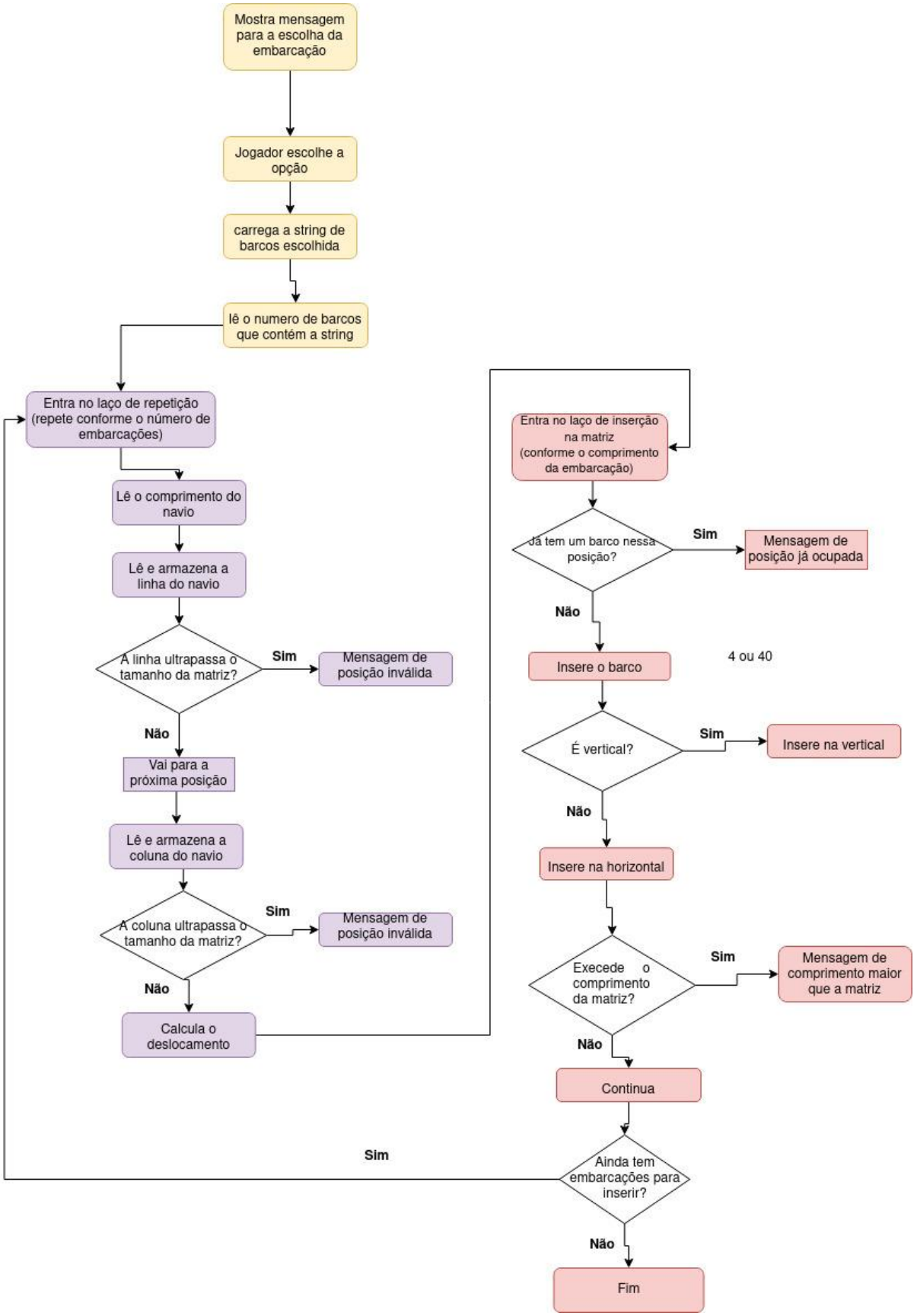
embarcação for inserida na matriz vai ter o seu comprimento somado. Segunda posição é um auxiliar que vai somando os barcos que o jogador atingiu. Se a primeira posição e a segunda se igualarem significa que o jogo acabou. As demais posições são os demais barcos exemplo: barco 1, barco 2, etc.



- Após obtermos todas as informações sobre o barco calculamos o deslocamento utilizando a seguinte fórmula:
- **Deslocamento = (Linha* quantidade de colunas + Coluna) * 4**
- Novamente utilizamos um laço que irá executar conforme o comprimento do barco para assim inserir toda a embarcação na posição correta.
- Antes de inserirmos o barco verificamos se na posição já temos alguma embarcação, utilizamos a seguinte lógica:
  - Se a posição é diferente de 0 então há uma embarcação e portanto chamamos a função de sobre_invalido que irá mostrar a seguinte mensagem **"Posição inválida por estar sobrescrevendo navio existente"**, isso executamos na função corpo_laco_ins_h.
  - Na função incremento_controle_ins_h verificamos se o barco será inserido na horizontal ou na vertical, utilizamos o **beq** se **a5** (variável que contém o valor da disposição) for igual a 0 então

chamamos a função horizontal_ins se for diferente o algoritmo jump(pula) para vertical_ins.

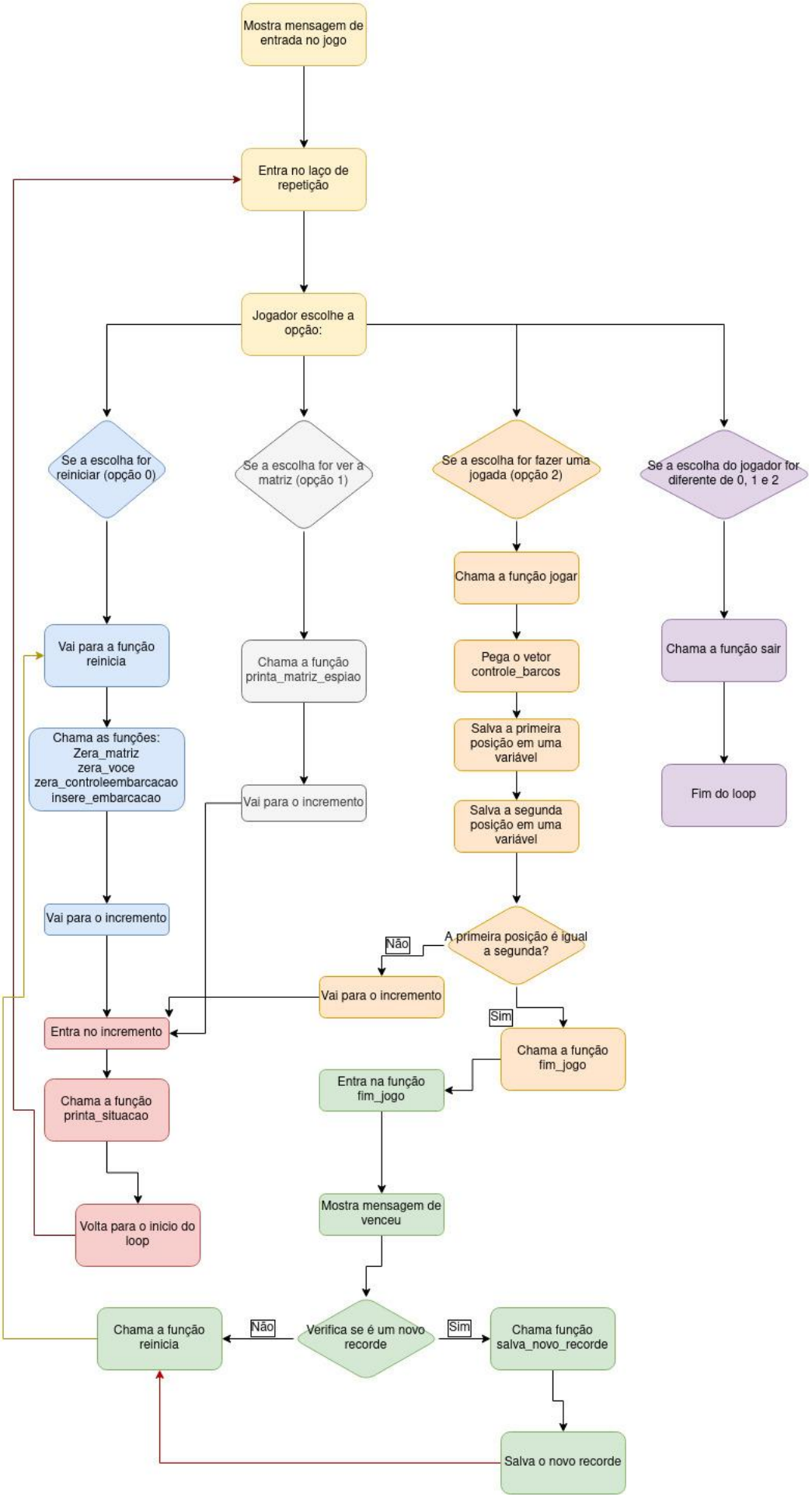
- Dentro das funções horizontal_ins e vertical_ins verificamos se o comprimento está de acordo com o tamanho da matriz.
- Utilizamos a seguinte lógica se for horizontal: coluna inicial + comprimento do navio se o resultado final for maior que 9 então ultrapassa o comprimento da matriz. Se for na vertical, fazemos a linha inicial + comprimento do navio e utilizamos a mesma condição: maior que 9 é invalido e menor que 9 é válido.
- Exemplo: **navios2: .string "2\n0 3 5 8 \n0 4 3 0**, montando a equação 8 + 3 = 11 é maior que 9 sendo assim excede o tamanho da matriz.
- Algoritmo da função:


Mostra mensagem para a escolha da embarcação
Jogador escolhe a opção
carrega a string de barcos escolhida
lê o numero de barcos que contém a string
Entra no laço de repetição (repete conforme o número de embarcações)
Lê o comprimento do navio
Lê e armazena a linha do navio
A linha ultrapassa o tamanho da matriz?
Sim
Mensagem de posição inválida
Não
Vai para a próxima posição
Lê e armazena a coluna do navio
A coluna ultrapassa o tamanho da matriz?
Sim
Mensagem de posição inválida
Não
Calcula o deslocamento
Entra no laço de inserção na matriz (conforme o comprimento da embarcação)
Já tem um barco nessa posição?
Sim
Mensagem de posição já ocupada
Não
Insere o barco
4 ou 40
É vertical?
Sim
Insere na vertical
Não
Insere na horizontal
Execede o comprimento da matriz?
Sim
Mensagem de comprimento maior que a matriz
Não
Continua
Ainda tem embarcações para inserir?
Sim
Não
Fim

Função jogo

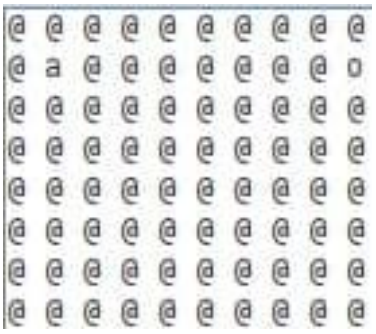
- Protótipo: int jogo(int controle_barcos[20], int voce[5], int recorde[3])
- Explicação detalhada:
    - Essa função é responsável por iniciar o jogo, mostrar a mensagem de boas vindas ao jogador.
    - O jogador tem 3 opções: fazer uma jogada, ver a matriz e reiniciar o jogo. O programa entra em um loop para mostrar as opções ao jogador e a condição de parada é quando o jogador digitar uma valor maior que 3 então irá sair do jogo.
    - Conforme a opção escolhida pelo jogador o programa vai direcionar para a função correspondente.
    - Se a opção for reiniciar, o programa remove as embarcações inseridas na matriz, zera a pontuação e também zera o controle de barcos.
    - Se a opção for visualizar a matriz o algoritmo chama a função printa_matriz_espiao que é responsável por imprimir a matriz;
    - Se a opção for fazer uma jogada o programa chama a função jogar, depois da jogada é verificado se o jogador já afundou todas as embarcações, se afundou todas as embarcações então chama a função fim_jogo_vitoria, caso contrário continua normal à execução.
    - Se vencer o jogo, o programa salva a pontuação. Caso tenha sido um novo recorde, que é a menor quantidade de tiros para afundar todas as embarcações, após reinicia o jogo.
    - Algoritmo:

Função jogo


Mostra mensagem de entrada no jogo
Entra no laço de repetição
Jogador escolhe a opção:
Se a escolha for reiniciar (opção 0)
Se a escolha for ver a matriz (opção 1)
Se a escolha for fazer uma jogada (opção 2)
Se a escolha do jogador for diferente de 0, 1 e 2
Vai para a função reinicia
Chama as funções: Zera_matriz zera_voce zera_controleembarcacao insere_embarcacao
Vai para o incremento
Chama a função printa_matriz_espiao
Vai para o incremento
Chama a função jogar
Pega o vetor controle_barcos
Salva a primeira posição em uma variável
Salva a segunda posição em uma variável
A primeira posição é igual a segunda?
Não
Vai para o incremento
Sim
Chama a função fim_jogo
Chama a função sair
Fim do loop
Entra no incremento
Chama a função printa_situacao
Volta para o inicio do loop
Entra na função fim_jogo
Mostra mensagem de venceu
Verifica se é um novo recorde
Não
Chama a função reinicia
Sim
Chama função salva_novo_recorde
Salva o novo recorde

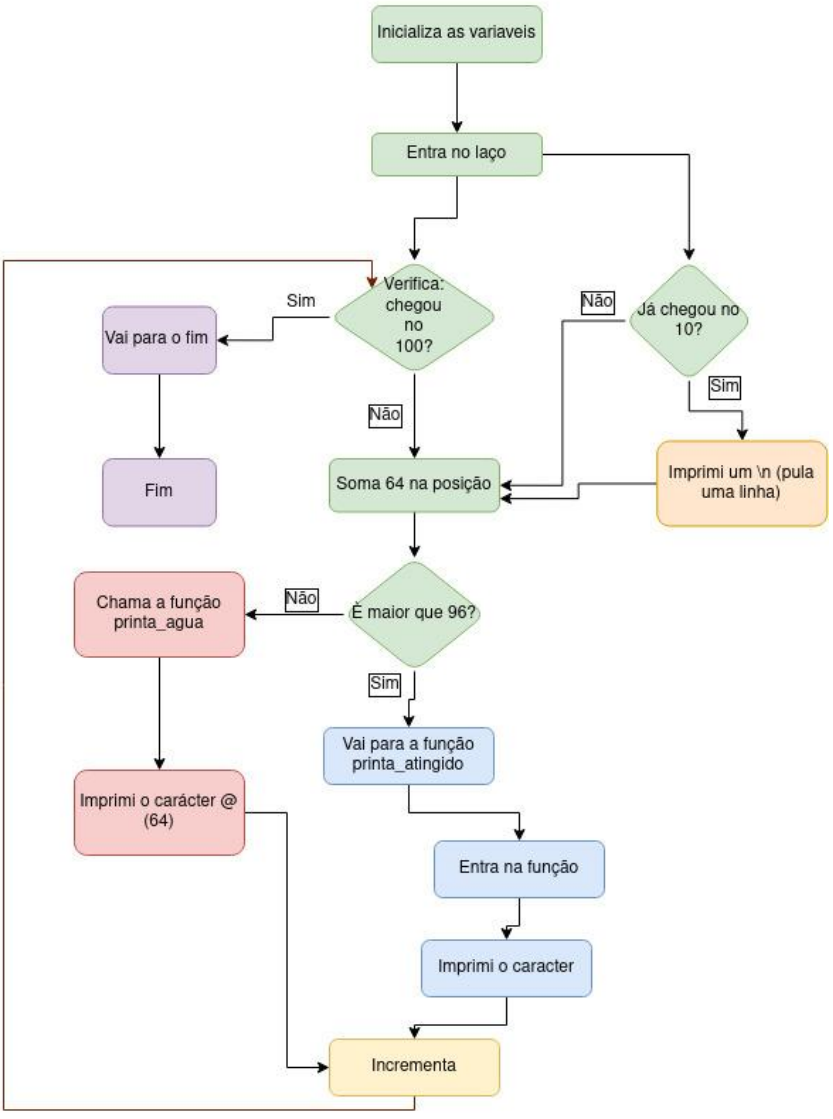
Funções para imprimir a matriz

- Implementamos 3 funções que imprimem a matriz, cada uma tem sua funcionalidade.
- Função printa_matriz_padrao**:**
  - Protótipo: void printa_matriz_padrao (int matriz[10][10]).
  - Funcionalidade: mostrar para o jogador se o tiro atingiu uma embarcação ou se atingiu a água. Se o tiro atingiu uma embarcação está marcado por uma letra minúscula “a”, “b” ou “c”, e se caiu na água marcado pela letra “o”.
  - Explicação do código: criamos 2 variáveis para controle, t1 que vai até 100 pois é uma matriz 10x10 e t3 que vai até 10 que é o tamanho da linha;
  - Utilizamos um loop para imprimir toda a matriz cuja condição de parada é 100, quando chega em 10 o programa dá um \n (quebra de linha);
  - No corpo do laço usamos a seguinte condição:
    - se a posição da matriz valor 64 (carácter @ conforme a tabela ASCII) for maior que 96 (a partir de 96 começa os caracteres a, b, c …) então a posição foi atingida, chamamos a função printa atingido que irá imprimir o carácter conforme a posição do tiro. Se o tiro não atingiu nada então printamos água (carácter @).
    - Após o programa dá o espaço (32 conforme a tabela ASCII) e incrementa, assim vamos repetindo até acabar toda a matriz.
    - Exemplo do print da matriz: conforme a imagem, um tiro acertou uma embarcação representado pela letra a e outro caiu na água representado pela letra o.



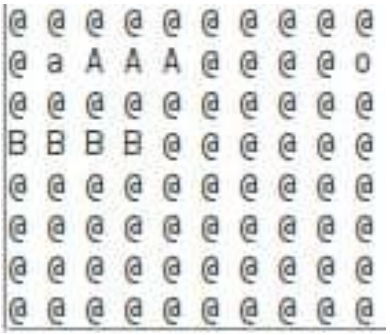
- Algoritmo:

Função de imprimir matriz



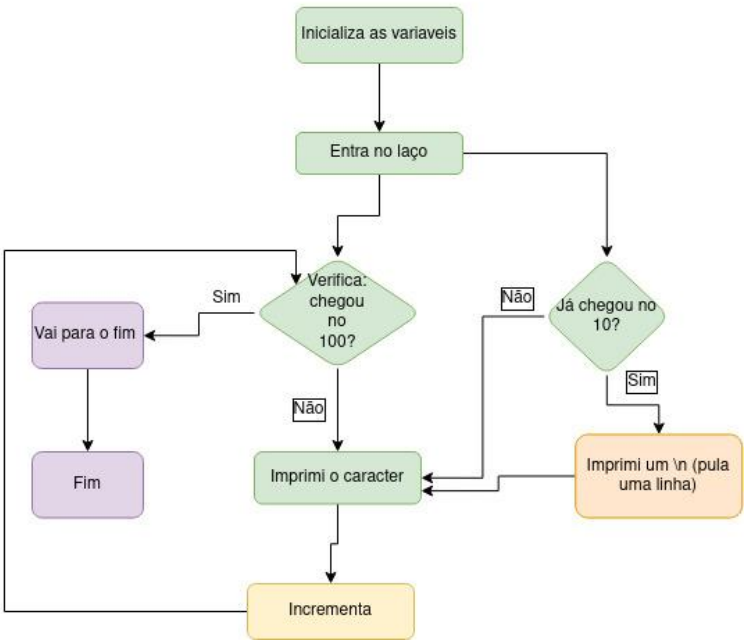
- Função print_matriz_espiao**:**
  - Protótipo: void printa_matriz_espiao (int matriz[10][10]).

- Funcionalidade: vai imprimir tudo o que está na matriz, a posição das embarcações e a posição dos tiros.
- A diferença para a função descrita acima é que não teremos a condição de ser maior que 94, pois iremos imprimir tudo o que está na matriz.
- Exemplo: podemos observar a posição dos barcos e os tiros.



- Algoritmo:

Função de imprimir matriz espião



- Função printa_situacao**:**
  - Protótipo: int printa_situacao(int matriz[10][10], int voce[5], int recorde[3])
  - Funcionalidade: essa função vai ser chamada logo após a jogada para mostrar a pontuação do jogo.

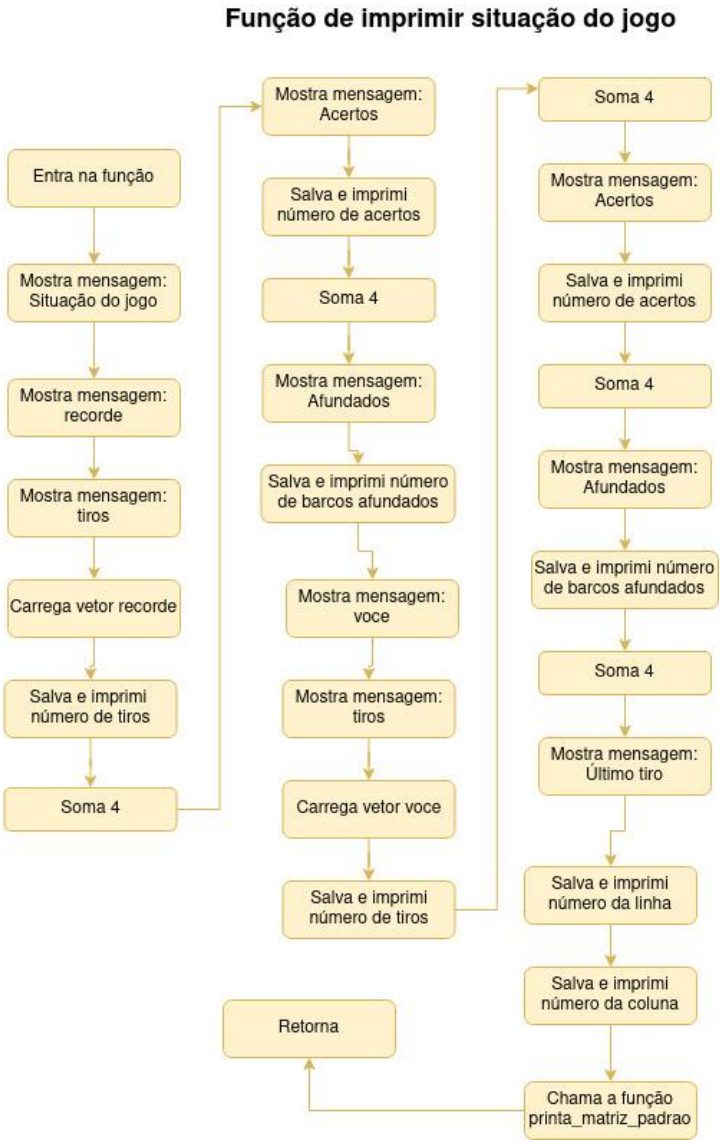
- A lógica da função é ir chamando a mensagem e ir salvando no vetor a pontuação conquistada, para isso sempre o programa vai andando uma posição.
- No vetor voce salvamos as seguintes informações:

```
Você
        Tiros: 1
        Acertos: 1
        Afundados: 0
        Último Tiro: 3 5
```

- No vetor recorde salvamos as seguintes informações:

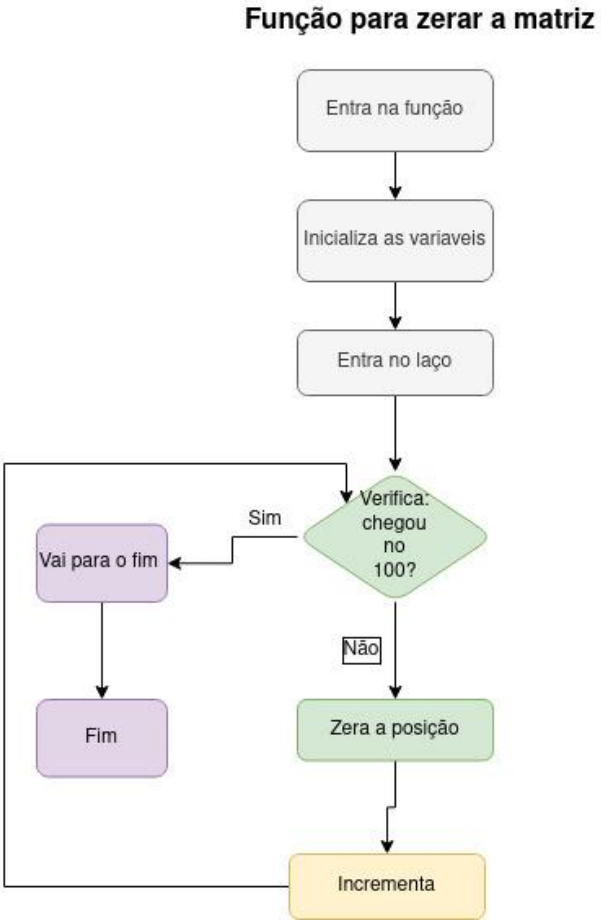
```
Recorde
        Tiros: 99
        Acertos: 99
        Afundados: 99
```

- Algoritmo:



Funções para reiniciar o jogo

- Função para zerar a matriz:
    - Protótipo: int zera_matriz(int matriz[10][10]);
    - Funcionalidade: remover todos os barcos da matriz para uma nova partida.
    - Lógica: para conseguir remover todas as embarcações utilizamos um laço de repetição que vai até 100 por conta do tamanho da matriz, em cada posição é colocado um 0 e no incremento adicionamos 4 para a próxima posição. Dessa forma zeramos todas as posições da matriz.
    - Algoritmo:



- Zerar a pontuação do jogo:
    - Protótipo: int zera_voce(int voce[5]);
    - Funcionalidade: zerar a estatística para reiniciar uma nova partida;

- Lógica: para zerar esse vetor utilizamos a lógica de ir de posição em posição, para fazer isso vamos somando 4 para conseguir de posição em posição.

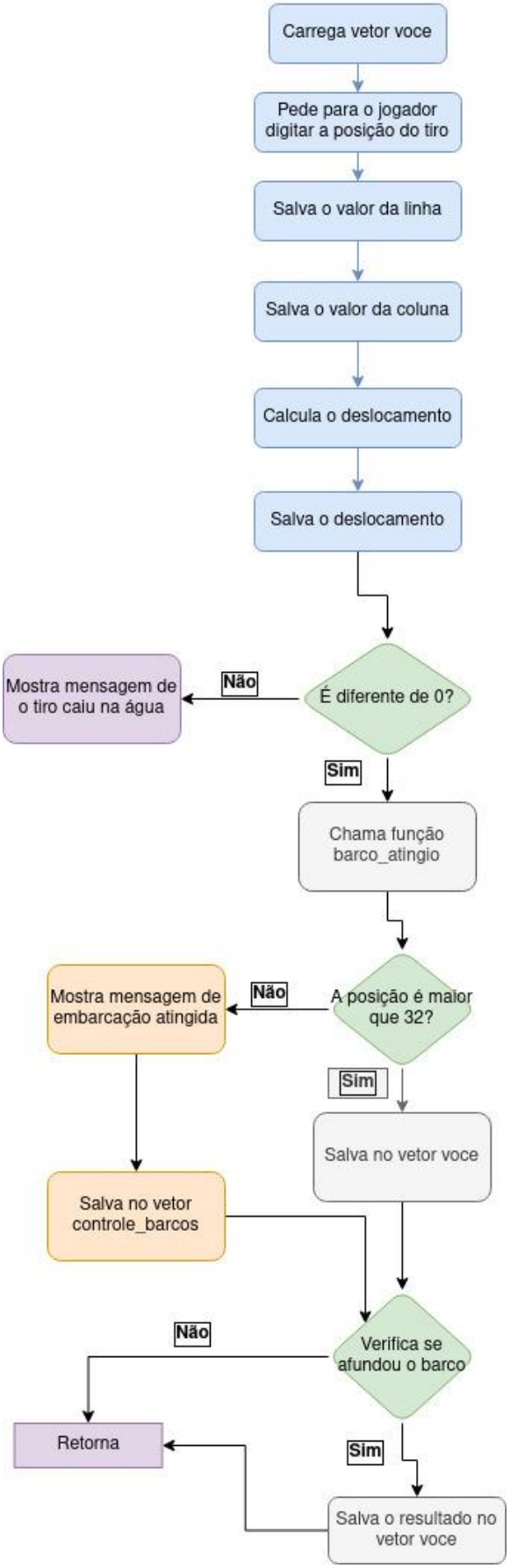
- Zerar o controle de barcos:
  - Protótipo: int zera_controledebarcos(int &controle_barcos[20]);
  - Funcionalidade: zerar o vetor controle de barcos;
  - Lógica: para zerar esse vetor utilizamos um laço de repetição que vai até o tamanho do vetor, nesse caso 20, e em cada posição é colocado um zero.
  - Algoritmo:

Função jogar

- Protótipo: int jogar(int voce[5], int matriz[10][10])
- Funcionalidade: solicitar as coordenadas do tiro e verificar se atingiu embarcação ou se atingiu água.

- Lógica: para salvar a pontuação do jogo utilizaremos o vetor voce de 5 posições que irá salvar as seguintes informações: tiros, acertos, afundados e a posição do último tiro (linha e coluna).
- A função inicia pedindo para o jogador inserir a posição do tiro, linha e coluna. Após o programa salva a linha e a coluna e já adiciona na última posição do vetor.
- Então com esses valores o programa calcula o valor do deslocamento com a mesma fórmula usada na função de inserir embarcações. Se a posição do tiro for diferente de 0 então o algoritmo chama a função barco_atingido.
    - Na função barco_atingido verifica se já tem algum valor somado na posição.
    - Se já tiver significa que já temos um tiro somado então o programa incrementa mais 1 na posição do tiro no vetor voce e depois verifica se afundou algum barco.
    - Se não tiver valor somado, então o programa soma o barco atingido no vetor controle_barcos (na segunda posição) e vai para o fim_jogar que verifica se o barco afundou.
    - Se afundou salva no vetor voce a contagem de afundados e vai para o fim.
- Se não atingiu um barco, então verifique se já tem algo somado nessa posição ou seja se é maior que 32.
    - Se tiver mostra a mensagem que o tirou caiu na água.
    - Se o valor for menor que 32 então soma 47 que depois será somado com 64 na função de imprimir, assim o valor será 111 que é o caráter **o**(na tabela ASCII) que representa tiro na água.
- Algoritmo:

Função jogar



3. Conclusões

O trabalho Batalha Naval foi um grande desafio para a implementação completa, sem ter o conhecimento de todos os recursos que seriam usados previamente, demandou várias trocas de E-mail com o professor para que fosse possível sanar todas as dúvidas, já que na internet geralmente não foi possível encontrar respostas sobre uso do conjunto de instruções RISC-V. Talvez tenha sido um erro não consultar o manual apresentado pelo professor, porém pode ser que não seja a forma mais eficiente de obter a resposta para pequenas e específicas dúvidas, ao qual demandaria longas leituras sem uma resposta clara, ou seja um mau hábito e dificuldade para ambos, mesmo.

Desconsiderando dúvidas, a implementação de fluxo do programa foi complexa e nada fácil, de fato, porém, o trabalho anterior forneceu grande conhecimento e facilidade para que a programação fosse possível. As dificuldades foram com uso de recursos ainda desconhecidos, e principalmente a *debugação* de um programa longo e complexo, escrito em um conjunto de instruções em vez de uma linguagem de alto nível. Se há uma certeza fazendo este trabalho, é que o Assembly nunca funcionaria de primeira tendo em vista as longas funções escritas antes do teste de execução, o que demandava longos períodos de concertos no código analisando o fluxo apresentado pelo simulador Rars.

Como aprendizado, o jogo Batalha Naval deixou instruções decoradas em mente (pelo menos as que foram usadas com frequência); uma facilidade maior ainda em programar em Assembly RISC-V, dada a experiência com o trabalho o qual é grande e complexo; um grande entendimento em como manipular memória para prints, inputs, inserções em vetor. Todos detalhes que não estavam tão claros nos trabalhos anteriores.

Conclui-se por fim que o trabalho Batalha Naval foi longo, desgastante e um grande produtor de entendimentos lógicos, os quais seriam várias vezes mais triviais em programas de alto nível, que não requerem o desenvolvimento de uma solução matemática para resolver um problema simples. Também foi um grande vetor para conhecimento do funcionamento de processadores em geral, fornecendo entendimento de como instruções implementadas em um processador, independente de qual sua arquitetura, são usadas para criar algo prático, não só um

programa direto, mas nos faz imaginar como usá-las para criação de uma linguagem de alto nível também.

Link do repositório no github:
https://github.com/alexzarp/OrganizacaoDeComputadores/tree/main/BatalhaNaval

4. Código fonte

```
# Trabalho feito por:
# Alex Sandro Zarpelon - 1911100039
# Bruna Gabriela Disner - 1911100007
   .data
matriz:      .word     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
controle_barcos:    .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
recorde:     .word     99,99,99 # tenho que partir de um valor maior que
0 para comportar a condição de salvamento
voce:        .word     0,0,0,0,0
space:       .space      46
space_4:     .space      4
situacaojogo_msg:   .string     "A sua situção de jogo atual se encotra
da forma:\n"
recorde_msg: .string     "Recorde\n"
voce_msg:    .string     "Você\n"
tiros_msg:     .string    "\tTiros: "
acertos_msg:   .string    "\tAcertos: "
afundados_msg:  .string     "\tAfundados: "
ultimotiro_msg:    .string     "\tÚltimo Tiro: "
navios1:      .string     "4\n0 4 1 1 \n0 4 3 0 \n1 2 7 3 \n1 5 3 5 " #
disposicao, comprimento, linha inicial, coluna inicial
navios2:      .string     "2\n0 4 1 1 \n0 4 3 0 " # posicao invalida de
teste
navios3:      .string     "1\n0 4 1 1 " # comprimento invalido de teste
msg_1:       .string     "Escolha o conjunto de posicionamento dos
navios (1, 2 ou 3): "
invalida_fora:      .string     "Posição inválida por estar fora da
matriz"
invalida_sobreposto:  .string    "Posição inválida por estar
sobreescrevendo navio existente"
invalida_maior:       .string     "Posição inválida pois o navio é
maior que a matriz"
msg_2:       .string     "Bem vindo à batalha naval!\n"
menu_jogo:        .string     "O que deseja fazer agora?\n0 - Reiniciar
o game\n1 - Mostrar o estado da matriz(espião)\n2 - Fazer uma jogada\n3
- Terminar o jogo\n$ "
```

```
tiro:       .string     "Insira as posições de tiro (linha coluna): "
atingiu:    .string     "Barco Atingido!\n"
errou:      .string     "O tiro caiu na água!\n"
afundou:    .string     "Você afundou o barco!\n"
terminou:   .string     "Você venceu! Todos os barcos foram
afundados!\n"
# br_n:       .string     "\n"
# space:      .string     " "
   .text
main:
   jal jogo
   j fim

# a função "insere_embarcações" recebe como parâmetro a "matriz" para
inserção dos barcos,
# "controle_barcos" para registro do comprimento e comprimento de soma
total dos barcos para fins de se afudou ou não,
# "navios1", "navios2" e "navios3" que descrevem como os barcos serão
colocados na "matriz"
insere_embarcacoes:
   la a0, msg_1
   li a7, 4
   ecall # mostra a mensagem para escolha de um conjunto de barcos

   li a7, 5 # a0 é o int
   ecall # solicita o inteiro

   addi t0, zero, 1
   beq a0, t0, carrega1
   addi t0, zero, 2
   beq a0, t0, carrega2
   addi t0, zero, 3
   beq a0, t0, carrega3 # switch que leva para o conjunto de barcos
escolhido

   carrega1: # -------
       la a1, navios1
       j continua_ins
   carrega2:
       la a1, navios2
       j continua_ins
   carrega3:
       la a1, navios3
```

```
   li a0, 10
   li a7, 11
   ecall # ------ continuação do switch de escolha do conjunto de
barcos

   continua_ins:
   addi s11, zero, 1 # contagem barco

   la a2, matriz # a2 navios
   lw a3, (a2) # matriz onde vai os navios
   lb a4, (a1) # string com a descricao dos navios

   addi a4, a4, -48 # toda vez que se faz -48, é para obter o valor
absoluto lido da string
   add t0, a4, zero # contagem de navios

   addi a1, a1, 2
   addi s9, zero, 32 # espaco na tabela ascii
   la s6, controle_barcos # para afundamento de barcos
   addi s6, s6, 8
   la s5, controle_barcos # para fim da partida (todos barcos
afundados)
   teste_condicao_ins:
       beq t0, zero, fim_ins
   corpo_laco_ins: # nessa parte é um loop que obtem a informação de
cada barco da string
       lb a4, (a1) # string com a descricao dos navios
       addi a4, a4, -48
       add a5, a4, zero # disposicao do navio

       addi a1, a1, 2
       lb a4, (a1) # string com a descricao dos navios
       addi a4, a4, -48
       addi t3, a4, 0 # comprimento do navio

       addi a1, a1, 2
       lb a4, (a1) # string com a descricao dos navios
       addi a4, a4, -48
       addi t4, a4, 0 # linha do navio
       addi a1, a1, 1
       lb a4, (a1)
```

```
        bne a4, s9, pos_invalida # invalidação por poscionamento fora da
matriz

        addi a1, a1, 1
        lb a4, (a1)
        addi a4, a4, -48
        addi t5, a4, 0 # coluna do navio | t0 = contagem de navios, t3 =
comprimento do navio, t4 = linha, t5 = coluna
        addi a1, a1, 1
        lb a4, (a1)
        bne a4, s9, pos_invalida # invalidação por poscionamento fora da
matriz

        sw t3, (s6)
        addi s6, s6, 8 # informações úteis para controlarmos quando um
barco será afundado

       # Deslocamento = (L * QTD_colunas + C) * 4
       # com os dados obtidos, calculamos e deslocamos para a posição
inicial correta na matriz
        addi t6, zero, 10
        addi s3, zero, 4
        mul s0, t4, t6 # multiplicação da linha por 9(número de colunas
da matriz)
        add s1, s0, t5 # soma de s0 com a coluna
        mul s0, s1, s3 # resultado final do deslocamento

        add a2, a2, s0

        teste_condicao_ins_h: # este loop é responsável inserir o barco
conforme seu comprimento e diposição
            beq t3, zero, fim_ins_h
        corpo_laco_ins_h:
            lw a3, (a2)
            bne a3, zero, sobre_invalido # aqui é validado se o barco
não sobreescreve outro já existente
            sw s11, (a2)
            lw s4, (s5)
            addi s4, s4, 1
            sw s4, (s5) # estou contando as posições do barco para no
final saber quando o jogo terminou
        incremento_controle_ins_h:
            addi t3, t3, -1
```

```
            beq a5, zero, horizontal_ins
            j vertical_ins
           # se for horizontal = coluna inicial + comprimento do navio
>9 invalido
           # se for vertical = linha inical + comprimento do navio >9
invalido
            horizontal_ins:
                add s8, t5, t3
                addi s7, zero, 10
                blt s7, s8, comp_invalido # testamos se o comprimeto
rompre a matriz

                addi a2, a2, 4
                j continua_ins_h
            vertical_ins:
                add s8, t4, t3
                addi s7, zero, 10
                blt s7, s8, comp_invalido # testamos se o comprimeto
rompre a matriz

                addi a2, a2, 40 # mesmo que 4 * 10 posições
            continua_ins_h:
                j teste_condicao_ins_h
        fim_ins_h:
            addi s11, s11, 1

    incremento_controle_ins:
        addi t0, t0, -1
        addi a1, a1, 2
        la a2, matriz
        j teste_condicao_ins
    fim_ins:
        ret
    pos_invalida: # aqui é notificada a invalidação por poscionamento
fora da matriz
        la a0, invalida_fora
        li a7, 4
        ecall

        li a0, 10
        li a7, 11
        ecall
```

```
      # add s10, zero, ra
      # jal zera_matriz
      # add ra, zero, s10
       j fim
      # ret
   comp_invalido: # aqui é notificada a invalidação por comprimento
rompendo a matriz
       la a0, invalida_maior
       li a7, 4
       ecall

       li a0, 10
       li a7, 11
       ecall

      # add s10, zero, ra
      # jal zera_matriz
      # add ra, zero, s10
       j fim
      # ret

   sobre_invalido: # aqui é notificada a invalidação por sobreposição
do barco
       la a0, invalida_sobreposto
       li a7, 4
       ecall

       li a0, 10
       li a7, 11
       ecall

      # add s10, zero, ra
      # jal zera_matriz
      # add ra, zero, s10
       j fim
      # ret

# a função "printa_matriz_padrao" recebe como parâmetro a "matriz",
# à partir da matriz imprime os barcos atingidos e os tiros na água com
base em uma soma que resulta em letras na tabela ASCII,
# "o" reprenta um tira na água, "a" ou "b" etc, reoresentam o
respectivo barco e sua posição atingida
printa_matriz_padrao:
```

```
    add t0, zero, zero # quando chegar em 100, termina
    addi t1, zero, 100
    add t2, zero, zero # a cada 10, um \n
    addi t3, zero, 10
    la a1, matriz
    teste_condicao_prin:
        beq t0, t1, fim_prin
        beq t2, t3, pula_prin
        j corpo_laco_prin
    pula_prin:
        add t2, zero, zero
        li a0, 10 # código ascii do espaço
        li a7, 11 # printa char
        ecall
    corpo_laco_prin:
        lb a0, (a1)
        addi a0, a0, 64
        addi a2, zero, 96
        blt a2, a0, printa_atingido # printa posições atingidas e tiros
na água
        j printa_agua
        printa_agua:
            li a0, 64
            li a7, 11
            ecall
            j continua_prin
        printa_atingido:
            li a7, 11
            ecall
        continua_prin:
        li a0, 32 # código ascii do espaco
        li a7, 11 # printa char
        ecall

    incremento_controle_prin:
        addi a1, a1, 4
        addi t0, t0, 1
        addi t2, t2, 1
        j teste_condicao_prin
    fim_prin:
        ret

# a função "printa_matriz_espiao" recebe como parâmetro a "matriz",
```

```
# é reposnsável por printar tudo o que está na matriz se esconder nada,
com base em soma na tabela ASCII gerando letras,
# "o" reprenta um tira na água, "a" ou "b" etc, reoresentam o
respectivo barco e sua posição atingida,
# "A" ou "B" etc, representam um barco flutuante não atingido na
determina posição
printa_matriz_espiao:
   add t0, zero, zero # quando chegar em 100, termina
   addi t1, zero, 100
   add t2, zero, zero # a cada 10, um \n
   addi t3, zero, 10
   la a1, matriz
   teste_condicao_esp:
       beq t0, t1, fim_esp
       beq t2, t3, pula_esp
       j corpo_laco_esp
   pula_esp:
       add t2, zero, zero
       li a0, 10
       li a7, 11
       ecall
   corpo_laco_esp:
       lw a0, (a1)
       addi a0, a0, 64
       li a7, 11
       ecall

       li a0, 32
       li a7, 11
       ecall

   incremento_controle_esp:
       addi a1, a1, 4
       addi t0, t0, 1
       addi t2, t2, 1
       j teste_condicao_esp
   fim_esp:
       ret
# a função "zera_matriz" recebe como parâmetro a "matriz",
# é reposável por fazer a limpeza de todos os barcos para que uma novo
jogo com novos barcos seja iniciado futuramente
zera_matriz:
   add s2, zero, zero # quando chegar em 100, termina
```

```
    addi s3, zero, 100
    la s1, matriz
    teste_condicao_zen:
        beq s2, s3, fim_zen
    corpo_laco_zen:
        sw zero, (s1)
    incremento_controle_zen:
        addi s1, s1, 4
        addi s2, s2, 1
        j teste_condicao_zen
    fim_zen:
        ret
# a função "zera_voce" recebe como parâmetro a .word "voce",
# responsável por zerar suas estatíticas de jogo ao final de uma
partida, para dar espaço para uma nova partida futura
zera_voce:
    la s0, voce
    sw zero, (s0)
    addi s0, s0, 4
    sw zero, (s0)
    addi s0, s0, 4
    sw zero, (s0)
    addi s0, s0, 4
    ret

# a função "zera_controledebarcos" recebe como parâmetro a .word
"controle_barcos",
# responsável por zerar a .word "controle_barcos" para permitir o
controle de afundamentos em uma nova partida futura
zera_controledebarcos:
    la s0, controle_barcos
    add t0, zero, zero
    addi t1, zero, 20

    teste_condicao_zera:
        beq t0, t1, fim_zera
    corpo_laco_zera:
        sw zero, (s0)
    incremento_controle_zera:
        addi s0, s0, 4
        addi t0, t0, 1
        j teste_condicao_zera
    fim_zera:
```

```
        ret
# a função jogo recebe como parâmetro todos os parâmetros das outras
funções, pois ela é um switch de opções,
# para seu uso interno, recebe como parâmetro a .word "voce" e .word
"recorde",
# faz o controle de fim de jogo e inicia uma nova partida
automaticamente
jogo:
   add s10, zero, ra
   jal insere_embarcacoes
   add ra, zero, s10

   la a0, msg_2 # mensagem de boas vindas
   li a7, 4
   ecall

   addi t1, zero, 1
   addi t2, zero, 2
   corpo_laco_jogo:
       addi t1, zero, 1
       addi t2, zero, 2
       la a0, menu_jogo
       li a7, 4
       ecall
       li a7, 5
       ecall # a0 é o int

       beq a0, zero, reinicia
       beq a0, t1, mostra_mat
       beq a0, t2, jogada
       j sair # switch para uso do jogo

       reinicia: # caso quisermos reiniciar em uma nova partida
           add s10, zero, ra
           jal zera_matriz # limpa as alterações da matriz
           jal zera_voce
           jal zera_controledebarcos
           jal insere_embarcacoes # insere novamente podendo escolher
um novo conjunto de barcos
           add ra, zero, s10
           j incremento_controle_jogo

       mostra_mat: # caso quisermos mostrar a matriz em modo espião
```

```
            add s10, zero, ra
            jal printa_matriz_espiao
            add ra, zero, s10
            li a0, 10
            li a7, 11
            ecall
            j incremento_controle_jogo

        jogada: # caso quisermos jogar
            add s10, zero, ra
            jal jogar
            add ra, zero, s10
           # verfica o fim abaixo
            la s0, controle_barcos
            lw s2, (s0)
            addi s0, s0, 4
            lw s1, (s0)
           # addi s1, s1, 1
           # sw s1, (s0)
            beq s1, s2, fim_jogo_vitoria
            j incremento_controle_jogo

        sair: # para sair do jogo, simplesmente dropa o programa
            j fim

    incremento_controle_jogo:
        add s10, zero, ra
        jal printa_situacao
        add ra, zero, s10
        j corpo_laco_jogo
    fim_jogo_vitoria: # somos direcionados para cá caso todos barcos
tenham sido afundados
        la a0, terminou
        li a7, 4
        ecall
        la s0, voce
        la s1, recorde
        lw s2, (s0)
        lw s3, (s1) # vou usar a menor quantidade de tiros como critério
de recorde, foi o que consegui ver como melhor
        blt s2, s3, salva_novo_recorde # se você deu menos tiros que o
recorde
        j reinicia
```

```
        salva_novo_recorde: # somos direionados para cá caso um novo
recorde seja aplicável
            sw s2, (s1)
            addi s0, s0, 4
            addi s1, s1, 4
            lw s2, (s0)
            sw s2, (s1)
            addi s0, s0, 4
            addi s1, s1, 4
            lw s2, (s0)
            sw s2, (s1)
            j reinicia
    fim_jogo:
        ret

# a função "printa_situação" recebe como parâmetro a .word "voce" e
.word "recorde", além de todos parâmetros da função,
# "printa_matriz_padrao", é reponsável após, cada jogada, mostrar a
pontuação de jogo, recorde e situação de atingimento na matriz
printa_situacao:
    la a0, situacaojogo_msg
    li a7, 4
    ecall

    la a0, recorde_msg
    li a7, 4
    ecall

    la a0, tiros_msg
    li a7, 4
    ecall
    la a1, recorde
    lw a0, (a1)
    li a7, 1
    ecall
    li a0, 10
    li a7, 11
    ecall

    addi a1, a1, 4
    la a0, acertos_msg
    li a7, 4
```

```
 ecall
 lw a0, (a1)
 li a7, 1
 ecall
 li a0, 10
 li a7, 11
 ecall

 addi a1, a1, 4
 la a0, afundados_msg
 li a7, 4
 ecall
 lw a0, (a1)
 li a7, 1
 ecall
 li a0, 10
 li a7, 11
 ecall
# ---------------------------------
 la a0, voce_msg
 li a7, 4
 ecall

 la a0, tiros_msg
 li a7, 4
 ecall
 la a1, voce
 lw a0, (a1)
 li a7, 1
 ecall
 li a0, 10
 li a7, 11
 ecall

 addi a1, a1, 4
 la a0, acertos_msg
 li a7, 4
 ecall
 lw a0, (a1)
 li a7, 1
 ecall
 li a0, 10
 li a7, 11
```

```
    ecall

    addi a1, a1, 4
    la a0, afundados_msg
    li a7, 4
    ecall
    lw a0, (a1)
    li a7, 1
    ecall
    li a0, 10
    li a7, 11
    ecall

    addi a1, a1, 4
    la a0, ultimotiro_msg
    li a7, 4
    ecall
    lw a0, (a1)
    li a7, 1
    ecall
    li a0, 32
    li a7, 11
    ecall
    addi a1, a1, 4
    lw a0, (a1)
    li a7, 1
    ecall
    li a0, 10
    li a7, 11
    ecall
   # ---------------------------------
    add s10, zero, ra
    jal printa_matriz_padrao
    add ra, zero, s10

    li a0, 10
    li a7, 11
    ecall

    ret
# a função jogar recebe como parâmetro a .word "voce" e a "matriz",
# é reponsável por solicitar as coordenadas do tiro e executá-las
jogar:
```

```
    la a5, voce
    lw a4, (a5)
    addi a4, a4, 1
    sw a4, (a5)

    la a0, tiro
    li a7, 4
    ecall
    la a0, space
    li a7, 4
    ecall
    li a1, 4
    li a7, 8 # vamos ler uma string
    ecall # a0 é a string

    add a1, a0, zero
    li a0, 10 # \n na tabela ascii
    li a7, 11 # printar char
    ecall # pula a linha

    addi t2, zero, 10
    addi t3, zero, 4
    lb a2, (a1) # linha
    addi a2, a2, -48 # a2 linha
    addi a1, a1, 2 # pulamos para a coluna
    lb a3, (a1) # coluna
    addi a3, a3, -48 # a3 coluna
    la a5, voce
    addi a5, a5, 12
    lw a4, (a5)
    sw a2, (a5)
    addi a5, a5, 4
    sw a3, (a5)

   # Deslocamento = (L * QTD_colunas + C) * 4
    mul a2, a2, t2
    add a2, a2, a3
    mul s0, a2, t3 # resultado de deslocamento em s0
   # add s9, zero, s10 # salva o deslocamento para na próxima limpar a
posição do tiro

    la a1, matriz
    add a1, a1, s0
```

```
    lw a2, (a1)
    bne a2, zero, barco_atingido
    addi t6, zero, 32
    bge a2, t6, pula
    addi a2, zero, 47 # representa o "o" após a soma com 64 feita na
hora da função de print
    pula:
    la a0, errou
    li a7, 4
    ecall
    sw a2, (a1)
    j fim_jogar
    barco_atingido:
        bge a2, t6, pulaa
        la a0, atingiu
        li a7, 4
        ecall

        la s0, controle_barcos
        addi s0, s0, 4
        lw s1, (s0)
        addi s1, s1, 1
        sw s1, (s0)

        la s0, controle_barcos
        add s1, zero, a2
        addi s2, zero, 8 # usando o proóprio valor do barco, pulamos de
duas em duas posições até encontrar o barco certo
        mul s1, s1, s2
        add s0, s0, s1
        lw s1, (s0)
        addi s0, s0, 4
        lw s2, (s0)
        addi s2, s2, 1
        sw s2, (s0)

        addi a2, a2, 32
        sw a2, (a1)
        pulaa:
        la a5, voce
        addi a5, a5, 4
        lw a4, (a5)
        addi a4, a4, 1
```

```
        sw a4, (a5)

    fim_jogar:
        beq s1, s2, barco_afundado
        ret

    barco_afundado:
        la a0, afundou
        li a7, 4
        ecall
        la a1, voce
        addi a1, a1, 8 # pulamos para a contagem de afundados
        lw a2, (a1)
        addi a2, a2, 1
        sw a2, (a1)
        ret

# fim é um flag para indicar o encerramento do programa
fim:
    nop
```