% INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL — Exame, 27/01/2015 (Tempo: 3h

Identificação do aluno:

Nome: _____ #: _____

I

1. Analise as seguintes funções escritas em Python e explique o que fazem. Não precisa de explicar o funcionamento interno da função.

a)
```
def f(x,y):
    if x==[]:
        return None
    z = f(x[1:],y)
    if z==None or y(x[0],z):
        return x[0]
    return z
```

Através de uma função de comparação y, pegando no último elemento de x vamos comparar com o anterior e por aí adiante. É retornado o valor mais recente a satisfazer a função y.

b)
```
def g(x,y):
    if x==[] and y==[]:
        return []
    if x!=[] and y!=[]:
        z = g(x[1:],y[1:])
        if z!=None:
            return [(x[0],y[0])]+z
```

Se as listas x e y tiverem o mesmo comprimento, a função retorna uma lista de tuplos que são os dois elementos da mesma posição

2. Implemente em Python as seguintes funcionalidades relevantes para sistemas inteligentes:

a) Como sabe, o problema do "caixeiro viajante" (TSP) tem muitas aplicações. Ele consiste em determinar um caminho que, partindo de uma cidade, passa por um conjunto de outras cidades sem as repetir, e termina na cidade inicial. No contexto da resolução do TSP através de pesquisa A*, é necessário implementar uma heurística. Desenvolva uma função que, dada a cidade actual, a cidade final (que foi também a cidade inicial), uma lista de cidades ainda não visitadas e uma função de cálculo de distâncias entre cidades, devolve o valor da heurística. A função de cálculo das distâncias recebe como entradas duas cidades e devolve a distância em linha recta entre elas. Para não complicar muito o seu trabalho, a heurística é calculada multiplicando o número de transições que ainda faltam [illegible] quaisquer cidades.

```
def heuristic(start, finish, not_visited):
    smallest_dist = math.inf
    list = start + finish + not_visited
    for city1 in list:
        for city2 in list:
            if city1 != city2:
                dist = distance(city1, city2)
                if dist < smallest_dist:
                    smallest_dist = dist
    return (len(not_visited) + 1) * smallest_dist
```

b) No contexto da implementação de algoritmos de pesquisa em árvore, considere que os nós da árvore são representados por tuplos *(Id,State,AtomCust,Hour,IdParent)*, em que *Id* é um identificador do nó, *State* é o estado desse nó, *AtomCust* é o custo acumulado desde o estado inicial até *State*, *Hour* é o custo estimado para chegar à solução a partir do nó *Id*, e *IdParent* é o identificador do nó pai. Considerando que se está a usar uma estratégia de pesquisa A*, programe uma função que, dada a lista de nós abertos (nós que aguardam expansão) e dada uma lista de novos nós (que acabam de ser criados por expansão de outro nó), produza a nova lista de nós abertos.

II

1. O agente artificial Comilão necessita de alimento. O seu mundo baseia-se numa grelha de dimensão 10x10, existindo biscoitos dispersos pela grelha. Há também o Zarpadaqui, um agente predador que pode comer o Comilão. Cada célula da grelha pode estar livre, ou então ocupada pelo Comilão, pelo Zarpadaqui ou por um biscoito. O Comilão pode mover-se na horizontal ou na vertical usando a acção $mov(\Delta c, \Delta l)$, em que $\Delta c \in \{-1,0,1\}$, $\Delta l \in \{-1,0,1\}$ e $|\Delta c| \neq |\Delta l|$, não podendo ultrapassar os limites da grelha. Sempre que o Comilão se desloca para uma célula com um biscoito, come-o automaticamente e um novo biscoito surge numa posição aleatória da grelha. Quando o Comilão se move para uma posição adjacente à do Zarpadaqui, este come-o. O Comilão consegue saber onde está o predador bem como os biscoitos, e consegue ainda saber qual a célula que corresponde ao centro geométrico da distribuição dos biscoitos. Sendo insaciável, o seu objectivo é ir comendo o maior número possível de biscoitos, e claro, fugir às garras do predador. Sempre que o Comilão vem um biscoito em posição adjacente a si próprio e não adjacente ao predador, desloca-se para essa posição. Caso contrário, tenta aproximar-se do centro geométrico da distribuição de biscoitos.

a) Identifique e caracterize um conjunto de predicados que possam ser usados para descrever situações em que se encontre o Comilão. Identifique também as variáveis de estado, caso seja necessárias.

b) Usando as condições e variáveis da alínea anterior, especifique um conjunto de regras situação-acção que definam um comportamento adequado do Comilão. Pode fazê-lo na forma de uma tabela com as seguintes colunas:

- Situação - uma conjunção de condições em lógica de primeira ordem
- Actualização - actualização das variáveis de estado, caso existam
- Acção - acção a executar pelo agente na situação indicada

c) Considerando que o Comilão entretanto adquiriu capacidade deliberativa, nomeadamente para planeamento de caminhos que maximizem o número de biscoitos ingeridos, especifique um operador STRIPS para a acção *move(dx,dy)*. [Nota: para este efeito, considere que a grelha é infinita.]

d) Foi desenvolvido um agente reactivo simples, controlado por um conjunto de regras, para participar no Concurso Ciber-Rato. As condições das regras envolvem certos parâmetros cujo valor é necessário afinar. Caso queira utilizar uma técnica de pesquisa para determinar valores satisfatórios ou mesmo óptimos para os parâmetros, que técnica usaria? Como aplicaria essa técnica?