UNIVERSIDADE FEDERAL DE SERGIPE - *CAMPUS* SÃO CRISTÓVÃO

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

RELATÓRIO DE PROGRAMAÇÃO IMPERATIVA

GUILHERME FERREIRA AMÂNCIO

JOSÉ MARCOS BITTENCOURT OLIVEIRA PRADO

PIETRO OLIVEIRA LIMA

SIBELE OLIVEIRA CRUZ

São Cristóvão - SE

2025

UNIVERSIDADE FEDERAL DE SERGIPE - *CAMPUS* SÃO CRISTÓVÃO

GUILHERME FERREIRA AMÂNCIO
JOSÉ MARCOS BITTENCOURT OLIVEIRA PRADO
PIETRO OLIVEIRA LIMA
SIBELE OLIVEIRA CRUZ

Relatório referente ao Projeto “Labirinto” apresentado à disciplina de Programação Imperativa, ministrada pela professora Dra Beatriz Trinchão Andrade, como requisito parcial para aprovação, dos cursos de Bacharelado em Sistemas de Informação e Bacharelado em Ciência da Computação, da Universidade Federal de Sergipe - *Campus* São Cristóvão.

São Cristóvão - SE
2025

SUMÁRIO

1. INTRODUÇÃO

O presente relatório tem como objetivo apresentar o processo de desenvolvimento do projeto "Labirinto" realizado pelos discentes do segundo período de Ciência da Computação - Guilherme Ferreira Amâncio, Pietro Oliveira Lima e José Marcos Bittencourt Oliveira Prado. Assim como pela discente do segundo período de Sistemas de Informação - Sibele Oliveira Cruz. O projeto "Labirinto" trata-se de um meio de avaliação para a disciplina de Programação Imperativa, que teve como objetivo a aplicação dos principais conceitos difundidos sobre programação em linguagem C, tais como sintaxe, tipos, usos de estruturas condicionais e de repetição, manipulação de arquivos, funções, ponteiros, e principalmente o uso de estruturas como structs (que foi bastante utilizada na construção do trabalho).

Perante a elaboração do programa, foram experimentados alguns reveses, os quais tentamos solucionar da melhor forma e seguindo os padrões exigidos. A próxima seção detalha melhor sobre as etapas de desenvolvimento, as decisões adotadas e os resultados obtidos.

2. DESENVOLVIMENTO

2.1. Primeira Etapa

A princípio foram criados os labirintos a serem posteriormente utilizados e gerado um esqueleto da função main, a qual se fixou como base para o restante do trabalho. A partir dessa estrutura base, iniciamos as tentativas de manipulação de arquivo, com isso foi inserido o texto do menu do jogo apresentado as funções a serem implementadas (1 - Resolver o labirinto com uma tentativa; 2 - Resolver o labirinto até obter sucesso; 3 - Salvar o labirinto resolvido em um arquivo; 4 - Sair do programa.).

Para a primeira entrega, referente a uma versão parcial do projeto, usamos um arquivo .txt para receber o labirinto, logo depois, passamos a receber da linha de comando e a salvá-lo num arquivo quando a opção 3 é selecionada. Além disso, criamos uma função para salvar o labirinto num arquivo .txt com o nome recebido como argumento na linha de comando. Entre os pontos irregulares relacionados à essa etapa estavam, por exemplo, que o código não recebia ou imprimia o labirinto com espaços entre os elementos, ademais também encontramos alguns bugs, tais como: ao receber um inteiro da linha de comando, o while acabava rodando infinitamente caso recebesse um caractere.

Em seguida, ao constatarmos esta falha, consertamos recebendo um caractere ao invés de um inteiro da linha de comando, com isso, passamos a receber e imprimir o labirinto com espaços, também fizemos uso do fscanf() para que o programa ignorasse os espaços entre os elementos no momento que o labirinto fosse armazenado.

2.2. Segunda Etapa

No que diz respeito à concepção do programa para atender aos requisitos da segunda etapa, foram criados os structs para melhor classificar e organizar os elementos do jogo, sendo eles: um para posição/vetor, um para o personagem e um para os demais elementos do labirinto. Posteriormente, foi finalizada a função de movimento, implementada a primeira versão da visualização por meio da função "sleep", pertencente a uma biblioteca não especificada que foi posteriormente substituída por outra disponível numa biblioteca padrão de C, atendendo assim ao que foi requisitado.

Em seguida, criamos os structs e usamos typedef para criar: vetor de posição na matriz do labirinto, personagem e elementos. Logo após criamos funções para preencher os elementos, elaboramos a função de combate a qual é baseada no sorteio de um número aleatório de 1 a 10 somado com a força do personagem que precisa ser maior que 10. A força começa em 5 e é incrementada (+1) cada combate vitorioso. Em seguida, a função de movimento aleatório para que assim permitisse a movimentação do personagem, primeira foi considerada uma geração de números aleatórios no intervalo de -5 a 5. Dessa forma, os valores negativos representavam um decréscimo de uma unidade nas posições x ou y, o valor zero indicava ausência de movimento e os valores positivos representavam um acréscimo. Contudo, essa abordagem permitia o deslocamento diagonal do personagem, o que não era desejado. Por conseguinte, fizemos uma adaptação porém mantivemos a ideia de utilizar números aleatórios, restringindo-os ao intervalo de 1 a 4, em que cada número passou a representar uma direção específica: cima, baixo, esquerda e direita. Além disso, foi implementado um sistema de verificação para garantir que a nova posição estivesse dentro dos limites válidos da matriz, evitando deslocamentos fora da área permitida.

Também foram feitas correções diante dos erros de compilação e mudança da função de movimentação e da função para lidar com o combate para um escopo mais global. Tratando-se da função para fazer com que a movimentação em conjunto com o combate funcione, a lógica seguida visa dois cenários principais: a presença de um inimigo na posição de destino e

a ausência dele. Primeiramente, é feita uma verificação para saber se a nova posição (posNova) contém um inimigo:

- Caso haja um inimigo presente, inicia-se um combate por meio da função combate(heroi). Se o personagem vencer a batalha, a função retorna "1" e a posição é marcada como já percorrida, sendo que aqui, o caractere '!' é usado para indicar que o personagem enfrentou um inimigo naquela posição e venceu. Em seguida, a posição do herói é atualizada para a nova posição e o movimento é considerado bem-sucedido. No entanto, se perder o combate, a derrota é informada e o caractere '+' marca a posição onde o personagem foi derrotado. Consequentemente, o labirinto é atualizado e impresso, e duas variáveis de controle são modificadas:
- Se ele perdeu: "1" (sinaliza que o personagem foi derrotado);
- Se ele andou: "1" (garante que o movimento foi processado, evitando que o sistema interprete a derrota como um erro de movimentação).

Por fim, caso não haja inimigo na nova posição, o código segue por outro caminho.
Logo após, notamos que a função imprimeLabirinto não condicionava a impressão dos demais caracteres além do "@" para contextos específicos referentes ao personagem, então, essa questão foi resolvida para que ocorresse a troca de caracteres com base na situação em que o personagem e ou inimigos se encontram. A seguir, elaboramos o Loop para corresponder a opção 2, na qual o jogo rodaria até o personagem resolver o labirinto. Contudo, uma dificuldade vivenciada nesse processo foi que no início, o personagem não estava voltando a sua posição original. Por último, tal reves quanto ao retorno do personagem para sua posição inicial a cada morte/perda, foi solucionado com criação de variáveis e funções que permitiram armazenar a posição inicial e replicá-la a cada processo de reinicialização.

3. CONCLUSÃO

O projeto "Labirinto" permitiu ao grupo uma melhor consolidação dos conhecimentos adquiridos durante a disciplina de Programação Imperativa, principalmente por termos trabalhado diretamente com a aplicação dos conceitos apresentados. Ainda assim, uma das maiores dificuldades vivenciadas no decorrer da elaboração do trabalho, estava relacionada ao fato de existirem limites muito específicos para o desenvolvimento, o que, por vezes, conteve uma maior liberdade criativa do grupo.