Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Programação

Trabalho Prático – Jogo do Semáforo

Docente: Francisco Pereira

Pedro Jorge Fernandes Morais – 2018020733 – LEI

quarta-feira, 2 de junho de 2021

Cofinanciado por:

Índice

Cofinanciado por:

Índice de Figuras

Cofinanciado por:

1 Introdução

Este relatório tem como principal objetivo demonstrar os conhecimentos obtidos na unidade curricular de Programação. Para tal, foram utilizados diversos conteúdos lecionados no decorrer deste segundo semestre, nomeadamente memória dinâmica, *arrays* dinâmicos e listas ligadas.

Assim sendo, este relatório é realizado no âmbito do trabalho prático da unidade curricular Programação lecionada no segundo semestre do primeiro ano do curso de Engenharia Informática do Instituto Superior de Engenharia de Coimbra (ISEC).

O trabalho é composto pela seguinte forma:

- 1ª Parte – Estruturas de dados utilizadas;
- 2ª Parte – Estruturas Dinâmicas Implementadas;
- 3ª Parte – Opções Tomadas durante a Implementação;
- 4ª Parte – Manual de Utilização.

Cofinanciado por:

2 Estruturas de Dados

No decorrer da realização deste trabalho prático existiu a necessidade de criar algumas estruturas de dados de forma a manter os dados de algumas partes do projeto mais organizados. Para tal foram criadas três diferentes estruturas de dados.

2.1 Estrutura Tabuleiro

A primeira estrutura de dados que foi criada visa guardar os dados sobre o tabuleiro de jogo, nomeadamente qual o seu tamanho, o número de linhas e de colunas pelas quais este é constituído, e o tabuleiro em si com os dados inseridos pelo utilizador. Para tal foi utilizada a seguinte estrutura:

```
//estrutura do tabuleiro
struct Tabuleiro {

    //guarda num array bidimensional dinâmico o estado do tabuleiro
    char **tabuleiro;

    //guarda o número de linhas e de colunas do tabuleiro
    int nLinhas;
    int nColunas;
};
```

Figura 1 - Estrutura de dados para o tabuleiro de jogo

Como é possível verificar na Figura 1 na estrutura tabuleiro podemos encontrar três variáveis, a primeira é um ponteiro de ponteiros do tipo caractere, sendo esta um ponteiro de ponteiros devido à alocação de memória dinâmica, pois, o tamanho do tabuleiro poderá variar com o decorrer do programa. Podemos ainda verificar que temos duas variáveis do tipo inteiro, para o número de linhas e de colunas do tabuleiro.

Cofinanciado por:

2.2 Estrutura Jogadores

A segunda estrutura criada tem como objetivo armazenar os dados de um jogador, sendo que para isso esta vai guardar a sua identificação, se é o jogador A ou B, bem como o número de pedras restantes que poderá colocar no tabuleiro de jogo e ainda o número de vezes que pode realizar uma operação que afete o tamanho do tabuleiro. Para isso foi desenvolvida a seguinte estrutura:

```
//estrutura para guardar os dados de um jogador
struct dadosJogadores{

    char identificacao;
    int pedra;
    int aumentarTabuleiro;

};
```

Figura 2 - Estrutura de dados para um jogador

Através da Figura 2 podemos verificar que existem três variáveis, sendo a primeira do tipo caractere e armazena a identificação do utilizador. As restantes duas variáveis são do tipo inteiro e armazenam as restantes vezes que o utilizador pode colocar pedras no tabuleiro, e as restantes operações que o utilizador pode realizar que afetem o tamanho do tabuleiro.

Cofinanciado por:

2.3 Estrutura Jogada

Para concluir a estruturas de dados criadas temos a estrutura Jogada em que armazena toda a informação relevante sobre uma jogada realizada no decorrer do programa. Nesta estrutura de dados podemos verificar que existem diversas variáveis entre elas uma para armazenar o tabuleiro da jogada, uma para identificar o jogador da jogada, outra para identificar o número da jogada, bem como outras duas para identificar a linha e coluna afetada pela jogada e uma pequena descrição sobre a jogada que ocorreu. Temos ainda uma última variável do tipo ponteiro para esta mesma lista, pois esta estrutura é utilizada para a criação de uma lista ligada.

```
//estrutura para guardar as jogadas numa lista ligada
struct Jogada{

    //guarda um tabuleiro
    tabuleiro tab;
    //guarda o caractere que identifica o jogador
    char jogador;
    //guarda o número da jogada, linha e coluna afetada
    int nJogada;
    int linhaAfetada;
    int colunaAfetada;
    //guarda ainda uma pequena descrição da jogada
    char infoJogada[100];

    //ponteiro para o próximo nó da lista ligada
    jogada *next;

};
```

Figura 3 - Estrutura de dados de uma jogada que ocorreu

Como podemos ver na Figura 3 a estrutura Jogada cria uma variável do tipo estrutura tabuleiro, com o objetivo de armazenar o tabuleiro daquela jogada. De seguida temos uma variável do tipo caractere que armazena a identificação do jogador que realizou a jogada. Temos ainda três variáveis do tipo inteiro, em que a primeira armazena o número da jogada, e as seguintes a linha e coluna afetada pela jogada. Podemos ainda encontrar uma outra variável do tipo caractere que armazena uma *string* com cem caracteres com uma breve descrição da jogada. Para terminar temos um ponteiro para uma estrutura deste mesmo tipo, jogada, em que esta variável é necessária, pois, esta estrutura tem como objetivo ser utilizada para a criação de uma lista ligada.

Cofinanciado por:
CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

3 Estruturas Dinâmicas Implementadas

No decorrer do projeto foram implementadas duas estruturas dinâmicas. Estas foram necessárias devido ao tamanho das mesmas poder variar durante a execução do trabalho. Embora ambas sejam estruturas dinâmicas estas não são iguais, temos uma em que fazemos uso de uma lista ligada e a outra utilizamos um *array* bidimensional dinâmico.

3.1 *Array* dinâmico bidimensional

Foi utilizado um *array* dinâmico bidimensional para resolver o problema do tabuleiro de jogo, tabuleiro este que no decorrer do programa poderá sofrer alterações no seu tamanho. Foi escolhida a solução de uma *array* bidimensional, pois, como são realizadas diversas operações de pesquisa aleatória ao *array* torna-se mais simples, prático e rápido o uso de um *array*, quando comparado com uma lista ligada que apenas tem acesso sequencial. Este *array* bidimensional dinâmico encontra-se na estrutura tabuleiro, Figura 1.

```
//função que cria um estrutura do tipo tabuleiro e a inicializa
tabuleiro inicializarTabuleiro()
{

    //declara a estrutura do tipo tabuleiro
    tabuleiro tab;

    //recebe um número random entre 3 e 5, valor este que vai ser o número de linhas/colunas do tabuleiro de jogo
    int tam = intUniformRnd(3,5);

    //cria array dinâmico com 'tam' número de linhas
    tab.tabuleiro = (char**)malloc(sizeof(char*)*tam);

    //verifica se a alocação de memória foi realizada com sucesso
    if(tab.tabuleiro == NULL)
    {
        printf("Ocorreu um erro a alocar memória!");
        return;
    }

    //para cada linha cria 'tam' número de colunas
    for (int i=0; i<tam; i++)
    {
        //aloca o espaço para cada coluna
        tab.tabuleiro[i] = (char*)malloc(sizeof(char)* tam );

        //verifica se estas foram alocadas com sucesso
        if(tab.tabuleiro[i] == NULL)
        {
            printf("Erro na alocacao de memoria");
            free(tab.tabuleiro);
            return;
        }

        //inicializar o tabuleiro com espaços em branco
        for (int j = 0; j < tam; j++)
        {
            tab.tabuleiro[i][j] = ' ';
        }

    }

    //inicializa o número de linhas e colunas com o número random recebido em "tam"
    tab.nLinhas = tam;
    tab.nColunas = tam;

    return tab;
}
```

Figura 4 – Inicialização do *array* dinâmico

No código da Figura 4 é possível observar todo o código necessário para a criação do array bidimensional dinâmico.

Inicialmente é criada uma estrutura do tipo tabuleiro onde serão armazenados os dados. Posteriormente é gerado um valor aleatório entre 3 e 5, como pedido no enunciado, para determinar o tamanho do tabuleiro, sendo este inicialmente quadrado. De seguida é alocado espaço para as linhas, sendo o número de linhas dado pelo valor aleatório. É realizada uma verificação à alocação de memória. Depois vamos a cada linha da tabela e alocamos memória para as colunas, valor dados pelo número aleatório, e é também realizada a verificação à alocação. Caso corra mal é libertada a memória da tabela. Para terminar é inicializada cada coluna com um espaço em branco e no fim os valores das linhas e das colunas é igual ao número aleatório.

Para além desta função existem outras duas que influenciam o tamanho do tabuleiro, sendo que estas realizam operações de realocação de memória para incrementar as linhas ou colunas do tabuleiro sem ter perda de dados.

3.2 Lista ligada

O segundo tipo de estrutura dinâmica implementada é uma lista ligada, que é constituída por nós, em que cada nó guarda a posição do próximo, tendo por isso acesso puramente sequencial. Esta solução foi optada, pois, apesar de ser pedido no enunciado, faz sentido ser usada, devido à lista ligada ser sempre consultada sequencialmente e não ser necessário ver apenas uma jogada especifica.

```
//estrutura para guardar as jogadas numa lista ligada
struct Jogada{

    //guarda um tabuleiro
    tabuleiro tab;
    //guarda o caractere que identifica o jogador
    char jogador;
    //guarda o número da jogada, linha e coluna afetada
    int nJogada;
    int linhaAfetada;
    int colunaAfetada;
    //guarda ainda uma pequena descrição da jogada
    char infoJogada[100];

    //ponteiro para o próximo nó da lista ligada
    jogada *next;

};
```

Figura 5 - Estrutura de dados utilizada para criar a lista ligada

Cofinanciado por:

Como é possível verificar na Figura 5 existem diversos dados que serão guardados, inclusive uma estrutura dinâmica, Array dinâmico bidimensional. Podemos ainda observar que no fim da estrutura é possível encontrar um ponteiro para o mesmo tipo de dados que é criado, sendo este o ponteiro referido em cima como o que guarda a posição do próximo nó. Para uma explicação mais detalhada sobre as variáveis, esta encontra-se em Estrutura Jogada.

Na criação de cada nó da lista ligada existem dois pontos importante, a alocação de memória e a inserção do nó na lista.

```
//cria dois ponteiro, um para o novo nó da lista e outro auxiliar para manipular a lista ligada
ponteiroJogadas nova, aux;

//aloca memória para o novo nó da lista
nova = malloc(sizeof(jogada));

//verifica se a memória foi bem alocada
if (nova == NULL)
{
    printf("Erro na realocacao!\n");
    return listaJogadas;
}
```

Figura 6 - Alocação de memória para o nó da lista ligada

Como é possível observar na Figura 6 são criados dois ponteiros do tipo Jogada, sendo um para armazenar os dados do novo nó da lista e o outro para manipular a lista de forma a inserir este novo nó no local correto. É alocada memória para o tamanho de uma estrutura jogada e verifica-se se esta alocação foi realizada com sucesso.

Cofinanciado por:

```
//se a lista fornecida estiver vazia coloca o novo nó no inicio da lista
if(listaJogadas == NULL)
{
    listaJogadas = nova;
} else
{
    //caso contrário percorre a lista fornecida até chegar ao último nó
    aux = listaJogadas;

    while(aux->next != NULL)
    {
        aux = aux->next;
    }

    // e coloca o novo nó como o último nó da lista
    aux->next = nova;
}
```

Figura 7 - Inserção do novo nó na lista ligada

Na Figura 7 podemos verificar que a inserção do nó será feito no final na lista, sendo que para isso tem duas condições, na primeira caso a lista ligada se encontre vazia insere o nó logo ao inicio, caso contrário vai percorrer toda a lista ligada com a variável auxiliar, criada anteriormente, e quando chegar ao ultimo nó presente na lista vai adicionar o novo nó criado, sendo que os dados para este novo nó são inseridos entre o código da Figura 6 e da Figura 7.

Cofinanciado por:

4 Opções Tomadas na Implementação

Na realização deste trabalho prático existiram momentos onde se viu necessário a procura de estratégias para a resolução de diversos problemas. Em vários momentos foram encontradas várias opções para a resolução de um certo problema. Nestes casos teve de existir uma tomada de decisão de qual melhor se adapta à visão do projeto em geral e não apenas para o problema específico. Alguns destes problemas já foram referidos anteriormente, nomeadamente, o formato do tabuleiro de jogo, para além deste existiram ainda mais, como, a forma como é realizada a alternância entre os jogadores e o formato dos dados ao serem guardados em um ficheiro binário.

4.1 Formato tabuleiro de jogo

Este problema já foi referido antes, no tópico *Array* dinâmico bidimensional, neste tópico são faladas as duas opções ideais, um *array* bidimensional dinâmico ou uma lista ligada. Para além destas estratégias foi ainda pensada numa outra que foi implementada numa fase inicial do projeto, sendo ela a utilização de uma *array* unidimensional dinâmico. Esta estratégia tem alguns problemas, embora seja mais simples de perceber devido a só ter uma dimensão terá uma enorme complexidade na questão de incrementar colunas. Visto que as linhas estariam seguidas seria necessário que ao incrementar as colunas existisse uma alteração aos dados sendo que teriam de alterar de posição para manter o estado do tabuleiro o mesmo. Para além disto seria mais complexo a verificação de vitória. Estes dois motivos levaram à alteração para um *array* bidimensional dinâmico.

Cofinanciado por:

4.2 Alternância de jogadores

O segundo problema encontrado foi como seria feito a alternância entre os dois jogadores no decorrer do jogo. Foram pensadas duas soluções, mas ambas se baseavam na mesma ideia, a paridade do número de jogadas, ou seja, verificar se o número de jogadas é par ou ímpar e alternar o jogador ativo em conforme. Para aplicar esta ideia foram pensadas duas soluções, sendo elas, um ponteiro para o jogador atual ou um *array* com os jogadores e seria utilizado como índice de acesso ao *array* o resto da divisão inteira do número de jogadas por dois. Na primeira solução, do ponteiro, seria necessária uma verificação através de um "*if"* e através deste alterar a variável apontada pelo ponteiro. Para a segunda solução é apenas necessário criar um *array* onde o jogador A representa o índice '0' e o jogador B o índice '1'. Com esta solução é apenas necessário utilizar o *array* com índice "nJogadas%2", que significa que devolve o resto da divisão inteira do número de jogadas por dois. É ainda importante referir que o número de jogadas começa em zero, pois, caso contrário seria o jogador B o primeiro a jogar. Ambas as soluções são válidas, sendo que foi optado pela estratégia do *array*.

4.3 Gravação de dados no ficheiro binário

Para terminar temos o problema de quando um jogador desejar colocar o jogo em pausa para continuar mais tarde. Neste problema para as variáveis "simples" não surgiu um problema, pois, é apenas guardada a variável. O que se demonstrou um problema foi guardar o estado do tabuleiro de jogo, sendo que o problema é derivado de este ser uma variável do tipo ponteiro de ponteiros. Para solucionar esta questão foram pensadas duas soluções, guardar cada célula do tabuleiro individualmente ou guardar uma linha completa de cada vez, pois, não se demonstrou possível guardar a estrutura que guarda o estado do tabuleiro, nem a variável do estado do tabuleiro. Esta questão relevou-se não apenas para guardar o estado do tabuleiro de jogo atual, bem como para guardar os diversos estados do tabuleiro guardados no registo de jogadas. Para tal, foi optada por a solução de guardar linha a linha do tabuleiro, pois, guardar cada célula individualmente necessitaria de mais código, e possivelmente seria uma solução com *performance* inferior.

Cofinanciado por:

5 Manual de Utilização

Para ajudar os jogadores será mostrado um pequeno manual de utilização do jogo. Este manual tem como objetivo mostrar uma pequena descrição às jogadas que o utilizador poderá realizar.

5.1 Menu Principal

Para começar temos o menu principal.

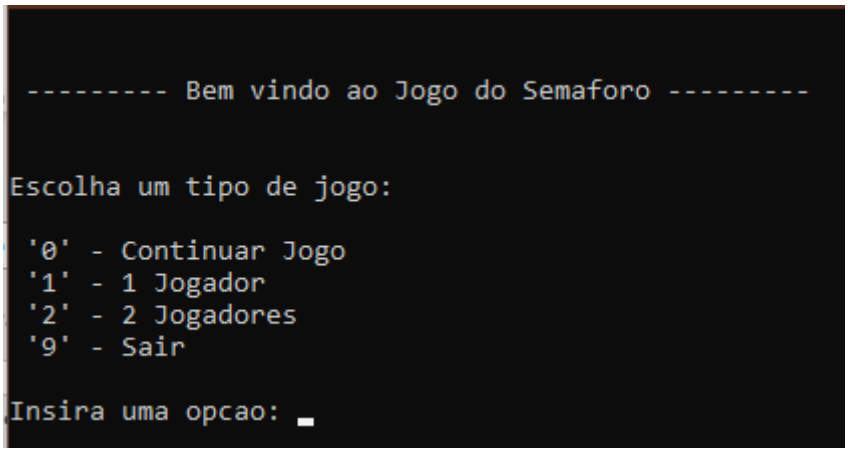

Figura 8 - Manual de utilização, Menu Principal

Como é possível ver na Figura 8 existem quatro opções.

A primeira opção, ‘0’, é a opção que permite continuar um jogo interrompido anteriormente. Esta opção irá apenas aparecer caso exista o ficheiro “jogo.bin”.

Na segunda opção,’1’, temos o modo de jogo de uma pessoa, ou seja, contra o computador.

A terceira opção, ‘2’, permite-nos fazer um duelo entre duas pessoas, dois jogadores humanos.

Para terminar, opção ‘9’, permite sair do jogo.

Cofinanciado por:

5.2 Continuar Jogo

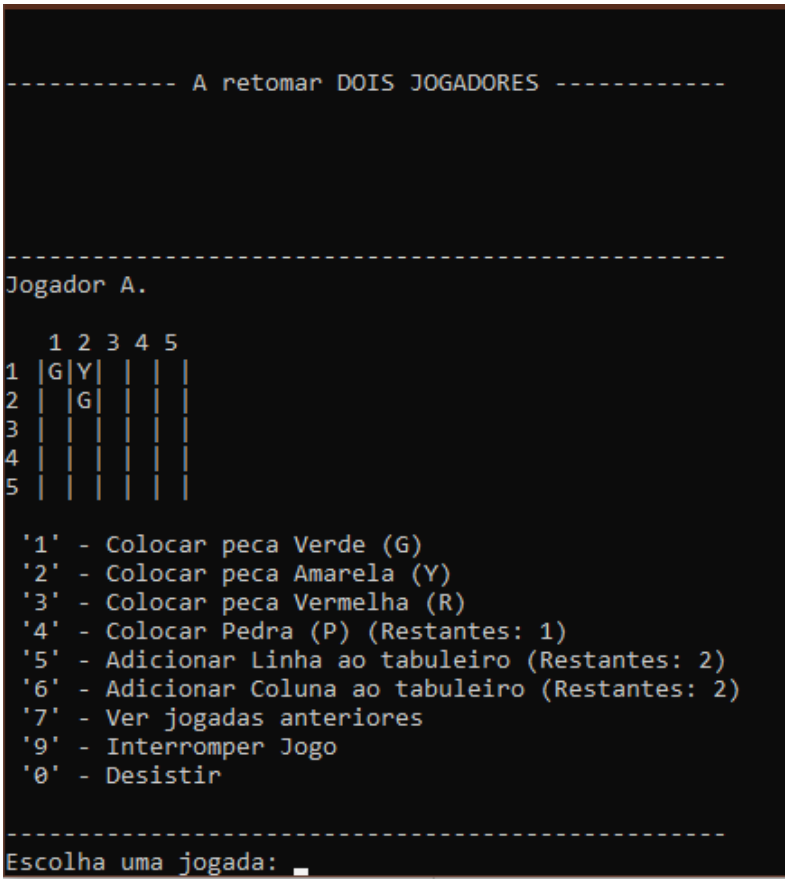

Figura 9 - Manual de utilização, Continuar Jogo

Ao escolhermos a opção "0 – Continuar Jogo" será lido o ficheiro com os dados de jogo. Após ser realizada a leitura irá aparecer no ecrã o tipo de jogo, neste caso dois jogadores, de seguida qual o jogador ativo de momento, o tabuleiro de jogo e as diversas opções do utilizador.

Cofinanciado por:

5.3 Um jogador

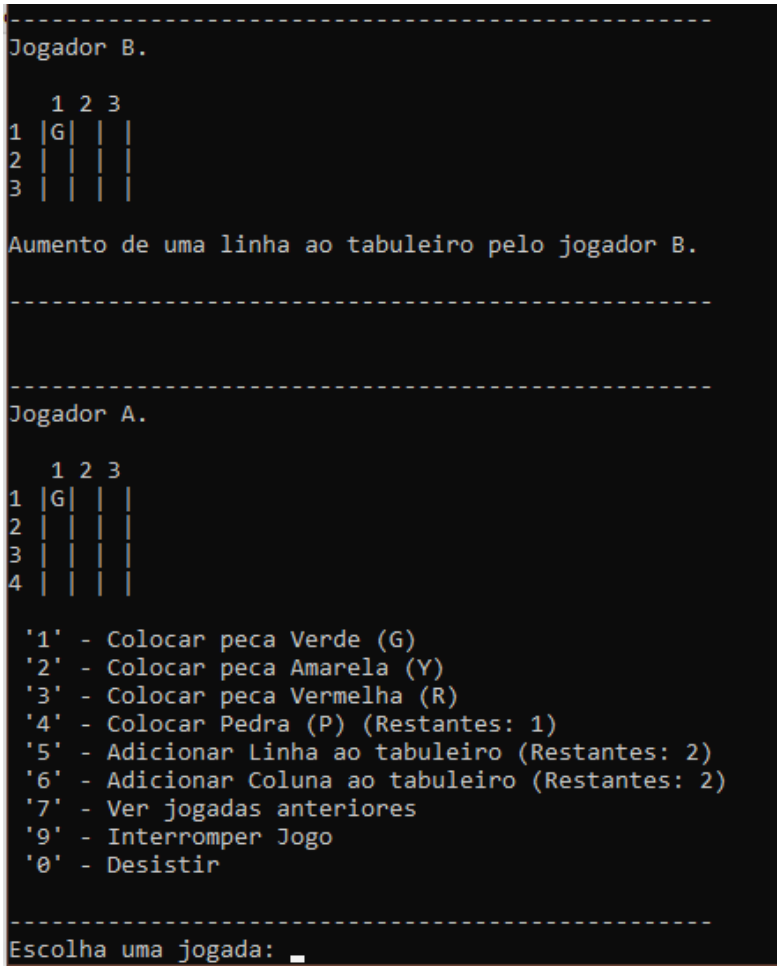

Figura 10 - Manual de utilização, Um Jogador

Se escolhermos a opção "'1' – 1 Jogador", o utilizador será designado por "Jogador A", e o computador por "Jogador B". Quando o jogador A realizar uma jogada, logo de seguida será mostrada a indicação da vez do jogador B, será mostrado o tabuleiro após a ação do utilizador e será ainda mostrado um texto descritivo da jogada realizada pelo computador. O efeito desta jogada no tabuleiro será mostrado na vez do utilizador.

Cofinanciado por:

5.4 Dois Jogadores

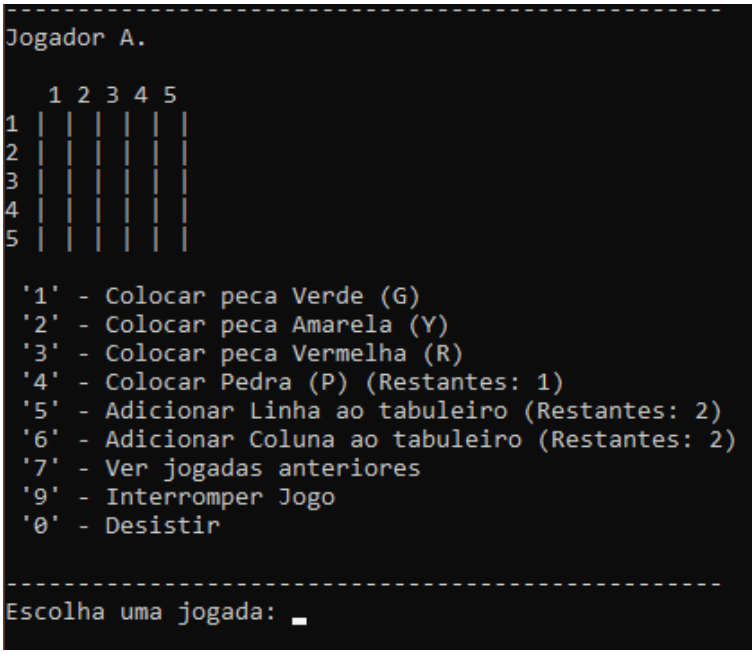

Figura 11 - Manual de utilização, Dois Jogadores

Se escolhermos a opção "'2' – 2 Jogadores" será mostrado sempre o tabuleiro com a jogada do jogador anterior, e são mostradas as possíveis jogadas que o utilizador pode realizar. Sempre que um utilizador realizar uma jogada válida existirá a alternância para a vez do outro jogador.

Cofinanciado por:

5.5 Opções do Jogador

Um jogador pode realizar nove possíveis ações, sendo que duas delas não contam como jogadas, pois, não implicam que passe para a vez do outro jogador. Para uma melhor compreensão destas diversas opções será feita uma breve descrição de cada uma.

5.5.1. Colocar Peça Verde

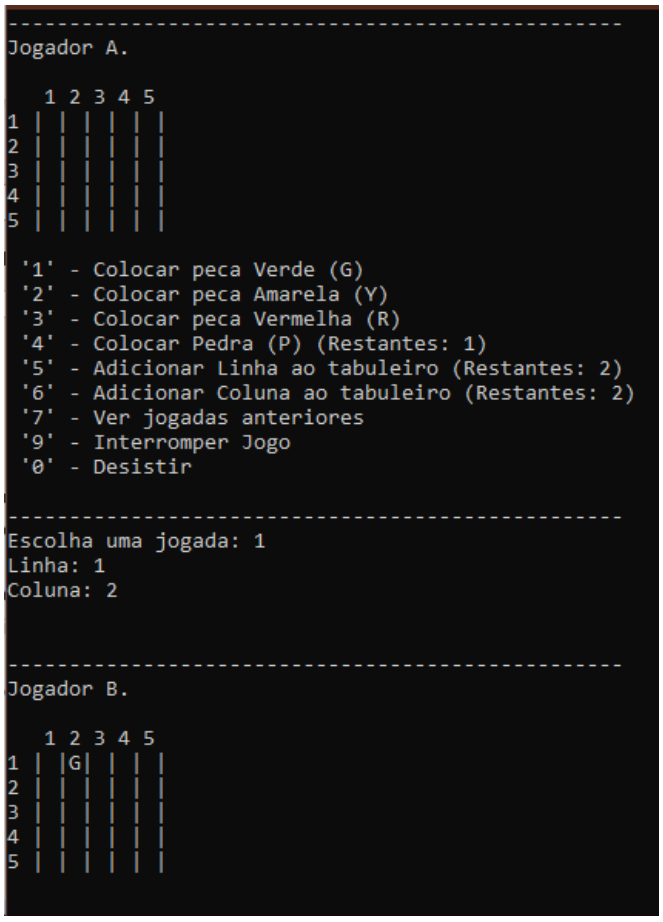

Figura 12 - Manual de utilização, Inserir Peça Verde

Na Figura 12 é possível verificar que o utilizador escolheu a opção '1'. Esta opção indica que este pretende inserir uma peça de cor verde no tabuleiro. Após inserir qual a operação que pretende realizar será questionado qual a posição no tabuleiro onde pretende que esta tome lugar, ou seja, onde pretende que a peça verde seja colocada. Para tal são realizados uma série de testes, desde a verificação se a opção inserida é válida, se o valor para as linhas e colunas é um valor válido, e se esta posição do tabuleiro apresenta todas as condições necessárias para que seja colocada uma peça verde, sendo que neste caso é necessário que a posição do tabuleiro se encontre vazia.

Cofinanciado por:

5.5.2. Colocar Peça Amarela

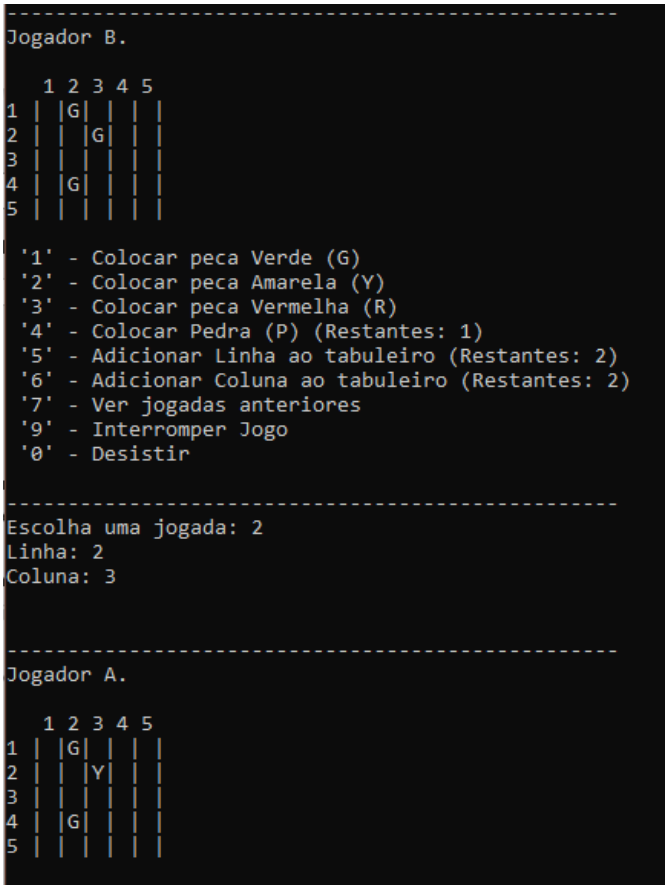

Figura 13 - Manual de utilização, Inserir Peça Amarela

Tal como para a peça Verde, para a peça amarela é pedido as coordenadas no tabuleiro, linha e coluna, onde se pretende que seja inserido esta peça. Também para esta é realizada a verificação da opção do menu inserida e das coordenadas inseridas. A diferença de esta operação para a operação de Colocar Peça Verde é apenas o requisito para a posição do tabuleiro, sendo que neste caso se deve encontrar na posição designada pelo utilizador uma peça verde, caso contrário, será uma jogada inválida e continuará a ser o mesmo jogador a jogar.

Cofinanciado por:

5.5.3. Colocar Peça Vermelha

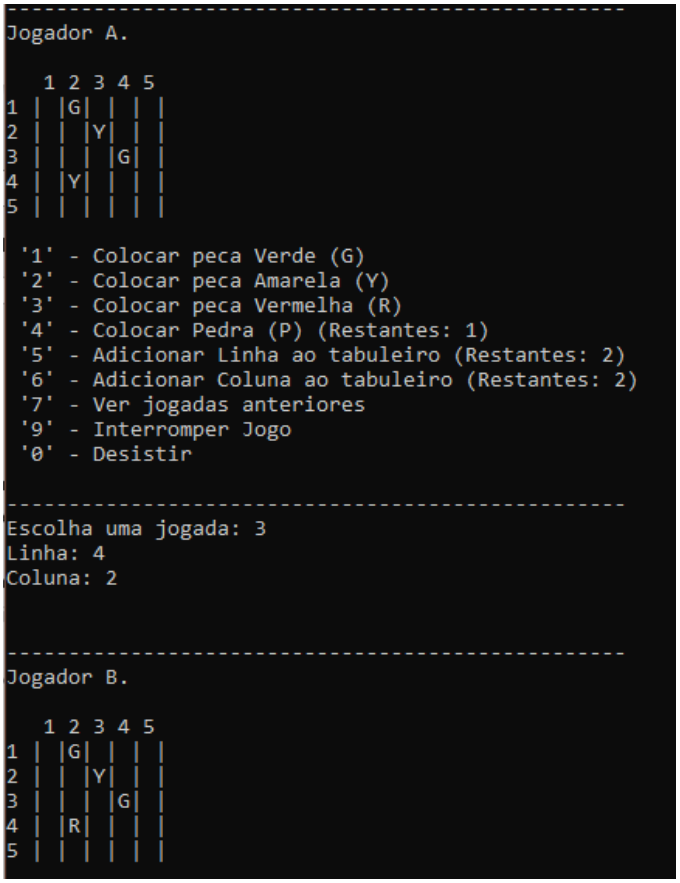

Figura 14 - Manual de utilização, Inserir Peça Vermelha

A operação colocar peça vermelha é bastante semelhante às operações colocar peça verde e amarela, tal como nas outras são pedidas as coordenadas do tabuleiro onde pretende inserir a peça, e são realizadas as verificações necessárias. A diferença para estas duas outras operações consta no requisito do tabuleiro, sendo que para colocar uma peça vermelha é necessário ter nessa posição uma peça amarela, colocada anteriormente.

Cofinanciado por:

5.5.4. Colocar Pedra

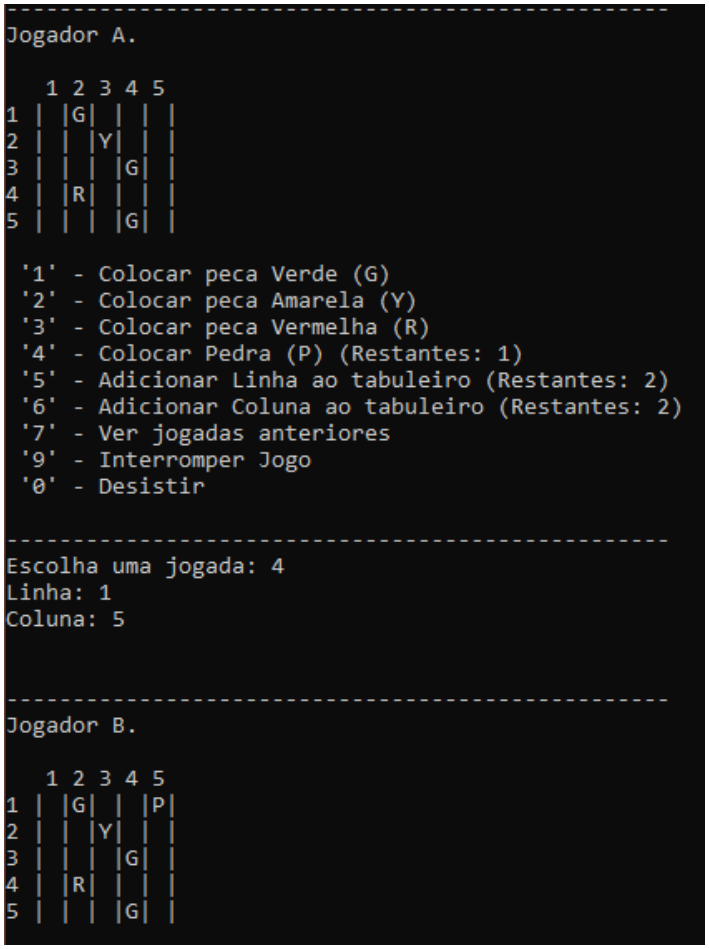

Figura 15 - Manual de utilização, Inserir Pedra

Ao contrário das operações de colocar peça de cor, a operação de Colocar Pedra tem não uma, mas duas restrições, sendo que para que seja possível colocar uma pedra é necessário que as coordenadas fornecidas pelo utilizador apontem para uma célula vazia do tabuleiro, e para além disto é necessário tomar em conta que cada utilizador pode apenas colocar uma pedra no tabuleiro por jogo. Cada jogador pode verificar quantas pedras lhe restam nas opções do menu, sendo que esta atualiza conforme as jogadas realizadas. São também pedidas coordenadas e são verificados os valores inseridos pelo utilizador.

Cofinanciado por:

5.5.5. Adicionar Linha ao Tabuleiro

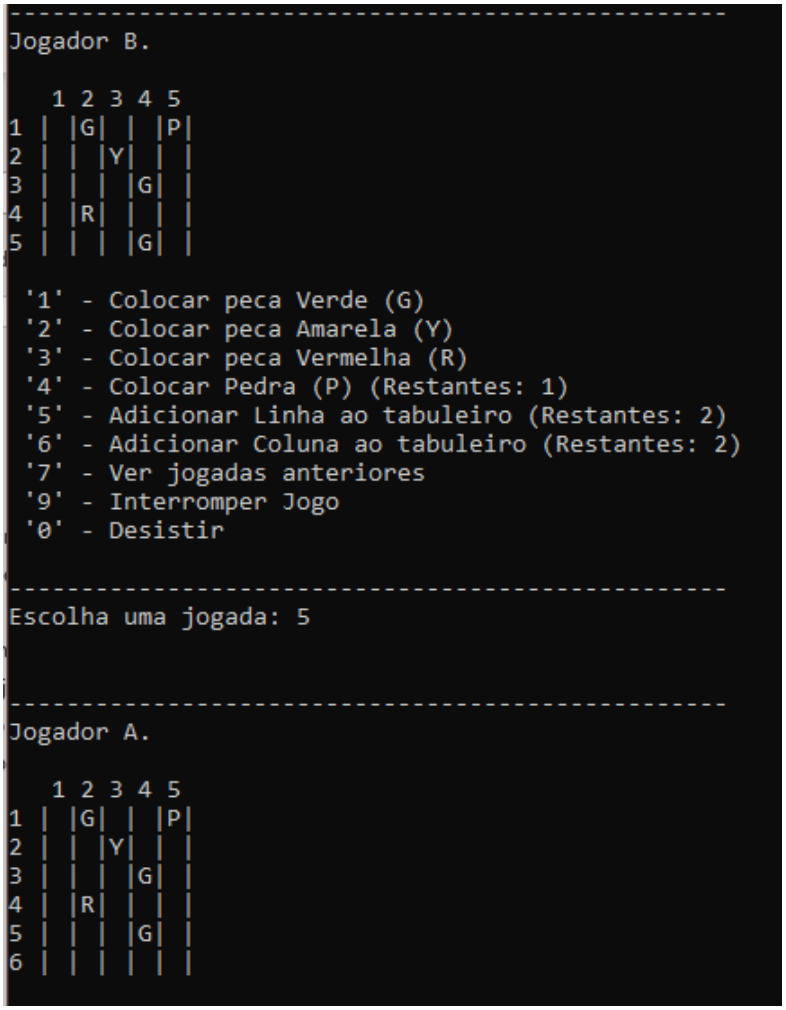

Figura 16 - Manual de utilização, adicionar linha ao tabuleiro

Para a opção Adicionar Linha ao Tabuleiro não é necessário especificar coordenadas, pois, esta será sempre adicionar ao final do tabuleiro. Esta opção partilha de uma condição com a opção Adicionar Coluna ao Tabuleiro, sendo que só é permitido que sejam realizadas duas vezes estas operações, duas vezes uma delas ou uma vez cada uma. Esta condição é aplicada a cada jogador.

Cofinanciado por:

5.5.6. Adicionar Coluna ao Tabuleiro

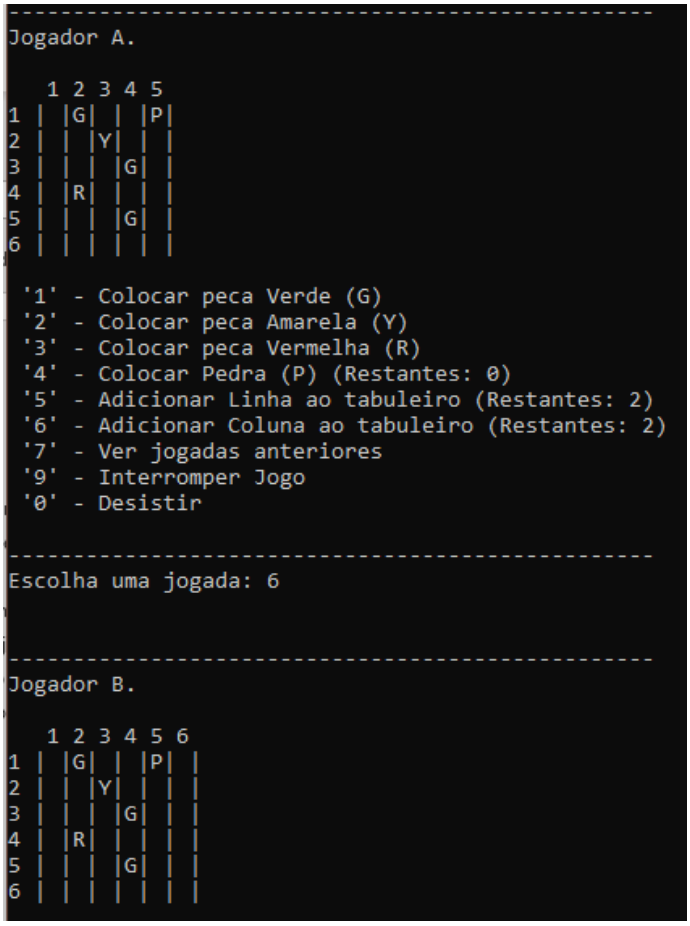

Figura 17 - Manual de utilização, adicionar coluna ao tabuleiro

Tal como para a operação Adicionar Linha ao Tabuleiro a coluna inserida por esta operação será ao final do tabuleiro, e tal como referido em cima, só é possível realizar duas operações deste tipo, para as linhas e para as colunas.

Cofinanciado por:

5.5.7. Ver jogadas anteriores

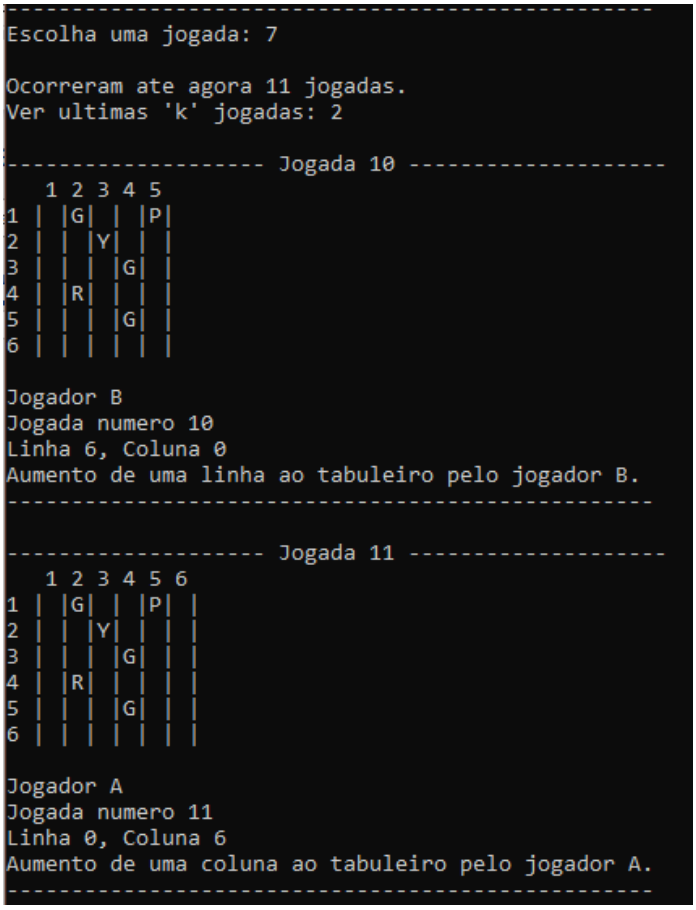

Figura 18 - Manual de utilização, ver jogadas anteriores

A operação ver jogadas anteriores é uma das referidas ações que não contam como jogadas, ou seja, o jogador que consultar o registo de jogadas irá voltar a jogar de seguida, pois não se justifica que consultar o registo de jogadas conte como uma jogada.

Após selecionar a opção ver jogadas anteriores são mostradas ao utilizador o número de jogadas que ocorreram, e de seguida são pedidas as ‘k’ últimas jogadas que o utilizador pretende visualizar. É realizada uma verificação ao valor inserido e são mostradas as últimas ‘k’ jogadas ao utilizador. O registo de jogadas guarda o tabuleiro da jogada, bem como o jogador que efetuou a jogada, qual o número da jogada, a linha e coluna afetada e uma breve descrição da jogada que o utilizador realizou.

Cofinanciado por:
CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

5.5.8. Interromper Jogo

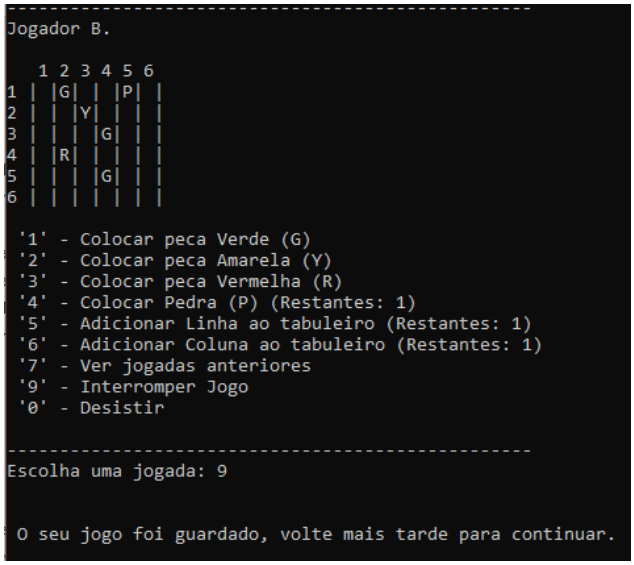

Figura 19 - Manual de utilização, interromper jogo

Figura 20 - Manual de utilização, retomar jogo

A opção Interromper Jogo é a segunda opção que não conta como jogada, ou seja, o jogador que realize esta operação continuará a jogar na próxima jogada.

A operação interromper jogo guarda em um ficheiro “jogo.bin” os diversos dados necessários para retomar mais tarde o estado atual do jogo, de forma que os jogadores possam pausar o jogo e posteriormente retomar no mesmo estado onde deixaram, sendo que funciona para jogos de um ou dois jogadores. Quando esta operação é realizada aparecerá uma nova opção no Menu Principal, sendo ela a Continuar Jogo. Quando esta opção for selecionada será retomado o jogo, sendo que após isto será eliminado o ficheiro contendo todo o conteúdo do estado do jogo.

Cofinanciado por:

5.5.9. Desistir

Figura 21 - Manual de utilização, desistir

A opção desistir, como o nome indica, permite que um jogador desista do jogo, dando a vitória ao outro jogador.

Tal como aconteceria caso um jogador ganhe pelo preenchimento do tabuleiro é mostrada uma mensagem com a identificação do jogador vencedor e é pedido um nome de ficheiro para guardar os dados do registo das várias jogadas.

Cofinanciado por:
CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

6 Conclusão

A realização deste relatório esteve em concordância com o decorrer da unidade curricular de Programação e é possível concluir que é possível encontrar várias soluções para um único problema, sendo que cada uma tem as suas vantagens e desvantagens.

No decorrer do trabalho foram utilizados vários conteúdos lecionados ao longo da unidade curricular, entre os quais: *arrays* dinâmicos, listas ligadas, estruturas de dados, leitura e escrita em ficheiros.

Foram cumpridos todos os requisitos propostos pelo professor, sendo que foi sempre aplicada a melhor estratégia encontrada para a resolução de cada problema.

Este trabalho foi bastante importante para o aprofundamento do meu conhecimento sobre a programação na linguagem C, sendo que me permitiu desenvolver as minhas capacidades de investigação, organização e resolução de problemas, uma vez que se demonstrou necessário organizar o trabalho total em fases, para uma melhor compreensão do projeto, e para cada uma destas fases foi necessário procurar uma ou mais soluções para cada um dos problemas encontrados.

Cofinanciado por: