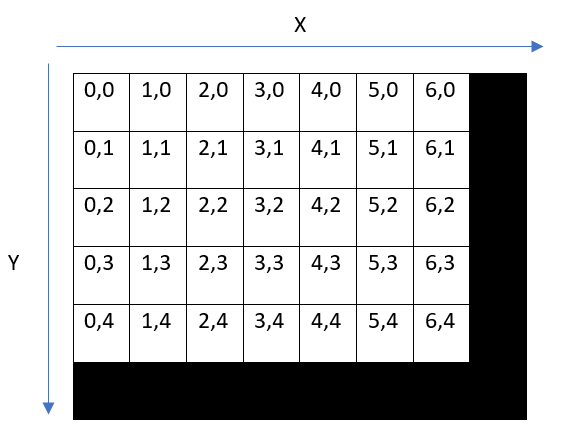
# **ANALISI DEI REQUISITI**

Il requisito fondamentale per poter affrontare la soluzione del problema presentatoci dal cliente è che il ddr robot **“rbr”** si muova all’interno di un perimetro. Lo sprint 0 nasce per esaurire l’esigenza di poter rappresentare tale robot.

# **ANALISI DEL PROBLEMA**

Per soddisfare il requisito precedente, ci sono diversi problemi da analizzare. A tale scopo si evidenziano le successive note:

1. Il **robot** è dotato di una specifica **lunghezza e larghezza**, che si suppone siano **uguali**.
2. Un **passo del robot**, detto anche rbr, corrisponde al suo spostamento in una direzione che dura tanto tempo quanto serve per coprire una distanza pari alla sua lunghezza.
3. Il customer ci ha fornito un **ambiente virtuale** in cui coesistono il robot e una **stanza già preconfigurata**. Tale stanza ha anch’essa una specifica lunghezza e larghezza.
4. Le proprietà dimensionali simmetriche del robot e la composizione spaziale della stanza richiedono di trovare un **sistema di riferimento appropriato** per poterne ricavare le **coordinate**. Viste le dimensioni prefissate del robot, è utile dividere la stanza in **celle quadrate** grandi quanto il robot stesso per agevolare gli spostamenti e la loro rappresentazione.
5. **Un esempio** di come può essere suddivisa la stanza è il seguente**: la stanza risulta essere 8 celle in larghezza e 6 celle in lunghezza compresi i muri, quindi ha 5x7 celle accessibili.** Nel nostro SdR l’asse Y è verticale, l’asse X è orizzontale e l’origine (0,0) è in alto a sinistra.



1. Le dimensioni della stanza sono una proprietà essenziale del “**modello della stanza**” così come la posizione e la direzione sono caratteristiche peculiari del “**modello del robot**”. Risulta quindi necessario rappresentare tali proprietà in una base di conoscenza: una per la stanza e una per il robot.
2. Nel modello del robot la sua **posizione** è rappresentata da una **coordinata X,** da una **coordinata Y** e **da una direzione** che può essere su, giù, destra e sinistra. Tale posizione deve essere mantenuta come “**stato del robot**”. È evidente la necessità di avere un’entità che **si muove** (corpo) e al tempo stesso mantiene il suo **stato** (mente) seguendo la **filosofia mente-corpo**.
3. **Un passo di rbr** equivale perfettamente allo spostamento dalla **cella di partenza** alla **cella di arrivo.**

**Ipotesi di Product Backlog:**

1. Redazione del work plan preliminare
2. Creazione del metamodello/modello eseguibile
3. Creazione Knowledge Base della room
4. Pianificazione di un test plan
5. Implementazione ed esecuzione del test

**Test Plan:**

In generale ci aspettiamo che quando il robot si sposta da A a B che siano soddisfatte le seguenti condizioni:

* **Pre condizioni**: il robot è fermo e si trova nella posizione A = (X1, Y1) con direzione D1.
* **Post condizioni**: il robot è fermo e si trova non più in A, ma nella posizione B = (X2, Y2) con direzione D2.

Nello specifico, in questo primo Test Plan vogliamo verificare che:

* A = (0,0) con direzione sud
* B = (2,1) con direzione sud

Lo spostamento che effettueremo è determinato da una sequenza di mosse decisa a priori.

# **PROGETTO**

Partendo dall’ipotesi di Product Backlog e da quanto scritto in precedenza, ci sono ovviamente diverse possibilità di realizzazione. Noi, in particolare, abbiamo scelto di adottare **l’infrastruttura QAK** realizzata tramite un **internal DSL**.

Infatti, grazie all’utilizzo di un internal DSL è possibile realizzare programmi concisi e analizzabili auto-esplicativi che disincentivano l’ambiguità della comunicazione con il cliente.

Nello specifico utilizzare l’infrastruttura QAK ha diversi vantaggi:

* Introduce un **metamodello formale** all’interno del quale possiamo esprimere la **struttura**, **l’interazione** e il **comportamento** del sistema
* Include la capacità di gestire **azioni reattive/proattive**
* Implementa una soluzione basata sugli attori: gli **attori** sono entità che lavorano come **automi a stati finiti** e hanno una propria **base di conoscenza** (estendibile dinamicamente e realizzata in Prolog). Possiedono anche una propria capacità computazionale e una **coda** in entrata per accumulare messaggi in arrivo
* Supporta la **distribuzione in sistemi eterogenei**

Aggiungiamo di seguito **alcune convenzioni** che abbiamo deciso di utilizzare come **Best Practice** per agevolare la suddivisione del lavoro e il conseguente merge dei progetti:

* Quando un **Dispatch** 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 separatamente, anche a costo di rendere ripetitivo il codice.