25/06/2020

PIAZZA_SPRINT 1

Sprint Goal

Far muovere il robot virtuale all'interno della TeaRoom.

Requisiti

-Vogliamo essere in grado di muovere il ddr virtuale del progetto it.unibo.virtualRobot2020 all'interno della Tea Room.

-Vogliamo che, impostando una posizione all'interno della tea room come goal, il robot sia in grado di raggiungerla in autonomia.

Assunzione 1: il robot conosce già la mappa della Tea Room.
Assunzione 2: non possono essere impostate come goal delle posizioni contenenti un ostacolo. Di conseguenza, non essendoci ostacoli mobili, il robot non colliderà mai.

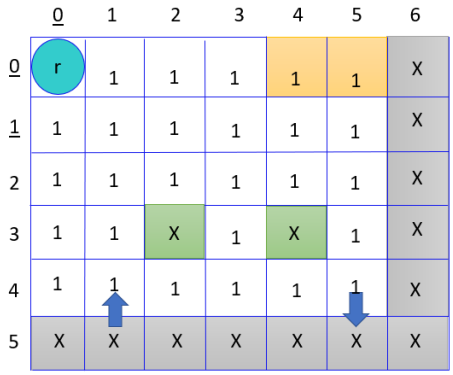
Analisi dei Requisiti

Per poter formalizzare i requisiti e poter impostare sin da ora dei Test Plan è necessario **formalizzare il concetto di mappa della TeaRoom.**

Da requisiti iniziali si tratta di una stanza rettangolare. Possiamo pensare di rappresentarla come una matrice di NxM celle quadrate, ciascuna di lato uguale al diametro della circonferenza circoscrivente il robot waiter.

Possiamo anche pensare di dare un significato al contenuto delle celle. In particolare:
1: cella esplorata.
0: cella non esplorata.
X: cella sopra la quale si trova un ostacolo (un oggetto o una parete).
r: cella in cui si trova il robot.

La mappa della TeaRoom risulta essere una matrice 6x7 in cui la riga 5 e la colonna 6 non sono percorribili dal robot in quanto vi si trovano delle pareti.



Individuando ciascuna cella con due coordinate (X,Y) le celle significative sono:

home in (0,0)
teatable1 in (2,3)
teatable2 in (4,3)
entrancedoor in (1,4)
exitdoor in (5,4)
servicedesk in (4,0) e (5,0)

BOZZA DI TEST PLAN

1. Se (X,Y) = r e inviamo un comando 'w' al robot dobbiamo ottenere (X,Y) = 1 poiché il robot si sarà spostato (assunto che non abbia un ostacolo davanti a lui).
2. Possiamo testare lo spostamento del robot in alcune celle significative. Se diamo come goal una cella (X,Y) = 1 ci aspettiamo che, al termine dello spostamento, valga (X,Y) = r.

NOTA: Ai fini dei test possiamo sfruttare il fatto che i QActor sono risorse Coap osservabili.

Analisi del Problema

Movimento step-wise : Affinché il robot possa muoversi consapevolmente all'interno della Tea Room, cella dopo cella, dovrà essere in grado di muoversi facendo degli step. Ogni step verrà fatto secondo una direzione (down | left | right | up) e porterà il robot nella cella successiva lungo quella direzione (se è libera).

Planning : Il problema poi richiede che il robot sia in grado di pianificare un serie di azioni che gli permettano di raggiungere una posizione goal a partire dalla posizione in cui si trova.

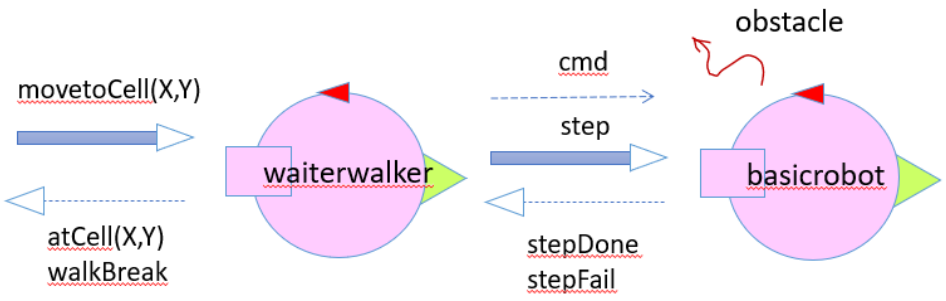
Dal momento che pianificare una sequenza di azioni ed eseguirle sono due concetti logicamente distinti possiamo pensare di modellare il sistema in due entità distinte seguendo il paradigma *mente-corpo* .
Così facendo otterremo due componenti che, avendo una singola responsabilità, saranno dei **servizi riusabili** anche per altri scopi.

> Abbiamo quindi **basicrobot**, che ha il compito di interfacciarsi con il robot virtuale per attuare i comandi. È in grado di far muovere il robot virtuale in modo step-wise. Poi avremo il planner che chiamiamo **waiterwalker**, ossia colui che pianifica le mosse da eseguire e ne richiede l'esecuzione al basicrobot.

Architettura Logica

Si delinea quindi la seguente architettura logica del sistema:

Test Plan



→ basicrobot: si può fare riferimento al testBasicrobot.kt, nel progetto *it.unibo.qak20.basicrobot/test.*

→ waiterwalker: Dai TestPlan elencati nell'overview iniziale era emersa la necessità di **tenere traccia degli spostamenti del robot e della sua posizione attuale.**

Quindi ho introdotto *due stati logici in cui può trovarsi il waiterwalker* :

-movingTo(cell(X,Y)) -at(cell(X,Y))

Possiamo testare lo stato logico trattando il waiterwalker come risorsa osservabile Coap:

```
fun checkResource(value: String){
  if( waiterWalker != null ){ assertTrue(waiterWalker!!.geResourceRep() == value)}
}
```

Si veda testWaiterWalker.kt nel progetto *it.unibo.iss.sprint_1/test.*

Progetto

Utilizzerò il seguente software sviluppato nel corso delle lezioni e perfettamente in linea con le analisi svolte:

-*it.unibo.qak20.basicrobot*

- Può ricevere dei Dispatch (per i vari comandi elementari w | s | l …) e delle richieste *step*.
- È in grado di emettere eventi *obstacle* quando il supporto di basso livello rileva una collisione.
- Interagisce con un supporto *robotSupport.kt* che funziona da adapter verso il robot (virtuale o fisico) dopo aver configurato opportunamente il file di configurazione basicRobotConfig.json
- Sono presenti nelle resources alcune console in grado di inviare comandi al basicrobot tramite protocolli TCP, MQTT e COAP.

-it.unibo.planner20-1.0.jar

- È una libreria nata come supporto alla pianificazione di azioni di un robot che lavora in una mappa composta da celle quadrate grandi quanto il robot stesso.
- Sfrutta algoritmi di intelligenza artificiale della libreria AIMA_3.0

-waiterwalker.qak:

- Riceve delle richieste *movetoCell(X,Y),* pianifica una sequenza di mosse per raggiungere la posizione goal a partire dalla posizione corrente e le comanda al *basicrobot* con delle richieste step. Una volta raggiunta la posizione risponde con *atCell(X,Y)*. Se qualcosa va storto risponde con *walkBreak*.
- Per pianificare le azioni e gestire la roomMap sfrutta la libreria *it.unibo.planner20-1.0.jar.*