Programação Declarativa

Trabalho - RPG - Relatório

António Peixe - 34164

22 de Janeiro de 2016

Conteúdo

1 Introdução

No âmbito da unidade curricular de Programação Declarativa, integrada no programa da licenciatura em Engenharia Informática, foi proposto pelo professor a elaboração de um trabalho prático que apresentasse uma implementação de um jogo de role-playing, tipo "Dungeons & Dragons", na sua versão base.

Este trabalho tem como principal objectivo aprofundar os conhecimentos leccionados durante o semestre sobre programação lógica e a linguagem Prolog.

O programa deve permitir uma boa jogabilidade. Como qualquer jogo, tem de ter um objectivo final, assim decidi que seria um jogo de sobrevivência e procura de um item que dá a vitória.

O presente relatório serve por isso para apresentar o trabalho efectuado e como está estruturado o programa.

2 Mapa e Predicados

Nesta implementação foram adicionados alguns predicados ao ficheiro `map.pl`, para além dos base, de modo a existirem mais *features* no jogo. O mapa implementado no ficheiro enviado está de acordo com a seguinte imagem:

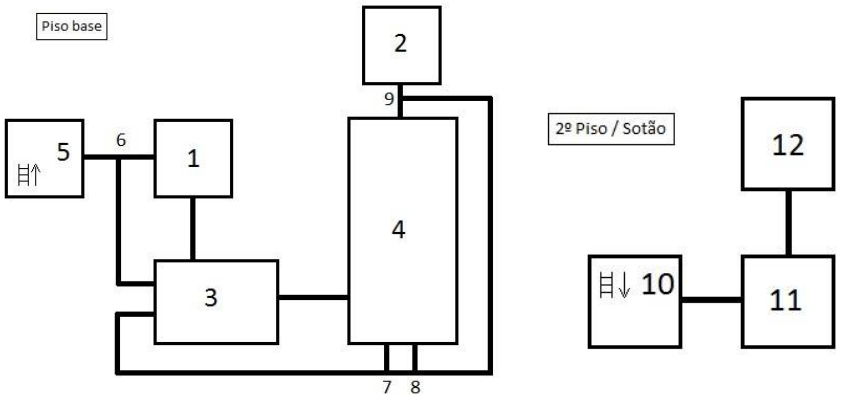

Figura 1: Mapa

- no(IDno, NOME)

  Qualquer local no jogo é considerado um nó e, só posteriormente, especificado como quarto ou cruzamento de passagem.

- cruza(IDno)

  Define quais dos nós do mapa são cruzamentos de passagens.

- quarto(IDno)

  Define quais dos nós do mapa são quartos.
  Para a implementação actual bastava ter apenas um dos predicados `cruza(IDno)` ou `quarto(IDno)` porque um exclui o outro, no entanto, pensando que no futuro podem haver outras categorias de nós, considero que esta é a melhor abordagem.

- light(IDno)

  Define quais dos nós têm luz (todos os restantes têm a característica oposta).

- passagem(IDno1, IDno2, DIR)

  Define qual a direcção a tomar para ir de `IDno1` para `IDno2`, isto não significa que o inverso aconteça no sentido oposto.

- item(NOMEitem, DESCRICAO)

  Define os itens existentes.
  `NOMEitem` é único por item.

- itemvalue(NOMEitem, PRECOVENDA, PRECOCOMPRA)

  Define os valores dos itens na loja.

- food(NOMEitem, NUTRICAO, QUANTliquidos)

  Define que itens são comida e como vão afectar a fome e a sede.

- itempos(NOMEitem, IDno)

  Define a posição inicial dos itens, em nós ou no vendedor.

- inicial(IDno)

  Define a posição inicial do jogador.

- password(X)

  Define a password para entrar em modo *wizard*.

- vendorpos(IDno)

  Define os nós onde se encontram vendedores ou lojas.

- winitem(NOMEitem)

  Define qual é o item que garante a vitória quando apanhado.

3 O Jogo

A implementação do motor do jogo é feita no ficheiro `dungeonrpggame.pl`. No início do ficheiro encontra-se a inicialização do mapa, a criação dos predicados dinâmicos e a inicialização do jogo propriamente dito. De seguida estão todos os predicados jogáveis e por fim alguns predicados auxiliares que não podem ser acedidos enquanto se joga.
O predicado que inicia o jogo é `:-rpg.`.
Tal como referido em 1, decidi que o jogo passasse por ter um aspecto de sobrevivência e de procura de um item que permitisse ganhar. Tendo isto em mente introduzi no jogo duas condicionantes para o jogador: a fome e a sede. Assim, para algumas acções, como por exemplo ir de um local para o outro, há um aumento da fome e da sede que têm de ser contrariados para que seja possível continuar a jogar.
De modo a arranjar métodos para o jogador saciar a fome e a sede, criei uma loja na qual pode comprar alimentos. Sendo que, só os poderá adquirir se tiver dinheiro. Para arranjar dinheiro, o jogador deve coleccionar itens e vende-los na loja, aumentando a pequena quantia inicial. Vender itens à loja "remove-os"do jogo pois deixam de estar em qualquer nó do mapa e a loja também não os guarda.
Devido à forte característica de procura que o jogo tem, achei essencial o inventário ter um limite, assim para todas as acções que tentem adicionar itens ao inventário é verificado se este já esta cheio. Um item que esteja equipado, apesar de não estar no inventário, continua a entrar na contagem pois a qualquer momento pode ser desequipado.
Outra característica do jogo é o facto de existirem zonas do mapa que não estão iluminadas, nestas o jogador não consegue ver nada, este efeito pode ser cancelado se o jogador apanhar uma lanterna e a equipar. Pelo contrário, nas zonas com luz o jogador pode ver todas as saídas que pode tomar, bem como os itens disponíveis para apanhar.

3.1 Predicados Dinâmicos

- pos/1

  Guarda a posição do jogador.

- invent/1

  Guarda os itens do inventário.

- reco/0

  Controla se o modo gravação está ligado.

- trackedactions/1

  Guarda as acções gravadas.

- wiz/0

  Controla se o modo *wizard* está activo.

- itemlocal/2

  Guarda, dinâmicamente, a posição dos itens no mapa.

- inhand/1

  Guarda o item que está equipado na mão.

- vendoritems/1

  Guarda os itens que estão à venda na loja.

- money/1

  Guarda o dinheiro que o jogador tem.

- hunger/1

  Guarda a fome que o jogador tem.

- thirst/1

  Guarda a sede que o jogador tem.

Existe também o predicado `reset.` (não jogável) que é chamado no início do jogo para garantir que todos os predicados dinâmicos estão do modo pretendido e ainda o predicado `placeitems.` (não jogável) que corre o predicado `itempos(NOMEitem, IDno).` (visto em 2) e coloca os items nos respectivos predicados dinâmicos.

3.2 Predicados Jogáveis

3.2.1 Predicados de Movimento

- go(N).

  Visto o jogador nunca saber o ID do quarto ou cruzamento onde está, apenas descrições, optei por abordar este predicado como sendo ir numa direcção. Deste modo, N toma os valores das saídas dos quartos, ou seja, **n, s, e, o, up, down** em que todos podem ter índices caso haja mais do que uma saída nessa direcção.
  Esta jogada aumenta a fome e a sede e é gravável.

- look.

  Permite ao jogador ver onde está caso haja luz ou se uma lanterna estiver equipada.
  Informa das saídas e itens que se encontram na mesma posição que o jogador.

3.2.2 Predicados de Itens e Inventário

- inv.

  Permite ao jogador ver o seu inventário.

- get(X).

  Permite ao jogador apanhar o item X caso se encontre na sua posição e se o inventário não estiver cheio.
  Esta jogada aumenta a fome e é gravável.

- drop(X).

  Permite ao jogador largar o item X do seu inventário na sua posição actual.
  Esta jogada é gravável.

- equip(X).

  Permite ao jogador equipar o item X existente no inventário.
  Retira o item do inventário e coloca-o na mão.

- deequip.

  Permite ao jogador desequipar o item que tem na mão.

- equipped.

  Permite ao jogador verificar qual o item que tem na mão.

3.2.3 Predicados de Gravação

- record.

  Indica ao jogo que deve começar a gravar acções.

- forget.

  Indica ao jogo que deve deixar de gravar acções.

- track.

  Permite ao jogador ver as acções gravadas.

3.2.4 Predicados Modo Wizard

- wizardon(X).

  Sendo X a *password* definida para o mapa (visto em 2).
  Indica ao jogo que deve entrar no modo *wizard* caso X seja válido.

- wizardoff.

Indica ao jogo que deve sair do modo *wizard*.

- jump(N).

  Para N ≡ IDno.
  Apenas utilizável quando o modo *wizard* está activo.
  Permite ao utilizador saltar para N.

- warp(X).

  Para X ≡ NOMEitem.
  Apenas utilizável quando o modo *wizard* está activo.
  Permite ao utilizador puxar o item X de qualquer local do mapa para a sua posição actual.

- destroy(X).

  Para X ≡ NOMEitem.
  Apenas utilizável quando o modo *wizard* está activo.
  Permite ao utilizador destruir o item X existente na sua posição actual.

3.2.5 Predicados da Loja

- shop.

  Permite ao jogador ver a loja caso haja luz ou se uma lanterna estiver equipada.
  Informa o dinheiro que o jogador possui, quais os itens que pode comprar, o seu preço e ainda os itens que pode vender e qual o seu valor.

- sell(X).

  Permite ao jogador vender o item X existente no seu inventário, na loja da sua localização.

- buy(X).

Permite ao jogador comprar o item X existente na loja da sua localização, caso não tenha o inventário cheio.

3.2.6 Predicados Sinais Vitais

- vitals.

  Permite ao jogador verificar como estão os seu sinais vitais.

- use(X).

  Para X um alimento definido no mapa (visto em 2).
  Usa X se este estiver equipado, modificando os valores de fome e sede de acordo com o definido no respectivo predicado do mapa.

3.3 Predicados Auxiliares

- bigBorder. e smallBorder.

  Dois predicados que imprimem linhas tracejadas para ajudar na vizualização.

- view., viewitems(X), viewitems2(X) e viewvendor.

  Predicados usados no predicado `look.` para facilitar a implementação e minimizar o número de linhas.
  Facilitam, respectivamente, a escrita do local, dos items que se encontram nessa posição e da informação de existência de loja.

- inventSize(S).

  Retorna a quantidade de itens no inventário.
  Usado nos predicados `get(X).` e `buy(X)` para verificar se é possível apanhar ou comprar mais itens.

- shoppping., shopitems., sellitems.

  Predicados usados no predicado `shop.` para facilitar a implementação e minimizar o número de linhas.

Facilitam a escrita do dinheiro disponível, dos itens existentes na loja que podem ser comprados e dos itens que podem ser vendidos, respectivamente.

- changeHunger(X)., changeThirst(X).

  Predicados usados para modificar o valor da fome e da sede, respectivamente.

- alive.

  Predicado usado para verificar se o jogo deve prosseguir, é por isso chamado todos os ciclos.
  Implementado de maneira a que, se a fome ou a sede tomarem valores iguais ou superiores a 50, o jogo termine avisando que o jogador morreu.
  Verifica também se o item que permite ganhar se encontra no inventário, caso seja verdade, termina o jogo com uma mensagem de sucesso.

4 Conclusão

A elaboração deste trabalho permitiu complementar os conhecimentos sobre Prolog e programação lógica leccionados durante o semestre.
No decorrer da implementação decidi focar-me na criação de uma boa experiência de jogo em termos de quantidade de *features* disponíveis, de modo a dar um aspecto mais completo ao jogo. Devido a esta abordagem o aspecto de interacção jogador/jogo acabou por ficar de parte e, por isso, o jogo é um interpretador de comandos Prolog e não de linguagem natural, o que o tornaria mais *user friendly*. Logo é sempre necessário um ponto final no fim de cada comando.
O facto de ter implementado a fome e a sede torna o jogador mais cuidadoso nas jogadas que faz, pois não pode simplesmente percorrer o mapa apanhando todos os itens até encontrar o desejado. Isto leva-o a pensar sobre o que fazer e a tentar compreender o mapa à medida que se aventura.
Por fim, parece-me que o programa permite ter uma boa experiência desde que seja respeitada a condição de colocar um ponto no fim de cada acção que o jogador tente fazer.