Trabalho Prático 02 - Jogo RPG

Objetivo: Desenvolver um mini-jogo de RPG (Role-Playing Game). Neste jogo, os jogadores Humano e Computador entram em combate em turnos e o resultado do combate é informado na tela. O jogo termina quando o Humano consegue chegar até o destino ou quando ele morre.
Valor: 20 pontos.

Entrega: Ao final desta prática, você deverá enviar no **SIGAA** da disciplina um arquivo **.zip** contendo os arquivos de extensão **.c** e com extensão **.exe** (resultados das compilações) **E** a Documentação (Seção 4) em **.pdf**. O trabalho pode ser feito em grupo de até **três integrantes**.
Atenção: Apenas um integrante deve submeter o trabalho.

Formato da Entrega:
Defina o nome do arquivo .zip como TP_02_Nome1_Nome2_Nome3.zip.
Atenção: SOMENTE serão aceitos submissões com extensão .zip

Seu código deve ser desenvolvido em C, estar bem organizado e indentado, possuir funções ou procedimentos (se necessário) e conter comentários relevantes. Cópias de trabalho são inaceitáveis, podem ser detectadas automaticamente e serão repassadas ao conselho disciplinar.

1 *Gameplay*

Um RPG pode contar com diversos elementos de gameplayer, como puzzles, gerenciamento de inventário, combate, potion craft, etc. Neste mini-jogo de RPG iremos concentrar apenas no elemento de combate. Além disso, o combate será em turnos. Ou seja, em cada turno, um jogador tem direito a executar uma das duas ações: ataque ou medicar. A arma utilizada no ataque será uma espada.

No ataque, existem 3 opções:

1. Slash (Golpe horizontal/diagonal);
2. Strike (Golpe vertical);
3. Trust (Golpe com a ponta da espada);

A cada ataque, o jogador pode escolher qual é o tipo de defesa que pode anular o ataque:]

1. Bloqueio anula Slash;
2. Esquirvar anula Strike;
3. Desviar anular Trust;

Antes de iniciar o jogo os jogadores começam com health points (pontos de vida) e com 1 poção de medicar. Se o jogador escolher se medicar, então ele não de defende durante o ataque. Ganha o jogo quem matar o adversário primeiro :)

2 Implementação

O jogo deve ler as informações dos inimigos de um arquivo. Cada linha desse arquivo deve conter, no mínimo, as seguintes informações: nome;ataque;pontos_de_vida_maximo

Defina pelo menos 3 inimigos com características distintas.

Após ler uma linha, seu programa deve armazenar em uma struct que representa um inimigo. Seu jogo também deve ter uma struct que representa o Jogador. As informações dele devem ser lidas do teclado.

2.1 Leitura das Informações do Jogador

O programa deve ler as informações do Jogador pelo teclado e armazenar em uma struct. A função deve ter o seguinte protótipo:

```
1 Jogador lerJogador();
```

2.2 Carregamento das Informações de Inimigos

O programa deve ter uma função chamada carregarInimigos, que deve ler o arquivo dos inimigos e armazenar em um vetor de inimigos. Note que o inimigo deve ser definido utilizando uma struct. A função deve ter o seguinte protótipo:

```
1 int carregarInimigos(char arquivo[], int n, Inimigo inimigos[QUANTIDADE_INIMIGOS]);
```

A função retorna 0 (zero) caso o carregamento dos arquivos sejam feito com sucesso e 1 (um) caso ocorra um erro de carregamento. Se houver um erro o programa deve parar a execução. Considere que QUANTIDADE_INIMIGOS é uma constante com valor padrão igual a 3 (número de inimigos mínimo).

2.3 Combate

O jogo deve escolher aleatoriamente um inimigo e iniciar o combate. Você pode colocar os combates em sequência ou então deixar o jogador decidir se vai continuar o percurso, aplicar *heal*, etc. O combate deve ser em turnos e envolver o conceito de ataque e defesa. A cada turno um dos jogadores ataca e o outro se defende. O atacante pode escolher um tipo de ataque e o defensor pode escolher um tipo de defesa. Algumas defesas anulam o ataque, enquanto que outras são ineficazes. Na Seção 1 são detalhados os tipos de ataque e defesa.

2.4 Salvar informações

Após o jogo terminar seja por que o jogador perdeu ou ganhou, as informações da partida devem ser salvas em um arquivo. Isto é, nome do jogador, *health points* quando finalizou partida e se ganhou ou perdeu. Esse arquivo deve ser salvo no formato *append*.

3 Funcionalidades Adicionais

Abaixo está uma lista de desafios de implementação de funcionalidades adicionais que valerão, no total, 5 pontos extras. Todas as funcionalidades implementadas devem estar no mesmo programa. **Dica:** reservem uma versão da solução básica antes de se aventurarem nos desafios.

- **Desafio 1:** O jogador pode ver na tela as informações de ranking de pontuações. (1 ponto);
- **Desafio 2:** Criar uma *lore* para o jogo que dê sentido ao combate, *gameplay*, tipos de inimigos, etc. (1 ponto);
- **Desafio 3:** O jogador pode decidir prosseguir ou fazer *heal* após um combate terminar. O jogador deve ter uma quantidade limitada de poções ou comidas para heal em seu inventário. Os valores do inventário devem ser mostrados na tela (2 pontos);
- **Desafio 4:** Quando um inimigo morrer ele **pode ou não** "dropar" itens de *heal* que podem ser coletados pelo Jogador (2 pontos).

4 Documentação

O texto da documentação deve ser breve, de forma que, eu (professor) possa entender o que foi feito no código sem ter que entender linha a linha do programa. Implementações modularizadas deverão mencionar quais lógicas são implementadas em cada módulo (função). A documentação deve conter, no mínimo, os seguintes itens:

- Nomes dos integrantes e a contribuição de cada um;
- Uma descrição dos algoritmos, das principais funções, e as decisões de implementação;
- Decisões de implementação (ou considerações) que porventura estejam omissos na especificação do trabalho;
- Testes, mostrando que o programa está funcionando de acordo com a especificação, seguidos da sua análise. *Print screens* mostrando o correto funcionamento do programa e exemplos de testes executados;

O arquivo da documentação deve estar em **.pdf**. Não serão aceitos outros formatos.

5 Critérios de avaliação

Os trabalhos serão avaliados de acordo com os critérios:

- Implementação Correta [0 a 8 pontos].
- Documentação [0 a 4 pontos].
- Interface (Usabilidade, estética, diversão, etc) [0 a 5 pontos];
- Tratamento de erros e Organização do código [0 a 3 pontos].
- Desafios terão nota 0, 25%, 50%, 75% ou 100% dos pontos extras de acordo com o funcionamento e adequação ao restante do código.
- **Cópias terão nota zero!**
- Trabalhos que não compilarem terão notas entre 0 e 5.
- Caso necessário, haverá uma entrevista com os integrantes do grupo.