**Assunzioni**

1. Un passo di rbr corrisponde al suo spostamento in una direzione che dura tanto tempo quanto serve per coprire una distanza pari alla sua lunghezza: in questo modo, **un passo di rbr equivale perfettamente ad una cella.**
2. **Il pavimento della stanza è diviso in celle della stessa dimensione di rbr.**
3. **Le celle sono quadrate per una questione di simmetria**.
4. **La stanza risulta essere 8x6 compresi i muri, quindi 5 righe (asse Y verticale) e 7 colonne (asse X)**.
5. Di seguito, abbiamo rappresentato le dimensioni di una stanza che supponiamo essere nota:

X

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0,0 | 1,0 | 2,0 | 3,0 | 4,0 | 5,0 | 6,0 |  |
| 0,1 | 1,1 | 2,1 | 3,1 | 4,1 | 5,1 | 6,1 |  |
| 0,2 | 1,2 | 2,2 | 3,2 | 4,2 | 5,2 | 6,2 |  |
| 0,3 | 1,3 | 2,3 | 3,3 | 4,3 | 5,3 | 6,3 |  |
| 0,4 | 1,4 | 2,4 | 3,4 | 4,4 | 5,4 | 6,4 |  |
|  |  |  |  |  |  |  |  |

Y

1. Qualora le dimensioni della stanza non fossero note, abbiamo stabilito che **gli elettrodomestici si trovano ad una distanza fissa dagli angoli**. Dato che gli elettrodomestici non occupano nessuna cella della stanza, abbiamo salvato come loro posizione la cella di accesso.

Es. se la stanza è lunga 10 celle o 5, fridge sarà sempre nella penultima cella, quindi nella cella 9 o nella cella 4. Siccome l’origine parte da (0,0), alle nostre coordinate dobbiamo sottrarre un ulteriore valore.

* + Fridge: X-2, 0 in generale (6,0 nel nostro caso)
  + Pantry: 0, Y-2 in generale (0,4 nel nostro caso)
  + DishWasher: X-2, Y-2 in generale (6,4 nel nostro caso)

1. Siccome table è centrale rispetto alla stanza, ma la stanza può non essere un quadrato equilatero, **abbiamo supposto** **che** quantomeno **table si trovi ad una distanza di almeno una cella su tutti i lati** e cioè che non abbia nessun lato adiacente ad un muro.

**Il table ha dimensione quadrata, non ha forme irregolari e occupa 4 celle.**

1. Qualora la posizione del table non fosse nota, è necessario pensare ad un algoritmo di esplorazione che ne trovi le coordinate.
2. **Le celle riguardanti gli elettrodomestici sono quelle che rbr può considerare come mete**, mentre per raggiungere il table, rbr ha 8 celle possibili e calcolerà il percorso migliore in base a dove si trova.
3. **La posizione di rbr** è rappresentata da una coordinata X, da una coordinata Y e da una direzione che può essere su, giù, destra e sinistra.
4. **Abbiamo supposto che non ci siano spostamenti accidentali del table**.
5. **Abbiamo stabilito che basta che una cella sia in qualche modo occupata, indipendentemente dalla percentuale di occupazione, e quella cella deve essere marcata come non accessibile.**
6. Il fridge è smart, quindi è in grado di esporre il suo contenuto e di rispondere a domande su di esso. Per comunicare utilizza il protocollo CoAP.

Pantry e dishwasher non sono smart, ma non escludiamo che possono diventarlo in futuro.

1. Il fridge e rbr sono in contesti differenti.
2. **Supponiamo che il rbr abbia capacità infinita** e che quindi con un solo viaggio possa portare tutto il necessario da un elettrodomestico/mobile all’altro.
3. In pantry ci sono solamente oggetti, mentre nel fridge ci sono solamente cibi.
4. Con il comando “prepare”, rbr porterà gli oggetti dal pantry al tavolo e successivamente porterà i cibi dal fridge al table (il set di oggetti e cibi da mettere in fase di preparazione è fisso). Al termine, rbr tornerà in RH.
5. Durante il party alcuni cibi termineranno in modo casuale simulando lo svolgimento dell’evento.

Con il comando “add food”, rbr verifica se il cibo è disponibile nel fridge tramite il suo food-code. Se disponibile, lo aggiunge al table e torna in RH, altrimenti avvisa il maître.

1. Con il comando “clear”, rbr prenderà tutto il cibo rimasto nel table per rimetterlo in fridge e metterà tutti gli oggetti nel dishwasher. Al termine, rbr tornerà in RH.
2. Il **maître sa se un determinato cibo è in fridge** perché può vederne il contenuto in qualsiasi momento. Nonostante questo, **può comunque chiedere a rbr di aggiungere un cibo non presente in fridge:** quindi rbr deve controllare che il cibo esista prima di andarlo a prendere.
3. Il **maître** **sa in ogni momento dove si trovano gli oggetti e i cibi.** Il cibo che non è in fridge e non è sul table, sappiamo che è sul rbr; stessa cosa per gli oggetti che non sono né in pantry né sul table.
4. Rbr può essere fermato in qualsiasi momento dal maître e successivamente supponiamo che sia in grado di poter riprendere da dove era stato interrotto grazie all’aiuto di un pianificatore nel quale memorizziamo il goal corrente e il percorso per arrivarci.
5. Lo stato iniziale della stanza è il seguente:
   * Il **maître** è un **ostacolo sempre presente fisicamente** nella stanza.
   * Non ci sono altri ostacoli mobili.
   * Il fridge e pantry contengono oggetti e cibi prefissati.
   * Dishwasher e table sono vuoti.
   * Rbr è in RH.
6. Escludendo la fase di esplorazione della stanza nel caso in cui non sia nota, durante l’esecuzione di tutto il programma rbr deve evitare gli ostacoli mobili; tali ostacoli possono essere incontrati SE E SOLO SE sta compiendo un passo in avanti.

Ogni volta che rbr incontra un ostacolo, applica la seguente strategia: si ferma, attende X secondi e ritenta.

**Assunzioni di sviluppo**

* **Per effettuare i test**, abbiamo inserito due logger: uno per verificare gli spostamenti di rbr e uno per mantenere le modifiche dello stato della stanza.

**FileLog**

**Per evitare problemi di concorrenza, abbiamo deciso di utilizzare un unico file di log comune a tutti gli attori**. Ciò ci permette anche di vedere cronologicamente quali attori fanno quali azioni.

**Ad ogni avvio del programma, viene creato un nuovo file di log**.

**Struttura del file di log:**

* Ogni volta che viene pianificato un nuovo obiettivo, lo scriviamo nel file.

Esempio: GOAL: RH | 0, 0.

* Ogni volta che viene raggiunto un obiettivo, lo scriviamo nel file.

Esempio: GOAL REACHED: RH | 0, 0.

* Tutte le altre righe sono così composte:

***“Nome dell’attore | Vecchia posizione del robot | Mossa da fare | Nuova posizione del robot”.***

**FileLogKb**

**Per evitare problemi di concorrenza, abbiamo deciso di utilizzare un unico file di log comune a tutti gli attori**. Ciò ci permette anche di vedere cronologicamente quali operazioni vengono fatte sui cibi e sugli oggetti.

**Ad ogni avvio del programma, viene creato un nuovo file di log con lo stato della stanza**.

**Struttura del file di log:**

* Ogni volta che viene aggiunto un oggetto, lo scriviamo nel file.

Esempio: +.OT.knives.

* Ogni volta che viene rimosso un oggetto, lo scriviamo nel file.

Esempio: -.OR.knives.

* Ogni volta che viene aggiunto un cibo, lo scriviamo nel file.

Esempio: +.FD.wurstel,c001,19.

* Ogni volta che viene rimosso un cibo, lo scriviamo nel file.

Esempio: -.FD.wurstel,c001,19.

* Alla ricezione di ogni comando, rbr scrive il task corrente.
* Al termine del programma, viene scritto lo stato della stanza.
* Quando un messaggio non deve avere un contenuto utile, scriviamo nella dichiarazione: Dispatch nomeMessaggio : nomeMessaggio (N)
* Tutti i nomi dei contesti sono fatti in questo modo: ctxNomecontesto o ctxDummyForNomecontesto per gli attori esterni.
* Tutti i nomi delle variabili sono in inglese e tutti i commenti sono in italiano.
* onMsg non funziona se il messaggio non ha nulla nel payload, perciò, anche per una questione di sicurezza abbiamo creato alcuni messaggi contenenti il nome dell’attore mittente: in questo modo non solo possiamo usare la onMsg, ma siamo anche sicuri che il messaggio sia proprio quello che vogliamo gestire noi.
* In ogni stato se abbiamo diversi payload nel messaggio abbiamo subito gestito tale messaggio, anche a costo di rendere ripetitivo il codice.
* RoomButler creato nello sprint 6 seguendo l'architettura master/slave: il master è una simulazione del frontend e dà gli ordini, lo slave, cioè il nostro Room Butler, è quello che dice agli altri attori cosa fare
* Javascript: controllo lato inserimento della stringa ben formata per il codice del cibo cxxx. Se non corretto, appare una modal.
* Sprint8: Noi abbiamo previsto una funzione di controllo della direzione statica basata sulle celle che sappiamo essere quelle attorno al tavolo. Lo sappiamo perché conosciamo le dimensioni della stanza e la posizione del tavolo.

Eventualmente una estensione dinamica potrebbe operare come segue:

1. con una solveTable(L) ottengo la lista delle celle accessibili
2. ordino le celle in base alla coordinata X e a parità di coord. X, in base alla coord. Y
3. divido le celle in 4 gruppi:

* un gruppo avrà la stessa x1
* un gruppo avrà la stessa x2 ma diversa dalla x1
* un gruppo avrà la stessa y1
* un gruppo avrà la stessa y2 ma diversa dalla y1

1. a questo punto assegno la direzione:

* le celle con stessa x1 < x2 avranno direzione destra
* le celle con stessa x2 > x1 avranno direzione sinistra
* le celle con stessa y1 < y2 avranno direzione giù
* le celle con stessa y2 > y1 avranno direzione su

Es. Celle: 41, 51, 62, 63, 44, 54, 32, 33

Gruppo 1: 32, 33 --> stessa x1, direzione destra

Gruppo 2: 62, 63 --> stessa x2, direzione sinistra

Gruppo 3: 41, 51 --> stessa y1, direzione su

Gruppo 4: 44, 54 --> stessa y2, direzione giù

* Sprint 8: abbiamo inserito un attore “greedy” che mangia un cibo per volta ogni 10 secondi, simulando il vero e proprio buffet. Attualmente greedy dice alla kb che vuole mangiare un determinato cibo: se la kb riesce a fare la solve, lo toglie dal tavolo, altrimenti non succede nulla. Una possibile estensione potrebbe essere la seguente:
  + Greedy dice alla kb che vuole mangiare un determinato cibo
  + Se il cibo c’è, si procede come prima.
  + Se il cibo non c’è, si fa una richiesta al master che appare sul frontend con scritto “Puoi aggiungere questo cibo nel buffet?”, con un popup o qualcosa del genere.
    - Se il maitre non accetta, non succede niente.
    - Se il maitre accetta, si scatena automaticamente l’addfood senza che il maitre debba scrivere il codice nell’input text e prema addFood: basta che prema OK nel popup.
* Per aiutare il maître nell'inserimento del codice lato frontend, una estensione futura è quella di una casella dotata di autocompletamento.
* Per popolare la lista di codici da suggerire, si potrebbe anche utilizzare un file.