Relatório

Nome: José Barros – 23142

Miguel Simões - 23135

Data: 18-01-2021

Professor: Nuno Mendes

Introdução

O objetivo era desenvolver o jogo das minas em pseudocódigo, mas no contexto como se fosse para programar em C, então usei ponteiros em alguns momentos e mesmo o design do campo do jogo é muito simples feito com "loops" e "print's" de caracteres.

Ao longo do pseudocódigo usei alguns termos que não demos nas aulas mas que precisei ao longo do desenvolvimento do jogo:

O retorno da função quando é "void" em C, ou seja quando não retorna nada eu declarei como "nulo".

Quando tenho "-conteúdo-" significa que estou a declarar alguma função e que no conteúdo tem o significado do que ela faz exemplo:

-Função para dar clear no ecrã / terminal- : é como se fosse um system("cls") em C.

O texto "\n" vai representar um parágrafo como em C.

Dando uma variável com o nome "p" –

p^ - vai significar para declarar um ponteiro, ou para passar um valor para o ponteiro.

Exemplos:

inteiro ^p – significa que estou a declarar uma variável do tipo ponteiro que aponta para um número inteiro; em C seria: "int *x;"

^p ← 15 – significa que lemos o conteúdo do endereço apontado por "p" recebe 15;

>x – vai significar que obtém o endereço de uma variável.

Exemplos:

p ← >x – significa que a variável "p" vai armazenar o endereço de memória da variável "x".

Funcionamento do jogo

O jogo funciona através de um menu principal, onde caso o utilizador pretenda jogar deverá clicar no 1. Depois de clicar é lhe apresentado as escolhas do campo e das minas. Posteriormente, é que começa o verdadeiro jogo, onde ele vai escolhendo as coordenadas onde quer jogar, por exemplo AA (a coordenada mais superior e à esquerda possível). Este pode decidir marcar ou abrir atráves do 'M' ou 'A' respetivamente. A lógica por de trás implementada foi ter duas matrizes, uma que tem a solução de onde as minas estão e outra com os passos das coordenadas onde o jogador vai escolhendo. E a base do jogo é feita com comparações entre as duas matrizes. A que é apresentada ao jogador é a matriz "jogo", e a que contém as soluções e é apresentada no final do jogo com as soluções é a matriz "campo".

O jogo tem outras opções no menu para poder ver as pontuações que estão armazenadas no ficheiro "data.txt" e outra opção para sair do jogo.

Explicação das funções

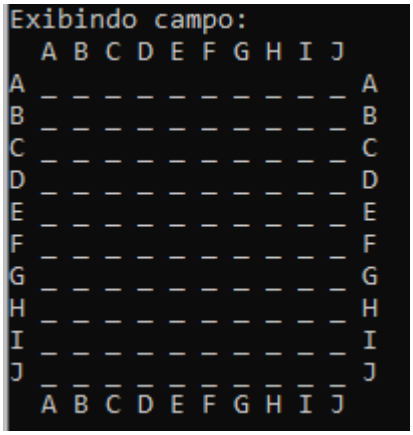
Função "ler_dimensões" declarei que o limite do campo tinha de ser no máximo 20x20, porque trabalho com valores ASCII não poderia ultrapassar as letras do alfabeto no contexto do meu problema e da forma de como o jogo está designado. Uma vez o utilizador metendo valores superiores ao limite que defini (20) este volta a chamar a mesma função recursivamente. Também usei condições para verificar se as dimensões são positivas. Usei as dimensões x e y como ponteiros para conseguir usar-los globalmente ao longo do programa, mesmo não estando na função, porque tendo o endereço de memória que quero mudar faço o que quero com a variável.

Função "ler_minas" tem como parâmetros os valores máximos das coordenadas do campo(x e y), e passo o endereço de memória das minas para conseguir mudar ao longo de todo o programa e assim mesmo depois saindo fora da função o valor das minas introduzido pelo utilizador continua o mesmo, porque estou a mudar o valor que está naquele endereço de memória.

Função “sortear_mina” tem como parâmetros o campo, a quantidade de minas e as coordenadas dos limites do campo(x e y). Dentro da função eu basicamente calculo um valor aleatório para as coordenadas do x e do y. Praticamente todas as linguagens de programação, incluindo a C tem funções de gerar números aleatórios e é o que estou a dizer com a função “rand()” , em prática poderiam pegar no valor dos milissegundos no dia e prontos gera um valor praticamente aleatório, mas como as funções existem é mais prático chamar por elas. Depois de ter esse numero aleatório eu uso o “MOD” para retornar o resto da divisão do número aleatório pelo número máximo das coordenadas máxima e dessa forma tenho a certeza que o valor final aleatório nunca será maior que x ou y(coordenadas máximas que fazem o campo) e assim não corro o risco de meter uma mina fora do campo. Depois tenho uma condição que verifica se já existe alguma mina naquele local e se existir ele chama de novo a função recursivamente para gerar outro x e y até encontrar um ponto onde não existam minas.

Função preencher_minas basicamente preenche o campo todo onde o jogador pode jogar com espaço (“ “) e depois por cada mina ele chama a função sortear_mina para declarar as coordenadas da mina, uma vez tendo passa para a próxima e sempre assim porque está dentro de um loop até ao número de minas.

Função exibir_campo vai exibir os caracteres que formam o retângulo ou quadrado do campo metendo dentro deste os espaços para ver onde pode jogar. Em baixo podemos ver como ficaria o campo:



Na função contar_vizinhos percorro todos os pontos e verifico se em todos os pontos adjacentes incluindo diagonais ou seja nos 8 existentes, se existe algum desses pontos com mina, se este existir deixo no ponto verificado o número total das minas adjacentes aquele ponto e se não existerem minas perto não faço nada ou seja deixo como está.

Na função preencher_jogo estou a meter todos os valores da matriz "jogo" para o caracter "_" ou seja é onde o jogador pode dicidir abrir ou dizer que é mina e represento dessa forma. A matriz "jogo" representa o que o jogador vê, enquanto a matriz "campo" é a solução que em vez dos "_" vai ter espaços em branco ou números de bombas adjacentes ou "m" que representeará as bombas.

Na função escolher_coordenada o jogador escolhe o valor do X que é a linha e o Y que é a coluna. As coordenadas são processadas como caracteres mas depois a fazer a computação passo-as para os valores ASCII para verificar se estão dentro dos valores normais para as dimensões do campo.

Na função abrir_ou_marcar o utilizador depois de escolher a coordenada dicide se quer abrir aquela coordenada ou marcar, ou seja, se pensar que lá existe bomba.

Na função abrir_brancos esta vem caso o utilizador escolha uma coordenada onde não exista nem números nem mina, ou seja branco, esta função é chamada no último "else" da função "jogar" ou seja com base naquela coordenada vai abrir os brancos chamando a função abir_brancos que por sua vez passa o valor dessa coordenada a branco e depois verifica em todos os seus lados adjacentes se existem espaços em branco também e se no jogo está '_' ou seja disponível para jogar. Caso haja a função chama-se a si mesma ou seja recursivamente passando como argumentos a nova coordenada que é branco e para abrir para o jogador ver. A função só termina de se chamar quando todos os brancos adjacentes chamados recursivamente acabarem.

Na função guarda_pontuacoes têm como argumentos o tempo que o jogador demorou a ganhar(0 em caso de perda), e o numero de minas que este escolheu. Em caso de perda a função guarda_pontuacoes é chamada com o primeiro parâmetro a 0 (tdiferenca) e ao ser 0, eu dou a pontuação através do segundo parâmetro que é as minas que ele marcou e estavam correctas, sendo as minas marcadas x 10. A pontuação máxima que um perdedor poderá ter é 100, porque no máximo só posso ter 10 minas independetemente da escolha do jogador. Caso vença eu chamo a função guarda_pontuacoes com os dois argumentos de quanto tempo demorou a ganhar e quantas minas havias. Para atribuir a pontuação a um vencedor fiz (n*10) como no inicio e prontos assim a diferença de um perdedor para um vencedor será a adição da pontuação do tempo. Achei que era justo assim porque mesmo que um jogador perca no nível difícil onde tem 10 minas, e marcou por exemplo 9 minas, acho justo ter mais pontuação que um jogador que ganhou no modo fácil e demorou muito tempo, então prontos defini essa formula que achei que era a melhor que se ajustava. Na formula quantas mais minas ele tinha mais pontos terá e quanto menos tempo demorar mais pontuação.

No final adiciono o nome pedido ao utilizador em conjunto com a respetiva pontuação ao ficheiro onde guardo os registos e depois chamo a função para ordenar novamente as pontuações.

Na função "jogar" chamo todas as funções como exibir campo pedir as coordenadas etc para o utilizador escolher uma coordenada. Depois que ele escolhe e se não tiver entrado numa das três condições de paragens (game over, todas as minas marcadas ou todos os campos abertos) a mesma volta-se a chamar até atingir umas das três condições de paragem.

Na função ordenar_pontuações optei por usar a ordenação bublesort por ser mais eficaz mesmo só tendo 10 registos. E basicamente a função lê todos os registos existentes no ficheiro e ordena-os do maior para o menor. Depois volto a escrever tudo de novo com os registos ordenados descrente. Caso houvesse mais de 10 registos só vou escrever 10 registos, se não em caso de haver menos registos eu contabilizo-os através da variável "tot" e apenas os retiro e volto-os a introduzir, mas com a garantia que estão ordenados.

Na função ver_pontuacoes abro o ficheiro em modo leitura e printo cada linha do ficheiro para o jogador conseguir ver a função. Esta função é chamada no menu principal, opção 2.

No função inicializadora é onde tudo acontece tenho um menu para ele escolher se quer jogar, ver as pontuações ou sair do programa. Processo as respostas do utilizador para as funções pretendidas em conjunto com validações das respostas se estão dentro das conformidades. De notar que caso o jogador queira jogar novamente no "caso 1" como está num loop verdade(ou seja infinito) se o jogador dicidir repetir o algoritmo apenas dá clear porque será chamado tudo de novo, mas caso ele não queira e digitar algo diferente de 'S' o programa vai quebrar o loop que é sempre verdade e voltar para o menu principal.

Problemas encontrados

Os problemas que encontramos na realização do algoritmo foi no início percebermos e implementar a lógica do jogo em código, onde tínhamos que ter várias verificações e condições e por vezes principalmente na de abrir brancos foi um pouco difícil, mas a solução que encontramos foi usar o debug do devc++ e ver o que acontecia em tempo real, a cada jogada.

Posteriormente tivemos um problema na lógica para decidir se um jogador ganhou ou não e era essencialmente porque não fazíamos verificações do numero de bandeiras que havia, ou seja o jogador poderia facilmente colocar bandeiras em todo o lado que a condição das bandeiras em caso de ganho ainda não verificava isso, ou seja so fazia as comparações das duas matrizes e se tivesse bandeira onde

tivesse mina ele verificava e dizia que ganhou. A solução implementada foi fazer um loop para ver quantas bandeiras o jogador tinha colocado e posteriormente melhorar a condição para verificar se as bandeiras colocadas ultrapassam as das minas. Nesse aspeto também tivemos que adicionar mais uma nova funcionalidade que era a de o jogador poder retirar uma bandeira, ou seja se ele tivesse um '?' ao clicar no 'M' outravez na mesma coordenada este iria passar a como se ainda não tivesse jogado '_'. Mais uma vez foi uma condição logo no inicio para verificar o valor da coordenada que o jogador escolheu.

Para finalizar para fazer as pontuações no inicio pensamos em fazer com condições, por exemplo se tempo menor que 20segundos então pontuação 500, porém não era algo personalizado e seria comum jogadores terem a mesma pontuação nos 10 maiores jogadores, então a solução foi fazer uma fórmula matemática para dar a pontuação o mais personalizada possível aonde o tempo jogado era o crucial para a variação.

No fim tivemos dificuldade em passar o nosso algoritmo para C, onde sabíamos a lógica, mas a implementação não foi fácil, onde tínhamos que concetenar para o ficheiro o nome a pontuação(que vinha em string do ficheiro) mas para comparar no bublesort precisava dele em float, depois com new lines para ficar a data organizada entre outros. A solução foi pesquisar no google e descobrirmos algumas funções que fazia algum do trabalho que queríamos como fazer "split" ou "concatenar".

Divisão das tarefas

Miguel – Desenvolvimento do programa em C, aperfeiçoamento do pseudocódigo e desenvolvimento do relatório

Bernardo – passagem do código de C toda para pseudocódigo e ajuda no relatório

Conclusão

Achamos um projeto bastante interessante, onde propiciou-nos bastantes conhecimentos nesta área de desenvolvimento de um jogo. Passamos a perceber como é a lógica por de trás de um jogo e como tirar partido disso. O uso do "debug" e do desenvolvimento do jogo deu-nos uma perceção muito boa de como alguém consegue hackear um jogo, seja por erros do desenvolvedor, ou saber onde executar o código malicioso quebrando a nossa lógica, foi mesmo porreiro ver a cada jogada que fazíamos como se comportavam as variáveis em debug e pensar em maneiras de defender o nosso programa contra possíveis hackers. Concluindo, achamos que foi o melhor projeto que nos poderiam ter atribuído porque aprendemos imenso e deu gosto ver como isto tudo funciona em "background", mudou totalmente a nossa visão e para melhor, obviamente.

Pseudocódigo e explicação

Também se encontra a mesma versão no notepad "algoritmo" da pasta enviada.

```
Função ler_dimensoes (inteiro ^x, inteiro ^y): nulo
Inicio
Escrever(" Digite as dimensoes do campo: \n ")
Ler(x, y)

Se ( ( ^y > 20 ) AND ( ^x > 20 ) ) então:
        Escrever(" Nao pode ultrapassar o limite de",20,  "linhas ou colunas!\n ")
        ler_dimensoes(x, y)

senão se ( ( ^y < 0 ) AND ( ^x < 0 ) ) então:
        Escrever(" O valor de linhas e colunas precisa ser positivo\n ")
        ler_dimensoes(x, y)
Fim
```

```
Funçao ler_nivel(inteiro ^x, inteiro ^y, inteiro ^n) : nulo

        Vars:
                res_menu : inteiro

        Início

                Escrever("Digite o nível pretendendo\n Fácil ( opção 1 ) \n Médio ( opção 2 )
\n Difícil ( opção 3 )")
                ler(res_menu)
                Enquanto(res_menu <> 1 && res_menu != 2 && res_menu != 3) fazer:
                        Escrever("Opção inválida")
                        ler(res_menu)
                Se( res_menu = 1 ) então:
                        ^x = 5
                        ^y = 5
                        ^n = 2
                Se( res_menu = 2 ) então:
                        ^x = 10
                        ^y = 10
                        ^n = 4
                Senão:
                        ^x = 20
                        ^y = 20
                        ^n = 10

        Fim
```

```
Função ler_minas (inteiro ^n, inteiro x, inteiro y): nulo
Inicio

Vars:
        max_minas ← 0

Escrever(“ Digite a quantidade de minas:\n “)
Ler(n)

max_minas ← (x*y) / 2

Se ( ^n > x * y / 2 ) então:
        Escrever(“ Muitas minas, no máximo pode ser metade do campo, ou seja”,
max_minas,  “minas\n “)
        ler_minas(n, x, y)


Senão se ( ^n < 0 ) então:
        Escrever(“ A quantidade de minas precisa de ser positiva\n “)
        ler_minas(n, x, ,y)
Fim
-------------------------------------------------------------------------------------------------------------------------
Função sortear_mina(caracter campo[ ][20], inteiro x, inteiro y): nulo

Vars:
         x_m : inteiro
        y_m : inteiro
Inicio

x_m ← rand( ) MOD x
y_m ← rand( ) MOD y


Se ( campo[x_m][y_m] = ‘m‘ ) então:
        sortear_mina(campo, x, y)

senão:
        campo[x_m][y_m] ← ‘m’

Fim
-------------------------------------------------------------------------------------------------------------------------
```

```
Função preencher_minas(caracter campo[ ][ 20 ], inteiro n, inteiro x, inteiro y): nulo

Vars:
        i, j: inteiro

Inicio

Para i = 0 enquanto  i < x fazer:  (i++)
        Para j = 0 enquanto j < y fazer:  (j++)
                campo[ i ][ j ] ← ' '

Para i = 0 enquanto i < n fazer:  (i++)
        sortear_mina(campo, x, y)

Fim
-----------------------------------------------------------------------------------------------------------------------
Função exibir_campo(caracter campo[ ][ 20 ], inteiro x, inteiro y): nulo
Vars:
        i, j: inteiro
        lin, col: caracter
Inicio

lin ← 'A'
col ← 'A'

Escrever(" A mostrar o campo:\n ")

        Para i = 0 enquanto i < y fazer:  (i++) (col++)
                Escrever(" ", col)
        Escrever("\n")

        Para i = 0 enquanto i < x fazer:  (i++), (lin++)
                Escrever(lin)
                Para j = 0 enquanto j < y fazer:  (j++)
                        Escrever (campo[ i ][ j ] )
                Escrever (lin)
                Escrever (" \n ")

        Escrever (" ")
        Para i = 0, col = 'A' enquanto i < y fazer: (i++) (col++)
                Escrever(col)
        Escrever (" \n ")

Fim
-----------------------------------------------------------------------------------------------------------------------
```

```
Função contar_vizinhos(char campo[ ][ 20 ], inteiro x, inteiro y): nulo

Vars:
        i, j: inteiro
        n: caracter
Inicio

Para i = 0 enquanto i < x fazer:  (i++)
        Para j = 0 enquanto j < y fazer:  (j++)
                n ← '0'
                Se ( campo[ i ][ j ] <> 'm' ) então:
                        Se ( j – 1 >= 0 AND  campo[ i ][ j – 1] = 'm' ) então:
                                n ← n + 1
                        Se ( j + 1 < y  AND campo[ i ][ j + 1 ] = 'm' ) então:
                                n ← n + 1
                        Se ( i + 1 < x AND j – 1 >= 0 AND campo[ i + 1 ][ j – 1 ] = 'm' ) então:
                                n ← n + 1
                        Se ( i + 1 < x AND  j + 1 < y AND campo[ i + 1 ][ j + 1 ] = 'm' ) então:
                                n ← n + 1
                        Se ( i – 1  >= 0 AND j – 1 >= 0 AND campo[ i – 1 ][ j – 1 ] = 'm' ) então:
                                n ← n + 1
                        Se ( j + 1 < y AND i – 1 >= 0 AND campo[ i – 1 ][ j + 1 ] = 'm' ) então:
                                n ← n + 1
                        Se ( i + 1 < x AND campo[ i + 1 ][ j ] = 'm' ) então:
                                n ← n + 1
                        Se ( i – 1 >= 0 AND campo[ i – 1 ][ j ] = 'm' ) então:
                                n ← n + 1
                        Se ( n <> '0' ) então:
                                campo[ i ][ j ] ← n;

Fim
-------------------------------------------------------------------------------------------------------------------------
Função preencher_jogo (caracter jogo[ ][ 20 ], inteiro x, inteiro y): nulo

Vars:
         i, j: inteiro
Inicio

        Para i = 0 enquanto i < x fazer:  (i++)
                Para j = 0 enquanto j < y fazer:  (j++)
                        jogo [ i ][ j ] ← '_'

Fim
-------------------------------------------------------------------------------------------------------------------------
```

```
Função escolher_coordenada(caracter ^cx, caracter ^cy, inteiro x, inteiro y) :nulo

Vars:
        xmax, ymax : inteiro

Início
        xmax ← (inteiro) 'A' + x - 1
        ymax ← (inteiro) 'A' + y - 1

        Escrever (" Escolha a coordenada (linha e coluna):\n ")
        Ler (cx, cy)

Se ( ( ^cx < 'A' ) AND ( (inteiro)^cx > xmax ) AND ( ^cy < 'A' ) AND ( (inteiro)^cy > ymax ) )
então :
        Escrever (" Coordenadas invalidas! Escolha de novo.\n ")
        escolher_coordenada(cx, cy, x, y)

Fim




Função abrir_ou_marcar(caracter ^c) :nulo

Inicio

        Escrever (" Abrir ou marcar mina? A = abrir, M = marcar.\n ")
        Ler (c)
        Se ( ( (^c <> 'A') || (^c <> 'a') ) AND ( (^c <> 'M') || (^c <> 'm') ) ) então:
                Escrever (" Escolha invalida!\n ")
                abrir_ou_marcar(c)

Fim
-----------------------------------------------------------------------------------------------------------------------------
Função abrir_brancos(caracter jogo[ ][ 20 ], caracter campo[ ][ 20 ], inteiro x, inteiro y,
inteiro cx, inteiro cy): nulo

Início

        jogo[ cx ][ cy ] = '  ';

        Se ( cy – 1 >= 0 ) então :
                Se ( campo[ cx ][ cy – 1 ] = '  ' AND jogo[ cx ][ cy – 1 ] = '_' ) então :
                        abrir_brancos(jogo, campo, x, y, cx, cy-1)
Senão
        jogo[ cx ][ cy – 1 ] = campo[ cx ][ cy – 1 ]

Se ( cy + 1 < y ) então :
        Se ( campo[ cx ][ cy + 1 ] = ' ' AND jogo[ cx ][ cy + 1 ] = '_' ) então :
                abrir_brancos(jogo, campo, x, y, cx, cy+1)
```

```
Senão
        jogo[ cx ][ cy + 1 ] = campo[ cx ][ cy + 1 ]

Se ( cx + 1 < x ) então :
        Se ( campo [ cx + 1 ][ cy ] = ‘ ‘ AND jogo[ cx + 1 ][ cy ] = ‘_’ ) então :
                abrir_brancos(jogo, campo, x, y, cx+1, cy)
Senão
        jogo[ cx + 1 ][ cy ] = campo[ cx + 1 ][ cy ]

Se ( cx – 1 >= 0 ) então :
        Se ( campo[ cx – 1 ][ cy ] = ‘ ‘ AND jogo[ cx – 1 ][ cy ] = ‘_’ ) então :
                abrir_brancos(jogo, campo, x, y, cx-1, cy)
Senão :
        jogo[ cx – 1 ][ cy ] = campo[ cx – 1 ][ cy ]

Se ( cx + 1 < x AND cy – 1 >= 0 AND campo[cx + 1][cy – 1] = ‘ ‘ AND jogo[cx + 1][cy – 1] =
‘_’ AND ( campo[cx + 1][cy] = ‘ ‘ AND campo[cx][cy – 1] = ‘ ‘ ) ) então :
        abrir_brancos(jogo, campo, x, y, cx+1, cy-1)

Se ( cy + 1 < y AND cx +  1 < x AND campo[cx + 1][cy + 1] = ‘ ‘ AND jogo[cx + 1][cy + 1] =
‘_’ AND ( campo[cx+ 1][cy] = ‘ ‘ AND campo[cx][cy + 1] = ‘ ‘ ) ) então :
        abrir_brancos(jogo, campo, x, y, cx-1, cy-1)


Se ( cy – 1 >= 0 AND cx – 1 >= 0 AND campo[cx – 1][cy – 1] = ‘ ‘ AND jogo[cx – 1][cy – 1] =
‘_’ AND ( campo[cx – 1][cy] = ‘ ‘ AND campo [cx][cy – 1] = ‘ ‘ ) ) então :
        abrir_brancos(jogo, campo, x, y, cx-1, cy-1)

Se ( cy + 1 < y AND cx – 1 >= 0 AND campo[cx – 1][cy + 1] = ‘ ‘ AND jogo[cx – 1][cy + 1] =
‘_’ AND ( campo[cx – 1][cy] = ‘ ‘ AND campo[cx][cy + 1] = ‘ ‘ ) ) então :
        abrir_brancos(jogo, campo, x, y, cx-1, cy-1)

Fim
```

Função ordenar_pontuacoes() : nulo

Vars:
^filePointer, ^fp : ficheiro
bufferLength, i,j,k,tot : inteiro
buffer[15][bufferLength], swap[bufferLength], pontuacao_linha[10][10], pontuacoes_string[11][100], each_line[11][100], ^^token : caracter
pontuacoes_por_linha[11], int_swap : decimal

Início
bufferLength ← 255
i ← 0
k ← 0
token ← -Alocar memória para 30 espaços do tipo caracter-

filePointer ← -Abrir o ficheiro "data.txt" em modo de leitura-
-Trazer cada linha do ficheiro do ficheiro e armazenar no vetor buffer com a ajuda do iterador "i", também tenho que colocar o último caracter a "/0" para dizer ao programa que a string acaba ali-

Tot ← -iterador i-

-Copiar o conteúdo do buffer todo para outro lado para não perder, ou seja para o vetor "each_line"
- Para todas as linhas do ficheiro (contablizo através da variável "tot"), separar o conteúdo de cada linha pelo caracter "-" para conseguir só os valores das pontuações, e neste caso como vai estar no final é o segundo grupo, e armazenar no vetor pontuações_string, através do iterador "i" que será o loop que vai percorrer todas as linhas do ficheiro e que em cada uma pega a pontuação-
-Passar as pontuações para float porque veem em string-

Para i=tot enquanto i>=1 fazer: (i--)
Para j=0 enquanto j<i fazer: (j++)
Se(pontuacoes_por_linha[j] < pontuacoes_por_linha[j+1] então:
-Fazer o swap do conteúdo que está armazenado no vetor each line[j] com o each_line[j+1], que é os nomes e as pontuações e trocar as pontuações em baixo com o swap para o bublesort funcionar corretamente, tendo que mudar de lugar as pontuações também como em baixo refere-
int_swap ← pontuacoes_por_linha[j]
pontuacoes_por_linha[j] ← pontuacoes_por_linha[j+1]
pontuacoes_por_linha[j+1] ← int_swap

fp ← -Abrir o ficheiro "data.txt" em modo de escrever por cima-
Se(tot>10) então:
Para i=0 enquanto i<10 fazer: (i++)
-Escrever each_line[i] no ficheiro apontado por fp-
-Escrever uma "\n" no ficheiro apontador por fp-
Senão:
Para i=0 enquanto i<tot fazer: (i++)
-Escrever each_line[i] no ficheiro apontado por fp-

-Escrever uma “\n” no ficheiro apontador por fp-

-Fechar o ficheiro apontado por fp-

Fim

Função ver_pontuacoes() : nulo

Vars:
^filePointer : ficheiro
bufferLength : inteiro
buffer[bufferLength] : caracter

Início

bufferLength ← 255
filePointer ← -Abrir o ficheiro “data.txt” em modo leitura

-Ler cada linha do ficheiro apontado pelo apontador filePointer até ao máximo defenido no bufferLenght, e escrever o buffer que vai ser o resultado de cada linha-
-Fechar o ficheiro-

Fim

Função guarda_pontuacoes(inteiro tdiferenca, inteiro n) : nulo

Vars:
i : inteiro
pontuação : decimal
nome[100], buffer[100] : caracter
^fp : ficheiro

Início

Se(tdiferenca = 0) então:
pontuacao ← n * 10
Senão:
pontuacao ← (n * 10) + (5000/tdiferenca)

Escrever(“Pontuacão final : “ ,pontuacao)
Escrever (“Digite o seu nome para registar a pontuação”)
Ler(nome)
fp ← -Abrir o ficheiro “data.txt” em modo de “append” ou seja adicionar-
-Concatenar o nome com o “- “ e a pontuação apenas com duas casas decimais e no final um \n”
-Adicionar ao ficheiro o conjunto da concetenação-
-Fechar o ficheiro-

```
            ordenar_pontuacoes();
        Fim
```

```
Função jogar(caracter campo[ ][ 20 ], caracter jogo[ ][ 20 ],inteiro x, inteiro y, inteiro
tinicio, inteiro minas) : nulo

Vars:
        cx, cy, ca: caracter
        k,contador_flags,conta_bandeiras,i,j, count, tdiferenca,tfim : inteiro
Inicio
        count ← 0
        contador_flags ← 0
        conta_bandeiras ← 0

        -Função para dar clear no ecrã / terminal-
        exibir_campo(jogo, x, y)
        escolher_coordenada(>cx, >cy, x, y)
        abrir_ou_marcar(>ca)

        Se ( ca = 'A' || ca = 'a' ) então:
                Se ( campo[ cx – 65 ][ cy – 65 ] = 'm' ) então:
                        Para k=0 enquanto k < x fazer:  (k++)
                                Para v=0 enquanto v < y fazer:  (v++)
                                        Se(campo[k][v] = 'm' && jogo[k][v] = '?') então:
                                                contador_flags = contador_flags + 1
                        -Função para dar clear no ecrã / terminal-
                        exibir_campo (campo, x, y, minas)
                        Escrever (" Abriste uma mina! Game Over\n ");
                        guarda_pontuacoes(0, contador_flags)
                        -Retornar ao menu principal-

                Senão se ( campo[ cx – 65 ][ cy – 65 ] <> '  ' ) então :
                        jogo[ cx – 65 ][ cy – 65 ] ← campo[ cx – 65 ][ cy – 65 ]

                Senão :
                        abrir_brancos(jogo, campo, x, y, cx-65, cy-65)

                Para i = 0 enquanto i < x fazer:  (i++)
                        Para j = 0 enquanto j < y fazer: (j++)
                                Se ( ( campo[ i ][ j ] <> 'm' ) AND ( jogo[ i ][ j ] = '_' ) ) então:
                                        count ← count + 1
                                Se ( count = 0 ) então:
                                        -Função para dar clear no ecrã / terminal
                                        exibir_campo(campo, x, y, minas)
                                        tfim ← -Função para retornar o tempo exato no
momento em que é chamada-
                                        tdiferenca ← tfim - tinicio
```

```
                                        Escrever (“ Todos os campos foram abertos. Ganhaste
o jogo em” tdiferenca, “segundos! Parabéns!\n “ )
                                        guarda_pontuacoes(tdiferenca,minas)
                                        -Retornar ao menu principal-

        Senão:
                Se(jogo[(inteiro) cx – 65 ] [ (inteiro) cy – 65 ] = ‘?’) então:
                        jogo[(inteiro) cx – 65 ] [ (inteiro) cy – 65 ] ← ‘_’
                Senão:
                        jogo[ (inteiro)cx – 65 ][ (inteiro)cy – 65 ] = ‘?’

                Para i = 0, enquanto i < x fazer:  (i++)
                        Para j = 0, enquanto j < y fazer  (j++)
                                Se ( ( campo[ i ][ j ] = ‘m’ ) AND ( jogo[ i ][ j ] = ‘_’ ) ) então:
                                        count ← count + 1
                                Se( jogo[i][j] = ‘?’ ) então:
                                        conta_bandeiras ← conta_bandeiras + 1

                Se ( count = 0  && conta_bandeiras <= minas) então :
                        -Função para limpar o ecrã-
                        exibir_campo(campo, x, y,minas)
                        tfim ←  -Função que retorna o tempo exato no momento-
                        Escrever(“ Todas as minas foram marcadas. Ganhaste o jogo em”
,tfim – tinicio, “segundos! Parabéns!\n “ )
                        tdiferenca ← tfim – tinicio
                        guarda_pontuacoes(tdiferenca,minas)
                        -Função para retornar ao menu principal/ sair da função-
                Senão se( conta_bandeiras > minas ) então:
                        Escrever(“Tens mais bandeiras marcadas que minas existentes”)
                        -Manter a mensagem durante 1 segundo para o jogador conseguir
ver-

        jogar(campo, jogo, x, y, tinicio,minas)
Fim
-------------------------------------------------------------------------------------------------------------------------------

Início(função principal)

        Vars:
                 x, y, minas, tinicio, res_menu, opcao: inteiro
                campo[ 20 ][ 20 ] de caracter
                jogo[ 20 ][ 20 ] de caracter
                resposta: caracter
        Repetir :
                Escrever(“Opções do menu: \n”)
                Escrever(“Opção 1: Jogar\n”)
                Escrever(“Opção 2: Ver pontuacao\n”)
                Escrever(“Opção 0: Sair \n”)
                Ler(opcao)

                Escolha(opcao) :
```

```
                    Caso 1:
                        Enquanto verdade fazer:
                            -Limpar terminal-
                            Escrever("Digite um dos 3 níveis predefinidos ( opção 1
) ")
                            Escrever("Escolher detalhadamente o campo e as
minas ( opção 2 )")
                            Escrever("Voltar ( opção 0 )")
                            Ler(res_menu)
                            Se(res_menu = 0) então:
                                -Limpar o terminal-
                                -Quebrar o enquanto verdade e voltar ao menu
principal-
                            Enquanto(res_menu <> 1 && res_menu <> 2) então:
                                Escrever("Opção inválida")
                                Ler(res_menu)
                            Se(res_menu = 2) então:
                                ler_dimensoes(>x, &y)
                                ler_minas(>minas, x, y)
                            Senão:
                                ler_nivel(>x,>y,>minas)

                            preencher_minas(campo, minas, x, y)
                            contar_vizinhos(campo, x, y)
                            preencher_jogo(jogo, x, y)
                            tinicio = -Função para determinar o momento exato-
                            jogar(campo, jogo, x, y, tinicio,minas)
                            Escrever(" Deseja jogar novamente? S= sim, N= não.\n
")
                            Ler(resposta);

                            Se ( resposta = 'S' || resposta = 's' ) então:
                                -Função para dar clear ao ecrã-
                            Senão:
                                -Função para quebrar o loop do enquanto e sair-
                        -Quebrar o caso-

                    Caso 2:
                        -Limpar o terminal-
                        ver_pontuacoes()
                        -Quebrar o caso-

                    Caso 0:
                        Escrever("A sair...")
                        -Fechar o programa-

                    Outros casos:
                        Escrever("Opção inválida, tenta novamente")
                        -Quebrar o caso-
Fim
```