Nome: Tális Breda

Disciplina: INE5408 - Estruturas de dados

Relatório do projeto I

Explicação resumida

A solução para o problema apresentado no projeto foi pensada de forma a minimizar o uso de memória e tempo de execução.

O algoritmo, resumidamente, funciona da seguinte forma:

- Lê o arquivo e retorna seu conteúdo em formato string, e checa cada um dos caracteres desta string.
- Checa a sintaxe e a estrutura do XML, retornando ‘erro’ caso algo esteja errado.
- Pula as matrizes, armazenando apenas seus dados (nome, altura, largura, x do robô, y do robô, índice do carácter inicial).
- Se não houver nenhum erro na estrutura, será feita uma iteração para cada conjunto de dados de matrizes armazenado.
- Cada matriz então é construída a partir do seu primeiro elemento, e o algoritmo que encontra a quantidade total de espaços que o robô percorre é executado em cima dessa matriz

Dessa forma, o algoritmo fica otimizado para que não precise percorrer as matrizes desnecessariamente, para os casos onde há erro(s) na estrutura XML. Também não há uso excessivo de memória, pois apenas uma matriz é armazenada por vez.

Explicação detalhada

Iniciando pelo main:

```
int main() {
    std::string filePath;
    std::cin >> filePath;

    std::string fileContents = readFileToString(filePath);

    std::vector<std::string> pilha;

    bool inTag = false;
    bool inClosingTag = false;
    bool xmlCorrect = true;

    int lineCounter = 1;
    std::string tag;

    std::map<std::string, std::vector<int>> matrixDataMap;
    std::string s_altura;
    std::string s_largura;
    std::string nome;
    std::string s_x;
    std::string s_y;

    std::vector<std::string> tagSequence = {"matriz", "y", "x", "robo", "largura",
"altura", "dimensoes", "nome", "cenario", "cenarios"};

    std::map<std::string, std::function<void(char)>> functionMap = {
        {"altura", [&s_altura](char c) {appendToString(s_altura, c);} },
        {"largura", [&s_largura](char c) {appendToString(s_largura, c);} },
        {"nome", [&nome](char c) {appendToString(nome, c);} },
        {"x", [&s_x](char c) {appendToString(s_x, c);} },
        {"y", [&s_y](char c) {appendToString(s_y, c);} },
    };
```

Aqui são definidas as variáveis que serão utilizadas durante a execução do programa.

- **fileContents** é string com o conteúdo do arquivo xml
- **pilha**: pilha que será utilizada para verificar se as tags estão sendo fechadas corretamente
- **inTag** e **inClosingTag** são flags que dizem se o caractere atual está dentro de uma tag, sendo aberta ou sendo fechada, respectivamente.
- **xmlCorrect** diz se a estrutura do XML está correta, e no momento que ela se torna *false*, o programa fecha.

- **matrixDataMap** armazena os dados de cada matriz: índice do caractere inicial, x e y iniciais do robô, altura e largura da matriz, e o nome do cenário.
- As strings **s_altura**, **s_largura**, **nome**, **s_x** e **s_y** armazenam seus respectivos valores em formato string, depois sendo convertidos para inteiro.
- **tagSequence** representa a sequência, em ordem invertida, das tags que o XML deve conter

readFileToString()

```
std::string readFileToString(const std::string& filePath) {
    std::ifstream file(filePath);

    if (!file.is_open()) {
        std::cerr << "Error opening file: " << filePath << std::endl;
        return "";
    }

    std::stringstream buffer;
    buffer << file.rdbuf();

    std::string fileContents = buffer.str();

    file.close();
    return fileContents;
}
```

Essa é uma função simples, que lê o arquivo e retorna seus conteúdos em formato string.

Loop principal

```
for (std::string::size_type i = 0; i < fileContents.size(); ++i) {
       char aux = fileContents[i];

       if (aux == '\n') {
           lineCounter++;
       }

       xmlCorrect = checkSyntax(aux, inTag, inClosingTag, tag, pilha, tagSequence);
       if (!xmlCorrect) break;

       if (!pilha.empty() && pilha.back() == "altura" && isdigit(aux)) {
           appendToString(s_altura, aux);
       }
       if (!pilha.empty() && pilha.back() == "largura" && isdigit(aux)) {
           appendToString(s_largura, aux);
       }
       if (!pilha.empty() && pilha.back() == "nome" && aux != '<' && aux != '>' &&
!inClosingTag) {
           appendToString(nome, aux);
       }
       if (!pilha.empty() && pilha.back() == "x" && isdigit(aux)) {
           appendToString(s_x, aux);
       }
       if (!pilha.empty() && pilha.back() == "y" && isdigit(aux)) {
           appendToString(s_y, aux);
       }

       if (!pilha.empty() && pilha.back() == "matriz" && isdigit(aux)) {
           i += skipMatrix(i, lineCounter, s_x, s_y, s_altura, s_largura, nome,
matrixDataMap);
       }
   }
```

Esse é o loop que vai passar por cada um dos caracteres da string.

Cada vez que um caractere ‘\n’ (quebra de linha) for identificado, **lineCounter** aumentará em 1.

xmlCorrect receberá o valor retornado de **checkSyntax()**, que é a função que verifica a estrutura das tags. Caso essa função retorne *false*, o programa será fechado.

Após a verificação de estrutura, é checado qual o último elemento da pilha. Caso seja “altura”, “largura”, “x” e “y”, e o caractere sendo checado seja um número, este é colocado na sua respectiva string, para ser convertido posteriormente.

No caso do “nome”, é verificado se o caractere não é ‘<’ ou ‘>’, e se este não está dentro de uma tag sendo fechada.

Caso o topo da pilha seja “matriz” e o caractere atual seja um número, a função **storeMatrixDataAndSkip()** é chamada. Esta armazena os dados da matriz e

retorna a quantidade de caracteres a serem pulados. Será explicada detalhadamente posteriormente.

checkSyntax()

```
bool checkSyntax(
       char aux,
       bool &inTag,
       bool &inClosingTag,
       std::string &tag,
       std::vector<std::string> &pilha,
       std::vector<std::string> &tagSequence) {

   if (inTag) {

       if (aux == '/') {
           inClosingTag = true;
           inTag = false;
           return true;
       }

       if (aux == '<') {
           printf("erro\n");
           return false;
       }

       if (aux != '>') {
           tag += aux;
       } else {
           if (!pilha.empty() && tag == pilha.back()) {
               printf("erro\n");
               pilha.pop_back();
               return false;
           } else {
               if (tag == tagSequence.back()) {
                   pilha.push_back(tag);
                   tagSequence.pop_back();
               } else {
                   printf("erro\n");
                   return false;
               }
           }
           tag = "";
           inTag = false;
           return true;
       }
   }

   if (inClosingTag) {
       if (aux != '>') {
           tag += aux;
       } else {
           inClosingTag = false;
           std::string top = pilha.back();
           if (top == tag) {
```

```
                tag = "";
                pilha.pop_back();
                if (tagSequence.empty()) tagSequence = {"matriz", "y", "x", "robo",
"largura", "altura", "dimensoes", "nome", "cenario"};
                return true;
            } else {
                printf("erro\n");
                tag = "";
                return false;
            }
        }
    }

    if (aux == '<') {
        inTag = true;
    }

    return true;
}
```

Essa é a função um pouco mais complexa e difícil de entender do programa.

Primeiramente, ela checa se o caractere atual é ‘<’. Se for, significa que uma tag está sendo aberta ou fechada, e a flag **inTag** é ativada.

Após isso, verifica se o caractere após esse é ‘/’. Se for, significa que uma tag está sendo fechada, então a flag **inTag** é desativada, e a flag **inClosingTag** é ativada.

Enquanto a flag **inTag** estiver ligada e o caractere verificado não for ‘>’, este é adicionado à string **tag**. Assim que chegar no caractere ‘>’, verifica se **tag** é diferente do último elemento da **pilha** e se é igual ao último elemento de **tagSequence**. Caso satisfaça essas condições, quer dizer que a **tag** está no lugar correto dentro da sequência e é válida, então é adicionada à **pilha**, e o último elemento de **tagSequence** é removido.

A mesma lógica se segue enquanto **inClosingTag** estiver ligada, exceto que quando chega ao final da **tag**, é verificado se **tag** é igual ao último elemento da **pilha**, e caso seja, ela está correta e o último elemento da **pilha** é removido. Caso **tagSequence** esteja vazio, ele será resetado, pois um novo cenário estará se iniciando. Neste caso, **tagSequence** receberá todas as tags, exceto “cenarios”.

A função retorna *true* nos casos onde tudo ocorre como esperado, e *false* quando ocorre um problema.

appendToString()

```
void appendToString(std::string &s, char aux) {
   std::string auxString(&aux, 1);
   s.append(auxString);
}
```

Essa função é bastante simples, apenas instancia uma string a partir do endereço do caracter passado como parâmetro, e o adiciona na string principal, também passada como parâmetro.

storeMatrixDataAndSkip()

```
int storeMatrixDataAndSkip(
       int posInicial,
       int& lineCounter,
       std::string &s_x,
       std::string &s_y,
       std::string &s_altura,
       std::string &s_largura,
       std::string &nome,
       std::map<std::string, std::vector<int>> &map)
   {
   int altura = std::stoi(s_altura);
   int largura = std::stoi(s_largura);
   int x = std::stoi(s_x);
   int y = std::stoi(s_y);

   std::vector<int> vetor = {posInicial, x, y, altura, largura};

   map[nome] = vetor;

   s_x.clear();
   s_y.clear();
   s_altura.clear();
   s_largura.clear();
   nome.clear();
   lineCounter += altura;

   return altura * (largura + 1)-1;
}
```

Essa função armazena os dados necessários da matriz. Ela recebe strings contendo **x** e **y** do robô, **altura** e **largura** da matriz, **nome** do cenário e **índice** do primeiro caractere da matriz.

O método **std::stoi** converte as strings para números inteiros, armazena estes junto com a posição inicial em um vetor, e armazena esse vetor dentro de **matrixDataMap**, com a chave sendo o nome do cenário. Após isso as strings são resetadas e soma-se altura no **lineCounter**, pois a matriz será pulada.

O retorno é a multiplicação da altura da matriz com (largura+1), representando o total de caracteres da matriz juntamente com os caracteres de quebra de linha. -1 no final é um ajuste.