TÉCNICO LISBOA

Fundamentos da Programação

Projecto – Segunda Parte (corrigido)

22 de Novembro de 2012

Robot Explorador

Com este projecto pretende-se desenvolver um programa em Python que permita controlar um *robot* explorador que recolhe o maior número possível de objectos, explorando um mundo desconhecido[1].

O *robot* encontra-se num mundo que não conhece, e deve construir um mapa desse mundo, à medida que o vai explorando. Para isso devem ser criados vários tipos de informação que permitam representar a informação relevante acerca do mundo, podendo esta informação ser consultada posteriormente.

1 Descrição do problema

O mundo que este *robot* vai explorar possui obstáculos, ameaças à integridade do *robot*, e fontes de energia necessárias à continuação da exploração. Para além disso, existem uma série de objectos que devem ser recolhidos pelo *robot*.

Dada a dificuldade em testar os *robots* numa situação real, será fornecido um simulador do mundo[2]. A tarefa no projecto é desenvolver o programa que vai controlar o *robot*. Para isso, é necessário saber quais as características do ambiente em que o *robot* vai ser colocado e quais as suas capacidades.

2 Características do ambiente

O ambiente onde o *robot* vai ser colocado consiste numa área rectangular plana, composta por várias posições. Em cada posição apenas pode existir um objecto em cada instante (ou nenhum). Os objectos existentes, para além do *robot*, dividem-se em quatro categorias: paredes, comida, prémios e monstros. O número de objectos de cada categoria é desconhecido e estão distribuidos de forma aleatória. A área em que o *robot* se desloca é delimitada por paredes a toda a volta, formando um rectângulo, podendo existir ainda outras paredes no interior do rectângulo. Se o *robot* tentar deslocar-se para uma posição onde existe um outro objecto choca com esse objecto e é destruído. As paredes são

[1] O enunciado deste projecto foi produzido pelos Professores António Leitão e João Cachopo, aos quais agradecemos reconhecidamente.

[2] O simulador será publicado à parte com instruções de como deve ser utilizado.

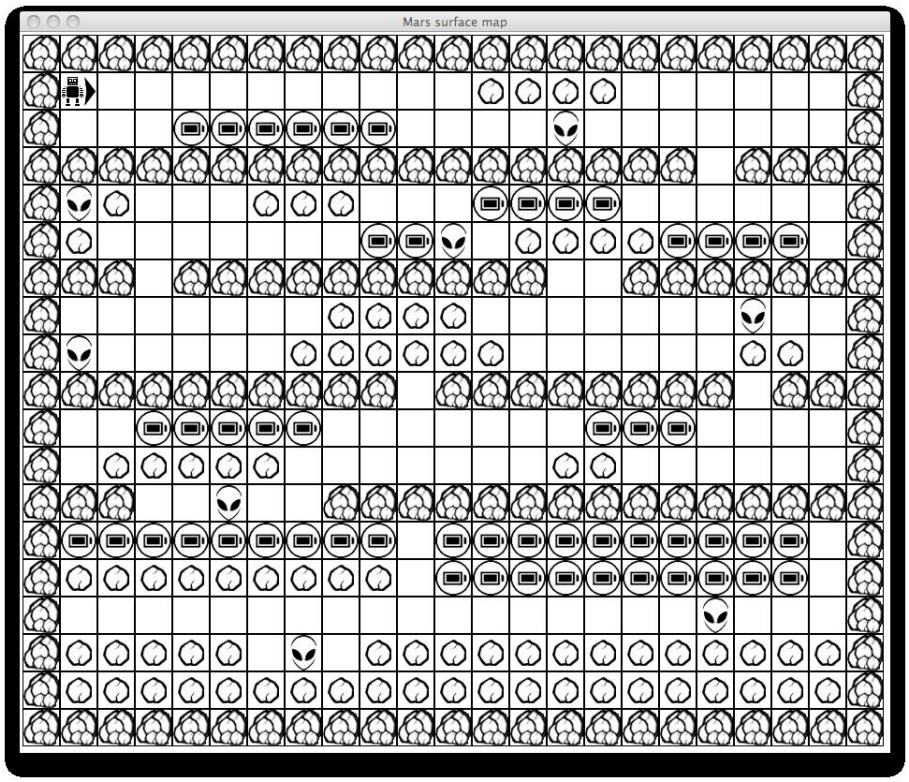

Figura 1: Exemplo de um ambiente.

indestrutíveis e inultrapassáveis. Os monstros podem ser destruídos com um raio disparado pelo *robot*. Os prémios e a comida podem ser apanhados pelo *robot*. Quando o *robot* apanha comida, obtém energia.

Na Figura 1 encontra-se um exemplo de um ambiente, na casa em que se encontra o *robot* há uma seta que indica a direcção para a qual está voltado, as caixas correspondem aos prémios, as formas quase circulares correspondem à comida, os rostos alienígenas correspondem aos monstros, as posições vazias são casas não ocupadas, e o resto são paredes).

O *robot* começa sempre na primeira linha e primeira coluna do mundo (ou seja, no canto superior esquerdo encostado à parede esquerda e superior) virado para este. Repare-se que a parede que limita o ambiente à esquerda se encontra na coluna zero, e a parede que limita o ambiente em cima se encontra na linha zero.

3 Capacidades do robot

O *robot* tem um conjunto de sensores que lhe permitem obter informação sobre ele próprio (a sua energia) e sobre o que está à sua frente. Para além disso pode efectuar uma série de outras acções como andar e virar, por exemplo. A utilização dos sensores e a realização de acções consomem energia do *robot*. Quando a energia se esgotar, o *robot* desliga-se.

Para iniciar a comunicação com o *robot* deve usar-se a função `inicia_comunicacao`, que não tem argumentos e que devolve um objecto computacional que representa o canal de comunicação com o *robot*. Note-se que esta função é invocada pelo código do simulador e não deve ser invocada pelo código de controlo a desenvolver no contexto do projecto.

As operações disponíveis para o *robot* podem ser executadas através da aplicação das seguintes funções em Python que têm como único argumento o objecto computacional devolvido pela função `inicia_comunicacao`:

- `robot_sente_espaco`, `robot_sente_monstro`, `robot_sente_parede`, `robot_sente_comida`, `robot_sente_premio`

  estas funções devolvem `True` ou `False`, respectivamente, conforme exista ou não o objecto do tipo correspondente na posição à frente do *robot*;

- `robot_anda`

  esta função desloca o *robot* uma posição em frente, na direcção em que está virado. Se na posição em frente existir algum objecto o *robot* é destruído;

- `robot_apanha`

  esta função apanha o objecto que se encontra à sua frente, desde que seja um prémio ou comida. Se tentar apanhar um monstro ou uma parede o *robot* é destruído;

- `robot_vira_direita`, `robot_vira_esquerda`

  esta função vira o *robot* 90 graus para a direita ou esquerda, respectivamente, mantendo-se na mesma posição;

- `robot_energia`

  esta função devolve um número inteiro que corresponde às unidades de energia ainda disponíveis no *robot*;

- `robot_dispara`

  esta função dispara um raio de pequeno alcance para a casa imediatamente à sua frente, destruindo o objecto que lá se encontre, excepto se for uma parede;

- `robot_desliga`

  esta função desliga o *robot*.

Todas estas acções consomem uma unidade de energia ao *robot*, exceptuando as acções `robot_energia` e `robot_desliga`, que não consomem qualquer energia, e a acção `robot_dispara`, que consome 4 unidades.

A acção `robot_apanha`, quando aplicada a uma fonte de energia, aumenta a energia do *robot* em 10 unidades.

Estas funções encontram-se no ficheiro robot.pyc disponibilizado na página da disciplina no arquivo proj2.zip, este arquivo contém também um ficheiro README com instruções relativas à utilização do código disponibilizado. O seu programa deverá conter a instrução from robot import *, a qual torna disponíveis todas as funções acima descritas.

Deve ser desenvolvida a função chamada *controla_robot* que controla o *robot*. Esta função tem como argumentos: (1) o objecto computacional que representa o canal de comunicação com o *robot* e (2) o estado[3] do mundo "conhecido". A função ao terminar deve devolver uma cadeia de caracteres correspondente ao nome da última acção efectuada (nome da função correspondente descrita acima) e ainda o estado actualizado de acordo com última acção efectuada.

É esta a função que é chamada pelo simulador enquanto o *robot* estiver activo, isto é, enquanto: tiver energia, não tenha sido destruído ou não se tiver desligado. No processo de decisão o *robot* deve começar por: (1) consultar a informação que mantém sobre o estado do mundo "conhecido", contida no argumento estado fornecido à função, e decidir qual a acção a realizar das acções descritas acima, (2) deve realizar a acção escolhida invocando a correspondente função descrita acima e (3) antes de terminar deve actualizar o estado de acordo com última acção efectuada.

4 Tipos de informação

O principal objectivo do *robot* é explorar o mundo em que se encontra, de modo a criar um mapa que lhe permita deslocar-se de forma mais eficiente, e apanhar todos os prémios que encontrar. Para que ele possa deslocar-se o mais longe possível, deve também apanhar toda a comida que encontrar.

O mapa a criar pelo *robot* corresponde a uma instância de um tipo que deve ser criado, e que tem um conjunto mínimo de operações que são especificadas abaixo. Para além do tipo para representar o mapa, é necessário definir outros tipos, por exemplo, as posições e os caminhos.

Descrevem-se alguns desses tipos e as operações que devem ser implementadas, bem como o funcionamento pretendido.

Para além destes tipos, podem ser criados todos os outros que considerarem necessários.

4.1 Tipo *objecto*

O tipo *objecto* permite representar o que existe em cada posição do mundo.

Devem ser definidos os nomes `espaco`, `comida`, `premio`, `monstro`, `parede` e `desconhecido`, bem como os reconhecedores (predicados) `espaco_p`, `comida_p`, `premio_p`, `monstro_p`, `parede_p` e `desconhecido_p`.

O objecto `espaco` serve para indicar que o *robot* sabe que não se encontra nada numa determinada posição, e o objecto `desconhecido` serve para indicar que o *robot* não sabe o que se encontra numa determinada posição.

[3] O tipo *estado* é especificado na página 7.

4.2 Tipo *direcção*

O tipo *direcção* tem quatro valores: `norte`, `sul`, `este` e `oeste`. Devem ser definidas quatro nomes em Python para representar cada um destes valores (usando os nomes `norte`, `sul`, `este` e `oeste`), e os respectivos reconhecedores (predicados) `norte_p`, `sul_p`, `este_p` e `oeste_p`.

Para além disso devem ainda ser definidas as seguintes operações:

- *vira_esquerda : direcção* $\mapsto$ *direcção*
- *vira_direita : direcção* $\mapsto$ *direcção*

Estas operações recebem uma *direcção* e devolvem a *direcção* que resulta de se virar para a esquerda ou direita, respectivamente, a partir da direcção dada.

4.3 Tipo *posição*

Os valores do tipo *posição* correspondem às posições que os vários objectos podem ocupar num determinado mundo. Cada posição é definida com base na sua linha e coluna.

Devem existir, pelo menos, as operações:

- *posicao : inteiro* $\times$ *inteiro* $\mapsto$ *posição*
- *posicao_linha : posição* $\mapsto$ *inteiro*
- *posicao_coluna : posição* $\mapsto$ *inteiro*
- *posicao_igual : posição* $\times$ *posição* $\mapsto$ *lógico*
- *posicao_relativa : posição* $\times$ *direcção* $\mapsto$ *posição*

A operação *posicao* é o construtor do tipo, e as operações *posicao_linha* e *posicao_coluna* são os selectores. A operação *posicao_igual* é a operação de teste que permite comparar duas posições, indicando se são iguais ou não.

A operação *posicao_relativa* recebe uma posição e uma direcção e deve devolver a posição adjacente à posição dada na direcção dada.

O tipo *posicao* deve ser implementado como uma classe Python. O construtor deve verificar se os argumentos são do tipo esperado, se não forem deve gerar um `TypeError`.

4.4 Tipo *mapa*

O tipo *mapa* permite representar a informação relativa ao mundo explorado pelo *robot*.

Este tipo deve apresentar, pelo menos, as operações:

- *mapa :* $\mapsto$ *mapa*

- *mapa_poe_objecto_em : mapa* $\times$ *posição* $\times$ *objecto* $\mapsto$ *mapa*
- *mapa_objecto_em : mapa* $\times$ *posição* $\mapsto$ *objecto*
- *mapa_altura : mapa* $\mapsto$ *inteiro*
- *mapa_largura : mapa* $\mapsto$ *inteiro*

A operação *mapa* constrói um novo mapa ainda não explorado, e a operação *mapa_poe_objecto_em* cria um novo mapa, a partir de um mapa dado, em que na posição dada está o objecto dado.

A operação *mapa_objecto_em* permite saber que objecto existe numa dada posição do mapa. Note-se que esta operação pode devolver o valor `desconhecido`.

As operações *mapa_altura* e *mapa_largura* devolvem as dimensões do *rectângulo* que envolve a área explorada, incluindo as paredes envolventes.

O tipo *mapa* deve ser implementado como uma classe Python.

4.5 Tipo *caminho*

Um *caminho* corresponde a uma sequência de posições adjacentes no mundo, tendo uma origem e um destino. Consideram-se posições adjacentes apenas as quatro posições que se encontram a norte, sul, este e oeste de uma determinada posição (ou seja, não se consideram as diagonais).

Devem existir, pelo menos, as operações:

- *caminho : posição* $\mapsto$ *caminho*
- *caminho_junta_posicao : caminho* $\times$ *direcção* $\mapsto$ *caminho*
- *caminho_origem : caminho* $\mapsto$ *posição*
- *caminho_apos_origem : caminho* $\mapsto$ *caminho*
- *caminho_destino : caminho* $\mapsto$ *posição*
- *caminho_antes_destino : caminho* $\mapsto$ *caminho*
- *caminho_comprimento : caminho* $\mapsto$ *inteiro*
- *caminho_contem_ciclo : caminho* $\mapsto$ *lógico*
- *caminho_elimina_ciclos : caminho* $\mapsto$ *caminho*

As operações *caminho* e *caminho_junta_posicao* são os construtores. No caso da operação *caminho*, é criado um caminho apenas com uma posição, onde a origem e o destino são coincidentes. A operação *caminho_junta_posicao* altera o caminho existente, juntando-lhe um novo destino, adjacente ao destino do caminho dado na direcção dada.

As operações *caminho_origem* e *caminho_apos_origem* recebem um caminho e devolvem a posição origem do caminho e o caminho sem a origem, respectivamente. As operações *caminho_destino* e *caminho_antes_destino* são semelhantes mas para o destino do caminho.

A operação *caminho_comprimento* devolve o comprimento do caminho, ou seja, quantas posições existem no caminho.

A operação *caminho_contem_ciclo* deve devolver o valor `True` se o caminho dado passa duas vezes pela mesma posição, e `False`, em caso contrário.

A operação *caminho_elimina_ciclos* deve devolver um caminho que resulta do caminho dado, depois de eliminar todas as posições que constituem ciclos no caminho, ou seja, posições que se encontrem entre duas ocorrências da mesma posição.

O tipo *caminho* deve ser implementado como uma classe Python. O construtor deve verificar se o argumento é do tipo esperado, se não for deve gerar um `TypeError`.

4.6 Tipo *estado*

O *tipo* estado representa a situação actual em que o *robot* se encontra, isto é, a *posição* em que ele se encontra, a *direcção* para onde está voltado, o *mapa* que já explorou e o *caminho* que percorreu.

Devem existir, pelo menos, as operações:

- *estado : posição* $\times$ *direcção* $\mapsto$ *estado*
- *estado_robot_avanca : estado* $\mapsto$ *estado*
- *estado_robot_vira_direita : estado* $\mapsto$ *estado*
- *estado_robot_vira_esquerda : estado* $\mapsto$ *estado*
- *estado_robot_apanha : estado* $\mapsto$ *estado*
- *estado_robot_dispara : estado* $\mapsto$ *estado*
- *estado_posicao_robot : estado* $\mapsto$ *posição*
- *estado_direccao_robot : estado* $\mapsto$ *direcção*
- *estado_mapa : estado* $\mapsto$ *mapa*
- *estado_caminho_percorrido : estado* $\mapsto$ *caminho*

A operação *cria_estado* recebe a posição e a direcção inicial do *robot* e deve devolver um estado em que o *robot* se encontra na posição e direcção dadas, sem ter explorado ainda nada (excepto a posição inicial).

As operações *estado_robot_avanca*, *estado_robot_vira_direita*, *estado_robot_vira_esquerda*, *estado_robot_apanha* e *estado_robot_dispara* recebem um estado e devolvem um novo estado que corresponde à situação que resulta de efectuar a acção respectiva (avançar, virar, etc) no estado dado.

As operações *estado_posicao_robot*, *estado_direccao_robot*, *estado_mapa* e *estado_caminho_percorrido* são selectores que, dado um estado, devolvem a posição actual do *robot*, a direcção para onde está virado, o mapa já explorado e o caminho já percorrido pelo *robot*, respectivamente.

O tipo *estado* deve ser implementado como uma classe Python. O construtor deve verificar se os argumentos são do tipo esperado, se não forem deve gerar um `TypeError`.

5 Sugestões

1. Comece por desenvolver os tipos de informação pedidos pela ordem listada na secção anterior. Teste cada um dos tipos cuidadosamente antes de passar ao seguinte, de modo a garantir que funcionam correctamente em todos os casos possíveis.
2. Só depois de ter completado o desenvolvimento de todos os tipos deve desenvolver a função de controlo do *robot*.

6 Relatório

No relatório deverão ser discutidos vários aspectos relacionados com o desenvolvimento deste projecto. Alguns dos aspectos a focar são:

- Especificação de todos os tipos de informação.
- Descrição das decisões de implementação mais importantes do projecto, incluindo os tipos de dados.
- Descrição da estratégia utilizada pelo *robot* para explorar o mundo.
- Limitações do programa.
- Possíveis melhoramentos ao programa.

O relatório deve incluir também uma listagem completa do código do projecto. A listagem deve ser em fonte monospace sem caracteres não ASCII.

7 Aspectos a evitar

Os seguintes aspectos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projecto):

1. Não pense que o projecto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá ver a Lei de Murphy mesmo em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).

2. Não pense que um dos elementos do seu grupo fará o trabalho por todos. Este é um trabalho de grupo e deverá ser feito em estreita colaboração (comunicação e controle) entre os vários elementos do grupo, cada um dos quais com as suas responsabilidades. Tanto uma possível oral como perguntas no segundo teste sobre o projecto, servem para despistar estas situações.
3. Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
4. Não se esqueça que as funções com muitas linhas são penalizadas no que respeita ao estilo de programação.
5. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
6. Quando o programa gerar um erro, preocupe-se em descobrir *qual* a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.

8 Classificação

A nota do projecto será baseada nos seguintes aspectos:

1. Execução correcta (40%).

   Esta parte da avaliação é feita recorrendo a um programa de avaliação automática.
2. Facilidade de leitura, nomeadamente abstracção procedimental, abstracção de dados, nomes bem escolhidos, paragrafação correcta, qualidade (e não quantidade) dos comentários e tamanho das funções (25%).
3. Estilo de programação (5%).
4. Relatório (30%).

9 Condições de realização e prazos

A segunda parte do projecto deve ser entregue até às **15:00** horas do dia **20 de Dezembro de 2012**, na sala de estudo do DEI ou na portaria do Tagus (consoante o campus que corresponda ao grupo do projecto), e deverá constar de um relatório, incluindo uma listagem do código (numa fonte "monospace"como, por exemplo, a fonte "courier", sem caracteres não ASCII). O relatório deve ser entregue dentro de uma capa apresentando visivelmente o número do grupo e número e nome dos seus autores, na capa.

A partir das **15:00** horas do dia **20 de Dezembro de 2012** não se aceitam quaisquer projectos, seja qual for o pretexto.

Para além disto, a submissão do código por via electrónica, através do sistema Fénix, é obrigatória e deverá ser feita nos mesmos prazos que a entrega do relatório.

O código do projecto deve estar contido num único ficheiro com o nome (note a utilização do caracter '_' no nome!) `FP1213_parte2_grupo<n>.py`, em que `<n>` é o

número do seu grupo. Por exemplo, o ficheiro do grupo número 5 deverá chamar-se `FP1213_parte2_grupo5.py`. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo.

Importante: no texto do programa não devem ser utilizados carateres acentuados ou qualquer caracter que não pertença à tabela ASCII, isto inclui comentários e cadeias de carateres. Programas que não cumpram este requisito serão penalizados em três valores.

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Projectos iguais, ou muito semelhantes, serão penalizados com a reprovação na disciplina. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projecto.