TEXT RPG

O Text RPG consiste em um jogo no qual o player comanda praticamente tudo que ele deseja realizar durante o jogo. Ele já começa recebendo uma lista com as 8 classes que o jogo possui e irá escolher 4 dentre todas elas para compor o seu time que enfrentará uma torre com 5 andares de lutas, além de dar um nome personalizado para cada um destes membros da sua equipe. Logo após terminar sua composição, ele receberá 3 pontos de atributos em cada personagem para distribuir entre Força, Destreza, Inteligência e Constituição e se preparar para as batalhas que virão.

Começando a batalha da torre, o player já pode ser surpreendido com um ataque dos inimigos caso a velocidade total do seu time seja menor que a dos mesmos, ou irá surpreendê-los caso contrário, quando for seu turno, o jogador poderá informar qual ação da lista ele deseja realizar e em qual alvo ele quer direcionar o mesmo. Os danos recebidos e causados serão impressos na tela, assim como uma atualização em tempo real das informações da equipe do player, o desafio está em não saber as informações dos inimigos que está enfrentando. Ao fim de cada batalha você receberá mais 3 pontos de atributos para distribuir e se preparar para os desafios seguintes até vencer o quinto e último andar.

* **Jogo.h:**

Classe principal de nosso jogo, contém objetos das classes de equipe jogadora (**Equipe.h**) e das equipes inimigas (**EquipeInimiga.h**), bem como métodos que retornam informações dessas equipes. Também possui métodos que definem eventos básicos do jogo, tais como seu início, as batalhas, tratamentos de exceção, final do jogo, etc. A classe possui os seguintes métodos:

* **Construtor**

Ao instanciar um objeto da Classe Jogo, este irá instanciar todos os inimigos dos 5 andares da torre a qual o jogador irá percorrer, cada um com uma equipe inimiga, composta por 4 inimigos, como se segue:

Andar 1: Possui 4 Slimes inimigos.

Andar 2: Possui 4 Skulls inimigos.

Andar 3: Possui 4 Trolls inimigos.

Andar 4: Possui 1 Slime, 1 Skull, 1 Troll e 1 Witcher inimigos.

Andar 5: Possui 1 Skull, 2 Trolls e 1 Witcher inimigos.

* **iniciar()**

O método **iniciar()**, através de mensagens de texto na tela, da boas vindas ao jogador e, após isso, apresenta uma tabela que apresenta ao jogador as 8 classes de personagens disponíveis em nosso jogo: Arqueiro, Bardo, Bárbaro, Guerreiro, Ladino, Mago, Paladino e Sacerdote, bem como seus respectivos atributos base (Ataque, Defesa, Vida, Mana, Dano Mágico, Taxa de Acerto e Velocidade). Em seguida, pede para que o jogador crie 4 personagens baseados nas classes aos quais foi apresentado, bem como pede para que atribua um nome a cada um deles. Feito isso, uma tabela com os dados dos personagens criados pelo jogador é impressa na tela para fins de consulta.

* **pre\_batalha()**

O método **pre\_batalha()** da oportunidade para que o jogador melhore um atributo de sua escolha (Dentre Força, Destreza, Constituição e Inteligência) para cada um dos seus 4 personagens criados.

* **batalha():**

O método **batalha()** inicializa as batalhas que ocorrem no jogo, as ordenando por turnos, alternando entre a vez do jogador e a vez do inimigo. Em cada turno é pedido ao jogador que escolha uma ação (via método **acao()**), após isso é apurado as consequências dessa ação. Passado esse turno, passa para a vez do inimigo, cujo alvo do time do jogador é escolhido aleatoriamente, após isso executa um ataque (via método **ataque\_inimigo()**), depois disso se apura as consequências desse ataque. O turno volta para o próximo personagem da equipe do jogador, alternando-se entre os inimigos e os personagens do jogador que ainda não tiveram vez, e sendo essa dinâmica repetida até que uma das equipes tenham todos seus indivíduos mortos.

* **acao(int numero, EquipeInimiga &equipe\_torre)**

O método **acao()** recebe dois argumentos, sendo o primeiro o número referente ao personagem o qual é a vez de se realizar aquela ação, e o segundo, o objeto da classe referente a equipe inimiga (**EquipeInimiga.h**). o método imprime na tela as opções possíveis de ação que o jogador pode executar naquele turno e aguarda até que o jogador faça a escolha desejada, seguido de outra mensagem, que pergunta a qual personagem inimigo ou aliado o jogador deseja direcionar aquela ação, novamente é aguardado até que o jogador digite sua escolha. O método segue com o executando algoritmo adequado, a depender da ação que foi escolhida pelo jogador e o tipo de personagem que está a executando, tais algoritmos frequentemente fazem acessos a atributos de personagens do jogador e inimigos, bem como métodos que implementam ataque, defesa e habilidades desses personagens.

* **ataque\_inimigo(EquipeInimiga &equipe\_torre, int posicao\_inimigo, int posicao\_jogador)**

O método **ataque\_inimigo()** recebe três argumentos, o primeiro recebe o objeto referente a classe da equipe inimiga (**EquipeInimiga.h**), o segundo é um número inteiro referente a qual inimigo esta realizando o ataque, o terceiro é um número inteiro referente ao personagem da equipe do jogador que irá receber esse ataque. O método utiliza de um algoritmo que calcula e aplica o dano realizado pelo inimigo, a depender de atributos do inimigo, atributos do personagem, e ambos sendo utilizados em conjunção de números aleatórios visando dar variedade ao resultado do ataque. O método imprime ao usuário o resultado do sucesso ou falha do ataque, bem como a quantia de dano causado caso o ataque tenha sido bem sucedido.

* **ordena\_batalha(EquipeInimiga &equipe\_torre)**

O método **ordena\_batalha()** recebe como argumento um objeto da classe da equipe inimiga (**EquipeInimiga.h**). Esse método é utilizado pelo método **batalha()**, e retorna um valor inteiro 1 ou 2. Esse valor é utilizado pelo método **batalha()** para determinar qual equipe, jogadora ou inimiga, irá jogar o primeiro turno. O método toma como base o valor de velocidade dos personagens da equipe jogadora e da equipe inimiga, aquela cujo somatório de todas velocidades for maior ganha o primeiro turno.

* **informacoes\_jogador()**

O método **informacoes\_jogador()**  imprime na tela informações sobre atributos e estado de cada personagem da equipe jogadora, para que o jogador acompanhe o estado de sua equipe durante sua jornada e durante as batalhas.

* **informacoes\_inimigos(EquipeInimiga equipe\_torre)**

O método **informacoes\_inimigos()** recebe um objeto da classe referente a equipe inimiga (**EquipeInimiga.h**). É análogo ao método **informacoes\_jogador()**, imprimindo na tela informações sobre atributos e estado de cada personagem da equipe inimiga, para que o jogador acompanhe o estado da equipe inimiga durante as batalhas.

* **EquipeInimiga.h**

Classe que define a equipe de inimigos do nosso jogo, possui os seguintes atributos:

* **\_tamanho:** Valor inteiro referente ao tamanho do vetor que comporta os personagens da equipe inimiga (**\_equipe\_inimiga**).
* **\_equipe\_inimiga:** Vetor de Inimigos(**Inimigos.h**), do tipo vector, cada posição comporta um objeto da classe **Inimigos**, referente aos personagens da equipe inimiga.

A classe possui os seguintes métodos:

* **Construtor:**

O construtor da classe inicializa o vetor **\_equipe\_inimiga**, atribuindo a cada uma das 4 primeiras posições um objeto do tipo **Inimigo**, bem como atribui à variável **\_tamanho** o tamanho do vetor.

* **get\_tamanho\_equipe\_inimiga()**

O método **get\_tamanho\_equipe\_inimiga()** retorna o valor do atributo **\_tamanho** da classe.

* **get\_inimigo(int ordem)**

O método **get\_inimigo()** recebe como parâmetro um valor inteiro referente a posição desejada, sua função é retornar o um ponteiro que aponta para a posição do vetor de **\_equipe\_inimiga** referente ao valor inteiro recebida como parâmetro.

* **IfInimigo\_morrer()**

O método **IfInimigo\_morrer()** remove do vetor de inimigos **\_equipe\_inimiga** o objeto referente ao indivíduo inimigo que morra numa batalha, fazendo a modificação necessária no atributo **\_tamanho**.

* **Equipe.h**

Classe análoga a classe **Inimigo.h**, definindo por sua vez a equipe do jogador, possuindo os seguintes atributos:

* **\_tamanho:** Valor inteiro referente ao tamanho do vetor que comporta os personagens da equipe jogadora (**\_equipe**).
* **\_equipe:** Vetor de Personagem(**Personagem.h**), do tipo vector, cada posição comporta um objeto da classe **Personagem**, referente aos personagens da equipe jogadora.

A classe possui os seguintes métodos:

* **Construtor 1: Equipe(int personagem1,std::string nome1,int personagem2,std::string nome2,int personagem3,std::string nome3,int personagem4,std::string nome4)**

Um dos construtores da classe que inicializa o vetor de equipe do jogador (**\_equipe**), utilizando para cada posição do vetor o método **cria\_personagem** que será explicado abaixo, bem como registra no atributo **\_tamanho** o tamanho do vetor. Recebe como parâmetros um valor inteiro referente ao tipo de personagem que se deseja criar, segundo de um valor string com o nome que se deseja atribuir ao personagem criado, repetido desse mesmo par de atributos por mais três vezes, sendo que cada par será utilizado na criação de um personagem, totalizando quatro.

* **Construtor 2: Equipe(const Equipe& eq)**

Sendo uma classe polimórfica, o segundo construtor da classe recebe como atributo apenas um objeto da classe **Equipe.** Esse construtor refaz o vetor de personagens do jogador (**\_equipe**), utilizando-se do método **realoca()** que será explicado abaixo.

* **Destrutor**

O destrutor da classe desaloca os personagens criados alocados dinamicamente pelo construtor 1.

* **cria\_personagem(int personagem, std::string nome)**

O método **cria\_personagem()** possui dois parâmetros, o primeiro um valor inteiro que indica o tipo de personagem que se deseja criar, o segundo um valor string que se refere ao nome que será atribuído a esse personagem. O método cria o personagem com as características de tipo e nome recebidas como parâmetro, alocando dinamicamente um objeto do tipo de personagem escolhido, em seguida insere o objeto criado no vetor de equipe do jogador (**\_equipe**).

* **realoca(const Equipe& eq)**

O método **realoca()** possui como parâmetro único um objeto da classe Equipe, sua função é fazer a realocação de personagens dentro do vetor **\_equipe**, já que assim que um personagem morre ele é removido do vetor, evitando assim possíveis erros de execução do jogo durante seu funcionamento.

* **Equipe& operator=(const Equipe &eq)**

O método **operator()** possui como parâmetro um objeto da classe **Equipe** passado por referência. Ele realiza, através da sobrecarga do operador “=”, a realocação do vetor de equipe do objeto passado como parâmetro, utilizando-se do método **realoca()**, afim de se evitar erros de execução do programa. Possui função semelhante a do construtor 2.

* **get\_tamanho\_equipe ()**

O método **get\_tamanho\_equipe ()** retorna o valor do atributo **\_tamanho** da classe.

* **get\_personagem(int ordem)**

O método **get\_personagem()** recebe como parâmetro um valor inteiro referente a posição desejada, sua função é retornar o um ponteiro que aponta para a posição do vetor de **\_equipe** referente ao valor inteiro recebida como parâmetro.

* **IfPersonagem\_morrer()**

O método **IfPersonagem\_morrer()** remove do vetor de inimigos **\_equipe** o objeto referente ao indivíduo da equipe jogadora que morra numa batalha, fazendo a modificação necessária no atributo **\_tamanho**.