ENGENHARIA INFORMÁTICA

PROGRAMAÇÃO DECLARATIVA

2015/2016

Discente:

André Figueira, 31626

Docente:

Salvador Abreu

1. Introdução

Este trabalho consiste em implementar um simulador de RPG, para tal vão ser aplicados todos os conhecimentos adquiridos ao longo do semestre de forma a que se consiga atingir o objectivo da melhor maneira possível. Para além do conjunto base, nota-se também a adição de extensões, e extensões adicionais para tornar o simulador mais realista, e interessante.

O objectivo do jogo é adquirir a Power Stone que provém do boss final Samael .

2. A história

O herói, tem conhecimento que numa Dungeon, existe uma pedra, esta pedra é guardada por uma criatura de incrível poder de nome desconhecido. Esta Dungeon é conhecida por ser extremamente perigosa e quem entrou nunca saiu.

O herói, aventureiro e ambicioso por poder e fama decide aceitar o desafio e tentar obter essa Power Stone, sem saber o que lhe espera.

O jogador ao entrar nesta Dungeon, não sabe o que lhe espera, e encontra que vários monstros no seu percurso, sendo alguns destes aventureiros que perderam a vida.

Numa sala existe um Guardião, este guardião tem uma chave, que se for derrotado a deixa cair e o heroi a pode apanhar. Esta chave abre uma porta que o herói tem de descobrir.

No mapa existem items especiais, estes items sao essenciais para derrotar o boss final. Mas o herói não tem conhecimento deles. Pois se ninguém saiu desta Dungeon então não há maneira de saber. Estes items dao ao herói incrivel poder.

Se o herói encontrar a sala, então, encontra-se com Samael que se o herói encontra a espada lendária então diz lhe que a luta poderá ser interessante. Se não a tiver encontrado então, diz lhe que o herói morre no fim e mata o.

Se tiver a espada a luta inicia, ao derrotar o Samael, então Samael diz lhe que ele é digno da Power Stone, e dá lhe, mas com uma contrapartida, com a Power Stone vem uma maldição, quem adquirir essa Power Stone passa a ser Samael, libertando o anterior da maldição e este não poderá sair até ser libertado por outro herói.

3. Start Game

O jogo inicia chamando o predicado :-rpg.

Ao iniciar pede ao utilizador o mapa a usar, caso não tenha já pedido (isto porque causa do comando stop, assim o utilizado não tem de voltar a introduzir o mapa. Após introduzir o mapa, então chama begin. Este predicado inicializa a lista de track ([]), e o record (passa a 0), e chama start, que verifica se já existe uma posição em pos(X) e retoma o jogo, caso contrário, começa no inicio (inicial(X)).

Se o mapa existir então entra-se em loop que só termina se:

A vida do jogador for menor ou igual a 0,
O nivel de thirst e hunger for menor ou igual a 0,
O jogador adquirir a Power Stone,
O jogador introduz no input, stop.

Se nenhum destes se verificar, pede input de um comando qualquer:

Exemplo:

```
Hero: What should I do now? look.
```

4. Predicados Dinâmicos

4.1 Predicados dinâmicos no trabalho.pl

```
:-dynamic(record/1).            % tells if recorder is on or off
:-dynamic(track/1).             % tracking list
:-dynamic(pos/1).               % current pos
:-dynamic(inventory/1).         % player's inventory
:-dynamic(hp/1).                % hero's life
:-dynamic(attack_power/1).      % hero's attack damage
:-dynamic(equiped_w/1).         % equiped weapon
:-dynamic(equiped_a/1).         % equiped accessory
:-dynamic(hunger/1).            % hunger level
:-dynamic(thirst/1).            % thirst level
:-dynamic(loaded/1).            % is the map already loaded
:-dynamic(godmode/1).           % is god mode enabled
```

record/1 - indica o estado de recorder. 0 não está ligado, 1 está ligado.

track/1 - contém a lista do track (que contém todos os comandos utilizados pelo utilizador se o recorder estiver ligado)

pos/1 - indica a posição actual do jogador

hp/1 - indica a vida do jogador

attack_power/1 - indica o ataque do jogador

equip_w/1 - indica a arma que está equipada

equip_a/1 - indica os acessorios que estão equipados

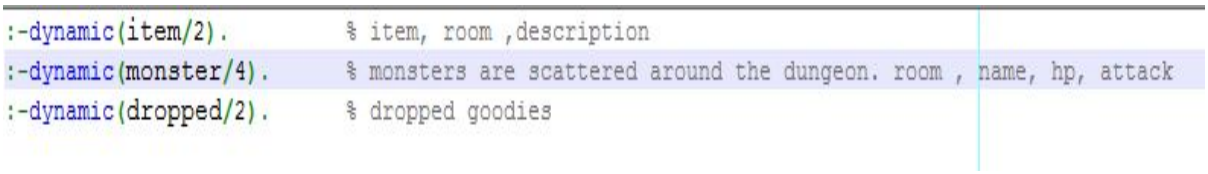
hunger/1 - indica o nivel de fome do jogador

thrist/1 - indica o nivel de sede do jogador

loaded/1 - indica se o mapa está foi carregado ou não. 1 significa carregado, 0 indica não.

godmode/1 - indica se o jogador está em godmode ou não. 1 significa está desactivado, 0 significa desactivado.

4.2 Predicados dinâmicos no map.pl:



item/2 - item(X,Y), sendo X o ID da sala e Y o nome do item.

monster/4 - monster(SALA, NOME, ATT, HP), sendo SALA o ID da sala, NOME, o nome do monstro, ATT o ataque do monstro, HP a vida do monstro

dropped/2, - dropped(SALA, NOME), sendo SALA o ID da sala, NOME o nome do item.

5. Os comandos

Nesta secção refere-se a implementação e descrição de todos os comandos obrigatórios e adicionais que foram adicionados ao longo da realização deste trabalho.

5.1. Conjunto base

5.1.1. go(N).

Verifica a posição actual do jogador. Ao chamar o comando, é feita uma verificação, se N for um número (number(N)), e esta for verdade então verifica se existe uma passagem entre N e a posição actual, ou seja, verifica a existência de passagem(X,N,_) sendo X a posição actual. Caso seja verdade, então pode chamar o predicado move(N).

Caso uma das verificações seja falsa, então envia uma mensagem de erro ao jogador, e o jogador mantém-se na sala onde estava inicialmente.

Cabe ao jogador submeter outro comando mas desta vez correctamente.

5.1.2. n, s, e, w, nw, sw, ne, se, up, down.

Verifica inicialmente a posição do jogador, isto é, o ID da sala onde se encontra e calcula o número de caminhos possíveis, através de getDir(DIR, Paths,Lenght), sendo DIR a direcção, Paths o numero de caminhos possíveis, e Lenght o tamanho da lista Paths. Dependendo do numero total de caminhos nessa direcção há uma interecção diferente.

Se o número de caminhos for 0, significa que passagem(X,_,DIR) não existe, e envia mensagem de erro.

Se o numero de caminhos for 1, significa que existe uma única passagem, nesse caso chama-se move(X) e altera-se a posição.

Se o numero de caminhos for superior a 1, significa que existe mais que uma passagem na mesma direcção, nesse caso chama-se o predicado chooseOne(Paths,DIR), para pedir ao utilizador para escolher uma das hipóteses da lista. Caso a hipótese não esteja na lista, é enviado uma mensagem de erro, caso contrário chama o predicado move(Y), sendo Y uma das hipóteses da lista.

5.1.3 inv.

Faz write do conteúdo do inventario. Passa todos os resultados de inventory(X), sendo este um predicado dinâmico, para uma lista usando findall e faz write dos seus elementos.

5.1.4 drop(X).

Verifica a posição do jogador.
Executa várias verificações.

No primeiro IF ELSE, verifica se X é atom(X), se for verdade então verifica se este é um item que faz parte do inventário (usando inventory(X)). Se ambas as verificações forem verdades, faz retract do inventory e faz asserta item(Y,X), sendo Y o quarto e X o item, passando este item X a fazer parte da lista de items no quarto Y. Se for falsa, então, envia mensagem de erro, e não ha alterações nenhumas.

No segundo IF ELSE se este item tiver equipado, seja acessório ou arma, então remove-se os atributos adicionados por estes items (só o ataque). Consegue-se remover os atributos fazem retract de attack_power(ATT) subtrai pelo valor do ataque do item (sabe através de: se for arma, weapon(NOME, ATT_W), e se for acessório, accessories(NOME,ATT_A,HP_A).

5.1.5 get(X).

Verifica a posição do jogador,
Se X for atom (atom(X)) e corresponder um item que faça parte do quarto correspondente a posição do jogador, então retira o item (só uma cópia se houver mais que um igual, usando retract(item(Y,X)) e faz asserta inventory(X) guardando o item no inventário do jogador.

Se o item for a chave (Key to the Golden Chamber) então existe uma interacção especial (só dialogo).

Se uma das condições acima falhar então envia mensagem de erro, e não há alterações.

5.1.6 look.

Verifica a posição do jogador.
Verifica todos os adjacentes (usando findall que guarda numa lista), é feito um print desta lista para mostrar o conteúdo da sala, em seguida chama printItems que, guarda todos os resultados item(X,Y) sendo X o numero da sala em que está presente, usando findall,e imprime a lista. Se a lista estiver vazia então manda mensagem que a lista está vazia.

Exemplo:

Quarto com items:

```
Hero: What should I do now? look.

Hero: These are my options..

      n leads to room 11
      nw leads to room 16
      nw leads to room 15


Hero: And these are all over the place..

      Ancient Stone Sword, Vial of Water
```

Quarto sem items:

```
Hero: What should I do now? look.
Hero: These are my options..

      e leads to room 17
      se leads to room 18


Hero: And these are all over the place..

Hero: Ups.. Actually its Empty
```

5.2. Registo

Todos os comandos tem um tracking(X), sendo X o nome do comando, por exemplo get(Herb), mesmo que o comando não seja válido, com por exemplo, get(ITEM), é adicionado para a lista tracking.

5.2.1 record.

O sistema de recorde necessita da ajuda de um predicado dinâmico, este predicado dinâmico (record/1) tem a função de verificar o estado do record. Só pode ter dois valores, 1 ou 0, e só tem um resultado de cada vez.

Se for 0, então ao utilizador este comando altera o valor de 0 para 1, caso contrário 1 para 0.

5.2.2 track.

Se o record estiver a 1, então sempre que o utilizador executa um comando então, adiciona esse comando para uma lista. Quando o utilizado executa o comando track. Então faz-se um print, do conteúdo da lista de tracking. listando os comandos utilizador pelo utilizador quando o record estava a 1.

5.2.3 forget.

O inverso de record. Se estiver a 1 passa a 0 , caso contrário 1 para 0.

5.2.4 stop.

Para o interpretador, isto é, o ciclo termina.

5.3. Modo “wizard”

5.3.1 jump(N).

Se N for um número, então executa um teleport, para N usando move(N). Se N não for um número então, não há alterações e mantém-se onde está.

5.3.2 warp(X).

Verifica a posição do jogador.
Se X for atom, então faz aparecer um item no quarto, isto é, asserta(item(POSICAO,X)). Se X não for atom, então, não há alterações e envia mensagem de erro.

5.3.3 destroy(X).

Verifica a posição do jogador.
Se X for atom, então verifica se o item X está presente na sala. Se sim então faz retract do item. Se uma das condições falhar então, não há alterações, e envia mensagem de erro.

5.4. NPCs

Existem monstros, e um sistema de combate, para tal monstros é um predicado dinâmico pois estes monstros desaparecem quando morrem. Exemplo:

monster(1,'Fallen Rookie Hero',300,60).

Sendo o primeiro argumento a sala onde está o monstro.
O segundo argumento é o nome do monstro.
O terceiro argumento é a vida do monstro.
O quarto argumento é o ataque do monstro.

Combate:

O jogador ataca sempre primeiro;
O jogador pode ignorar o monstro mas, o monstro continuará na sala;
O jogador pode bloquear os ataques do jogador;
Após o jogador atacar, o monstro ataca.

Sempre que o jogador atacar o monstro é feito uma verificação, para verificar se a vida do monstro é superior ou inferior a 0. Se for menor ou igual a 0 então, se o monstro tiver items ligado a ele, então o monstro faz um drop do item e esse item passa a ser um item do quarto.

Se for maior que 0, o monstro ataca o jogador e actualiza a vida do jogador (o jogador pode morrer).

Se o monstro for Samael (o boss final) então existem interacções especiais, usando os comandos attack e block.

6. Extensões

Das extensões apresentadas estas foram as escolhidas:

6.1 Luz

O jogador tem de ter no inventário uma torcha para utilizador o usar comando look.

6.2 Várias passagens na mesma direção, distinguir

Implementação descrita na secção 5.1.2

6.3 Armadilhas

Sempre que executa o move(N), chama act_trap. Sempre que um utilizador vai para uma sala, tem uma chance de 10% que a sala esteja armadilhada, se o jogador tiver azar então, perde vida da armadilha. Caso contrário não acontece nada.

```
random(1,10,L),
(L = 1 -> retract(hp(X)), Y #= X - 20,
    (Y #=< 0 -> write(''); asserta(hp(Y)), nl,
    write('Hero: ARGHHH a trap! Better be careful next time!'),
    nl);
nl,write('Hero: Uh.. No trap..'),nl).
```

6.4 Níveis de caverna

O jogador pode navegar entre dois mapas ligados, usando os comandos up e down. Isto se, a sala em que está tem uma direcção up ou down, ou ambos.

6.5 Monstros

Descrito na secção 2.4. Não tem spawning de monstros.

6.6 Comida

Usa o predicado, change_food_thirst_levels/0.

Sempre que o utilizador faz move e usa os comandos attack ou block consome recursos. Estes recursos são hunger e thirst.

attack e block consomem mais recursos que movimentar entre salas.

Se a comida e sede atingirem 0 o jogador morre e o interpretador termina.

6.7 Percurso total dum nível, a partir dum dado ponto

Existem 3 comandos, estes comandos usam o predicado **caminho/4** para calcular todos os caminhos possíveis.

```
% finds all the possible paths A to B, works with cycles
caminho(A,A,_,[A]).
caminho(A,B,VISITED,[A,'->',X,'->'|K]):- passagem(A,C,X), \+(member(C,VISITED)),caminho(C,B,[C|VISITED],K).
```

O comando Path, este comando calcula o caminho entre a sala inicial e a sala final. Para tal necessita de inicial(X) e final_chamber(Y).

O comando Path(X), este comando calcula todos os caminhos possíveis entre X e a sala final. Para tal necessita de final_chamber(Y).

O comando Path(X,Y), estes comando calcula todos os caminhos possíveis entre X e Y.

Todos os comandos após terem a lista dos caminhos é feito um print dessa lista em que o utilizador pode visualizar todos os caminhos possíveis incluindo a direcção.

7. Comandos adicionais

7.1 Equip(X).

O utilizador pretende equipar o item X se este pertencer ao inventário, for uma arma ou um acessório e X for atom.

O utilizador só pode ter uma arma de cada vez equipada e até 3 acessórios equipados.

Se ainda não tiver item equipado:

Altera os valores de attack_power(ATT) e hp(HP), e adiciona o item, se for arma asserta(equiped_w(X)), se for acessório asserta(equiped_a(X)).

Se já tiver item equipado:

Para armas:

- Retira os atributos adicionados pela arma, e substitui pelos de X.

Para acessórios:

- Se o item não tiver já equipado, então adiciona atributos, caso contrário substitui.

Se tentar equipar um item que já está equipado então da mensagem de erro e diz que já está equipado.

Usa os predicados weapon/2 e accessory/3 para verifica se X é um acessório ou uma arma.

7.2 stats.

Chama os predicados hp/1, attack_power/1, thirst/1, hunger/1 e faz print destes resultados.

Exemplo:

```
Hero: What should I do now? stats.

    [Health Points: 150]
    [Attack Power: 20]
    [Hunger: 30]
    [Thirst: 30]
```

7.3 use(X).

Serve para consumir os items que regeneram thirst e hunger. Se for um destes dois, estiver no inventário, e X for atom. Então, consome e altera os valores de thirst ou hunger. Sabe o valor a adicionar através de,
food(X,HP_REC) para hunger e thirst(X,HP_REC) para thirst. (ver secção 8.3)

Usa o predicado food/2 e thirst/2 para verifica se X é food ou thirst

7.4 heal(X).

Verifica se X é atom, se está no inventário, e é uma poção. Se essas forem true, então faz retract do inventory, e utiliza, isto é, usa **potions(X,HP_RECOVERY)** e altera a vida actual do jogador para a actual mais HP_RECOVERY. SE alguma destas falhar, então envia mensagem de erro.

Usa o predicado potions/2 para verificar se X é uma poção

7.5 gm.

Activa godmode, implica que o utilizador tem um poder de ataque colossal, e pode matar tudo com um único ataque. Se já estiver em godmode então envia mensagem de erro.

7.6 nogm.

Desactiva godmode, retira os stats adicionais que foram acrescentados por gm. Se já estiver o godmode desactivado, então manda mensagem de erro.

8. Inicialização e informação

8.1 Jogador

```
hunger(30).             % starting hunger level
thirst(30).          % starting thirst level
hp(150).                % hero's health pool
attack_power(20).       % hero's attack power


% player's starting items
inventory('Torch').          % can only use look command with torch
inventory('Herb').           % healing
inventory('Vial of Water'). % thirst
inventory('Loaf of Bread'). % food
```

8.2 Geral

```
loaded(0).          % is file loaded? default -> no
godmode(0).             % player is not god when he first enters
```

8.3 Tipo de Items

```
special('Key to the Golden Chamber').
special('Samael is Power Stone').
special('The Destroyers Wrath').
special('Necklace of Justice').
special('Ancient Ring of Power').
special('Ancient bracelet of Power').

% equip weapons
weapon('Ancient Stone Sword',30).
weapon('Good ol generic RPG sword',40).
weapon('Flaming whip of vengeance',200).
weapon('The Destroyers Wrath',600). % only weapon that can kill samael

% consumables
potions('Herbs',15).
potions('Small Potion',30).
potions('Potion',50).
potions('Big Potion',60).

% food and water

food('Loaf of Bread',20).
food('Pickle',20).
food('Cheese',20).

thirst('Vial of Water',13).
thirst('Jug of Water',20).
thirst('Wine',10).
thirst('Canteen of water',30).

% accessories
accessories('Necklace of Justice',30,30).
accessories('Ancient Ring of Power',200,200).
accessories('Ancient bracelet of Power',200,200).
```

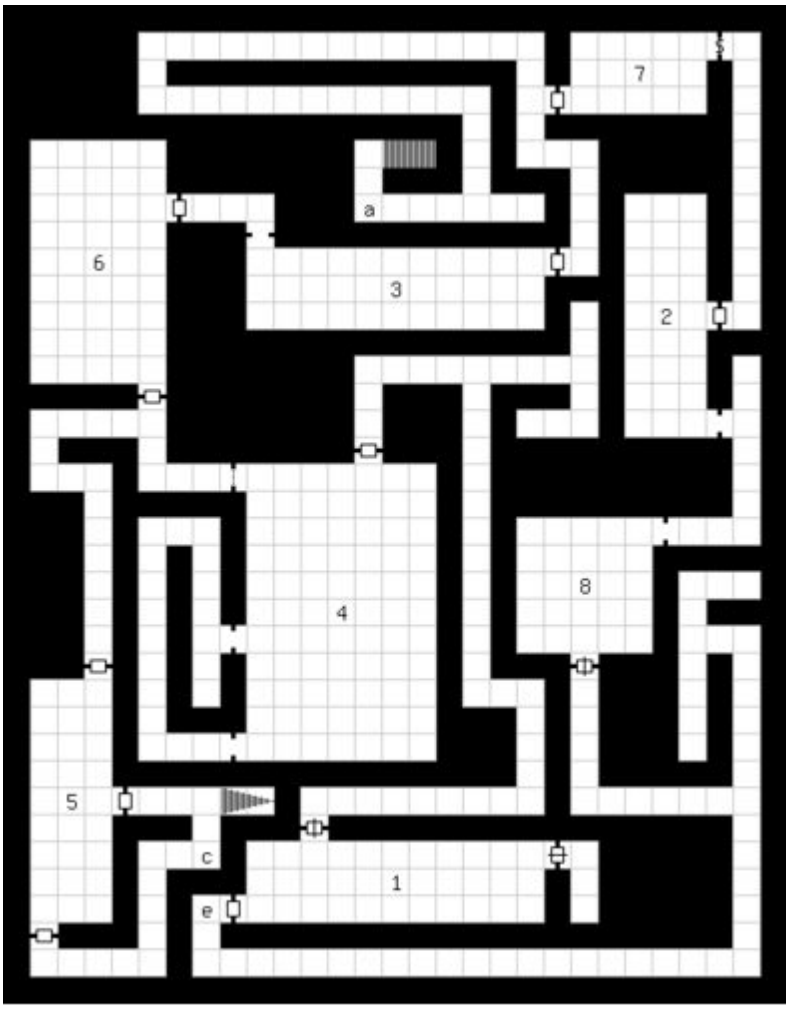
8.4 Monstros

```
monster(6,'Skeleton Warrior',300,40). % ROOM, NAME, HP, ATTACK
monster(3,'Skeleton Berserker',400,50). % ROOM, NAME, HP, ATTACK
monster(16,'Zombie',10,10).
monster(12,'Fallen Rookie Hero',30,15).
monster(4,'Fallen Hero',200,100).
monster(8,'Fallen Veteran Hero',300,150).
monster(14,'Skeleton Warrior',10,10).
monster(15,'Skeleton Beserker',50,30).
monster(1,'Fallen Rookie Hero',300,60).
monster(10,'Fallen Hero',50,50).
monster(7,'Fallen Veteran Hero',450,300).

% boss monsters
monster(2,'Guardian of the Golden Chamber',700,350). % guardian of the chamber
monster(5,'Samael',999,999). % final boss
```

8.5 Mapas

8.5.1 Mapa 1



8.5.2 Mapa 2

8.6 Move(X)

O predicado por detrás de todos os movimentos do jogador. Sendo X a nova posição do jogador.
Começa por fazer retract da posição do jogador, e se o jogador não estiver em god mode, altera os niveis de fome e sede.

Seguem-se três verificações.

- O jogador tem a chave e quer mover-se para a sala do boss. nesse caso há interecção espeicla com o monstro, há fala do boss e faz-se asserta da nova posição.

- O jogador não têm a chave e quer movimentar-se para a sala do boss. Recebe só mensagem de erro.

- O jogador quer movimentar-se para uma sala que nao seja a ultima. neste caso, verifica a existência de armadilhas usando o predicado act_trap, asserta da nova posição e verifica a existência de monstro.

A existência de monstro provém do predicado istheremonster(NX) que verifica usado monster(ROOM,NAME_MONSTER,_,_) a existência de um monstro.

9. Conclusão

A resolução deste trabalho foi certamente interessante. Ao ponto que dá vontade de acrescentar mais extras para tornar ainda mais interessante e balanceado, e outros extras pelo factor do “porque apetece e porque sim”.
Este trabalho penso que contribuiu para um maior conhecimento e compreensão através da utilização de todos os conhecimentos que foram adquiridos ao longo do semestre.