**UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE**

**INSTITUTO METRÓPOLE DIGITAL**

**TECNOLOGIA DA INFORMAÇÃO**

**IMD1012 - TURMA 04**

**INTRODUÇÃO ÀS TÉCNICAS DE PROGRAMAÇÃO**

**PROFESSOR: ANTONINO FEITOSA**

**PROJETO FINAL – UNIDADE 3 – 2021.2**

**Elbert Natan Fernandes Morais - 20210052554**

**João Victor Dantas do Nascimento - 20210054361**

1. **Introdução**

O presente relatório está sendo desenvolvido levando em consideração a implementacao de algoritmo realizada no contexto da disciplina de Introdução às Técnicas de Programação, do curso de Tecnologia da Informação na UFRN (Universidade Federal do Rio Grande do Norte).

Inicialmente, foi apresentado um desafio aos estudantes para que fosse possível programar uma mini “inteligência conduzida”, onde dado um jogo de pesca desenvolvido pelo Professor André Campos, ela poderia tomar decisões acerca de onde ir, como pescar, e como vender seu produto da maneira mais eficaz possível.

A lógica do jogo é simples: há em nossa tela um mapa, representado internamente por uma matriz nxn, onde o tamanho é definido aleatoriamente no início de cada rodada. Nesse mapa, há pequenas concentrações de cardumes espalhados, onde cada cardume corresponde a um tipo de peixe com sua respectiva quantidade. Uma vez que cada “bot” - que daqui pra frente trataremos como barco - pesca uma quantidade x de peixes, é necessário vender o produto em um porto, representado no mapa como um ponto preto. O jogo envolve, também, questões ambientais, por isso é necessário manter o controle da quantidade de peixes pescada em algum ponto do mapa, tendo noção de que é inviável e absolutamente proibido causar dano ambiental à espécie do cardume em questão.

Dada essa breve introdução acerca do desafio, esclarecidos os objetivos, partamos então à abordagem utilizada na implementação. Inicialmente, nos foi dado um arquivo de exemplo, com tudo aquilo que seria vital ao funcionamento de um barco no simulador (embora que sem tomada de decisão alguma, salvo um comando para explicitar como funcionaria o envio de dados). A partir desse arquivo, foi feita uma análise sobre como seria a maneira mais fácil de implementar uma solução inteligente que pudesse ser replicada para os demais barcos. Nesse contexto, foram estabelecidas as metas de como funcionaria cada barco, dividindo as operações necessárias em funções, afinal, para esse e muitos outros momentos, a divisão de responsabilidades pareceu algo racional e lógico.

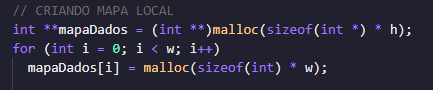
Foram criados, então, os esqueletos do que regeria o barco, a cada rodada os dados seriam atualizados com os da rodada passada, e assim seria possível tomar uma decisão (função readData foi adaptada para esse propósito). Uma vez que os dados da rodada estão disponíveis, é hora de descobrir qual ação será tomada. Levando isso em consideração, a funcao executarProximaAcao() foi criada, encapsulando tudo que era referente à lógica de escolha em uma só função separada da leitura de dados. Dessa forma, o método main() ficou muito mais limpo do que poderia eventualmente estar no caso de uma abordagem distinta. Na referida função, foram avaliados algumas das variáveis relativas às regras do jogo, para que não houvesse prejuízo algum que pudesse ser estabelecido pelo simulador. Além disso, dependendo do objetivo atual, o barco se movimentaria no eixo X ou no eixo Y, sabendo se é para pescar ou vender.

É importante ressaltar, a princípio, que os movimentos do barco em busca do peixe ou do porto foram previamente analisados, isto é, não se trata de um movimento aleatório ao redor do mapa. A matriz de dados (chamada na solução de mapaDados) é lida a cada rodada, dessa forma é possível que o barco leve isso em consideração para calcular qual é o melhor trajeto a ser tomado (foram criadas funções para calcular trajetos para portos e peixes).

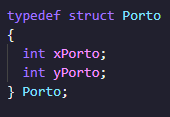
1. **Implementação**

Nesta seção será utilizada a seguinte abordagem: seguir o código da maneira como ele deve funcionar e explicar qual é a lógica por trás daquilo.

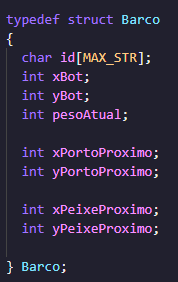
Após as configurações iniciais, realizadas pelo professor no arquivo de exemplo, foi alinhado que a técnica mais adequada para armazenar os dados da rodada seria utilizar uma matriz de inteiros, dessa forma, a primeira intervenção no código foi preencher essa matriz com o que seria necessário para que ela funcionasse utilizando a função malloc da biblioteca <stdlib.h>, a qual foi devidamente importada.



**Figura 1 -** “Instância” da matriz mapaDados

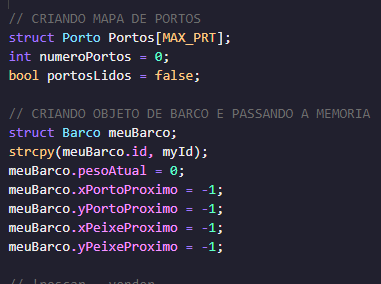
Antes de prosseguir para a dos portos e suas propriedades, é importante descrever como as estruturas de dados utilizadas foram formuladas, pois esse conceito foi amplamente utilizado dentro das funções implementadas.

O struct de porto é relativamente simples, foram utilizados apenas dois inteiros para armazenar as coordenadas x e y (respectivamente) do porto em questão. Uma vez que os portos não mudam de lugar, não é necessária uma grande cautela com a atualização desses dados.



No caso do struct de barco, a abordagem é diferente. É notável a diferença do numero de informações presentes nele em relação às do porto. De cara é perceptível que o id fornecido inicialmente é utilizado para identificar cada barco, fazendo com que assim seja possível alimentar cada barco com suia respectiva informação da rodada. Pelo fato de terem sido utilizadas funções para calcular qual peixe e qual porto estavam mais próximos naquele momento, temos essas informações também na estrutura, algo que será particular para cada barco em função de sua posição atual, ja que, como será explicado posteriormente, a cada rodada é possível que o objetivo mude: uma área pescável pode aparecer e alterar o foco do barco.

Introduzidos os structs, é possível continuar. O próximo passo tomado foi pensar: qual seria a maneira mais fácil, rápida e objetiva para guardar as informações de barcos e portos? Matriz de inteiros? Matriz de tipos genéricos? A resposta para essa pergunta nessa implementação foi usar uma matriz de portos - com suas respectivas propriedades - e uma única instância de barco, posto que nessa implementação o barco só sabe suas informações.



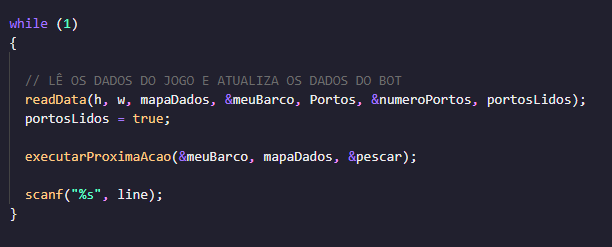
**Figura 2** - “Instância” de structs no contexto da implementação

Algumas observações precisam ser feitas, o número de portos foi inicialmente setado como 0 pois a esta altura, quando os dados ainda não foram lidos, não se sabe quantos portos existem no mapa, mais pra frente será necessário iterar sobre eles, dessa forma, essa informação deverá ser guardada para um uso posterior (o booleano portosLidos será utilizado nesse sentido). Além disso, na declaração de barco o peso atual foi setado como 0 pois nada foi pescado ainda. As demais prorpiedades foram setadas como -1 pelo fato de que nesse contexto a coordenada (x: -1, y: -1) não é válida, abrindo brecha para que isso fosse usado como validação nas funções que determinam posição lá na frente.

Para saber se é necessário pescar ou vender, foi criada uma variável booleana “pescar”, quando ela é falsa, naturalmente precisamos vender, e o caso contrário dispensa explicações.

/\*inserir aqui algo sobre arquivos - FILE \*fp = fopen("LogBots.txt", "w")\*/

No loop principal, que definirá a janela de funcionamento do jogo, há quatro linhas, das quais três delas são extremamente importantes e vitais para o funcionamento de cada parte do barco. Portanto, temos:



**Figura 3** - Estrutura principal do código (while da rodada)

Como comentado inicialmente, essa abordagem deixa o código bem mais limpo e legível, ao passo de que as adaptações realizadas na função readData competem apenas à ela. Na ordem, temos:

1. readData

Responsável por ler, interpretar, e atualizar os dados do barco que está chamando ela. É possível reparar que muitas variáveis são alteradas dentro dessa função, ela é vital para que o barco consiga se atualizar e se localizar em “tempo real”.

1. executarProximaAcao

Essa é, definitivamente, a função mais complexa que foi criada no escopo desse projeto. A responsabilidade dela é pegar os dados colhidos por readData e avaliar as potenciais situações. Nela é feita todo um processo de checagem, fazendo com que o barco respeite as premissas do jogo, tornando quase impossível que o mesmo seja eliminado por dano ambiental ou que perca muito dinheiro através de multas (tanto de evasão de terreno quanto de carga superior ao permitido).

1. scanf

Essa função nativa da linguagem está aí para capturar qual foi a resposta do simulador para a ação solicitada. Como descrito no repositório do projeto, deslocamentos e outros tipos de ação podem ser solicitadas ao simulador, que por sua vez, terá a responsabilidade de responder acerca do que está acontecendo nesse momento. A função scanf tem como atribuição ler essa informação e passar ela para a variável line.

Como comentado inicialmente. FALTA APROFUNDAR SOBRE AS FUNCOES E MOSTRAR AS FUNCOES COMPLEMENTARES NA TOMADA DE DECISÃO. ALÉM DISSO VAI FALTAR DESALOCAR A MATRIZ DE DADOS E REMOVER COMENTÁRIOS DESNECESSÁRIOS.

**Anexo (Código-Fonte)**

Nesta deve constar o código-fonte de toda a aplicação com instruções (comandos de compilação) para geração de 4 bots. Haverá um bônus de meio ponto no relatório caso seja utilizado Makefile.