A lojinha

Edgar Sampaio de Barros, 16/0005213
Grupo 12

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
Técnicas de programação 1

160005213@aluno.unb.br

1. Descrição do problema

O intuito do projeto desenvolvido e relatado neste trabalho é a criação de um jogo com temática RPG *roguelike*. *Roguelike* é um gênero de RPG onde a geração de níveis é aleatória e a morte é permanente, ou seja, caso o jogador morra no jogo ele deve começar outra partida e como os níveis são gerados aleatoriamente essa nova partida será totalmente diferente da anterior.

2. Regras de negócio

Como se trata de um jogo *roguelike*, com exceção de alguns dados dos personagens que serão inseridos pelo jogador, a maioria dos valores e até mesmo o nome das lojas serão gerados de forma aleatória para que assim seja praticamente impossível jogar duas partidas iguais. O objetivo do jogo é chegar no nível 10, cada nível será composto por três turnos de combate, onde o jogador deve usar as habilidades de seus personagens para derrotar os monstros que deixam cair moedas de ouro quando derrotados. Ao final de cada fase haverá uma loja que o jogador poderá gastar suas moedas em troca de itens que podem aumentar a força de seus personagens.

3. Diagrama de classe

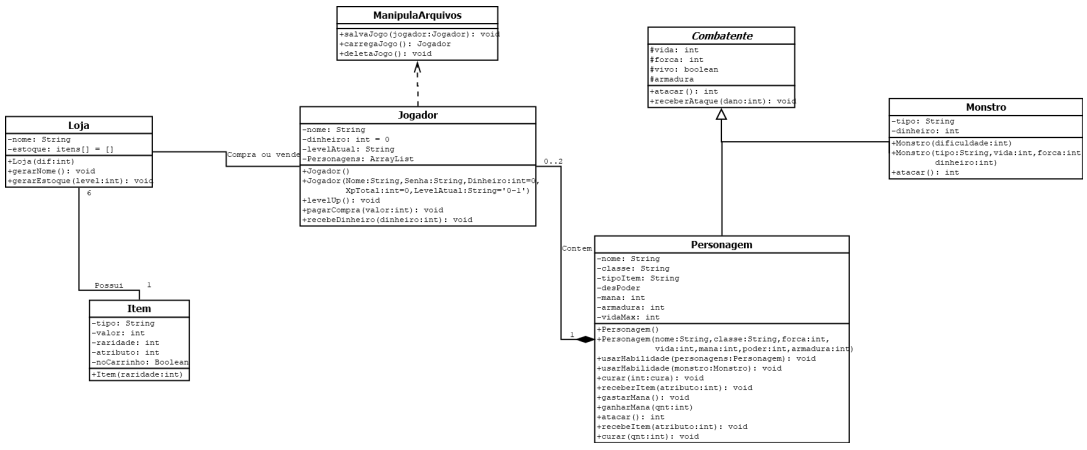

Figure 1. Diagrama de classes.

4. Classes

4.1. Classe Loja

A classe loja é responsável por gerar um estoque de itens aleatórios e armazenar para que o jogador possa realizar ações de compra. Tem como atributos uma String para o nome da

loja e um vetor de 6 posições de itens. O nome da loja é acompanhado da frase de efeito do vendedor e serve para personalizar a tela da loja. O método construtor abaixo é gerado com base na dificuldade dada como argumento, uma loja com todos os atributos gerados aleatoriamente.

```
    public Loja(int dif) {
        Random rand = new Random();
        String []nomes = {"Alfredo,mate os monstros que dao medo",
                          "Juvenal,compra ae namoral",
                          "Edgar,me da uns ponto pra eu passar",
                          "Creuza,pode vim que o estoque ta uma
beleza"};
        this.nome = nomes[rand.nextInt(nomes.length)];

        for (int i = 0; i < 6; i++)
            estoque[i] = new Item(dif);
    }
```

Listing 1. Classe loja

4.2. Classe Item

A classe item contém cinco atributos e além dos get's e set's apenas um método que é seu construtor pode ser visto na parte de código abaixo. O construtor recebe como parâmetro um inteiro referente a raridade do item e com base nele gera todos os atributos da classe de forma aleatória.

```
    public Item(int raridade) {
        Random rand = new Random();
        String []tipos = {"varinha", "espada", "arco"};

        valor = 2*(rand.nextInt(raridade) + raridade) + 1;
        tipo = tipos[rand.nextInt(tipos.length)];
        this.raridade = raridade;
        this.atributo = (raridade < 3 ? rand.nextInt(raridade+1) +1:
rand.nextInt(3) + (raridade+1)/2);
        this.noCarrinho = false;
    }
```

Listing 2. Classe Item

4.3. Classe Jogador

A classe jogador é onde fica armazenado as principais informações do jogo, nela contém o nome do jogador, o dinheiro e o level atual do jogo e um ArrayList dos personagens do jogador. Os métodos da classe são levelUp que é responsável por alterar os valores quando o personagem sobe de nível e é chamado sempre no final do level, pagarCompra e receberDinheiro que servem para manipular o atributo dinheiro de acordo com a situação, a seguir podemos ver a implementação do método levelUp.

```
    public void levelUp() {
        this.levelAtual++;
        for (Personagem personagem : personagens) {
            personagem.setVivo(true);
            personagem.setVidaMax(personagem.getVidaMax() + 5);
            personagem.curar(5);
```

```
            personagem.ganharMana(1);
        }
    }
```

Listing 3. Classe Jogador

4.4. Classe Combatente

A classe combatente é uma classe abstrata que serve como modelo para as classes Monstro e Personagem. Tem como objetivo implementar os atributos e métodos necessários para ocorrer o combate do jogo. Os métodos podem ser vistos na parte do código abaixo.

```
    public abstract int atacar();

    public void receberAtaque(int dano){
        if (dano > armadura/2)
            this.vida -= dano-armadura/2;

        if(this.vida <= 0){
            this.vivo = false;
            this.vida = 0;
        }
    }
```

Listing 4. Classe abstrata Combatente

4.5. Classe Monstro

A classe monstro é derivada da classe combatente com adição dos atributos tipo e dinheiro. O atributo tipo serve para a interface gráfica saber qual imagem deve colocar na tela de combate para ilustrar o monstro e o dinheiro é a quantidade de moedas de ouro que o monstro deixará para os personagens caso seja derrotado. O método construtor abaixo gera um monstro aleatório com base na dificuldade que é dada como parâmetro.

```
    public Monstro(int difi) { // gera um monstro aleatorio a partir da
    dificuldade dada
        String []tipos = {"esqueleto", "orc", "slime"};
        tipo = tipos[rand.nextInt(tipos.length)];

        this.forca = difi;
        this.dinheiro = 1*difi/2 + 5*(rand.nextInt(difi+1) + difi+1)/4;
        this.armadura = 3*difi/2;

        if(difi == 1)
            difi = 2;
        this.vida = Integer.max(difi + 5*(difi+rand.nextInt(difi+1)),
    difi + 5*(difi+rand.nextInt(difi+1))); // sorteia duas vezes e pega
    o maior

        this.vivo = true;
    }
```

Listing 5. Classe Monstro

4.6. Classe Personagem

A classe personagem também é derivada da classe abstrata combatente. Os atributos dela servem tanto para a customização da interface gráfica (nome, classe e descPoder - descrição do poder) quanto para ações durante o combate do jogo (mana, armadura e vidMax) o atributo tipoItem serve para saber se o personagem pode ou não receber um item comprado na lojinha. Nesta classe foi feito o uso do polimorfismo para a implementação da habilidade dos personagens, os personagens do classe mago podem curar os aliados, portanto o método recebe como parâmetro o personagem que a ser curado, já os personagens das classes guerreiro e arqueiro tem como habilidade realizar um ataque diferente, portanto o parâmetro do método é um monstro que irá receber o ataque, a seguir é possível ver o código implementado.

```
1    public boolean usarHabilidade(Personagem personagem){// habilidade
     do mago, curar
2        if(this.mana > 0){
3            personagem.curar(this.poder);
4
5            this.gastarMana();
6            return true;
7        }
8        return false;
9    }
10
11   public boolean usarHabilidade(Monstro monstro){// habilidade do
     arqueiro e do guerreiro, dar dois ataques ou um ataque carregado
12       if(this.mana > 0){
13           if(this.classe.equals("arqueiro"))
14               monstro.receberAtaque(this.atacar() + (this.poder+this.
     atacar())/2);
15           else
16               monstro.receberAtaque(poder+4*this.atacar()/3);
17           this.gastarMana();
18           return true;
19       }
20       return false;
21   }
```

Listing 6. Classe Personagem

4.7. Classe ManipulaArquivos

A classe ManipulaArquivos não tem atributos, ela possui apenas 3 métodos, sendo eles: salvaJogo, deletaJogo e carregaJogo que servem para manipular o arquivo que salva as informações do jogo. O método salvaJogo cria ou edita o arquivo save.dat com as informações do estado atual da classe Jogador e é chamado sempre que termina um nível, já o método deletaJogo serve para deletar o arquivo, é usado quando o jogador termina o jogo e por fim o método carregaJogo tem como função carregar os valores escritos no arquivo save.dat em um objeto da classe jogador caso o botão continuar no menu inicial seja pressionado.

5. Telas

A seguir podemos ver todas as telas do programa e algumas de suas variações. Durante o decorrer da execução do programa aparecem algumas mensagens para o usuário,

por exemplo, quando o botão novo jogo na tela inicial é pressionado aparece uma caixa de diálogo perguntando ao jogador seu nome, outro exemplo é quando o jogador tenta usar uma habilidade do personagem que esta sem mana ou quando o seu saldo de moedas é inferior ao valor da compra que está tentando realizar, em ambos os casos aparece um aviso dizendo que tal ação não pode ser executada no momento.

Figure 2. Tela inicial.

Figure 3. Tela de créditos.

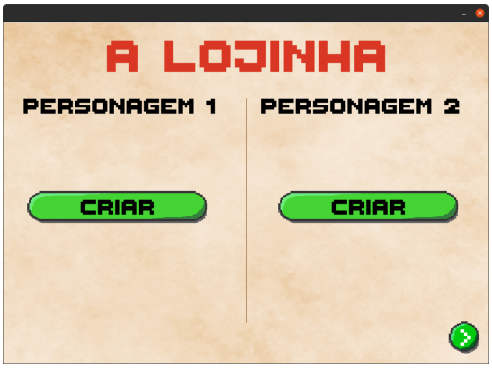

Figure 4. Tela de de personagens vazia.

Figure 5. Tela de de personagens com apenas um personagem criado.

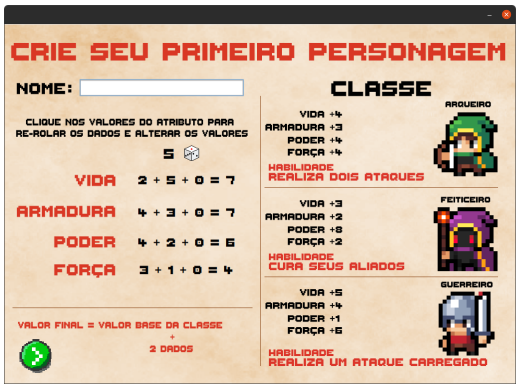

Figure 6. Tela de criação de personagem.

Figure 7. Tela da lojinha.

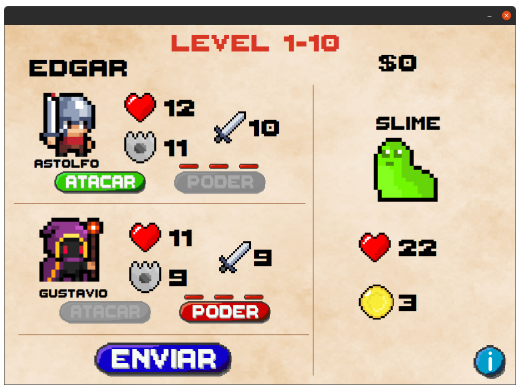

Figure 8. Tela de combate.

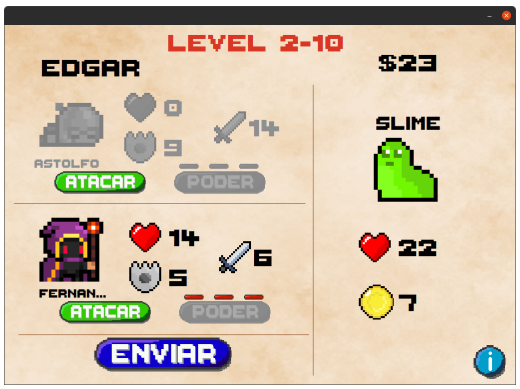

Figure 9. Tela de combate com personagem morto.

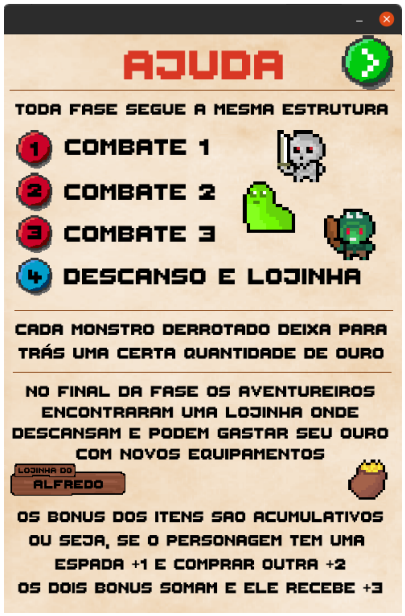

Figure 10. Tela de ajuda 1.

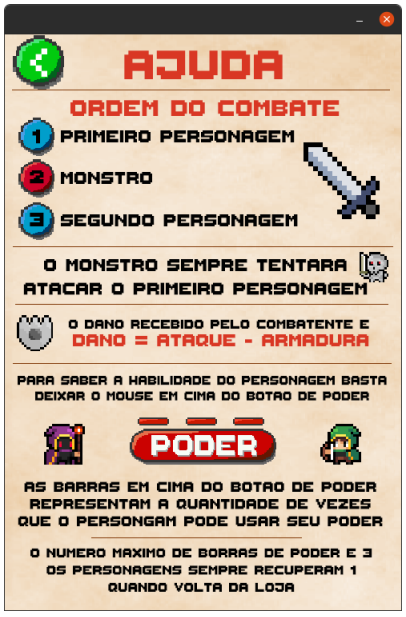

Figure 11. Tela de ajuda 2.

Figure 12. Tela de vitória.

Figure 13. Tela de derrota.