
Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

2020/2021

Relatório do Trabalho Prático



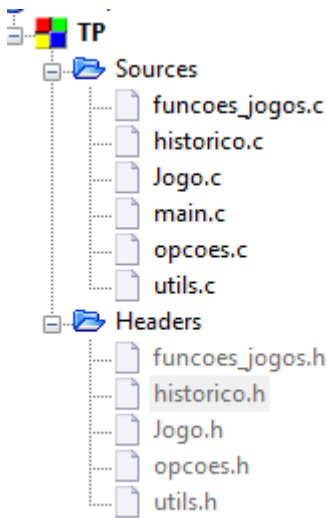
PROGRAMAÇÃO

Carolina Ferreira – 2018018459

Índice

1. Implementação.....	3
1.1 FUNÇÕES	4
○ funções do <i>opcoes.c</i>	4
○ funções do <i>funcoes_jogos.c</i>	6
○ funções do <i>Jogo.c</i>	9
○ funções do <i>historico.c</i>	10
2.2 ESTRUTURAS DE DADOS.....	13
2. Manual de Utilização	15

1. Implementação



O projeto divide-se em diversos ficheiros .c e .h de modo a organizar e agrupar, em cada ficheiro, um conjunto de funções de alguma forma relacionadas. O *main.c*, sendo o mais simples, contém apenas a inicialização dos campos da estrutura principal, a alocação de memória para a inicialização do tabuleiro de jogo e a chamada do menu principal para interação com o utilizador e, assim, começar um jogo. Esta função para menu, por sua vez, consta no ficheiro *opcoes.c*. Para além da função **opcoes()** para Menu Principal, apresenta-se também a **game_options()** como sub-menu, onde é possível escolher qual o tipo

de jogo que pretende iniciar. O header file correspondente, contém apenas a declaração destas funções. No ficheiro *Jogo.c*, podemos encontrar duas funções, chamadas pelo sub-menu anterior: **jogo_semaforo()** e **jogo_semaforo_vs_PC()**. Estas funções, como o nome assim o sugere, contém toda a lógica do jogo propriamente dito sendo que a primeira é chamada para o tipo de jogo jogador vs. jogador enquanto a segunda executa o jogo do tipo jogador vs. máquina. O header file correspondente, contém apenas a declaração destas funções.

No ficheiro *funcoes_jogo.c* estão todas as funções “auxiliares” que asseguram o funcionamento correto do jogo de acordo com a sua lógica e regras. Nestas funções enquadra-se a função de inicialização do tabuleiro assim como a função que o mostra, dando ao utilizador uma perceção visual do tabuleiro de jogo atualizado. Para além destas, é também neste ficheiro que podemos encontrar todas as funções de verificação. Nesta “categoria” de funções começamos pela **verifica_tentativa()** para validar as tentativas de jogada das peças no tabuleiro e acabamos com as funções que verificam a ocorrência de vitória ou empate. (Nota: Pelas regras do jogo expostas no enunciado, é sempre possível encontrar um vencedor para o jogo. O empate não acontece.). Como é possível vencer o jogo de três maneiras diferentes – linha completa, coluna completa ou diagonal completa – a verificação da vitória, por ser mais simples, conta com três funções: a **verifica_linha()**, **verifica_coluna()** e a **verifica_diagonal()**. Por fim, encontramos também neste ficheiro as funções **selecionar_jogada()** e **gera_jogada_automatica()**. À semelhança das funções do *Jogo.c*, estas duas últimas funções interagem com o utilizador apresentando-lhe um conjunto de opções que lhe permitem decidir o que pretende fazer no jogo e recorrem às funções de verificação para validar essas jogadas. Estas duas funções são semelhantes sendo que a **selecionar_jogada()** é chamada sempre que um jogador deve executar uma ação e a

gera_jogada_automatica() é para o caso particular da jogada automática da máquina na versão do jogo jogador vs. PC. O header file correspondente, além da declaração das funções, contém a estrutura tabuleiro.

No ficheiro *historico.c* encontram-se todas as funções que estão, de algum modo, relacionadas com o armazenamento de informação sobre o jogo. A função **add_historico()**, de nome sugestivo, é a responsável por guardar a informação de uma jogada/estado do tabuleiro sempre que uma jogada é, efetivamente, concretizada. Por sua vez, a **show_historico()** é apenas uma função de debug que permite ver o histórico completo de jogadas de uma partida. Quanto à função **ultimas_k_jogadas()**, ainda neste ficheiro, permite apresentar na consola a sucessão de jogadas nos últimos k turnos. Por fim, a função **guarda_txt()** implementa a lógica que permite guardar o histórico de jogadas de um jogo terminado num ficheiro txt. Para além da declaração das funções, no seu header file, consta a estrutura dinâmica do tipo lista ligada para o histórico.

A todos estes ficheiros, juntam-se o **utils.c** e **utils.h** fornecidos pelo professor Francisco Pereira contendo algumas funções auxiliares úteis. Neste trabalho, fiz uso de duas dessas funções: a **initRandom()** e a **intUniformRnd()** para gerar um tabuleiro de jogo quadrado de dimensão 3x3, 4x4 o 5x5 e ainda para gerar decisões aleatórias de peças e posições por parte da máquina nos jogos do tipo jogador vs. PC.

1.1 FUNÇÕES

No ponto anterior, todas as funções foram apresentadas e explicadas de um modo geral no sentido de dar a entender qual é o seu papel no programa. Aqui, entraremos mais em detalhe para compreender as lógicas implementadas na perspetiva de código.

- funções do *opcoes.c*

- **void opcoes(int* opcao);**

```
void opcoes(int* opcao){  
    printf("Prima uma das seguintes teclas:");  
    printf("\n\n*****");  
    printf("\n 1 Novo Jogo");  
    if( fopen("jogo.bin", "rb") != NULL )  
        printf("\n 2 Retomar Jogo");  
    printf("\n 3 Exit");  
    printf("\n*****");  
    printf("\n\n----> ");  
    scanf("%d", opcao);  
}
```

Sendo uma função de interação com o utilizador, esta função dá opções numéricas ao jogador e guarda a sua resposta. Ao escolher a opção 1, o jogador é redirecionado para um novo sub-menu de opções disponíveis no contexto de iniciar um Novo Jogo.

- **int game_options(int opcao, tabuleiro *tab);**

Esta função é também interativa. Sendo chamada após a **opcao()** no main, o valor da “opcao” é passado por argumento para a **game_options()** que, ficando a conhecer esse valor, agirá de acordo com ele por lógica de **case...switch**.

```
switch (opcao)
{
    case 1:
        do{
            printf("\nOPCOES DE JOGO");
            printf("\n\n*****");
            printf("\n1 Multiplayer");
            printf("\n2 Player vs. PC");
            printf("\n3 Voltar ao menu principal");
            printf("\n*****");
            printf("\n\n----> ");
            scanf("%d",&jogo);

            if(jogo==1){
                jogo_semaforo(&tab);
                break;
            }
            else if(jogo==2){
                jogo_semaforo_vs_PC(&tab);
                break;
            }
            else if(jogo==3){
                return 1;
            }
            else{
                printf("\nA opcao nao existe. Try again\n\n");
            }

        }while (jogo!=1 || jogo!=2);
        break;

    case 2:

        // Confirmar se jogo existe
        //jogoanterior=confirmajogo();
        if(retomar_jogo==1){ //Existe jogo anterior

        }

        break;
}
```

Para o caso de opcao==1, é apresentado um novo menu de opções ao jogador para este escolher o tipo de jogo que pretende jogar. Aqui, seguindo a lógica de if, else if...else, o programa poderá executar as diferentes versões do jogo do semáforo.

- funções do *funcoes_jogos.c*

- **void inicializacao_tabuleiro(tabuleiro **tab);**

```
void inicializacao_tabuleiro(tabuleiro **tab){  
  
    for(int i=0;i<((*tab)->linha);i++){  
        for(int j=0;j<((*tab)->coluna);j++){  
            (*tab)->tabuleiro[i][j]='_';  
            printf(" ");  
            printf("%c", (*tab)->tabuleiro[i][j])  
        }  
        printf("\n");  
    }  
}
```

Esta função inicializa o tabuleiro de jogo com a passagem da estrutura por referência. Como o parâmetro em si a passar já é um ponteiro, ao fazermos a passagem por referência ficamos, inevitavelmente, na situação de ponteiro para ponteiro. O primeiro **for** percorre as linhas e o segundo percorre as colunas. Desta maneira, é possível preencher os espaços alocados em memória anteriormente com o carater “_”, permitindo assim “criar” um tabuleiro “visual” com posições definidas de forma clara. Os **for** acedem corretamente ao número de linhas e colunas recorrendo aos campos da estrutura tabuleiro com esses valores guardados após a chamada das funções **initRandom()** e **intUniformRnd()** no main:

```
initRandom();  
tab.coluna=intUniformRnd(3, 5);  
tab.linha=tab.coluna;
```

- **void show_tabuleiro(tabuleiro **tab);**

Esta função, é em tudo semelhante à anterior à exceção da linha código que escreve os “_” no tabuleiro. Funciona para simplesmente imprimir o tabuleiro ao jogador, seguindo a lógica dos **for** da função anterior.

- **int verifica_tentativa(tabuleiro **tab, int peca);**

Esta função, chamada sempre que um jogador tenta seleccionar uma peça para jogar no tabuleiro, para além da estrutura tabuleiro, recebe também um inteiro peca passado por argumento da função **selecionar_jogada()**, onde é chamada.

Esta variável “peca” pode tomar valores de 1 a 4, sendo que 1 reflete a intenção do jogador usar uma peça verde, o 2 uma peça amarela, o 3 uma peça vermelha e o 4 uma pedra.

```
int verifica_tentativa(tabuleiro **tab, int peca){
    int empty_spaces=0, conta_g=0, conta_y=0;

    for(int i=0;i<((*tab)->linha);i++){
        for(int j=0;j<((*tab)->coluna);j++){

            if((*tab)->tabuleiro[i][j]=='_')
                empty_spaces++;
            if((*tab)->tabuleiro[i][j]=='G')
                conta_g++;
            if((*tab)->tabuleiro[i][j]=='Y')
                conta_y++;
        }
    }
}
```

Com os for a permitir percorrer todo o tabuleiro, o programa verifica o número de espaços não ocupados e o número de espaços preenchidos com peças verdes e amarelas.

```
if(peca==1 && empty_spaces>0)
    return 1;
if(peca==4 && empty_spaces>0)
    return 1;
if(peca==2 && conta_g>0)
    return 1;
if(peca==3 && conta_y>0)
    return 1;
if(peca==0)
    return 1;

return 0;
```

Seguindo as regras do jogo, se o jogador selecionar uma peça verde ou a pedra, a função retorna o valor 1 (de validação) se existirem espaços vazios para a colocar e 0 (de invalidação) se não se verificar. O mesmo acontece para a seleção das peças amarelas ou vermelhas não em relação a espaços vazios, mas sim a posições com peça verde prévia para trocar por amarela ou a posições com peça amarela prévia para o caso de o jogador querer colocar uma peça vermelha no tabuleiro.

- **void selecionar_jogada(tabuleiro **tab, int jogador, int *win, phist *historico);**

A função é chamada quando o programa está pronto para receber uma jogada por parte do jogador. O programa pergunta ao jogador qual o tipo de peça que pretende colocar no tabuleiro. Sendo que cada jogador só pode jogar uma pedra, a opção de colocar uma pedra em jogo só lhe é apresentada no menu de escolha após o programa verificar que o jogador ainda não colocou a sua pedra no tabuleiro. Para fazer isto, de forma muito simples, o programa recorre aos campos das pedras na estrutura tabuleiro e verifica se estão a 0 ou a

```
1. if( (num_jogada%2!=0 && ((*tab)->pedra1) < 1 ) || (num_jogada%2 == 0 && ((*tab)->pedra2)) < 1))
    printf("\n4 Pedra");
```

Uma outra opção que o jogador tem é a de ver o histórico de jogadas. Se seleccionar esta opção, o programa pede ao jogador que lhe indique quantas das últimas jogadas deseja ver e, de acordo com esse valor, chama a função **ultimas_k_jogadas()** para lhe mostrar.

Ultrapassando esta fase e estando a seleção do jogador validada pela chamada da **verifica_tentativa()**, o programa pede ao jogador que lhe indique a posição (linha e coluna) em que pretende colocar a peça e seguindo a lógica de if, else if...else escreve-a no tabuleiro, como se pode ver na figura seguinte.

```
//ifs para escrever pedras no tabuleiro
if(peca==1){

    if((*tab)->tabuleiro[line][column]=='_'){
        (*tab)->tabuleiro[line][column]='G';
        pecinha = 'G';

        validade=1;
    }else
        printf("Acao Impossivel");
}

else if(peca==2){

    if((*tab)->tabuleiro[line][column]=='G'){
        (*tab)->tabuleiro[line][column]='Y';
        validade=1;
        pecinha = 'Y';
    }else
        printf("Acao Impossivel");
}

else if(peca==3){

    if((*tab)->tabuleiro[line][column]=='Y'){
        (*tab)->tabuleiro[line][column]='R';
        pecinha = 'R';
        validade=1;
    }else
        printf("Acao Impossivel");
}

else if(peca==4){

    if((*tab)->tabuleiro[line][column]=='_'){
        (*tab)->tabuleiro[line][column]='P';
        pecinha = 'P';
        validade=1;
    }
    else
        printf("Acao Impossivel");
}
```


Após confirmar a jogada, o programa adiciona a mesma ao histórico recorrendo à função `add_historico()` cujo funcionamento é explicado mais abaixo. Para o jogador poder visualizar o tabuleiro já atualizado com a sua jogada, é chamada a função **`show_tabuleiro()`**.

No final, verifica-se se após a jogada a linha, coluna ou diagonal que inclui a posição da nova peça no tabuleiro ficaram preenchidas recorrendo às funções **`verifica_linha()`**, **`verifica_coluna()`** e **`verifica_diagonal()`**. Se sim, coloca-se o `*win` a 1, indicando ao programa que se verifica uma vitória.

```
if(num_jogada%2!=0) {
    *historico = add_historico(*historico, pecinha, line+1, column+1, 'A');
}
else {

    *historico = add_historico(*historico, pecinha, line+1, column+1, 'B');
}

//show_historico(historico);

printf("\n");
show_tabuleiro(&(*tab));
if (verifica_linha(&(*tab),line)==1 || verifica_coluna(&(*tab),column)==1 || verifica_diagonal(&(*tab))==1)
    *win=1;
```

- **`void gera_jogada_automatica(gera_jogada_automatica(tabuleiro **tab, int num_jogada, int *win, phist *historico);`**

Esta função é semelhante à **`selecionar_jogada()`** e é chamada no jogo do tipo jogador vs. PC, quando é o turno da máquina. Ao contrário da primeira que vai pedindo ao jogador que lhe diga qual a peça que quer jogar assim como a posição onde a quer colocar, nesta função a escolha é feita automaticamente recorrendo às funções **`initRandom()`** e **`intUniformRnd()`** do `utils.c`. A partir daqui segue a lógica da **`selecionar_jogada()`** até ao final.

○ funções do *Jogo.c*

- **`void jogo_semaforo(tabuleiro **tab);`**

Esta função é chamada para dar início a um jogo entre dois jogadores (A e B) e recebe por referência um ponteiro para o ponteiro que aponta para a estrutura.

Depois de inicializar um historico do tip phist (explicado nas estruturas mais abaixo) a NULL e pedir os nomes dos jogadores, o programa inicializa o tabuleiro de jogo pela **inicializa_tabuleiro()**. Depois disso, conta com um for para assegurar que o jogo termina quando se verifica vitória, ou seja, win=1. Este for incrementa-nos o número de jogadas (que se inicializa a 1) para permitir saber quem é o jogador a jogar no turno atual. Como o Jogador A é o primeiro a colocar peça, todas as jogadas de turno com número ímpar são suas. Ao conhecer qual é o jogador ao qual deve pedir para jogar, o programa chama a função **selecionar_jogada()** (ver acima) que fica encarregue da restante lógica do jogo até encontrar uma situação de vitória win =1. Quando isso acontece, o programa retorna à função atual e anuncia o vencedor (mais uma vez, pelo número da jogda) e pede ao utilizador que insira um nome para o ficheiro.txt onde irá guardar o histórico do jogo recorrendo à função **guarda_txt()**.

- **void jogo_semaforo_vs_PC(tabuleiro **tab);**

Esta função é chamada para dar início a um jogo entre um jogador e uma máquina (A e M). A função é em tudo semelhante à anterior com o pormenor de chamar a função **gera_jogada_automática()** (ver acima) nos turnos em que a máquina deve jogar. Fora desse turno, chama exatamente a mesma função **selecionar_jogada()** **para o Jogador A.**

- *funções do **historico.c***

- **phist add_historico(phist historico, char peca, int linha, int coluna, char player);**

Esta função, como o “phist” indica, devolve algo do tipo de “phist”. Este “phist” nada mais é do que um ponteiro para a lista ligada e, portanto, guarda o mesmo tipo de dados declarados nos seus campos. Ao receber acesso ao phist, cria os novos campos de peca, linha, coluna e player que ficam prontos a preencher com novos dados. Esses novos dados são referentes à nova jogada executada. A ver na figura abaixo:

```

phist add_historico(phist historico, char peca, int linha, int coluna, char player) {

    phist new = NULL, aux = NULL;
    new = malloc(sizeof(historico_S)); // lista ligada portanto é so 1
    new->peca = peca;
    new->linha = linha;
    new->coluna = coluna;
    new->player = player;
    new->prox = NULL;

    if(historico == NULL) {
        historico = new;
        return historico;
    }

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

    aux->prox = new;

    return historico;
}

```

- **void ultimas_k_jogadas(phist historico, int jogada);**

A função inicia com um ciclo **for** para percorrer a lista ligada até à posição a partir da qual deve mostrar a sucessão de jogadas na consola. Posteriormente, enquanto não chegar à posição final (ou seja, onde o ponteiro para a posição seguinte é NULL) faz o print da informação guardada mostrando qual a peça que foi jogada, onde e por qual dos jogadores.

```

void ultimas_k_jogadas(phist historico, int jogada) { // jogada = numero de jogada atual - k
    for(int i = 0; i < jogada; i++) {
        historico = historico->prox;
    }

    while( historico != NULL ) {
        printf("%c - %d - %d - %c\n",
            historico->peca, historico->linha, historico->coluna,
            historico->player);

        historico = historico->prox;
    }
}

```

Exemplo do print na consola:

```

Deseja visualizar o historico de quantas ultimas jogadas?
-->1
G - 2 - 2 - B

```

- **void guarda_txt(phist phistorico);**

Nesta função pede-se ao utilizador que introduza um nome para o ficheiro txt no qual pretende guardar o histórico do jogo. O programa abre esse ficheiro e, lá, escreve o histórico de jogadas à semelhança do que acontecia na função anterior diretamente na consola. Nesta função, a indicação é para que o programa comece a escrever no início da lista ligada e que continue a fazê-lo até chegar à última posição (que aponta para NULL já que não existe nó seguinte).

```
void guarda_txt(phist phistorico){
    FILE *f;
    char filename[15];

    printf("\nNome para o ficheiro a criar?");
    printf("\n-->");
    scanf("%14s", &filename);

    f = fopen(filename, "w");
    if(f == NULL){
        printf("\nOcorreu um problema.\n");
        return;
    }

    else{
        while (phistorico != NULL){
            fprintf(f, "%c-%d-%d-%c\n", phistorico->peca, phistorico->linha, phistorico->coluna, phistorico->player);
            phistorico = phistorico->prox;
        }
    }
}
```

Exemplo de ficheiro.txt criado:

```
Em que linha deseja jogar?
-->1

Em que coluna deseja jogar?
-->5

G G G G G
- - - - -
- - - - -
- - - - -
- - - - -

a venceu o jogo! :)

Nome para o ficheiro a criar?
-->ln
Prima uma das seguintes teclas:

*****
1 Novo Jogo
3 Exit
*****

-----> 3

Process returned 0 (0x0)   execu
Press any key to continue.
```

In - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

```
G-1-1-A
G-1-2-B
G-1-3-A
G-1-4-B
G-1-5-A
```

2.2 ESTRUTURAS DE DADOS

Depois de alguma análise e ponderação, concluí que seria suficiente criar duas estruturas para garantir o bom funcionamento do programa assim como a sua organização, simplicidade e a facilidade em manipular código.

A estrutura tabuleiro tem como campos um double ponteiro do tipo char uma vez que o tabuleiro é precisamente de caracteres e é alocado dinamicamente, um linha do tipo int para guardar o número de linhas, um coluna do tipo int para guardar o número de colunas, um int pedra1 e um intpedra2 para guardar o valor 0 ou 1 e assim saber, em todos os momentos, se cada um dos jogadores já fez uso da sua possibilidade de colocar uma pedra no tabuleiro e, por fim, um jogadas_efetuadas do tipo int para marcar o número da jogada atual. Estas jogadas_efetuadas é muito útil não só para contabilizar o número de jogadas mas também para permitir saber qual o jogador de cada turno não esquecendo que o jogador A joga sempre em turnos de número impar e o jogador B (ou automático) joga sempre em turnos de número par.

Nota: Funcionalidades como adicionar linhas ou colunas no tabuleiro não foram implementadas. Porém, caso tivessem sido, esta estrutura, contendo o int linha e o int coluna, permitia manter a dimensão do tabuleiro atualizada com os novos valores.

```
typedef struct tabuleiro tabuleiro;

struct tabuleiro{
    char **tabuleiro;
    int linha;
    int coluna;
    int pedral;
    int pedra2;
    int jogadas_efetuadas;
};
```

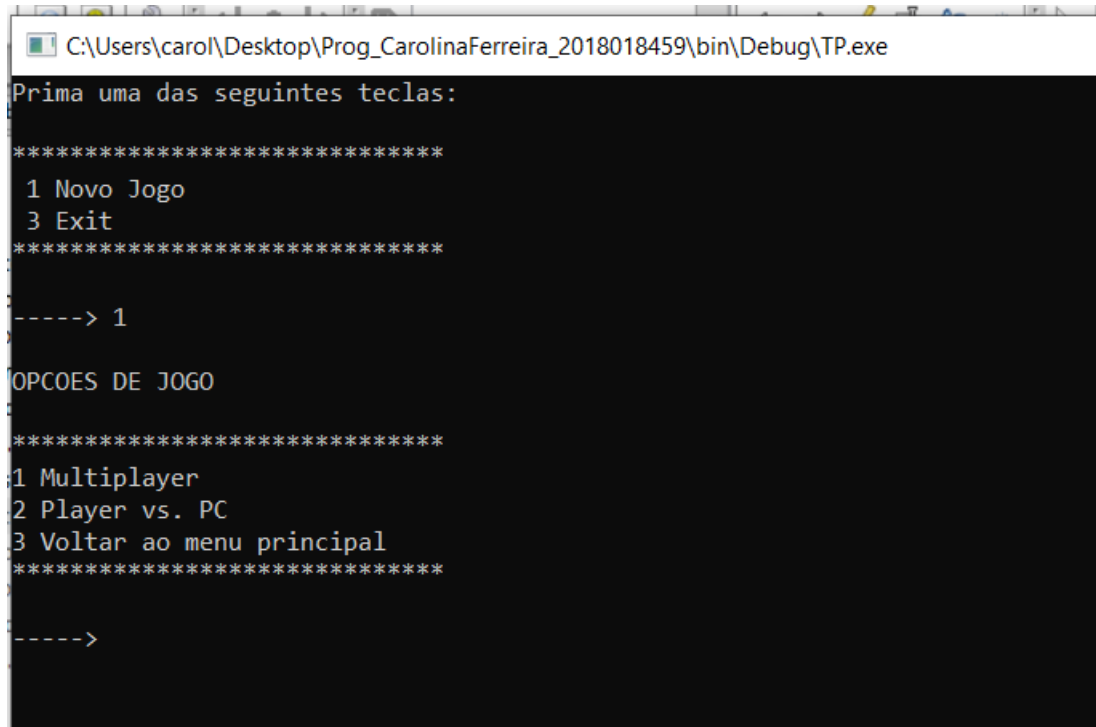
A estrutura historico, sendo uma estrutura dinâmica do tipo lista ligada, é usada para armazenar os dados relevantes de jogadas anteriores ou de um jogo terminado, tem como campos um int linha e um int coluna que, não semelhante à estrutura anterior, guardam as posições onde uma peça foi jogada, um char peca para guardar o tipo de peça que foi jogada – G, Y, R, ou P – e, por fim um ponteiro prox para o nó seguinte da lista ligada. É necessário que assim seja para possibilitar o armazenamento do

decorrer do jogo, ou seja, a sucessão de jogadas e, portanto, o estado do tabuleiro. Cada nó seguinte da lista ligada é criado quando ocorre a chamada da função **add_historico()** explicada acima. Na figura abaixo, apresenta-se a estrutura.

```
typedef struct historico historico_S, *phist;  
  
struct historico{  
    int linha;  
    int coluna;  
    char peca;  
    char player;  
    phist prox;  
};
```

2. Manual de Utilização

Resultado de execução:



```
C:\Users\carol\Desktop\Prog_CarolinaFerreira_2018018459\bin\Debug\TP.exe
Prima uma das seguintes teclas:

*****
1 Novo Jogo
3 Exit
*****

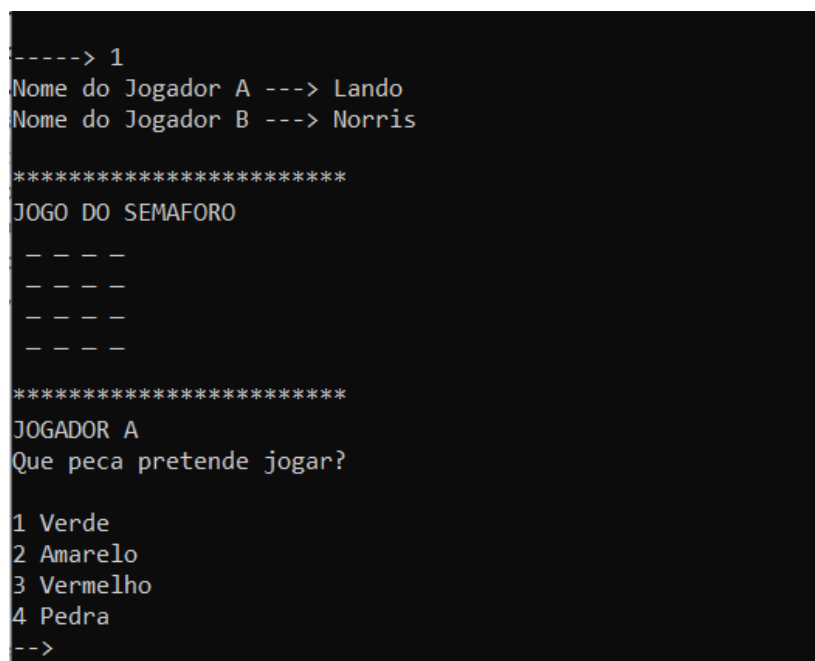
-----> 1

OPCOES DE JOGO

*****
1 Multiplayer
2 Player vs. PC
3 Voltar ao menu principal
*****

----->
```

- Aqui o jogador deverá escolher qual o tipo de jogo que deseja iniciar.



```
-----> 1
Nome do Jogador A ---> Lando
Nome do Jogador B ---> Norris

*****
JOGO DO SEMAFORO

- - - -
- - - -
- - - -
- - - -

*****
JOGADOR A
Que peca pretende jogar?

1 Verde
2 Amarelo
3 Vermelho
4 Pedra
-->
```

- O jogador A introduz o número correspondente ao tipo de peça que pretende colocar no tabuleiro.

```

Em que linha deseja jogar?
-->1

Em que coluna deseja jogar?
-->1

  G _ _ _
_ _ _ _
_ _ _ _
_ _ _ _

JOGADOR B
Que peca pretende jogar?

1 Verde
2 Amarelo
3 Vermelho
4 Pedra
0 Ver Historico de jogadas
-->

```

- O jogador B introduz o número correspondente ao tipo de peça que pretende colocar no tabuleiro.

```

Em que linha deseja jogar?
-->1

Em que coluna deseja jogar?
-->1

  G _ _
_ _ _
_ _ _

PC TURN
  G G _
_ _ _
_ _ _

JOGADOR A
Que peca pretende jogar?

1 Verde
2 Amarelo
3 Vermelho
0 Ver Historico de jogadas
--> 0

Deseja visualizar o historico de quantas ultimas jogadas?
-->1
G - 1 - 2 - M

Que peca pretende jogar?

```

- O jogador tem a opção de visualizar o histórico de jogadas. Para isso deve escolher a opção 0 e de seguida indicar quantas das jogadas mais recentes deseja ver.


```

Em que linha deseja jogar?
-->3

Em que coluna deseja jogar?
-->2

Y P P
_ _ _
G G G

Lando venceu o jogo! :)

Nome para o ficheiro a criar?
-->game
Prima uma das seguintes teclas:

*****
1 Novo Jogo
3 Exit
*****

----->
tems are up-to-date).
```

- O jogador tem a opção de visualizar o histórico de jogadas. Para isso deve escolher a opção 0 e de seguida indicar quantas das jogadas mais recentes deseja ver.