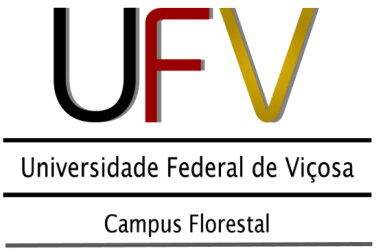

TRABALHO PRÁTICO 1
Disciplina de Projeto e Análise de Algoritmos

Daniel Fernandes Pinho - 2634
Samuel Aparecido Delfino Rodrigues - 3476
Vinicius Tadeu Silva Ribeiro - 2670

Segundo trabalho prático da disciplina Projeto e Análise de Algoritmos - CCF 331, do curso de Ciência da Computação da Universidade Federal de Viçosa - campus Florestal

Professor: Daniel Mendes Barbosa

Florestal - MG

2022

Sumário

1. Introdução

Este trabalho consiste na utilização do paradigma de algoritmo Backtracking. Através de um jogo onde devemos ajudar o jovem Ness (jogador) a andar por um mapa cheio de monstros com vários tipos de poderes e o objetivo é matar um monstro específico, o Giygas. Para isso o Backtracking deve encontrar o melhor caminho, onde cada caminho só pode ser percorrido uma única vez.

O programa deverá receber como entrada o nível de força do jogador, o nível de força dos inimigos, o nível de força que é aumentado a cada morte do inimigo e a quantidade de técnicas que podem ser usadas, juntamente isso, desenho do mapa do percurso indicando os caminhos que podem ser percorridos e a posição dos monstros.

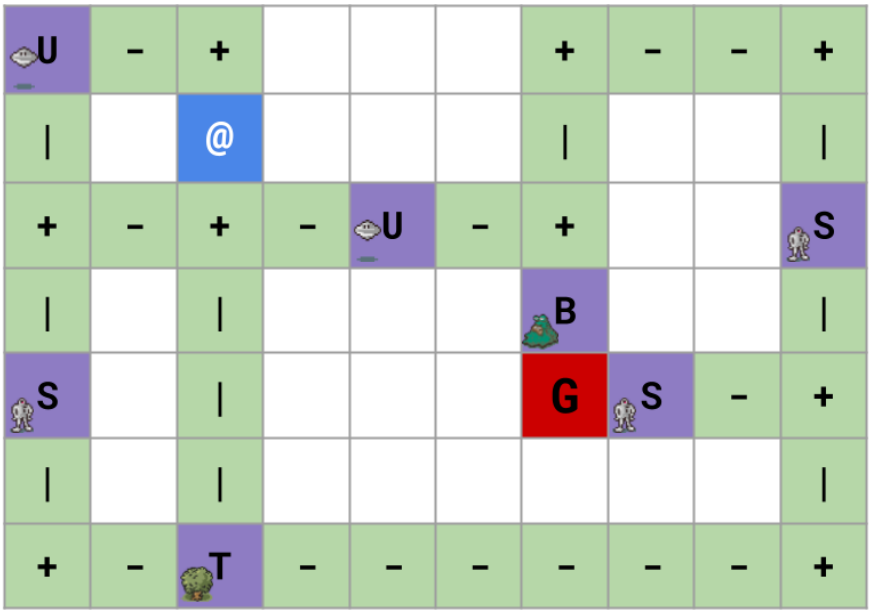

Figura 1. Representação do jogo exemplo

2. Desenvolvimento

Para a implementação do código, foi utilizada a linguagem de programação C, juntamente com a ferramenta Git para gerenciar controle das versões do código. Com intuito de manter a organização do código e facilitar o processo de análise dos algoritmos, criamos um TAD backtracking, sendo o mais importante, na qual temos o arquivo backtracking.h, nele está presente as funções para criar o labirinto, movimentar o Ness, verificar se é posição de batalha, verificar as extremidades e caminho do labirinto entre outras funções auxiliares para a movimentação do Ness. Temos também o TAD dados, para verificar se o Ness conseguiu sair do labirinto, a quantidade de movimentos e a impressão desses atributos. Foi

criado um gerador de labirinto para 3 modos: fácil, médio e difícil, na qual o labirinto vai crescendo a cada modo, respectivamente e um menu para a interação do usuário ao jogo.

Para a implementação do trabalho, utilizamos a estrutura de dados matriz, por ser uma estrutura mais parecida com um labirinto e alocação dinâmica para criar o labirinto, pois o tamanho dele será conhecido depois da leitura do arquivo.

```
int **IniciarLabirinto(int linha, int coluna, TipoNess *hess, int poder,
                       int especiaisRestantes);
void InserirPosicao(int ** labirinto, int linha, int coluna, int valor);
int EstudantePassou(int ** labirinto, int linha, int coluna);
int EstudanteEsta(int ** labirinto, int linha, int coluna);
int ChegouNoFim(int **labirinto, int i, int j);
int UltrapassouLimites(int i, int j, int linha, int coluna);
int LinhaEstudante(int ** labirinto, int linha, int coluna);
int ColunaEstudante(int ** labirinto, int linha, int coluna);
int EhParede(int ** labirinto, int linha, int coluna);
void MonstroEliminado(int ** labirinto, int linha, int coluna);
short EhPosicaoBatalha(int **labirinto, int linha, int coluna,TipoMonstroDatabase *monstroDatabase);
TipoMob IdentificaAmeaca(int **labirinto, int linha, int coluna, TipoMonstroDatabase *monstroDatabase);
int Movimenta_Estudante(int **labirinto, TipoNess *ness, int x, int y,int linha, int coluna, TipoDados *dados, TipoMonstroDatabase *monstrodatabase);
int Movimenta_Estudante_Analise(int ** labirinto, TipoNess *itens, int x, int y, int linha, int coluna, TipoDados *dados,long long int* NUM);
void ImprimirLabirinto(int ** labirinto, int linha, int coluna);
```

Figura 2. Funções para a manipulação do jogo do arquivo backtracking.h

2.1 Implementação

Na implementação do trabalho, o arquivo backtracking.h foi criado estruturas do *TipoNess* para o jogador, a estrutura do *TipoMob* para os monstros e a estrutura *TipoMonstroDataBase* para a identificação de cada monstro.

Para a implementação do arquivo dados.h foi criado uma estrutura com intuito do usuário receber os dados do que aconteceu no jogo e com o Ness , por exemplo, se ele saiu do labirinto, esse TAD é apenas um tad para informações ao usuário.

Os arquivos menu.h e gerador.h apresentam funções visuais para o usuário poder escolher fazer no jogo e gerar labirintos pequenos, médios e grandes, respectivamente.

A Partir das próximas seções serão apresentadas algumas funções mais importantes para a criação do projeto do jogo.

2.1.1 Função InserirPosicao

Nessa função será setado os monstros, identificados pelos números de 3 a 7, as posições que tem parede, identificado pelo número 2, a posição do Ness, identificado pelo número 0 e os caminho livre identificado pelo número 1.

```c
void InserirPosicao(char **labirinto, int linha, int coluna, int valor){
    if(valor == 0){ // posição inicial do estudante
        labirinto[linha][coluna] = '@';
    }else if(valor == 1){ // posição livre
        labirinto[linha][coluna] = 1;
    }else if(valor == 2){ //posição com parede
        labirinto[linha][coluna] = 2;
    }else if(valor == 3){ // posição com LiL UFO
        labirinto[linha][coluna] = 3;
    }else if(valor == 4){ // posição com Territorial Oak
        labirinto[linha][coluna] = 4;
    }else if(valor == 5){ // posição com Starman Junior
        labirinto[linha][coluna] = 5;
    }else if(valor == 6){ // posição com Master Belch
        labirinto[linha][coluna] = 6;
    }else{
        labirinto[linha][coluna] = 7; //posiçao com Giygas
    }
```

Figura 3. Função para inserir os elementos na matriz

2.1.2 Função MarcarPosicao

Essa função é uma das funções importantes, pois ela faz parte do algoritmo de Backtracking (para a movimentação do Ness no labirinto), ela irá marcar se o local já foi percorrido pelo estudante

```c
void MarcarPosicao(char **labirinto, int linha, int coluna){
  labirinto[linha][coluna] = '*';
}
```

Figura 4. Função para marcar a posição do Ness

2.1.3 Função NessPassou

Outra função que faz parte da movimentação do Ness, nela vai se verificar se o estudante passou por determinado caminho.

```
int NessPassou(char **labirinto, int linha, int coluna){
  if(labirinto[linha][coluna] == '*'){
    return 1;
  }
  return 0;
}
```

Figura 5. Função para verificar se o Ness passou pelo caminho

2.1.4 Função NessEsta

Nessa função, ele vai encontrar a posição inicial do Ness, caso precise voltar no backtracking.

```
int NessEsta(char **labirinto, int linha, int coluna){
  if(labirinto[linha][coluna] == '@'){
    return 1;
  }
  return 0;
}
```

Figura 6. Função para verificar a posição do Ness

2.1.5 Função EhParede

Função para verificar se determinada posição é uma parede (setado com o número 2).

```
int EhParede(int ** labirinto, int linha, int coluna){
  if(labirinto[linha][coluna] == 2){
    return 1;
  }
  return 0;
}
```

Figura 7. Função para verificar se é uma parede

2.1.6 Função EhPosicaoBatalha

Função para verificar se é uma posição de batalha, ou seja, posição que tem monstros (marcados pelos números 3 a 7).

```
short EhPosicaoBatalha(char **labirinto, int linha, int coluna) {
  if(labirinto[linha][coluna] >= 66 && labirinto[linha][coluna]<= 85){
    return 1;
  }
  return 0; // Nao tem monstro
}
```

Figura 8. Função para verificar se é uma posição de batalha

2.1.7 Função IdentficaAmeaca

Função para identificar qual o tipo de monstro em determinada posição, retornando ao tipo do monstro a ser eliminado.

```
TipoMob IdentificaAmeaca(int **labirinto, int linha, int coluna, TipoMonstroDatabase *monstroDatabase){
  if(labirinto[linha][coluna] >= 3 && labirinto[linha][coluna]<= 7){
    if(labirinto[linha][coluna] == 3){
      return monstroDatabase->LilUfO;
    }else if(labirinto[linha][coluna] == 4){
      return monstroDatabase->TerritorialOak;
    }else if(labirinto[linha][coluna] == 5){
      return monstroDatabase->StarmanJr;
    }else if(labirinto[linha][coluna] == 6){
      return monstroDatabase->MasterBelch;
    }else {
      return monstroDatabase->Giygas;
    }
  }
}
```

Figura 9. Função para verificar qual o tipo de monstro

2.1.8 Função UltrapassouLimites

Função onde se verificam as extremidades do labirinto (tamanho da matriz).

```
int UltrapassouLimites(int i, int j, int linha, int coluna){
  if(j >= coluna || i >= linha || j < 0){
    return 1;
  }
  return 0;
}
```

Figura 10. Função para verificar se ultrapassou os limites do labirinto

2.1.9 Função Batalha

Essa função vai realizar a batalha do Ness contra um monstro e modificar os valores do Ness e talvez do labirinto

```
short Batalha(char ** labirinto, TipoNess * ness, TipoMob monstro){
  if(monstro.identificador =='G'){
    if(ness->poder >= monstro.poder){
      ness->derrotouGiygas = 1;
      return 1;
    } else{
      ness->derrotado = 1;
      return 0;
    }
  }else{
    if(ness->poder >= monstro.poder){
      ness->poder += monstro.drop;
      return 1;
    }else{
      if(ness->EspeciaisRestantes > 0){
        ness->poder += monstro.drop;
        return 1;
      }
      ness->derrotado = 1;
      return 0;
    }
  }
}
```

Figura 11. Função para realizar uma batalha

2.1.10 Função Movimenta_Ness

Essa função é propriamente onde se faz o backtracking, nela se utiliza as funções mencionadas acima. Primeiramente verificando se o Ness chegou ao fim, depois se ele passou as extremidades do labirinto, em seguida verifica se não é parede e não é posição de batalha, significando que é um caminho válido, posteriormente verifica se é uma posição de batalha (ou seja tem um monstro) e identifica a ameaça, caso ele tenha especial ele elimina o monstro e logo após printando a posição na qual se encontra o monstro.

Em seguida verifica se o Ness esteja em uma posição válida, ou seja, nao seja parede, em um caminho que ele já foi e que não há monstro, o Ness marca essa posição e testa os movimentos para cima, para baixo, para direita e para esquerda, e a partir daí utiliza de forma recursiva a mesma função *Movimenta_Ness* para todos esses lados, assim aplicando o backtracking.

```
int Movimenta_Ness(char **labirinto, TipoNess **ness, int x, int y,int linha, int coluna, TipoDados *dados,TipoMonstroDatabase
    if(ChegouNoFim(*ness)){ /*O estudante chegou no final do labirinto*/
        DadosFinais(dados, y);
        MarcarPosicao(labirinto, x, y);
        printf("Linha: %d Coluna: %d\n", x, y);
        return 1;
    }
    if(UltrapassouLimites(x, y, linha, coluna)){ //Posição fora do espaço do labirinto
        dados->consegueSair = 0;
        return 0;
    }
    if(!EhParede(labirinto, x, y) && !EhPosicaoBatalha(labirinto, x, y)){ //posição valida
        dados->quantMovimentacao++;
        printf("Linha: %d Coluna: %d\n", x, y);
    }
    if(EhPosicaoBatalha(labirinto, x, y)){
      if(Batalha(labirinto,*ness, IdentificaAmeaca(labirinto,x,y,monstrodatabase)) == 0){

      }
      dados->quantMovimentacao++;
      printf("Monstro %c na Linha: %d Coluna: %d\n",IdentificaAmeaca(labirinto, x, y,monstrodatabase).identificador, x, y);
```

Figura 12. Função para a movimentação do Ness

```
if(!EhParede(labirinto, x, y) && !NessPassou(labirinto, x, y) && !EhPosicaoBatalha(labirinto, x, y)){
    MarcarPosicao(labirinto, x, y);
    if(Movimenta_Ness(labirinto, ness, x - 1, y, linha, coluna, dados,
                      monstrodatabase)); // para cima
    else{
        if(Movimenta_Ness(labirinto, ness, x, y + 1, linha, coluna, dados,
                          monstrodatabase)); // para a direita
        else{
            if(Movimenta_Ness(labirinto, ness, x, y - 1, linha, coluna,
                              dados, monstrodatabase)); // para a esquerda
            else{
                if(Movimenta_Ness(labirinto, ness, x + 1, y, linha, coluna,
                                  dados, monstrodatabase)); //para baixo
                else{
                    return 0;
                }
            }
        }
    }
}else{
  return 0;
}
return 1;
```

Figura 13. Continuação da função para a movimentação do Ness

2.1.11 Função Movimenta_Ness_Analise

Essa função é a mesma que a função *Movimenta_Ness*, utilizando as mesmas verificações, porém com o modo análise, analisando e printando a cada posição.

4. CONCLUSÃO

A implementação de um paradigma de algoritmo por meio de uma matriz labirinto para realização das tarefas solicitadas, conseguiu aprofundar de maneira prática os conteúdos vistos nas aulas. Além de incrementar o aprendizado sobre lógica de programação, elaborando quais caminhos poderiam ser utilizados de maneira prática e eficiente para execução do projeto.

Além do âmbito do aprendizado, para organização do grupo foram utilizadas algumas ferramentas como o Whatsapp no qual podemos nos comunicar sobre o trabalho e também a plataforma Github, que permitiu com que os membros do grupo pudessem ter acesso a todas as partes que estavam sendo elaboradas e controlar as versões do código.

Por fim, não tivemos grandes problemas na implementação do trabalho prático, visto que trabalhamos em um ambiente colaborativo. Portanto, por meio dos esforços do grupo, o trabalho prático foi concluído com sucesso.

5. REFERÊNCIAS

[1]Ziviani, N. (2010). *Projeto De Algoritmos: Com Implementações Em Pascal E C.* Cengage Learning.