Universidade da Beira Interior

Departamento de Informática

Engenharia Informática

Inteligência Artificial

Elaborado por:

João Domingos, nr 38023

Ricardo Domingos, nr 37906

Professor:

Luís Alexandre

2 de Janeiro de 2019

Conteúdo

Introdução

Estrutura do Relatório

O relatório encontra-se dividido em quatro capítulos:

Introdução:

Breve desrição do trabalho. Apresentando a estrutura do relatório, as motivações bem como os objetivos.

Implementação:

Mostra como foram implementadas as funções no programa e que linha de raciocínio utilizámos para resolver cada questão.

Conclusão:

Descreve como foi feita a partição do trabalho e relata as conclusões principais retiradas do mesmo.

Motivação

A principal motivação deste projeto é que os robôs comuns não são inteligentes ao fim de se mover num determinado ambiente, reconhecendo objetos, retirando informação do meio e gerar conclusões sobre este. O objetivo deste trabalho é desenvolver estas capacidades.

Objetivos

O meta deste trabalho é construir um programa que gere um robô móvel num determinado ambiente, capaz de responder a perguntas específicas sobre o mundo em que está inserido.

Implementação

Neste capítulo são descritas as funções utilizadas para resolver cada questão com exemplos do código.

Código Complementar

Os trechos de código seguintes são essenciais na resolução das perguntas.

Verificação da sala onde se encontra o agente atualmente. Sendo que a sala onde o agente nasce é a sala número um.

```
        #atributes current room <>
    global room_ant
    if x > -1:
        if y < 1.5 : room=1
        elif y<6.6 : room=2
        else: room=3
    elif x > -6:
        if y < 1.5 : room=4
        else: room=5
    elif x > -11:
        if y < 1.5 : room=6
        elif y<6.6 : room=7
        else: room=8
    else:
        if y < 1.5 : room=9
        elif y<6.6 : room=10
        else: room=11

    if room!= rm:
        adiciona_paths(str(room),str(rm))
        room_ant=rm
        rm=room
        print " room = %d" % room
```

Atribuição dos objetos à sala correspondente, utilizando uma lista de dicionários.

```
def callback1(data):
    global obj_ant
    global room_ant
    global salas
    global dala,rm
    obj = data.data
    if obj != obj_ant and data.data != "":
        print "object is %s" % data.data
        for i in range(len(data.data.split(","))):
            obj1=data.data.split(",")[i]
            obj_types=salas[rm-1].keys()
            if obj1.split("_",1)[0] in obj_types:

                if obj1.split("_",1)[1] not in salas[rm-1][obj1.split("_",1)
[0]]:

                    salas[rm-1][obj1.split("_",1)
[0]].append(obj1.split("_",1)[1])
            else:
                salas[rm-1][obj1.split("_",1)[0]]=[obj1.split("_",1)[1]]
    obj_ant = obj
```

Pergunta 1. How many different types of objects did you recognize until now?

Na resolução da quantidade de tipos de objetos diferentes reconhecidos foi utilizada uma função para percorrer as salas, verificando as *keys* diferentes, e guardando-as numa varíavel. Depois é devolvida a *lenght* da mesma.

```
def diftypes():
    global salas
    for i in range (len( salas)):
        if i == 0:
            obj=salas[i].keys()
        else:
            x=salas[i].keys()
            for j in range(len(x)):
                if x[j] not in obj:
                    obj.append(x[j])
    print len(obj)
```

Pergunta 2. Which objects were in the room you visited before this one?

Para mostrar que objetos estavam na sala anterior é recebido como parâmetro o número da sala anterior, de seguida é verificado se visualizou algum objeto. Se tiver visualizado mostra a quantidade de *keys* da sala anterior. Se não avisa que não detetou objetos.

```
def objects_in_room(room):
    obj=salas[room-1].keys()
    if len(obj)==0:
        print "nao visualizou objetos"
    else:
        for i in range(len(obj)):
            print obj[i]
```

Pergunta 3. What is the probability of finding 10 books in this world?

A fim de calcular a probabilidade de encontrar dez livros neste mundo foi usada a expressão $(\boldsymbol{L/O})^{(\mathbf{10}-\boldsymbol{L})}$ onde:

L - Número de livros encontrados

O - Número de objetos vistos

Foi criada uma função para a contar todos os objetos vistos e outra para contar quantos livros foram encontrados e calcular a probabilidade referida.

```
def nr_of_seen_objs():#Conta quantos objetos viu no mundo
    global salas
        nr_objs = 0
    for i in range(len(salas)):
        for key in salas[i].keys():
            nr_objs =nr_objs + len(salas[i][key])
        return (nr_objs)

def prob_ten_books():
        global salas
        count_book = 0
    for i in range(len(salas)):
            if "book" in salas[i].keys():
            count_book =count_book +  len(salas[i]["book"] )
    return( ( 1.0*count_book / (nr_of_seen_objs()) ) ** (10-count_book))
```

Pergunta 4. What type of object do you think is the one without identification, that appears close to Joe?

Com o objetivo de saber que tipo de objeto é o que está perto do Joe foi primeiramente utilizada uma função que descobre a sala onde um determinado objeto se encontra, devolvendo -1 se não for o caso.

```
def find_sala(type_obj,name):
    global salas
    for i in range(len(salas)):
        if(type_obj in salas[i].keys()):
            if name in salas[i][type_obj]:
                return i+1
    return -1
```

Recebendo a sala como parâmetro verifica quais as salas com maior proximidade, isto é, as salas que contêm o maior número de objetos iguais à sala dada como parâmetro. Obtendo estas salas são calculadas quantas vezes cada objeto aparece nessas salas. De seguida encontra-se o objeto que aparece mais vezes nestas salas.

```
def proxobj(sala):
    global salas
    proximidade={}
    obj={}
    for i in range(len (salas)):
        if i != sala:
            l=set(salas[i].keys()).intersection(set(salas[sala].keys()))
            if(len (l)!=0):
                m=0
                for x in l:
                    m+=len(salas[sala][x])-len(salas[i][x])
                if (m>=0):
                    if(m in proximidade.keys()):
                        proximidade[m].append(i)
                    else:
                        proximidade[m]=[i]
                else:
                    if(x in obj.keys()):
                        obj[x]=obj[x]+0-m
                    else:
                        obj[x]=0-m
                    m=0
                    if(m in proximidade.keys()):
                        proximidade[m].append(i)
                    else:
                        proximidade[m]=[i]
    maxprox=proximidade.keys()[0]
    for i in proximidade[maxprox]:
        for n in salas[i].keys():
            if n not in salas[sala].keys():
                if n in obj.keys():
                    obj[n]=obj[n]+len(salas[i][n])
                else:obj[n]=len(salas[i][n])
    maxobj=0
    obje=[]
    for j in obj.keys():
        if obj[j] > maxobj:
            maxobj=obj[j]
            obje=[j]
        elif obj[j] == maxobj:
            obje.append(j)
            maxobj=obj[j]
    print "O possivel objeto é:" + obj[0]
```

Pergunta 5. What is your estimate of the time it takes to visit all the rooms?

Para estimar o tempo que demora a visitar todas as salas foi usado um dicionário onde sempre que o agente muda de sala o tempo decorrido é calculado através da diferença entre o tempo atual e o tempo inicial. Este tempo inicial é atualizado sempre que muda de sala.

```
if room!= rm:
    adiciona_paths(str(room),str(rm))
    if(str(rm-1) not in times.keys()):
        times[str(rm-1)]=time.time()-t_in
        t_in=time.time()
    else:
        times[str(rm-1)]=times[str(rm-1)]+time.time()-t_in
        t_in=time.time()
```

Soma todos os tempos conhecidos, armazenando na variável *med*, em seguida verifica se a sala número cinco já foi percorrida, caso seja verdade divide a variável *med* por as salas que já foram percorridas mais um (pois a sala cinco vale por duas), e multiplica pelo número de salas mais um. Se não for verdade divide a variável *med* por as salas que já foram percorridas, e multiplica pelo número de salas mais um.
Se o agente não tiver saído da sala inicial é mostrado um aviso.

```
    elif data.data == "5":
        med=0
        for i in times.keys():
            med=med+times[i]
        try:
            if "4" in times.keys():
                print "O tempo estimado é :" +
str((med/(len(times.keys())+1))*12)
            else:
                print "O tempo estimado é :" +
str((med/len(times.keys()))*12)
        except:
            print("Tempo ainda nao e possivel de prever")
```

Pergunta 6. How many different paths can you take to go from the current room, back to the start room?

Com a finalidade de saber quantos caminhos diferentes o agente pode escolher ir da sala atual até à sala inicial foram utilizadas as duas funções.

Esta função recebe como parâmetros duas salas (*string*), sala1 e sala2. Verifica se a sala1 já está no dicionário *paths* anteriormente criado. Caso exista adiciona a sala2 a essa *key* (sala1), caso não exista cria a *key* (sala1) e atribui a sala2 à sala1. E vice-versa para a sala2.

```
#adicionar caminhos

def adiciona_paths(nodo_1,nodo_2):
  global paths
  if(nodo_1 in paths.keys()):
    if(nodo_2 not in paths[nodo_1]):
      paths[nodo_1].append(nodo_2)
  else:
    paths[nodo_1]=[nodo_2]
  if(nodo_2 in paths.keys()):
    if(nodo_1 not in paths[nodo_2]):
      paths[nodo_2].append(nodo_1)
  else:
    paths[nodo_2]=[nodo_1]
```

Esta função cria uma pilha. Enquanto essa pilha não estiver vazia, retira o primeiro elemento da pilha (*pop*) guardando o vértice e o caminho associado. Para todos os vértices associados a este elemento verifica se é o elemento final. Caso seja verdade devolve o caminho. Caso contrário adiciona à pilha o próximo elemento com o caminho associado.

```
#devolve caminhos de 'a' a 'b'

def dfs_caminhos(grafo, inicio, fim):
    pilha = [(inicio, [inicio])]
    while pilha:
        vertice, caminho = pilha.pop()
        for proximo in set(grafo[vertice]) -
set(caminho):
            if proximo == fim:
                yield caminho + [proximo]
            else:
                pilha.append((proximo, caminho +
[proximo]))
```

Pergunta 7. In what type of room is Mary in?

Com o objetivo de mostrar em que tipo de sala a Mary está foi primeiramente utilizada uma função que descobre a sala onde um determinado objeto se encontra, devolvendo -1 se não for o caso.

```
def find_sala(type_obj,name):
    global salas
    for i in range(len(salas)):
        if(type_obj in salas[i].keys()):
            if name in salas[i][type_obj]:
                return i+1
    return -1
```

Se Mary foi encontrada, então a seguinte função vai receber a sala onde a Mary está, tal como os objetos da mesma. De seguida verifica recorrendo a interseções e uniões, qual o tipo de sala onde a Mary está situada.

```
def roomtype(obj,sala):
    wr=['chair']
    str1=['chair','table','book','person']
    str2=['chair','table','book']
    compr=['table','computer','chair']
    meetr=['table','chair']
    if
set(wr).intersection(set(obj))==set(wr).union(set(obj)):
        return "Waiting Room"
    elif
set(str1).intersection(set(obj))==set(str1).union(set(obj)) or
set(str2).intersection(set(obj))==set(str2).union(set(obj)):
        return "Study Room"
    elif
set(compr).intersection(set(obj))==set(compr).union(set(obj)):
        return "Computer Room"
    elif
set(meetr).intersection(set(obj))==set(meetr).union(set(obj)):
        if len(sala['table'])==1:
            return "Meeting Room"
        else:
            return "Generic Room"
    else:
        return "Generic Room"
```

Pergunta 8. What is the probability of finding a chair in a room given that you already found a book in that room?

A fim de calcular a probabilidade de encontrar uma cadeira numa sala sabendo que já foi visto um livro nessa mesma sala foi utilizado a fórmula $P(B|A) = \frac{P(A \cap B)}{P(A)}$. Onde $P(A \cap B)$ é o número de salas onde foram encontrados livros e cadeiras, e $P(A)$ é o número de salas que têm livros.

```
def probbook():
    global salas
    count_book=0
    count_book_chair=0
    for i in range (len( salas)):
        if("book" in salas[i].keys()):
            count_book+=1
            if("chair" in salas[i].keys()):
                count_book_chair+=1
    prob_A_B=1.0*count_book_chair/len(salas)
    prob_A=1.0*count_book/len(salas)
    return 1.0*prob_A_B/prob_A
```

Conclusão

Partição do Trabalho

Ricardo Domingos – Perguntas 1, 2, 5, 6, 7, 8.

João Domingos – Perguntas 3, 4;

- Relatório;
- Apresentação.

Conclusões Principais

Concluimos com este trabalho que mesmo para perguntas aparentemente simples, a um agente móvel, num determinado ambiente, estas têm um grau de complexidade inesperada. Verificamos também que num mundo real existe ruído o que torna a implementações das funcionalidades mais difícil.