INSTITUTO POLITÉCNICO DO CÁVADO E AVE

TRABALHO PRÁTICO 1

RELATÓRIO

JORGE MIGUEL AREZES NORO | 15705

ENGENHARIA E DESENVOLVIMENTO DE JOGOS DIGITAIS

PARADIGMAS DE PROGRAMAÇÃO II

DOCENTE: LUÍS GONZAGA MARTINS FERREIRA

1º SEMESTRE

1. Estruturas de Dados Estáticas

Para o desenvolvimento da parte 1 do trabalho prático, a estrutura de dados estática capaz de armazenar todos os dados do cartão matriz será um *array* de inteiros. Uma possível solução, a que foi implementada neste trabalho, passou por separar as células do cartão matriz em *structs* chamados de *Nodes.* Cada *Node* terá um *array* de inteiros, em que será armazenada a sequência de números.

Para construir a matriz, usamos então outro *struct* chamado de *Row.* Este *struct* armazena o número de *Nodes* igual ao número de colunas definidas na matriz (ver *Types.h)*. O *struct Matrix* tem como membro um *array* do tipo *Row*, em que armazena o número de linhas definidas na matriz. É possível ver na imagem abaixo a relação dos *structs* entre si:

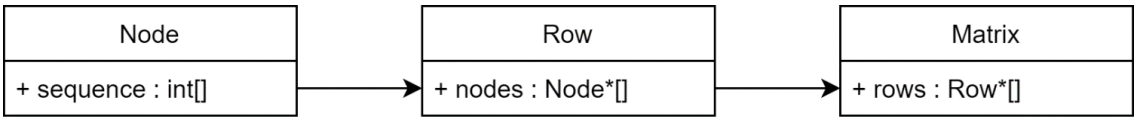

Figura 1-Relação das estruturas de dados.

A solução pode parecer complexa demais para a resolução deste problema simples, mas desta forma deixamos espaço para o crescimento da aplicação, como por exemplo, adicionar informação sobre determinado *Node* ou sobre determinada *Row*. Uma solução mais simples e básica podia ser implementar simplesmente um *array* tridimensional:

```
int matrix[ROWS_NUMBER][COLUMNS_NUMBER][SEQUENCE_SIZE];
```

Code Snippet 1- Possível implementação da matriz, usando um simples array tridimensional

As funções de alocação de memória destes objetos são encontradas no ficheiro *ObjectCreation.c*. Fica abaixo um exemplo (alocação em memória de uma matriz):

```
typedef struct {
	Row* rows[ROWS_NUMBER];
} Matrix;
Matrix* createMatrix() {
	Matrix* m = (Matrix*)malloc(sizeof(Matrix));
	for (int i = 0; i < ROWS_NUMBER; i++) {
		m->rows[i] = createRow();
	}
	return m;
};
```

Code Snippet 2- Alocação em memória de uma matriz.

O código chama a função *createRow()*, que por sua vez chama a função *createNode()*. Esta, vai popular o *Node* com uma sequência aleatória de números. Podemos ver essa função abaixo:

```
typedef struct {

	int sequence[SEQUENCE_SIZE];

} Node;

Node* createNode() {

	Node* n = (Node*)malloc(sizeof(Node));

	for (int i = 0; i < SEQUENCE_SIZE; i++) {

		n->sequence[i] = randomInt(0, 10);

	}

	return n;

};
```

Code Snippet 3- Alocação em memória de um Node

No que diz respeito a algoritmos, esta primeira parte do trabalho é relativamente simples. Podemos analisar como exemplo, a função em que pedimos ao utilizar que valide uma operação. Fica abaixo o código em excerto (algumas partes foram retiradas para auxiliar a leitura neste relatório, ver ficheiro *UtilityFunctions.c* para analisar o código completo):

```
int validateOperation(Matrix* matrix) {
	// error handling
	(...)
	int isValid = 0;
	for (int i = 0; i < SECURITY_LEVEL; i++) {

		int row, col, digit, ans;
		char rowChar;

		row = randomInt(0, ROWS_NUMBER - 1);
		col = randomInt(0, COLUMNS_NUMBER - 1);
		digit = randomInt(0, SEQUENCE_SIZE - 1);
		rowChar = 65 + row;

		// output to inform user
		(...)
		scanf("%i", &ans);
		getchar();
		isValid = validatePosition(ans, row, col, digit, matrix);

		// break after one failed attempt
		if (isValid == 0)
			break;
	}
	return isValid;
};
```

Code Snippet 4- Função para validar uma operação

A função gera números aleatórios para a linha, coluna e posição do dígito pretendido. De seguida, mostra ao usuário qual a célula e posição do dígito que se pretende e obtém a resposta. É chamada uma função que valida essa resposta. Vejamos abaixo:

```
int validatePosition(int ans, int row, int col, int digit, Matrix* matrix) {

	// error handling
	(...)

	// get the matrix digit in the given position
	int matrixDigit = matrix->rows[row]->nodes[col]->sequence[digit];

	if (matrixDigit == ans) {

		return 1;

	}
	else {

		return 0;

	}

};
```

Code Snippet 5- Função para validar escolha do utilizador

Ao realizar este trabalho, houve como ideal apelar à parametrização do programa. Existem várias variáveis que podem ser ajustadas conforme as preferências do utilizador. Ficam abaixo ditas variáveis com a descrição do seu papel no programa:

```
/*
Number of columns in the Matrix Card
*/
#define COLUMNS_NUMBER 8

/*
Number of rows in the Matrix Card
*/
#define ROWS_NUMBER 8

/*
Number of digits in a Node sequence
*/
#define SEQUENCE_SIZE 3

/*
Number of times the user will be asked for Matrix Card input
*/
#define SECURITY_LEVEL 3
```

Code Snippet 6- Variáveis para parametrização do programa