Trabalho da disciplina de Linguagem de Programação - 2019/1

Lucas Diniz da Costa - 201465524AC

[1]Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora (MG) – Brasil

lucasdinizcosta@ice.ufjf.br

1. Introdução

O objetivo deste trabalho envolveu-se na implementação do jogo popularmente conhecido como "campo minado" (*Mine sweeper*) a partir da linguagem de programação funcional denominada Haskell. Este relatório se divide na seção 1 exibindo uma introdução rápida sobre o conteúdo do trabalho, na seção 2 as especificações de software utilizados na realização da atividade, na seção 3 expõe detalhes técnicos sobre o desenvolvimento, na seção 4 será apresentado um pouco sobre as dificuldade encontradas na realização deste trabalho e na última seção são apresentadas algumas imagens de execução do jogo.

2. Especificações de software

- **Compilador:** GHCi - versão: 8.0.2.
- **Sistema operacional:** Linux Mint 19 Cinnamon.
- **Editor de texto definitivo:** Sublime Text - versão: 3.2.1, Build: 3207.
- **Editor de texto início:** Atom.
- **Versionamento:** git com suporte atráves da plataforma do GitHub.
  link de download do programa:
  (`https://github.com/lucasDinizCosta/campoMinadoHaskell`)

3. Desenvolvimento

3.1. Conceitual

3.1.1. Tipos de dados

Como trabalhar com o mapeamento do campo minado somente com os tipos de dados oferecidos pela linguagem Haskell estava gerando muitos problemas, a solução buscada foi criar um novo tipo denominado **Celula** no qual possui como construtor o texto que a célula irá exibir no mapa do console, identificação para linha e coluna da matriz do campo, estado de fechado caso a célula já tenha sido solicitada pelo jogador, um booleano responsável por exibir se a celula possui mina ou não, um booleano de marcação de mina do jogador na célula e um valor inteiro responsável por exibir a quantidade de vizinhos com minas.

3.1.2. Mapa

A ideia utilizada para fazer o mapeamento do campo minado foi em inicialmente trabalhar com matrizes no Haskell, mas devido a alguns problemas em controlar a estrutura, o processo sofreu uma adaptação de uma matriz para uma lista, de modo que a lista

de celulas é criada de maneira genérica a partir de list comprehension variando os indices de linha e colunas da matriz. Para acessar os elementos da matriz na lista(vetor), foi utilizado um conversor de indices em que ao passar os valores de linha, coluna da matriz é retornado a celula correspondente na posição determinada na lista. Na imagem a seguir é exposto a ideia de conversão de indices de matriz para uma lista(vetor) e vice-versa.

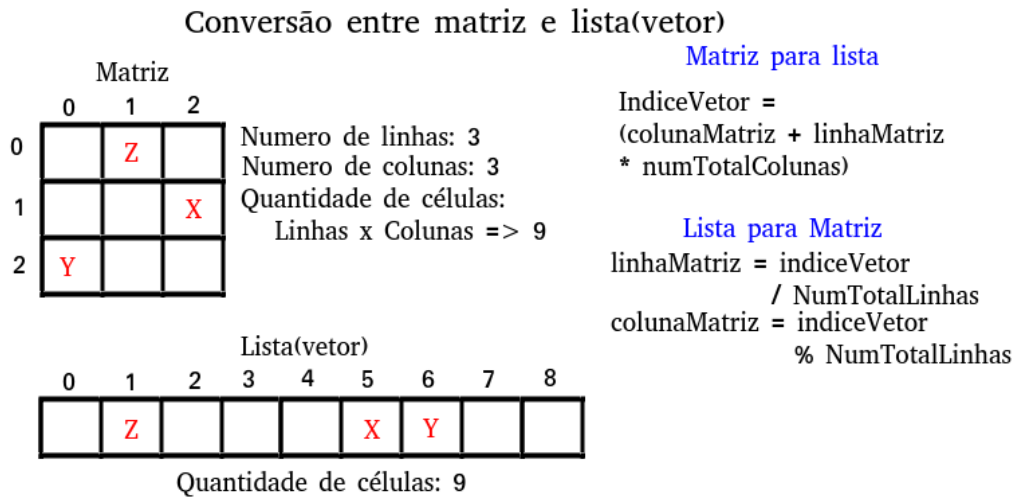

Figure 1. Conversão de índices entre matriz e lista(vetor)

3.2. Métodos

3.2.1. Iniciar

Método responsável por iniciar o jogo via console e disparar as ações dos outros métodos.

3.2.2. Posicionamento das minas

Foi bem complicado realizar esta tarefa, tanto em planejar a estratégia de posicionamento com um fator aleatório, quanto posicionar o conteúdo nas células do mapa propriamente dito. O primeiro passo determinado foi em planejar o fator de aleatoriedade, a ideia consiste em receber uma lista de posições sorteadas a partir do método “Lista Aleatória” e alterar todas as celulas com os indices sorteados para que apresentem a mina e ao final retorna o mapa com as minas posicionadas.

3.2.3. Lista Aleatória

O método embaralha a primeira lista de indices e armazena na lista sorteada a cabeça da lista de indices e em seguida o remove da lista original, fazendo assim não ter repetição de índices e ter variedade nas posições das minas, repete o processo até a lista sorteada ter o tamanho da quantidade de minas presentes no mapa.

3.2.4. Montar mapa

Método responsável por gerar o mapa incial de células a partir de um list Comprehension e fazer o posicionamento das minas e estabelecer vizinhos.

3.2.5. Print mapa

Função responsável por gerar, via console, o mapa da estrutura do campo minado.

3.2.6. Calcula vizinhos

Utiliza o método "retornaVizinhos" que volta a quantidade de vizinhos com minas próximas e em seguida, percorre o mapa todo armazenando o valor da quantidade de vizinhos com minas em cada célula que pode ser fechada.

3.2.7. Retorna vizinhos

Retorna a quantidade de vizinhos com minas em torno de uma célula passada como parâmetro.

3.2.8. Tratar minas

Está função trata a quantidade de minas informada pelo jogador conforme a especificação do enunciado do trabalho, sendo menor que 1, o método retorna o valor mais próximo e sendo maior que o valor máximo de minas, o valor a ser atribuído fica sendo o máximo possível no mapa.

3.2.9. Atualizar mapa

Se divide em três casos. O primeiro caso é relativo ao primeiro modelo de jogada em que o jogador informa a célula a ser aberta, e este caso modifica a escrita da célula para exibir a quantidade de vizinhos que apresentam minas. O segundo caso se guia para o modelo de posicionamento das minas pelo jogador, e o método muda o mapa preenchendo com "B" à célula escolhida com mina pelo jogador. E o último caso é voltado para a retirada de minas preenchidas pelo jogador, voltando a célula para o estado de não ter marcação do jogador e a escrita com **" * "**.

3.2.10. Revelar mapa

Percorre o mapa todo modificando o escrito das células de modo a exibir para o jogador todas as células no qual ele não explorou ou o pelo fato de ter perdido ou ter ganhado e não ver o posicionamento das minas.

3.2.11. Executar Jogada

Recebe a jogada informada pelo jogador e os mapas atualizados com as alterações posteriores e deixa as responsabilidades de tratamentos de jogada e mudanças do mapa para o método de "tratarJogada".

3.2.12. Tratar Jogada

É a função mais extensa do programa pois trata todos os erros de jogadas e caracteres informados pelo jogador, desde a conversão de uma letra maiúscula para a coordenada de coluna, até a verificação se o texto digitado posteriomente pode ser considerado um número e não obter problema ao tentar fazer uma conversão incorreta da String informada pelo jogador em um indice de inteiros para o número de linhas do mapa do campo minado. Guia todo o núcleo da partida, desde o controle das alterações do mapa, até a condições de vitória ou perda do jogador.

3.2.13. Termina partida

Apenas exibe ao jogador a possibilidade de querer reiniciar a partir ou finalizar o programa.

4. Dificuldades

O fato de Haskell ser uma linguagem de paradigma funcional dificultou bastante a implementação devido ao costume com a programação imperativa com Javascript, C++, Python, etc.

Outro problema bem recorrente foi a má identação do código que ao usar tabulação, o compilador apresenta problemas em identificar a identação do código. A solução encontrada foi migrar do Atom para o Sublime Text e substituir a tabulação para quatro espaços consecutivos.

Como não há variáveis, boa parte do tempo gasto neste trabalho foi em encontrar estratégias de armazenamento de contadores e modificação no mapa do campo minado, a ideia final encontrada foi no fato de o haskell trabalhar com recursão então deveriam ser passados estes valores sempre nas chamadas das funções, assim a "variável" atualizaria o seu valor. Uma ideia também utilizada para percorrer o campo e gerar a impressão do mesmo no console, foi em adaptar o comando *for* de repetições em linguagem imperativa para uma ideia de recursão no Haskell.

5. Exemplos de execução

Figure 2. Mapa com preenchimento nos espaços

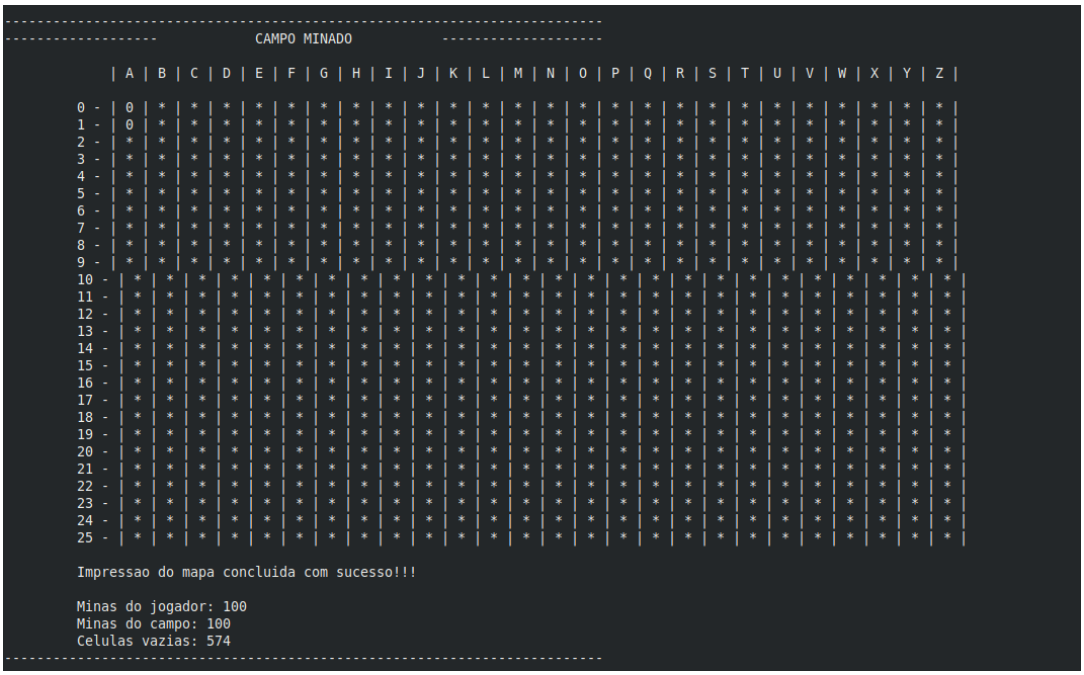

Figure 3. Mapas grandes

References