



UNIVERSIDADE FEDERAL DE SERGIPE  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
DEPARTAMENTO DE COMPUTAÇÃO  
PROGRAMAÇÃO IMPERATIVA

MAZE WALKER

GABRIEL DOS SANTOS  
RADUCHIU AMARAL  
RAUL ANDRADE

SÃO CRISTÓVÃO - SE  
2016

## **ESPECIFICAÇÃO**

Neste trabalho foi desenvolvido um código simples em Linguagem C, no qual envolvia ler um arquivo do computador e imprimir no terminal como forma de labirinto e de modo aleatório fazer animação de um boneco andando pelo labirinto.

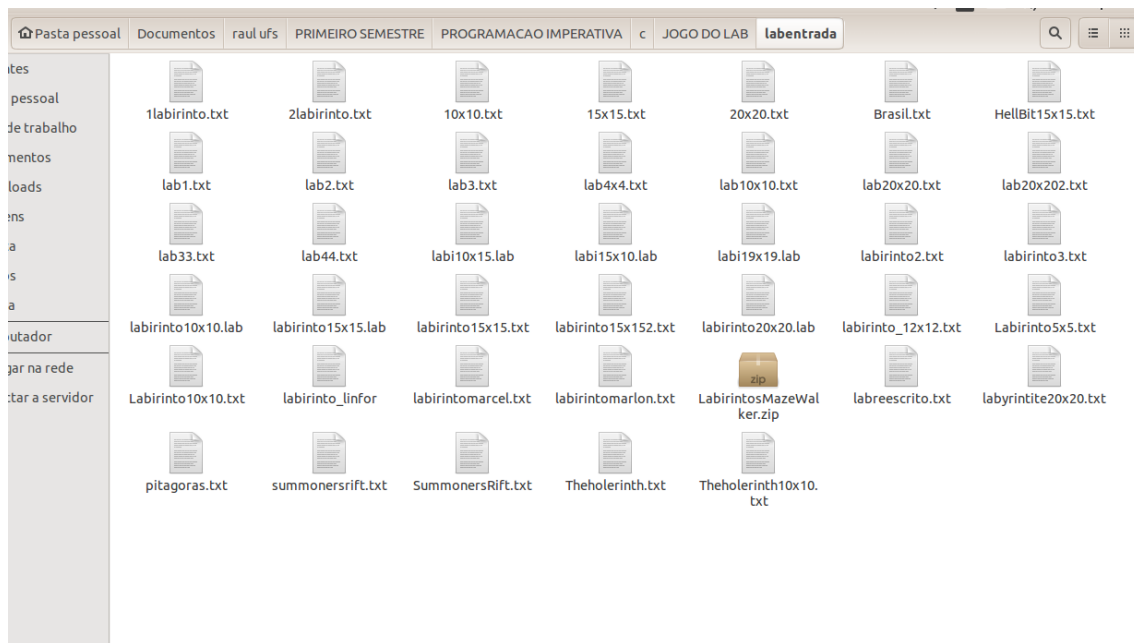
Este relatório tem como finalidade expor como foi o desenvolvimento do MAZE WALKER, que teve como intuito fazer os alunos vivenciarem desenvolvimento de algoritmo em equipe. O trabalho foi solicitado pela Professora Dr. Beatriz Trinchão Andrade para obtenção da terceira nota na disciplina de Programação Imperativa do primeiro semestre de Ciência da Computação, que teve como monitor Mateus Carvalho.

## RELATÓRIO

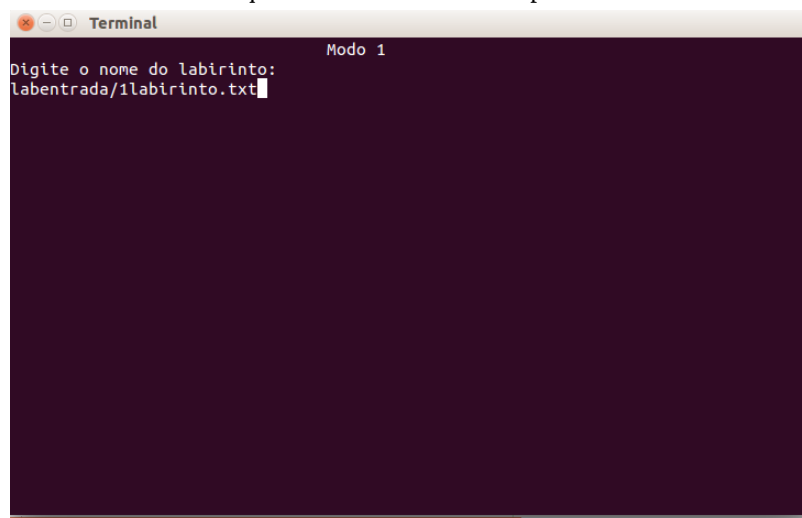
Primeiramente após estudar sobre arquivos e como eram feitas as leituras começamos a desenvolver nosso trabalho, o começo foi via aplicativos de mensagens, tivemos algumas ideias e assim no dia seguinte iniciamos a implementação do código.

Dando inicio a leitura de arquivos basicamente ela foi feita com um ponteiro que indicava um arquivo armazenado no computador, em seguida era recebido os dois primeiros números como parâmetros da matriz de caractere, após isso, o labirinto do arquivo de texto era salvo em uma matriz no programa.

Junto com a leitura do arquivo, foi criado 4 variáveis, duas delas para salvar as posições de início do personagem, e as outras duas para as posições finais do labirinto.

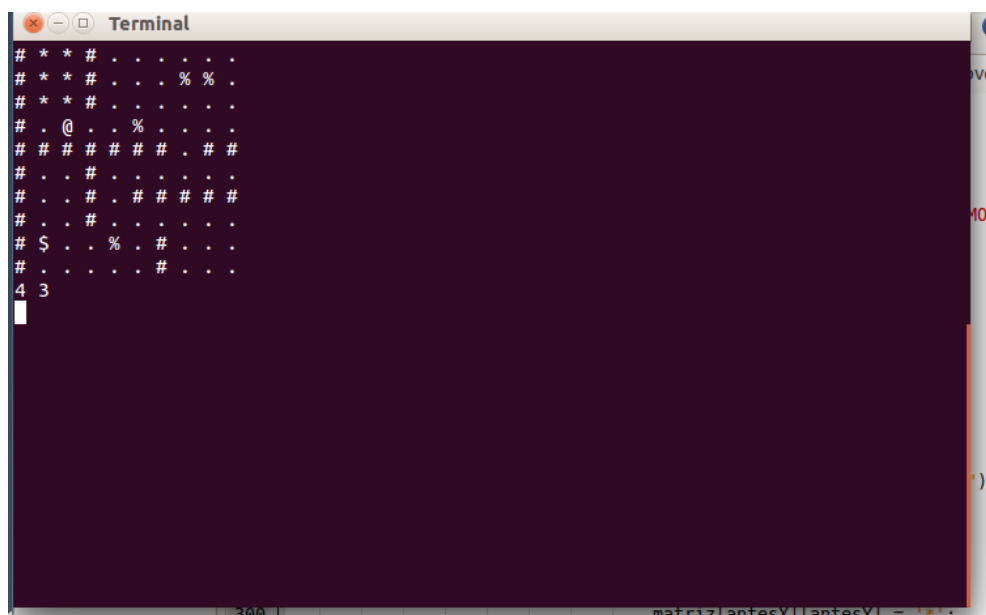


Arquivo Armazenado no computador



Chamando arquivo armazenado.

Quando o programa começou a ler e imprimir a matriz sem algum problema, foi dado início a implementação da estratégia para movimentar o personagem. Primeiramente foi criado um struct com dois números X e Y, que seriam as possibilidades que o boneco teria para se movimentar. Esse struct é um vetor de 5 posições, porém, não utilizamos a posição 0 por causa do contador que colocamos dentro das condições. Declaramos então o contador e uma série de condições para avaliar se o boneco poderia se movimentar (*ou não*) para a posição em questão. Cada condição tem uma posição, utilizando o struct com as posições X e Y, por exemplo "*cont++*; *possib[cont].x = x+1*; *possib[cont].y = y*;" nesse caso, se a condição fosse verdadeira, o contador seria incrementado antes, e o struct na posição do contador seria movimentar o boneco para cima. Além disso foi criado uma função randômica que recebia um valor como parâmetro e que gera um valor entre 1 e contador, então foi colocada uma variável para receber a função randômica com o contador como parâmetro. Depois disso o valor gerado seria utilizado para pegar o valor da posição do vetor da struct e movimentar o personagem para esse lugar.



```
Terminal
# * * # . . . . .
# * * # . . . % % .
# * * # . . . . .
# . @ . . % . . . .
# # # # # . # #
# . # . . . . .
# . . # . # # # #
# . # . . . . .
# $ . . % . # . . .
# . . . . # . . .
4 3
```

Personagem movimentando-se.

O personagem já estava se movendo sem problema algum, e o próximo passo era criar algumas saídas validas como por exemplo "*chegou ao final*", "*se perdeu*" ou "*morreu*". Após ler a matriz sabíamos onde cada caractere estava, após isso foi fácil descobrir onde se encontrava a posição final, bastava criar duas variáveis que recebiam as posições referente ao ponto final do labirinto, para isso utilizamos um comando de decisão. Como havíamos utilizados vários comandos de decisão para ele se movimentar, então quando não havia nenhuma possibilidade para caminhar ou movimentar-se (*quando o contador fosse 0*) era sinal que havia se perdido. Foi utilizado um comando randômico para simular a luta, assim quando ele encontrava o inimigo pelo caminho lutava se vencesse continuava o caminho, caso contrario o programa parava de ser executado.

```
Terminal
# * * # . . . . .
# ? * # . . . % % .
# * * # . . . . .
# . . . . % . . . .
# # # # # # . # #
# . . # . . . . .
# . . # . # # # #
# . . # . . . . .
# $ . . % . # . . .
# . . . . . # . . .
2 2
>>>0 personagem se perdeu...
labirinto de saida:
█
```

Personagem se perdendo.

```
Terminal
# * * # * * * * *
# * * # * . . % + *
# * * # * . . . . .
# . * * * % . . . .
# # # # # # . # #
# . . # . . . . .
# . . # . # # # #
# . . # . . . . .
# $ . . % . # . . .
# . . . . . # . . .
2 9
>>>0 personagem morreu em combate...
labirinto de saida:
█
```

Personagem morrendo.

```
Terminal
Tentativas: 302
# * . # . . . . .
# * . # . . . % ! *
# * . # . . * * * *
# * * * * ! * * * *
# # # # # # * # #
# . . # * * * * . .
# . . # * # # # #
# . . # * * . . . .
# V . . % * # . . .
# * * * * * # . . .

-----
(program exited with code: 0)
Press return to continue
█
```

Personagem vencendo. (ignore as 302 tentativas)

A luta foi implementada quase que por ultimo, tivemos divergência a respeito de como implementar a função de luta, mas no final estávamos falando praticamente a mesma coisa. A luta foi implementada através de duas funções. Uma delas avaliava o nível atual do personagem, e a outra gerava um número randômico que seria o nível do inimigo. A escala dos valores de luta foram implementados de 1 a 10. O personagem começa o labirinto com o nível de habilidade 5, e então quando encontrava um inimigo, um valor aleatório era gerado para o inimigo, e o que fosse maior, ganharia a batalha. E a cada batalha vencida pelo personagem era incrementado mais 10% (*na variável chamada Skill*) em habilidade de luta ficando assim cada vez mais forte, podendo até chegar a invencibilidade.

### **ALGUNS PROBLEMAS ENCONTRADOS DURANTE A IMPLEMENTAÇÃO:**

- **A questão da exclamação “!”:** Quando o personagem eliminava um inimigo, o mesmo acabava se perdendo do mapa, ficava somente a “!” e o “@” que no caso é o personagem, ele simplesmente desaparecia. Isso foi resolvido criando mais uma variável para armazenar a ultima posição acessada pelo “@” personagem.
- **A leitura do arquivo:** Depois de alguns dias implementando o código descobrimos que ele não estava lendo labirintos maiores que 10x10, porém esse erro era meio tolo pois foi somente durante a leitura que faltava um espaço no *"fscanf"*, esse erro tolo causou muita impaciência.
- **A arquivo de saída:** Mais um erro tolo, estávamos tentando salvar o arquivo de saída passando o parâmetro "W", e ele não imprimia nada no arquivo de texto, ficamos procurando e olhando varias vezes o código, quando na verdade faltava somente colocar um "W+" para poder criar e imprimir no arquivo.
- **Problema de movimentação:** Estávamos criando um código baseado em: “Pode andar ? Ande.” porem deu muito errado e com o passar do tempo isso ficou praticamente inviável. Então criamos um outro tipo de movimentação que se baseava em, quais são as possibilidades de andar? Escolha uma e ande, isso melhorou bastante e corrigiu muitos bugs.
- **Funções que não deram certo:** Outro código que não deu certo foi um no qual fizemos muitas funções e uma chamando a outra, porém chegou em um momento do código que ficamos perdidos/confusos, portanto concluímos que não seria viável ir por essa linha de raciocínio.