Universidade Federa do Espírito Santo – UFES

Centro Universitário do Norte do Estado – CEUNES

Caio Vianna Rizzo

Conceitos de RPG Aplicados em Prolog

São Mateus

2016

Caio Vianna Rizzo

Conceitos de RPG Aplicados em Prolog

Trabalho da disciplina Lógica 2

Prof. Luís O. Rigo Jr.

São Mateus

2016

Objetivo

O principal objetivo deste trabalho é criar um guia de batalha, baseado em um jogo MMORPG (Massive Multiplayer Online Role Playing Game) chamado Tree of Savior, com as principais mecânicas de dano, forças e fraquezas implementadas, de forma a auxiliar jogadores na escolha de monstros, chefes (Bosses) e dungeons, as quais ele leve vantagem pela sua build de skills ou equipamentos utilizados. Para tal foi utilizado a linguagem de programação Prolog, com o IDE SWI-Prolog para Windows.

O universo retratado

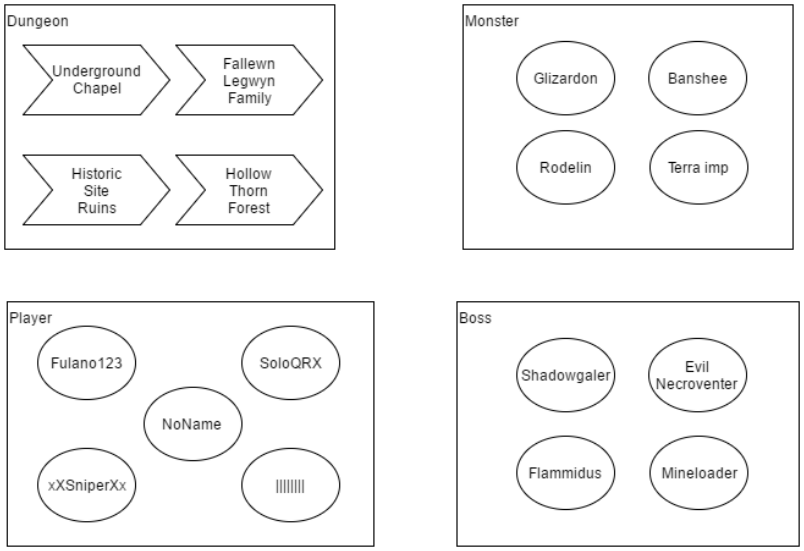
O jogo possui uma diversidade imensa de classes, portanto apenas algumas delas foram representadas, alem de algumas mecânicas de dano terem sido ajustados para melhor verificação no Prolog.

As principais forças e fraquezas presentes no jogo estão representadas nas tabelas abaixo, sendo a primeira relacionada a tipos de dano e armaduras e a segunda é entre elementos (fogo, gelo, raio, etc...).

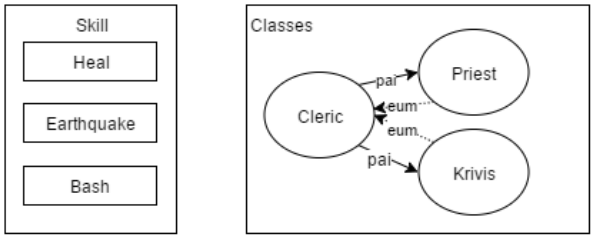
Figura 1: Tabela Dano/Armadura

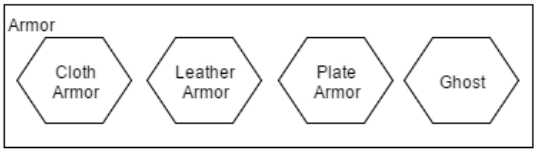
Figura 2: Tabela Elementos

As classes principais que compõem o universo do jogo são: player, a qual todos os nomes dos players do jogo são registrados; monster, que define os monstros que habitam as dungeons; boss, os chefes que ficam ao final das dungeons; dungeons, são mapas fechados que são repletos de monstros e possuem um chefe no final.



Cada player que começa a jogar deve escolher uma das classes principais, as que foram representadas neste programa são: Cleric, Swordsman e Wizard. A partir dessas classes iniciais ele avança para classes mais avançadas. Todas as classes possuem habilidades (skills), os quais possuem diversos tipos de dano (magic, slash, pierce, strike) e estes danos tem forças e fraquesas dependendo da armadura ou do elemento principal do adversário.





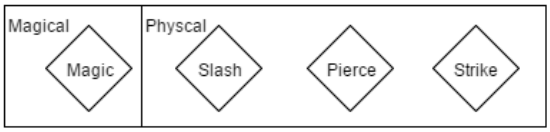
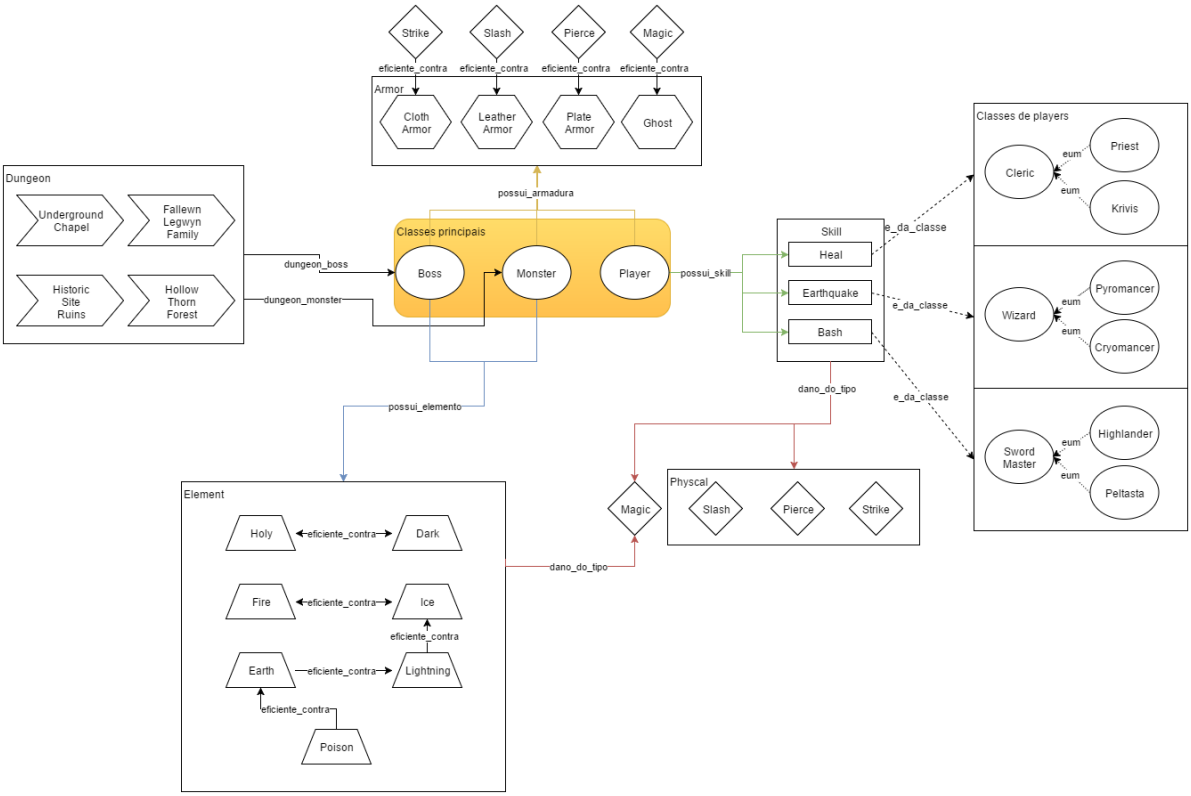


Diagrama Completo



Principais Classes

Classes Unárias:

player(PLAYER). – Classe unária que classifica e nomeia os players registrados;

dano(DANO). – Classe que lista os tipos de dano vistos na tabela acima;

element(ELEMENT). – Classe que lista os tipos de elementos vistos na tabela acima;

magical(X) :- (X = magical); element(X).– Agrupa os tipos de dano que são mágicos.

physical(X) :- dano(X), (X \== magic).– Agrupa os tipos de dano que são físicos;

tipo_armadura(ARMADURA). – Classe que lista os tipos de armadura;

skill(X) :- e_da_classe(X,_). – Utiliza uma classe binária que relaciona classe e skill, para definir os skills;

dungeon(X) :- dungeon_boss(X,_). – Classe que utiliza uma classe binária para classificar as dungeons;

boss(X) :- dungeon_boss(_,X). – Classe que utiliza uma classe binária para classificar os bosses das dungeons;

monster(M) :- monster-specs(M,_,_,_). – Classe binária que utiliza uma quartenária para especificar o nome do monstro;

Classes Binárias:

e_da_classe(SKILL,CLASSE). – Classe binária que relaciona os skills com as classes que os possuem;

e_um(CLASSE1,CLASSE2). – Predicado que identifica as descendências entre as classes de players.

dano_do_tipo(E,magic) :- element(E). – Classe que classifica os danos elementais como mágicos;

dano_do_tipo(SKILL,DANO). – Classe que relaciona os skills com os tipo de dano que eles causam;

dungeon_boss(DUNGEON,BOSS). – Classe que une as classes dungeon e boss;

possui_elemento(BOSS,ELEMENTO). – Classe que relaciona os chefes (Boss) com seus elementos;

dungeon_monster(D,M) :- monster_specs(M,_,_,D). – Classe que identifica a dungeon a qual um determinado monstro pertence;

possui_elemento(M,E) :- monster_specs(M,_,E,_). – Identifica qual o elemento do monstro;

possui_skill(PLAYER,SKILL). – Classe que relaciona os players e seus skills;

possui_armadura(PLAYER | BOSS | MONSTER, ARMADURA). – Classe que relaciona players ou boss ou monstros com as armaduras que possuem;

eficiente_contra(DANO, ARMADURA). – Classe que define se um tipo de dano é eficiente contra alguma das armaduras do jogo;

eficiente_contra(ELEMENTO,ELEMENTO). – Variação da classe acima que relaciona a eficiência de um tipo de elemento contra outro;

elementos_equivalentes(E1,E2) – Define os elementos que não possuem desvantagens entre si (holy, dark | ice, fire).

ineficiente_contra(DANO,ARMADURA) –Classe que define e um tipo de dano é ineficiente contra alguma das armaduras do jogo;

ineficiente_contra(E,E) :- element(E), (E \== dark), (E \== holy). – Variação da classe acima que representa que os danos elementais iguais (exceto dark,holy) são ineficientes.

ineficiente_contra(E1,E2) :- element(E1), element(E2), eficiente_contra(E2,E1), not(elementos_equivalentes(E1,E2)), (E1 \== ice). – Outra variação da classe acima que representa os elementos

Classe Quartenária:

monster_specs(NOME,ARMADURA,ELEMENTO,DUNGEON). – Define as principais características dos monstros.

Regras derivadas:

e_um(X,Y) :- player(X), classe(Y), skill(Z), e_da_classe(Z,Y), possui_skill(X,Z). – Regra que define qual a classe (classe de player) do player em questão;

build(P,S) :- player(P), possui_skill(P,S). – Regra que relaciona os players e seus skills;

causa_dano(P,D) :- build(P,S), dano_do_tipo(S,D). – Regra que une as regras de dano e de build para definir os tipos de dano que um determinado player pode causar;

vantagem_sobre(P,Y) :- causa_dano(P,D), tipo_armadura(A), possui_armadura(Y,A), eficiente_contra(D,A), !. – Classe que define sobre quais players, bosses ou monstros um determinado player tem vantagem, se baseando no dano causado pelo player e a relação deste com o tipo de armadura do adversário;

vantagem_sobre(P,Y) :- build(P,S), element(E), element(E2), possui_elemento(Y,E), dano_do_tipo(S,E2), eficiente_contra(E,E2),!. – Variação da classe vantagem_sobre, que define a vantagem de um player se baseando no dano elemental deste player e do elemento de seu adversário;

desvantagem_sobre(P,B) :- not(vantagem_sobre(P,B)), causa_dano(P,D), possui_armadura(B,A), ineficiente_contra(D,A), !. – A regra oposta da regra de vantagem, identifica as desvantagens de dano e armadura do player para com os inimigos;

desvantagem_sobre(P,B) :- not(vantagem_sobre(P,B)), causa_dano(P,E), element(E), possui_elemento(B,E2), ineficiente_contra(E,E2), !. – Variação da regra anterior, identifica as desvantagens elementais do player.

e_um(X,Y) :- player(X), classe(Y), skill(Z), e_da_classe(Z,Y), possui_skill(X,Z). – Regra que define quais as classes que um determinado player possui, baseando-se em seus skills.

melhores_dungeons(P,D) :- dungeon(D), boss(B), dungeon_boss(D,B), vantagem_sobre(P,B). – Regra que define as melhores dungeons para um player de acordo com suas habilidades específicas (skills).

piores_dungeons(P,D) :- dungeon(D), boss(B), dungeon_boss(D,B), desvantagem_sobre(P,B). – Oposta a regra das melhores dungeons, esta regra dita as piores dungeons para um determinado player.

Descrição dos pontos positivos e negativos

Pontos Positivos:

- O Prolog se mostrou uma linguagem simples de ser utilizada e de fácil aprendizagem.
- O tema escolhido possuía um conteúdo bem extenso e diversificado, por se tratar de um jogo real, facilitando ainda mais na procura de conteúdo, principalmente por existirem diversos sites dedicados ao jogo.
- Rapidez na exibição dos resultados, além da listagem das possíveis alternativas para o predicado.

Pontos Negativos:

- O tema necessita de muito conteúdo declarado para responder perguntas relativamente simples.
- O pouco tempo de contato com a linguagem fez com que certos predicados não fossem escritos da forma mais otimizada possível.
- Muito conteúdo do jogo teve de ser deixado de lado pelo fato de que seriam necessárias uma quantidade enorme de declarações para que pudessem ser aplicadas regras sobre eles.

Referencias

http://www.tosbase.com/

http://www.tradeofsavior.com/dungeons-bosses-loot-tree-of-savior/