DOCUMENTAÇÃO TRABALHO AED2

Marcos Gabriel Moreira Fonseca - 12221BCC043.
Felipe Santos Silva - 12221BCC042.
Breno Oliveira Cavalcante - 12221BCC011.

Link repositório Replit: https://replit.com/@felipesantos200/Jogo-Labirinto#Menu.c

Item 1 - Área:

1. Raciocínio Principal:

Cada área é um grafo representado na forma de lista encadeada, onde suas adjacências são as possíveis salas que o jogador pode se locomover, a leitura das áreas é feita ao chamar o jogo. Exemplo de área(Área 2.txt):

2. Estrutura de Dados:

A estrutura do projeto define um grafo direcionado representado por uma lista de adjacência. Cada vértice do grafo é representado por um índice no array Adj e cada elemento dessa lista de adjacência armazena os vértices adjacentes e os pesos das arestas que os conectam, sendo para essa área o peso igual a 0.

```
typedef struct TipoGrafo {
  TipoLista Adj[MAXNUMVERTICES];
  int NumVertices;
  int NumArestas;
} TipoGrafo;

typedef int TipoValorVertice;
typedef int TipoPeso;

typedef struct TipoItem {
  TipoValorVertice Vertice;
```

```
  TipoPeso peso;
} TipoItem;

typedef struct TipoCelula {
  TipoItem Item;
  struct TipoCelula *Prox;
} TipoCelula;
typedef TipoCelula *TipoApontador;

typedef struct TipoLista {
  TipoApontador Primeiro, Ultimo;
} TipoLista;
```

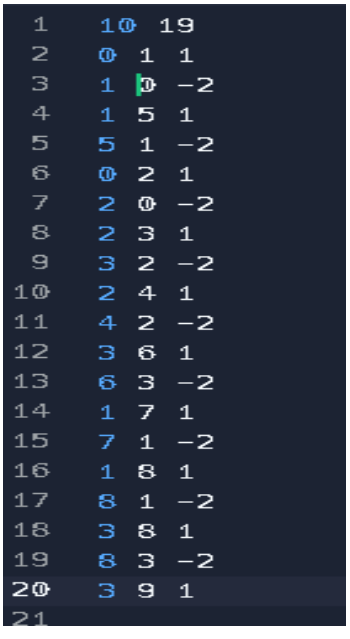
3. Funções:

(Explícitas dentro do código, Seção Grafos ‘jogo.h’):

Item 2 - Área Central:

1. Raciocínio Geral:

Segue o mesmo padrão da área descrita anteriormente, porém com algumas diferenças, na área central “é diferente das demais salas contabiliza uma pontuação para cada avanço e permite que o jogador retroceda uma sala sacrificando um ponto. A derrota só ocorre caso o jogador encontre uma sala vazia sem pontos acumulados para usar o retorno.” Seguindo essa requisição foi pensado em adicionar o vértice de retorno na lista de possíveis caminhos, uma pontuação é contabilizada para ida e uma maior pontuação é diminuída evitando assim que o jogo sempre tenha a mesma quantidade de pontos, para leitura dessa área especial



foi inserido um arquivo especial que contém mais conexões e pesos correspondentes. Situação:

2. Estrutura de Dados:

Assim como na área normal temos o mesmo uso das estruturas de dados, porém alteramos aqui o peso das vertentes para contabilizar pontuação.

3. Funções:

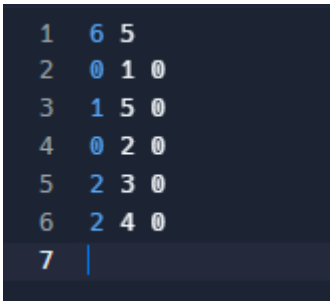
(Explícitas dentro do código, Seção Grafos ‘jogo.h’):

Item 3 - Dificuldade do percurso:

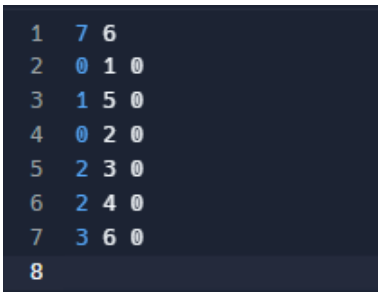
1. Raciocínio Geral:

A dificuldade do percurso aumenta de acordo com que se passa de fase. Tal aumento se deve pelo fato de os arquivos das áreas apresentarem mais caminhos com o subir de nível, ou seja, à medida que ocorre um avanço de fase, o número de salas da próxima área **aumenta** em **uma unidade** em relação à anterior. Sendo que a primeira área começa com 4 salas, a segunda com 5 salas, e vai aumentando de uma unidade até chegar na área central com um total de 10 salas.

Exemplo:



X



Item 4 - Mapa do labirinto:

1. Mapa do Labirinto:

O labirinto é composto por um conjunto de áreas conectadas por caminhos. Cada área é representada por um vértice no grafo.

O labirinto tem uma estrutura em que o jogador avança de região em região, completando todas as áreas de uma região antes de avançar para a próxima. Todas as áreas chegam a área central.

2. Progressão do Jogador:

O jogador avança pelo labirinto percorrendo as áreas e seguindo os caminhos disponíveis.

Ao completar todas as áreas de uma região, o jogador avança para a próxima região. O objetivo final é chegar à Área Central do labirinto.

Basicamente, utilizamos o percurso pós-fixado para tal, pois o último nó a ser visitado ou seja (percorrido a área) é a Área central, que também deve ter o maior número de salas, pois a cada avanço de FASE ocorre aumento de SALAS em uma unidade.

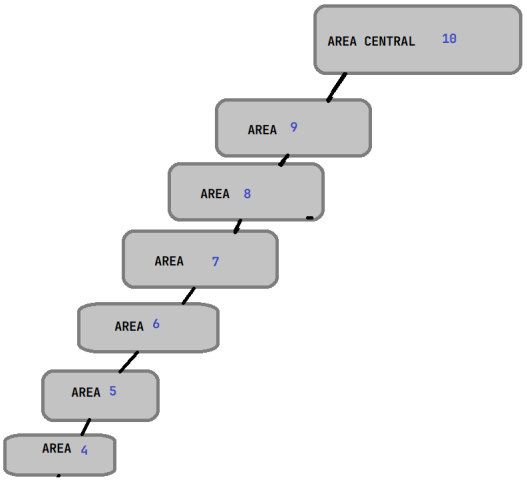
Com isso, temos que ter uma árvore degenerada para a esquerda, pois na raiz da árvore binária temos o maior elemento, e a partir dela, só temos elementos à esquerda da raiz de nossas sub-árvores, indicando que são menores que a raiz.

3. Visão do Percurso:

Após avançar para uma nova região, o jogador pode visualizar seu progresso.
Isso pode ser feito mostrando as áreas já percorridas e as áreas que ainda precisam ser exploradas.
Ele utiliza de uma seta, que mostra a área atual do jogador, por quantas áreas ele já passou, e mostra também quantas áreas faltam até a área central.

4. Representação:

A primeira área tem 4 vértices, a segunda área tem 5 vértices…



Leitura:

```
// Lê todas as ÁREAS e insere no MAPA
    for (int i = MAX_AREAS; i >= 1; i--) {
      sprintf(nome_arquivo, "AreasGame/Area%d.txt", i);
      registro.Chave.grafo = LerGrafoDoArquivo(nome_arquivo);
      Insere_binaria(registro, &Dicionario);
    }
```

Item 4- Sistema de Ranking:

1.Criação do Jogador:

Após o usuário inserir 1 no menu e começar a jogar, o programa pede para que o nome seja inserido, sendo assim, chamamos a função “chamaDadosJogador”, a qual contém a função “criarJogador”, a qual inicializa a struct Jogador com os dados zerados e o nome é recebido por parâmetro.

```
Jogador chamaDadosJogador(char *nomeBusca) {
  FILE *pont_arq;
  char nome[MAX_NOME];
  Jogador player1;
  criar_jogador(&player1, nomeBusca);
```

```
void criar_jogador(Jogador *jogador, char *nomeJogador) {
  strcpy(jogador->nome, nomeJogador);
  jogador->tempoTotal = 0.0;
  for (int i = 0; i < MAX_AREAS; i++) {
    jogador->tempoPorArea[i] = 0.0;
  }
  jogador->pontuacaoCentral = 0;
}
```

2.Arquivo e vetor:

Após a criação do jogador, guardamos seu nome, e após terminar o jogo, o inserimos em um vetor, e através dele, o inserimos no arquivo para que os dados se mantenham mesmo após o encerramento do programa. Sendo assim, percorremos o arquivo em modo de leitura, e após acharmos o nome do jogador com todos seus dados, o copiamos para o restante da Struct deste mesmo jogador (dados como: Tempo em cada área, pontos na área central etc.… os quais só podem ser obtidos após o uso do programa).

```c
// Função para adicionar um jogador ao ranking
void adicionarJogador(Jogador novoJogador) {
    ranking[numJogadores] = novoJogador;
    numJogadores++;
}
```

```c
void adicionarJogador_Arquivo(Jogador novoJogador) {
    FILE *arquivo;
    arquivo = fopen("Arquivos-Ranking/ranking.txt", "a");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        return;
    }
    fprintf(arquivo, "%s %.2lf %.2lf %.2lf %.2lf %.2lf %d\n",
            novoJogador.nome, novoJogador.tempoTotal,
            novoJogador.tempoPorArea[0], novoJogador.tempoPorArea[1],
            novoJogador.tempoPorArea[2], novoJogador.tempoPorArea[3],
            novoJogador.tempoPorArea[4],
            novoJogador.pontuacaoCentral);
    fclose(arquivo);
}
```

```c
    // Verificar se o nome buscado é igual ao nome lido do arquivo
    if (strcmp(nome, nomeBusca) == 0) {
        double tempoPorAREA[MAX_AREAS], tempTotal;
        int pontosNaCentral;
        // Ler os valores de vitórias, derrotas e pontos da linha
        sscanf(nome + strlen(nomeBusca) + 1, "%lf %lf %lf %lf %lf %lf %d",
               &tempoPorAREA[0], &tempoPorAREA[1], &tempoPorAREA[2],
               &tempoPorAREA[3], &tempoPorAREA[4], &tempoPorAREA[5], &tempTotal,
               &pontosNaCentral);

        // Fechando o arquivo
        fclose(pont_arq);

        // Atribuir os valores lidos à estrutura do jogador
        strcpy(player1.nome, nomeBusca);
        for (int it = 0; it < MAX_AREAS; it++)
            player1.tempoPorArea[it] = tempoPorAREA[it];
        player1.tempoTotal = tempTotal;
        player1.pontuacaoCentral = pontosNaCentral;

        return player1;
```

(Última foto em relação a parte da função “chamaDadosJogdaor”).

3. Ordenação:

Para a ordenação, foi utilizado o Insertion Sort, já que o algoritmo é estável e simples de implementar. Essas características tornam o método atrativo pois a estabilidade dele mantém o jogadores com pontuações e tempos iguais ordenados cronologicamente e pela simplicidade é interessante pela quantidade de jogadores normalmente ser baixa.

4. Na prática:

Na prática, pedimos para o usuário escolher o critério de ordenação, o qual pode ser feito por meio do tempo total que o jogador gastou nos labirintos e pontuação na Área Central.

Após o usuário inserir o método de ordenação, o programa ordena o vetor de jogadores pelo critério pedido, e após ser ordenado, o ranking é mostrado com os 10 primeiros elementos do vetor. Com isso, o arquivo não é ordenado, ele é utilizado apenas para permanência dos dados, a ordenação é feita pelo vetor.

```c
// Função para ordenar o ranking pelo tempo total
void ordenarRanking_TempoTotal() {
  int i, j;
  Jogador chave;

  for (i = 1; i < numJogadores; i++) {
    chave = ranking[i];
    j = i - 1;

    // Invertendo a condição para ordenar do menor tempo para o maior
    while (j >= 0 && ranking[j].tempoTotal > chave.tempoTotal) {
        ranking[j + 1] = ranking[j];
        j = j - 1;
    }
    ranking[j + 1] = chave;
  }
}

void ordenarRanking_PontuacaoCentral() {
  int i, j;
  Jogador chave;

  for (i = 1; i < numJogadores; i++) {
    chave = ranking[i];
    j = i - 1;

    while (j >= 0 && ranking[j].pontuacaoCentral < chave.pontuacaoCentral) {
        ranking[j + 1] = ranking[j];
        j = j - 1;
    }
    ranking[j + 1] = chave;
  }
}
```

Manual

Para explicar o funcionamento do jogo em si foi feito um manual que está presente dentro do próprio jogo em si.