INSTITUTO FEDERAL DO MARANHÃO

JEFFERSON FREITAS DOS SANTOS

JAILSON SOARES CANTANHEDE

IULANO SILVA DOS SANTOS

SISTEMAS DE INFORMAÇÃO

RELÁTORIO DE LINGUAGEM DE PROGRAMAÇÃO

JOGO FOGE-FOGE | PROGRAMAÇÃO ESTRUTURADA

SÃO LUÍS - MA

2019

RELÁTORIO DE LINGUAGEM DE PROGRAMAÇÃO

JOGO FOGE-FOGE | PROGRAMAÇÃO ESTRUTURADA

Trabalho referente a parte da terceira nota da disciplina linguagem de programação.

Prof.ª: EVALDINOLIA GILBERTONI MOREIRA

São Luís-MA

2019

INTRODUÇÃO TEÓRICA

O vigente trabalho, refere-se ao desenvolvimento do jogo foge-foge. Utilizamos conhecimentos no paradigma de programação estruturada adquiridos em sala de aula com a Profa. Dra. Evaldinolia Gilbertoni. O programa foi desenvolvido no ambiente CodeBlocks, em Linguagem C, onde o usuário foge de fantasmas tentando alcançar uma pílula que lhe faz vencer o jogo, e isto é feito em dois niveis FACIL e DIFICIL, sendo estes orientados a diretivas.

OBJETIVOS

1. Desenvolver o jogo com programação estruturada, em linguagem C;
2. Utilizar os conhecimentos e boas práticas adquidiros em sala de aula;
3. Documentar toda a modularização do jogo;

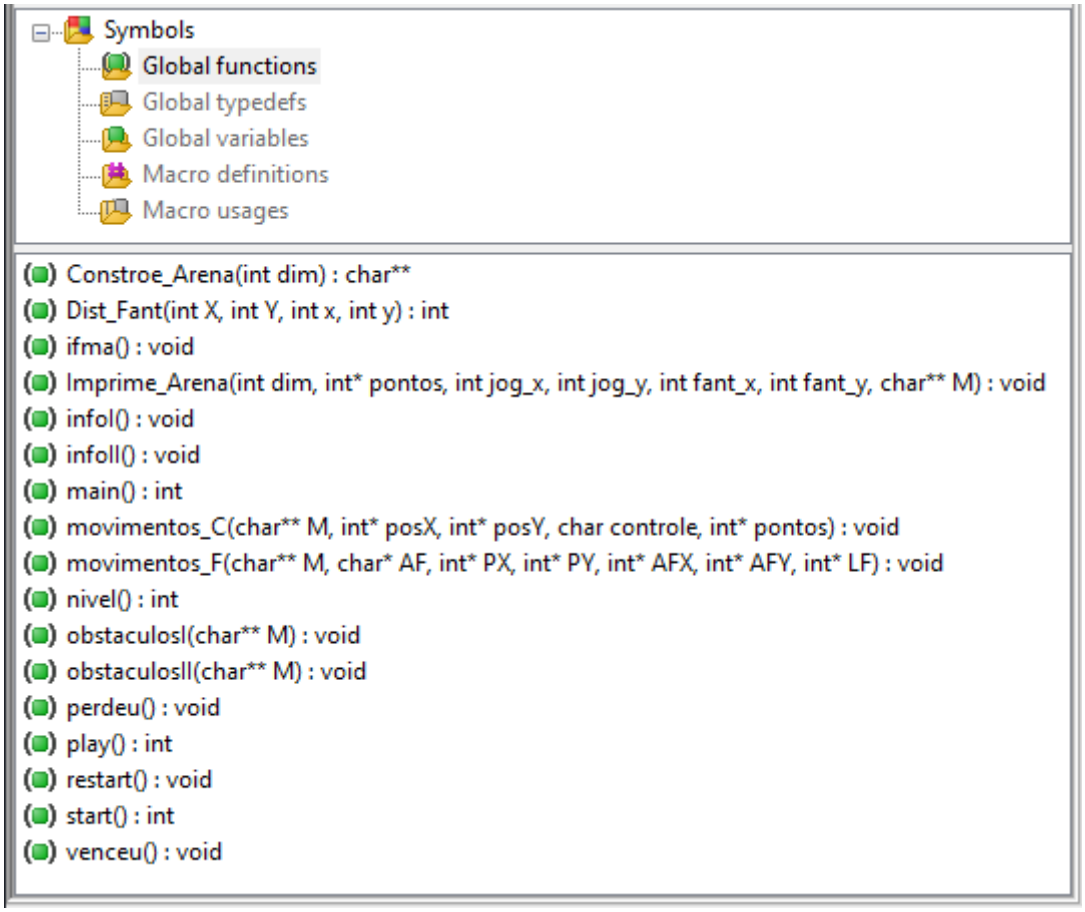
MATERIAL UTILIZADO

1. **Code:Blocks:** Um ambiente de desenvolvimento integrado de código aberto e multiplataforma. Ele foi desenvolvido em C++, usando wxWidgets. Sua arquitetura é orientada a plugin, de forma que suas funcionalidades são definidas pelos plugins fornecidos a ele.
2. **Estruturas Condicionais:** Instruções para testar se uma condição é verdadeira ou não. Elas podem ser associadas às estruturas que se repetem, após o cumprimento da condição (Loops).
3. **Funções:** Um conjunto de comandos que realiza uma tarefa específica em um módulo dependente do código. A mesma é relacionada pelo programa principal através do nome atribuído a ela. Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa específica.
4. **Ponteiros:** Variáveis que armazenam o endereço de memória de outras variáveis. Dizemos que um ponteiro “aponta” para uma varíável quando contém o endereço da mesma.
5. **Arquivo:** Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de texto.
6. **Alocação de memoria:** consiste no processo de solicitar/utilizar memória durante o processo de execução de um programa de computador.

RESULTADO E DISCUSSÕES

Utilizando os benchmarcks de programação estruturada, modularizamos 100% do nosso software com a aplicação de funções e diretivas de compilação, objetivando alto desempenho, otimização do código-fonte, alocação dinâmica de memória e referências com ponteiros, possibilitando o reaproveitamento de código:

Vide abaixo a estrutura funcional do nosso programa:

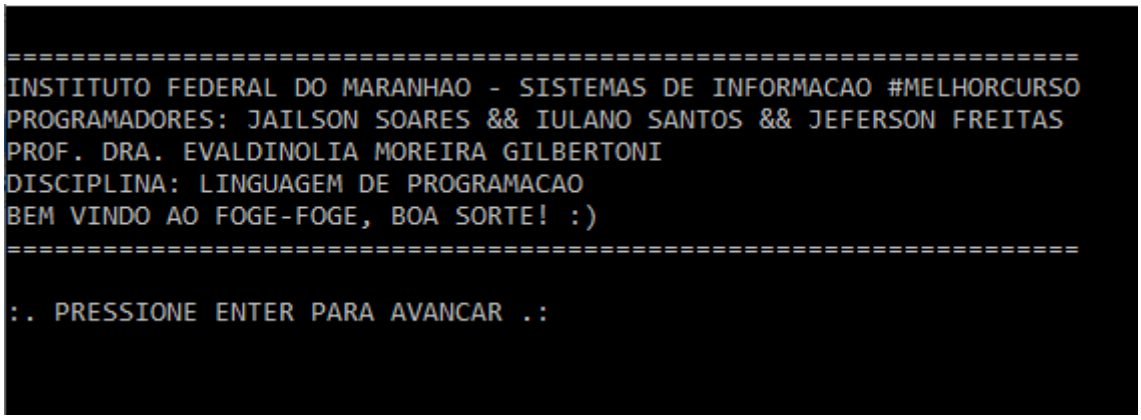


A main invoca apenas uma única função que encapsula todas as demais: int play( ), isentando o programa de variáveis locais e otimizando o consumo de memória:

```c
int main(){
  int play();
  play();
}

#define ArenaFacil 1
#define ArenaDificil 2
```

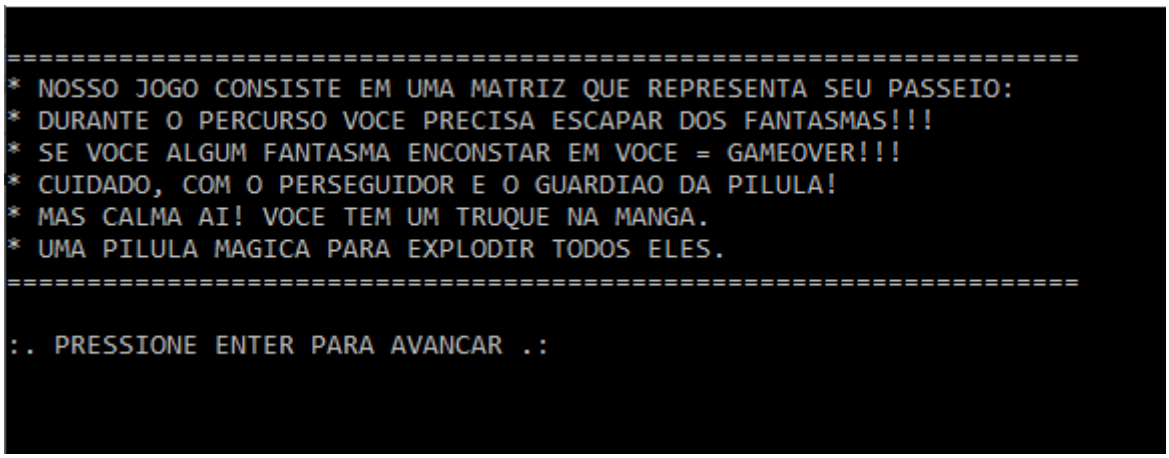
Como todo bom jogo, temos este splash que identifica os programadores, a organização na qual temos orgulho de estudar, e o nome do melhor curso deste lugar:



O splash é executado através da função void infoI( ). Vide abaixo o protótipo:

```c
void infoI(){
    printf("\n=====================================================================");
    printf("\nINSTITUTO FEDERAL DO MARANHAO - SISTEMAS DE INFORMACAO #MELHORCURSO");
    printf("\nPROGRAMADORES: JAILSON SOARES && IULANO SANTOS && JEFERSON FREITAS");
    printf("\nPROF. DRA. EVALDINOLIA MOREIRA GILBERTONI");
    printf("\nDISCIPLINA: LINGUAGEM DE PROGRAMACAO");
    printf("\nBEM VINDO AO FOGE-FOGE, BOA SORTE! :) ");
    printf("\n=====================================================================\n\n");
    printf(":. PRESSIONE ENTER PARA AVANCAR .:");
    getchar();
    system("cls");
}
```

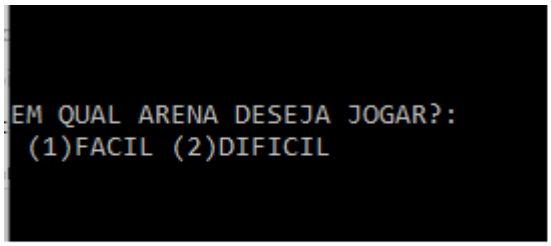
Logo em seguida, temos a chamada da função void infoII( ), com as regras do jogo:



Protótipo infoII( ):

```c
void infoII(){
    printf("\n=====================================================================");
    printf("\n* NOSSO JOGO CONSISTE EM UMA MATRIZ QUE REPRESENTA SEU PASSEIO:");
    printf("\n* DURANTE O PERCURSO VOCE PRECISA ESCAPAR DOS FANTASMAS!!!");
    printf("\n* SE VOCE ALGUM FANTASMA ENCONSTAR EM VOCE = GAMEOVER!!!");
    printf("\n* CUIDADO, COM O PERSEGUIDOR E O GUARDIAO DA PILULA!");
    printf("\n* MAS CALMA AI! VOCE TEM UM TRUQUE NA MANGA.");
    printf("\n* UMA PILULA MAGICA PARA EXPLODIR TODOS ELES.");
    printf("\n=====================================================================\n\n");
    printf(":. PRESSIONE ENTER PARA AVANCAR .:");
    getchar();
    system("cls");
}
```

Em seguida, usuário tem a opção de qual arena nosso herói vai jogar:

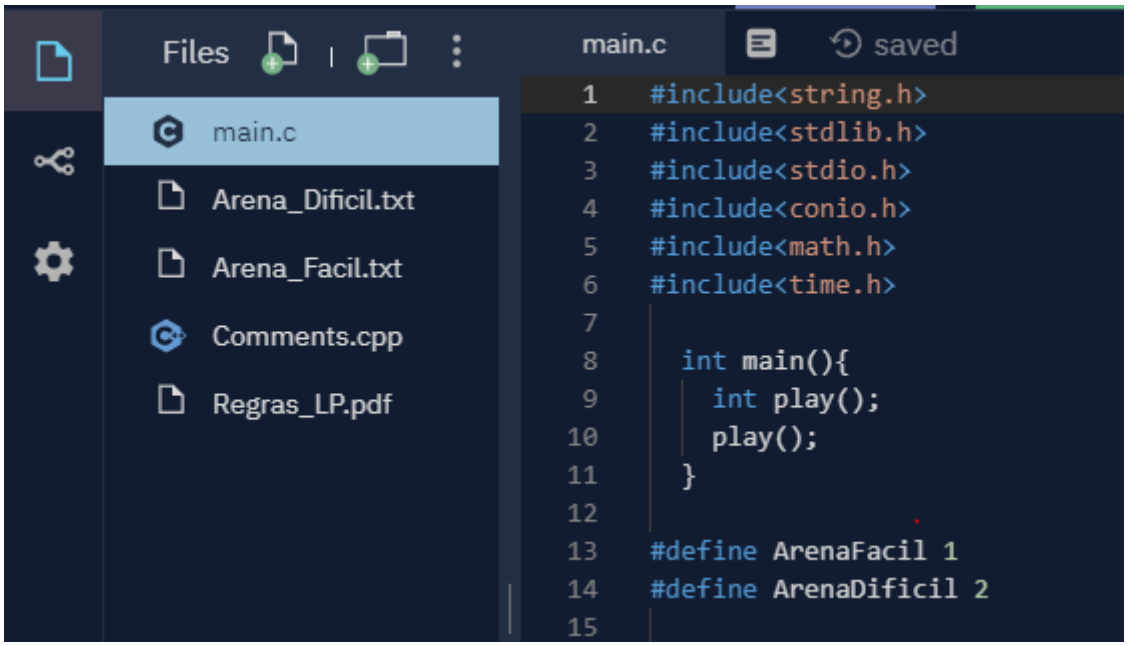


Esta escolha é encadeada em funções, que por sua vez, se baseiam em diretivas. Utilzamos também recursos como ponteiros e manipulação de arquivos para exibir o preview das arenas para o usuário, com base nas diretivas correspondentes:

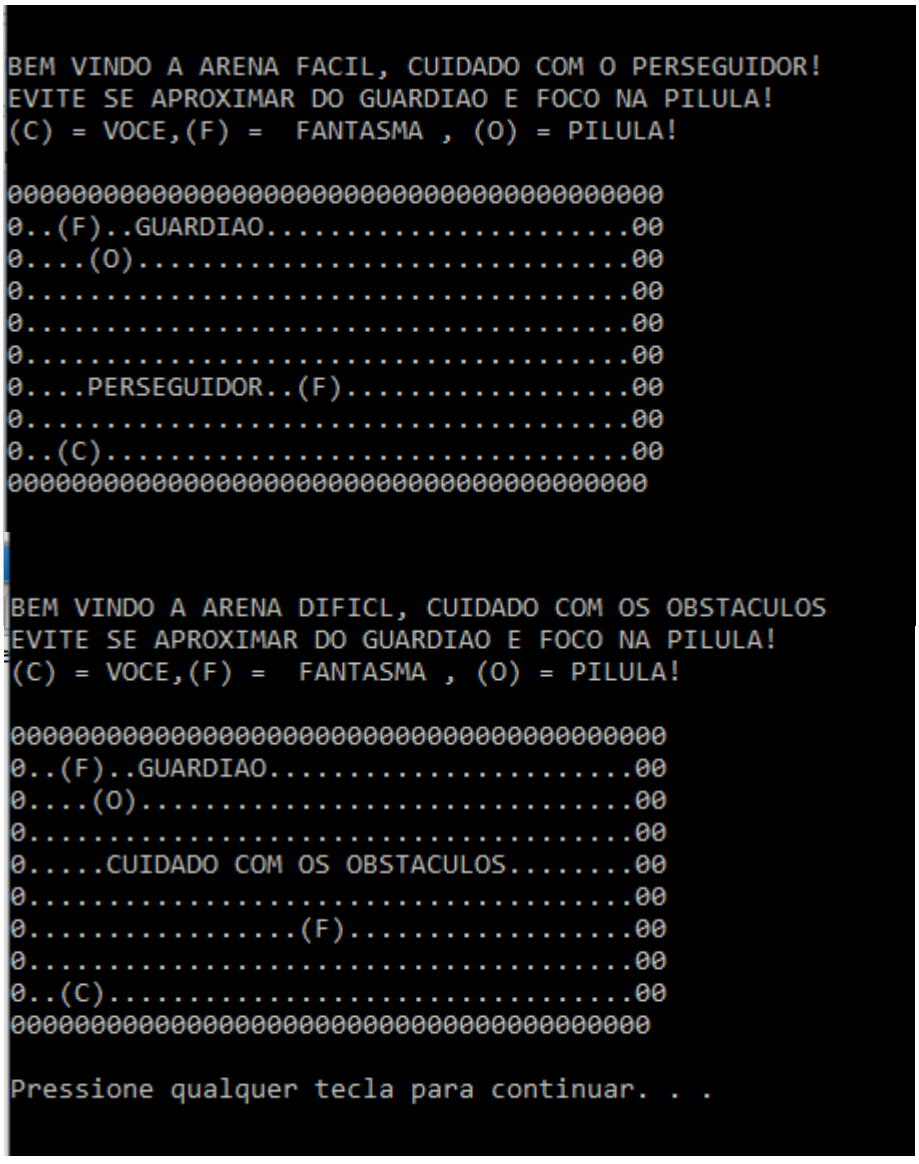
```c
#define ArenaFacil 1
#define ArenaDificil 2

int nivel(){
    int nivel = 0, dificuldade = 0;
    do{ printf("\n\n\n");
        printf("EM QUAL ARENA DESEJA JOGAR?:\n (1)FACIL (2)DIFICIL\n");
        fflush(stdin);
        scanf("%d",&nivel);
        system("cls");
    } while (nivel!= 1 && nivel!= 2);
    if(nivel==1){dificuldade = ArenaFacil;}
    if(nivel==2){dificuldade = ArenaDificil;}
  return (dificuldade);
}
```

Arquivos TXT contendo as arenas:



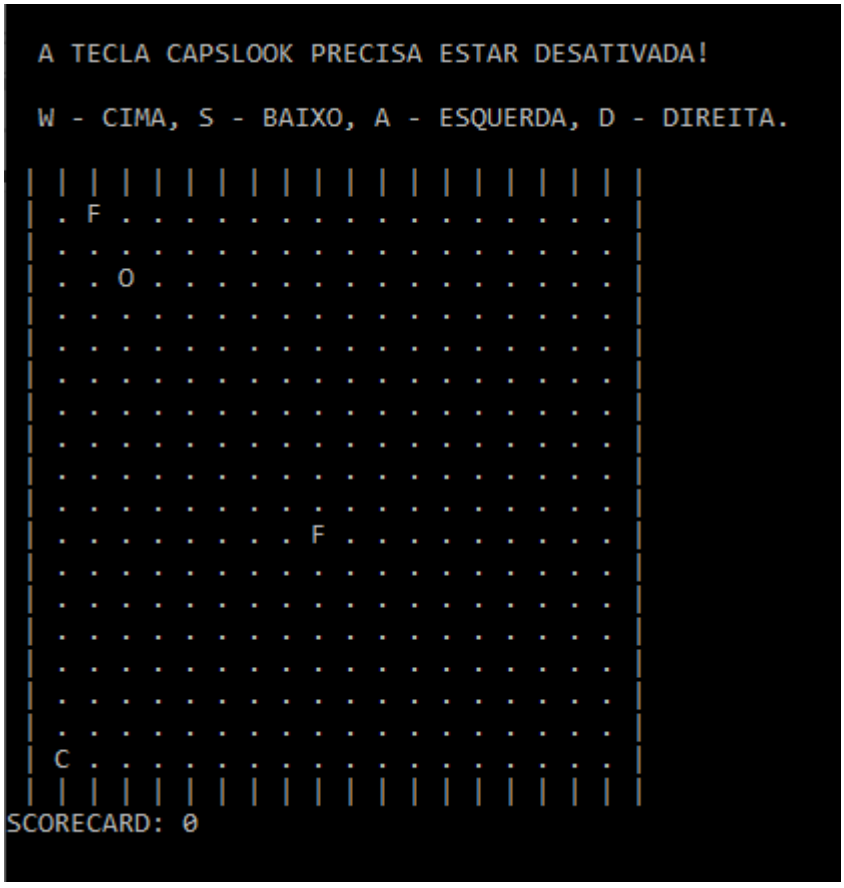
Preview das ARENAS FACIL e DIFICIL:

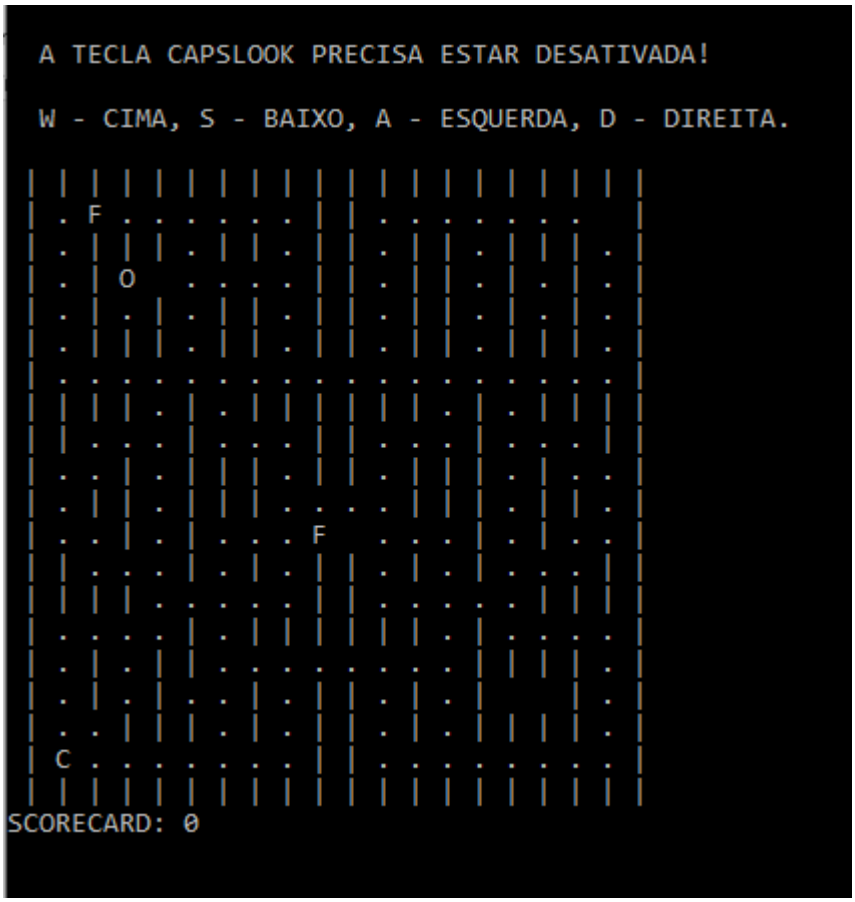


Como bem sabemos, todos os fabricantes de computadores do mercado atual produzem máquinas com Arquitetura X64 (64 bits), tornando desnecessária a alocação de memória em nosso jogo, contudo, resolvemos aplicar os conhecimentos adquiridos em sala de aula. Vide abaixo o protótipo da função:

```
char** Constroe_Arena(int dim){
    int i, j;
    char **M;
    M = (char**) malloc(dim *sizeof(char*));
    for(i = 0; i < dim; i++){
        M[i] = (char*) malloc(dim *sizeof(char));
        for(j = 0; j < dim; j++){
            if(i == 0 || i == (dim-1) || j == 0 || j == (dim-1) )
                M[i][j] = '|';
            else{
                M[i][j] = '.';
            }
        }
    }
    return M;
}
```

ARENA FACIL e DIFICIL que vem com a função obstaculos( ) em execução:





A exibição das arenas (FACIL OU DIFICIL) é feita através da chamada da função Imprime_arena( ):

```c
void Imprime_Arena(int dim, int *pontos, int jog_x, int jog_y, int fant_x, int fant_y, char **M){
  int i, j;

  M[jog_y][jog_x] = 'C';
  M[fant_y][fant_x] = 'F';
  system("cls");
  printf("\n  A TECLA CAPSLOOK PRECISA ESTAR DESATIVADA!\n");
  printf("\n  W - CIMA, S - BAIXO, A - ESQUERDA, D - DIREITA.\n");
  for(i = 0; i< dim; i++){
      printf("\n");
      for(j = 0 ; j<dim; j++)
          printf(" %c",M[i][j]);
  }
  printf("\nSCORECARD: %d",(*pontos));
}
```

Para jogar, o usuário precisa navegar na matriz livrando-se do fantasma perseguidor e do guardião que fica muito próximo da pílula. Para isto temos duas funções de movimento. Vide abaixo o protótipo da função de movimentos do herói, com estruturas condicionais feitas com base nos controles de navegação: A, W, S e D.

```c
void movimentos_C(char **M, int *posX, int *posY, char controle, int *pontos){
  if(controle == 'a' ){
    if(M[(*posY)][(*posX)-1] == '|'){
    }
    else{
      if(M[(*posY)][(*posX)-1] == '.')
        (*pontos)++;
      M[(*posY)][(*posX)] = ' ';
      (*posX) = (*posX) - 1;
      M[(*posY)][(*posX)] = 'C';
    }
  }
  if(controle == 'd' ){···
  }
  if(controle == 'w' ){···
  }
  if(controle == 's' ){···
  }
}
```

Protótipo da função de movimentos do fantasma e da função que calcula a distância:

```c
void movimentos_F(char **M, char *AF, int *PX, int *PY, int *AFX, int *AFY, int *LF){
  int i, j, DM;
  int PMX, PMY;

  DM = Dist_Fant((*AFX),(*AFY),(*PX),(*PY));
  PMX = (*AFX) ;
  PMY = (*AFY);
  if((M[(*AFY)][(*AFX) - 1] != '|') && (Dist_Fant(((*AFX)-1),(*AFY),(*PX),(*PY)) < DM)){···
  }
  if((M[(*AFY)][(*AFX) + 1] != '|') && (Dist_Fant(((*AFX)+1),(*AFY),(*PX),(*PY)) < DM)){···
  }
  if((M[((*AFY) - 1)][(*AFX)] != '|') && (Dist_Fant((*AFX),((*AFY)-1),(*PX),(*PY)) < DM)){···
  }
  if((M[((*AFY) + 1)][(*AFX)] != '|') && (Dist_Fant((*AFX),((*AFY)+1),(*PX),(*PY)) < DM)){···
  }
  if((*AFX) == PMX && ((*AFY) == PMY)){···
  }

  if((*AF) == ' ')···
  if((*AF) == '.')···
  if((*AF) == '|')···
  (*AFX) = PMX;
  (*AFY) = PMY;
  (*AF) = M[(*AFY)][(*AFX)];
  if((*AFY) == (*PY) && (*AFX)== (*PX))···
  }···
  }
}
```

Se nosso herói tocar no fantasma perseguidor ou no fantasma guardião ele perde o jogo:

```
============================================================
|                    VOCE PERDEU        :(                 |
|                    TENTE NOVAMENTE!                      |
============================================================
Pressione qualquer tecla para continuar. . .
```

Se alcançar a PÍLULA naturalmente nosso herói vence o jogo:

```
============================================================
|                    PARABENS(A)!!!                        |
|                    VOCE VENCEU!!!                        |
============================================================

Pressione qualquer tecla para continuar. . .
```

Abaixo estrutura condicional que invoca as funções ganhou( ) e perdeu( ):

```
        if(*LF == 0){
            system("cls");
            perdeu();
        }

        if(M[3][3] == 'C'){
        (*LF) = 0;
            if(*LF == 0){
            system("cls");
            venceu();
        }
    }

         if(M[1][2] == 'C'){
        (*LF) = 0;
            if(*LF == 0){
            system("cls");
            perdeu();
        }
    }
}
```

Em seguida a função restart( ) é invocada para reiniciar o jogo ou encerrá-lo:

```
DESEJA CONTINUAR ?
1 - SIM  QUALQUER TECLA - NAO
```

```c
void restart(){
    int operador;
    system("cls");
    printf("\n\n");
    printf("DESEJA CONTINUAR ?\n1 - SIM  QUALQUER TECLA - NAO\n");
    scanf("%d", &operador);
    if (operador == 1){
        system("cls");
        int main();
            main();
    }
    else{
        system("cls");
        printf("\nOBRIGADO POR UTILIZAR O NOSSO JOGO!!!\n\n");
        ifma();
    }
}
```

Se o usuário optar por encerrar o jogo a funçaõ ifma( ) é invocada e então o programa é encerrado:

```
OBRIGADO POR UTILIZAR O NOSSO JOGO!!!


___00__000000000__00000_________00000___0000000___
___00__000000000__00_00_________00_00__000000000__
___00__00_________00__00_______00__00__00_____00__
___00__000000_____00___00_____00___00__000000000__
___00__000000_____00____00___00____00__000000000__
___00__00_________00_____00_00_____00__00_____00__
___00__00_________00______000______00__00_____00__
___00__00_________00________________00__00_____00__

======== #SISTEMASDEINFORMACAO #IFMA ==========
======== #MELHORCURSO #MELHORPROFESSORA ========


Process returned 0 (0x0)   execution time : 438.642 s
Press any key to continue.
```

REFERÊNCIAS BIBLIOGRÁFICAS:

Aulas/Materiais disponibilizados na disciplina LP1 do curso de SI do IFMA:

https://suap.ifma.edu.br/edu/sala_virtual/21912/?tab=aulas

https://suap.ifma.edu.br/edu/sala_virtual/21912/?tab=materiais