Università degli Studi di Palermo

Facoltà di Ingegneria - Corso di Laurea in Ingegneria Informatica

TESINA DI ROBOTICA COGNITIVA

PROF. ANTONIO CHELLA
PROF. ROSARIO SORBELLO

A.A. 2003/2004

Allievi:

Maggio Paolo **Marcenò Leonardo** **Messina Antonio**

INDICE

Introduzione

La seguente tesina, presentata a corredo dell'implementazione software di un robot per la partecipazione alla Gara del corso di Robotica A.A. 2003-2004 tenuto presso la Facoltà di Ingegneria di Palermo, ne costituisce parte integrante per la piena comprensione delle scelte di realizzazione effettuate.

La nostra scelta è stata dunque quella di presentare in questa documentazione una cospicua parte di commenti riguardo le classi Java e i files Prolog consegnati nelle cartelle TRY_IT_LEFT e TRY_IT_RIGHT, ed inoltre offrire una panoramica sui principi teorici guida.

Nella prima parte ci siamo volutamente soffermati sugli innegabili vantaggi e gli importanti consensi che la Programmazione Logica ha riscosso nel mondo della Robotica.

Quindi successivamente l'attenzione si concentrerà sulla nostra strategia di gara ma non prima di aver delineato le differenze tra le varie tipologie di architetture robotiche e, avendone abbracciata una (la filosofia delle architetture ibride), aver dato la necessaria motivazione della scelta.

La quarta sezione è riservata, come detto sopra, ai commenti al codice. Abbiamo ritenuto utile, ai fini della chiarezza, soffermarci singolarmente su ogni classe e di ognuna tracciare le caratteristiche fondamentali attraverso l'analisi di parti del codice opportunamente riportate.

Infine, a conclusione di questa documentazione, è stato inserito il codice completo di tutte le classi del robot.

Programmazione Logica e Robotica

In questa sezione vogliamo occuparci di esporre quali siano state e siano tuttora le motivazioni che abbiano spinto molti laboratori di ricerca di Robotica ad un utilizzo sempre più preponderante di linguaggi ispirati alla programmazione logica.
Questo interesse crescente verso i linguaggi derivati dal Prolog deve essere ricercato nella natura stessa degli studi da condurre. Al fine di delineare a fondo le motivazioni di questa scelta sarà bene fare un passo indietro e analizzare innanzitutto Prolog.
La programmazione logica nasce intorno ai primi anni ’70 per merito di alcuni ricercatori francesi (Marsiglia), i quali in modo particolare introducono un’interpretazione procedurale delle clausole della logica che per la prima volta consente di ridurre il processo di dimostrazione di un teorema ad un tradizionale processo di computazione di un linguaggio di programmazione. Nel 1972 a Marsiglia, il gruppo di ricercatori di Alain Colmerauer mette a punto il primo interprete Prolog (PROgrammazione LOGica) della storia. La differenza sostanziale tra la programmazione logica e i normali linguaggi che seguono il paradigma imperativo (C, Pascal,…) è che in Prolog i problemi vengono descritti in maniera molto astratta grazie al ricorso alle formule della logica. Al contrario nei tradizionali linguaggi imperativi il programmatore fornisce un insieme di istruzioni che descrivono dettagliatamente le operazioni da eseguire in run-time per la risoluzione del problema proposto.
Già a questo punto siamo in grado di individuare i punti di forza della scelta di Prolog e dei suoi derivati (in modo particolare nel nostro caso sGolog) in robotica. In effetti è caratteristica fondamentale in ingegneria mantenere un solido contatto con la realtà: un robot è progettato per muoversi all’interno di un “suo mondo” (che può essere strutturato o meno), ed è chiaro che data l’imprevedibilità delle situazioni è pressoché impossibile per un programmatore poter prevedere ogni singola situazione che può presentarsi durante l’espletamento di un particolare compito. Di conseguenza è evidente che consentire al progettista di affidare al robot stesso il compito di scegliere l’azione giusta da compiere in un particolare momento è senza dubbio una prospettiva allettante oltre che notevolmente consigliabile, anche perchè se è possibile prevedere il verificarsi o meno di una situazione in certe condizioni ideali non è detto che poi lo scontro con la realtà produca i risultati voluti. Quanto appena detto corrisponde alla contrapposizione della programmazione logica da una

parte e dei linguaggi imperativi dall'altra. Infatti la prima consente di definire azioni e comportamenti complessi ad un livello molto alto di astrazione lasciando al calcolatore il compito di tradurre il tutto in azioni elementari; i linguaggi come C o Pascal invece richiedono di fornire "a priori" un set di istruzioni da applicare per ogni potenziale evento.

Il Golog

L'avvio della stagione della programmazione logica, fin da subito ben accetta negli ambienti di ricerca di Intelligenza Artificiale, ha portato allo sviluppo di linguaggi direttamente derivati dal Prolog ma che pongono attenzione a particolari aspetti dettati dalla ricerca. Durante il nostro corso di Robotica A.A. 2004 tenuto dal Prof. Antonio Chella, ci siamo particolarmente interessati del Golog (alGOl in LOGic).
Il Golog è un linguaggio che combina la potenza del Situation Calculus con le tradizionali strutture di controllo dei linguaggi di programmazione. Tale linguaggio può essere uno strumento potente nelle mani del progettista se usato sapientemente: in effetti, una volta specificato un modello del mondo in una situazione iniziale S_0, e una serie di primitive actions (che corrispondono alle azioni che è lecito compiere) e conseguentemente il risultato del loro verificarsi a discapito della situazione iniziale (ovvero quali modifiche hanno apportato al mondo) il Golog ritorna un piano (una sequenza di azioni legali) che consente di passare dalla situazione iniziale ad una situazione finale. E' bene notare che l' interprete Golog lavora in modalità off-line. Questo significa che l'interprete "valuta" quali effetti potrebbero essere apportati dal verificarsi di un'azione piuttosto che un'altra, e viene fornita una lista di azioni immediatamente eseguibili solo se il piano esiste. Tuttavia questo risulta uno dei grossi limiti di Golog, dal momento che alcune delle azioni che non risultano essere legali in modalità off-line potrebbero diventarlo in run-time. A sua volta esistono molti derivati di Golog stesso.
Fra tutti citiamo:

- Congolog: tale estensione arricchisce Golog di aspetti legati alla presenza di processi concorrenti e alla possibilità di ricorso agli interrupt;

- Indigolog: questo è il solo tra le versioni Golog che presenta una versione on-line (oltre a quella off-line). Indigolog rigetta dunque il meccanismo di backtracking tipico della programmazione logica;
- sGolog: questa versione arricchisce il Golog dal momento che consente il sensing delle azioni e l'interprete in output non produce situazioni bensì CAT (Conditional Action Tree).

Più avanti offriremo una panoramica sul linguaggio Golog "vero e proprio" e del suo derivato sGolog, al cui utilizzo è stato fatto ricorso per la stesura del codice del nostro robot.

✓ *Il Situation Calculus*

Il Situation Calculus è un linguaggio che fornisce un'integrazione ben riuscita tra ragionamento, percezione ed azione. In questo linguaggio (basato sulla logica dei predicati) tutti i termini presenti possono essere di tre soli tipi:

1. Oggetti ordinari;
2. Azioni;
3. Situazioni.

Inoltre S_0 è la costante che indica la situazione iniziale, momento in cui non è ancora intervenuto il verificarsi di nessuna azione. È definita una funzione binaria del linguaggio ***do(a, s)*** che indica lo stato successivo allo stato s dopo il verificarsi dell'azione a.
Sono inoltre definite delle relazioni il cui valore varia da situazione in situazione. Essi possono essere di due tipi: fluenti relazionali e fluenti funzionali. Un altro importante predicato è **Poss(a, s)** che fa rilevare la possibilità di eseguire l'azione **a** allo stato **s**.
A questo punto definiamo il concetto di **Basic Action Theory:** questa è un insieme di teorie che consentono di delineare quali siano i cambiamenti nel mondo avvenuti a seguito delle azioni compiute.
All' interno della BAT possiamo distinguere:

- precondizioni: mi indicano quando è possibile che le azioni cui sono associate possano essere realmente eseguite;

- assiomi: mi descrivono la situazione S_0;
- assiomi di nome unico per le azioni: mi garantisce che azioni aventi nome diverso sono azioni differenti;
- assiomi di stato successore: uno per ogni fluente F. Mi specifica quali azioni hanno effetto ed in quale modo.

✓ *sGolog*

Dopo questi brevi richiami sulle basi teoriche del Situation Calculus possiamo analizzare più a fondo il Golog. Esso unisce alla libertà espressiva fornita dalla logica dei predicati, la possibilità di definire azioni complesse utilizzando le strutture di controllo tipiche dei tradizionali linguaggi di programmazione.

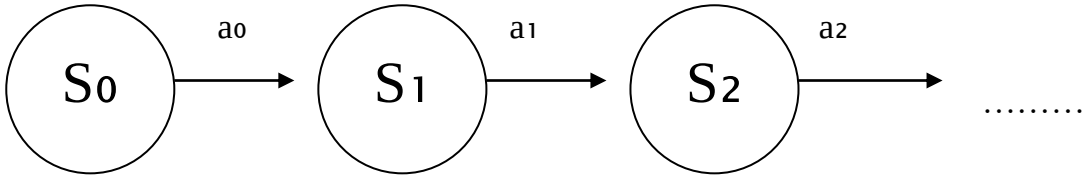
• A	primitive action
• ϕ?	testa una condizione
• (ρ_1;ρ_2)	sequenza
• (ρ_1|ρ_2)	non determinismo di azioni
• (πx. ρ)	non determinismo di argomenti
• ρ*	non determinismo di ciclo
• if ϕ then ρ_1 else ρ_2 endif	condizione
• while ϕ do ρ endwhile	ciclo
• proc ρ(x) endproc	procedura

Come si evince da una prima lettura dei costrutti base del linguaggio, nota caratteristica del Golog è il non determinismo. Questa peculiarità consente all'interprete di seguire un ordine di esecuzione dei programmi che non è dettato da nessun fattore prestabilito; se n esecuzioni sono parimenti attuabili Golog sceglie casualmente. In modo particolare il non

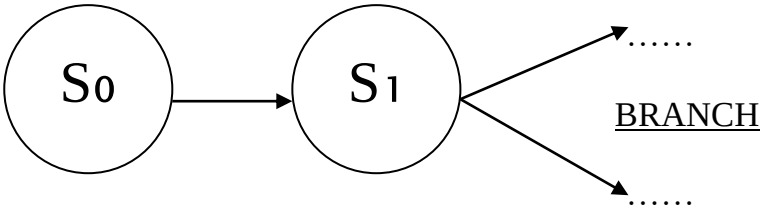
determinismo di ciclo consente al programmatore la libertà di non fornire una condizione di uscita, la quale sarà elaborata da Golog stesso.
Per quanto detto è possibile dunque affermare che il Golog, seguendo la logica fornita dal situation calculus, si muove nell'esecuzione del programma seguendo un piano in cui, individuato uno stato k e una sequenza di azioni che dallo stato k portano allo stato k+1, quest'ultimo è univocamente e completamente individuato. A questo punto possiamo introdurre sGolog.
Come detto sopra sGolog introduce il sensing al Golog: da un punto di vista tecnico vengono introdotte due nuove azioni che testano il valore del fluente:

- sense(x);
- branch_on(x).

Il valore di x deve essere fornito (ad esempio tramite rilevazione da parte di sensori).
L'azione di branch_on(x) ha come risultato una ramificazione (Conditional Action Trees) dovuta al test effettuato da sense(x): l'esecuzione del programma segue l'una o l'altra ramificazione a seconda che x abbia valore vero o falso. In simulazione, i valori vero o falso vengono assegnati ad x tramite le azioni **assm(x, 0)** e **assm(x, 1)**.
È evidente a questo punto la differenza sostanziale tra un programma Golog ed uno sGolog:



Nel situation calculus ci si muoveva su un piano;



SENSE

In sGolog individuo un albero (albero condizionale).

In ultima istanza si può sottolineare come la pianificazione in sGolog fondamentalmente differisca da quella in Golog solo per la presenza dei CAT. Dunque il **Do (a, s)** “ereditato” dal situation calculus nel Golog, adesso viene convertito in **Do (a, s, c)**, che indica di eseguire **a** nello stato **s** per produrre il CAT **c**.

La Strategia - Premessa

Nelle pagine seguenti spiegheremo quale sia stato il nostro approccio nella stesura del codice di implementazione del lato SW di un robot per la partecipazione alla gara di fine corso di Robotica Cognitiva.
Per una corretta visuale sul nostro lavoro sarà necessario però soffermarsi innanzitutto su quali siano stati i presupposti teorici che ci hanno guidato nelle scelte effettuate.

Architetture Robotiche

Essenzialmente il panorama della ricerca robotica mondiale vede contrapposte due diverse correnti che negli anni hanno concretizzato il loro lavoro in tre diverse tipologie di architetture robotiche:

- ✓ Architetture Deliberative;
- ✓ Architetture Reattive;
- ✓ Architetture Ibride.

In questa sede è nostro interesse fornire i concetti basilari su cui si basano le diverse filosofie ed in particolare, dalla loro analisi contrapposta, fare emergere il principio delle **Architetture Ibride.**

✓ Architetture Deliberative

L' architettura deliberativa si basa sul principio di voler dotare la creazione robotica di una capacità di ragionamento propria, che la possa rendere in grado di sapersi districare autonomamente in ogni situazione in cui possa venirsi a trovare. I progettisti di questo tipo di architettura fanno un grande ricorso alla necessità di fornire al robot un modello completo del mondo entro cui è destinato a muoversi. Dunque si può già notare uno dei punti deboli di questa prima tipologia di architettura: **il robot dipende completamente**

dall'accuratezza del modello del mondo che gli viene fornito. Questo vuol dire che se per una qualsiasi ragione il modello su cui il robot basa la sua pianificazione dovesse mutare nel tempo, l'entità robotica si troverebbe nell'impossibilità totale di riuscire a portare a termine i suoi compiti.

L'architettura deliberativa si basa pesantemente come già detto su un alto livello di "intelligenza" del sistema robotico: ma questo implica una notevole **complessità realizzativa** del sistema stesso, e soprattutto **tempi di risposta molto alti** agli stimoli esterni, che se per certe azioni sono tollerabili, per molte altre diventano troppo dispendiosi se non addirittura pericolosi (situazioni di pericolo). Ad ogni modo la prospettiva delle architetture deliberative è molto interessante proprio perché si pone come obiettivo la creazione di un sistema artificiale che a livello intellettivo sia molto vicino all'essere umano.

✓ Architetture Reattive

Le architetture reattive stanno riscontrando molto più successo nei centri di ricerca di robotica. Degno di nota è anche il fatto che due tra i più grandi ricercatori di robotica a livello mondiale (Brooks e Arkin) basano da anni i loro studi su questa tipologia di architettura. E infatti i modelli che maggiormente influenzano le ricerche in questo campo derivano proprio dai due scienziati di cui sopra:

- Subsumption Architecture (dovuta a Brooks);
- Motor Schemas (dovuta a Arkin).

I sistemi robotici di tipo reattivo sono progettati per potersi muovere all'interno di un ambiente pur **non avendo nessuna cognizione a priori**.

Tale libertà da un modello spaziale fornito si traduce d'altra parte in una maggiore **semplicità computazionale** e la possibilità di **associare una reazione immediata ad ogni stimolo esterno senza la necessità di complicate elaborazioni logico-matematiche.** Quanto appena detto costituisce la base concettuale delle architetture reattive: vengono individuati dei comportamenti di base, ognuno dei quali basandosi sull'analisi dei dati sensoriali elabora una risposta adeguata. L' unione dei diversi comportamenti (**modularità implementativa)** permette al robot di muoversi liberamente.

✓ Architetture Ibride

Questa tipologia di sistemi robotici si impone per la sua sapiente fusione delle caratteristiche peculiari di entrambe le architetture precedentemente viste.
In effetti, assodati gli svantaggi delle architetture deliberative, è innegabile che, se l'ambiente d'azione di un agente robotico non è sottoposto a notevoli mutamenti nel tempo, la presenza di un piano di azione preciso ed efficiente è di gran lunga preferibile rispetto all'imprevedibilità offerta da un sistema reattivo. Questo è solo uno dei punti di vista che rendono i sistemi deliberativi e quelli reattivi completamente affidabili solo in alcune circostanze: al contrario la possibilità di poter usufruire dei vantaggi di entrambe le architetture pone sicuramente l'accento sulla necessità di implementare dei sistemi che sappiano fare ricorso in maniera ottimale ad aspetti ora dell'una ora dell'altra parte.
Lo stesso Ronald Arkin è da anni impegnato in lavori che seguono la filosofia dei sistemi ibridi: un esempio fondamentale in questo senso nel mondo della robotica è stata l'implementazione di un sistema ibrido deliberativo e reattivo (basato sulla teoria degli schemi motori) per l'Autonomous Robot Architecture (AuRA).
In AuRA le due componenti reattiva e deliberativa convivono in maniera originale: all'avvio del sistema è la parte deliberativa (basata sulle tradizionali tecniche di Intelligenza Artificiale) ad assumere il controllo e a fare partire la sequenza di azioni del piano. Nel momento in cui il sistema raggiunge una fase di "tranquillità" il controllo è ceduto alla parte reattiva, chiaramente caratterizzata da una maggiore velocità operativa. Il controllo viene ripreso dalla parte deliberativa solo nel momento in cui dovesse verificarsi un failure, ovvero qualsiasi problema rilevato dalla parte reattiva nel normale svolgimento delle proprie azioni.

La Strategia e il Robot – Analisi

Alla luce di quanto esposto finora possiamo adesso concentrarci sulla strategia a cui abbiamo effettivamente fatto ricorso.
Il nostro robot segue la filosofia di un' architettura ibrida reattiva e deliberativa.

A grandi linee la nostra tattica di gara si basa, al momento dell'avvio, su un tentativo di "furto" di una bandierina per robot (quindi complessivamente due bandierine) in parti del campo di gioco prossime alla zona avversaria. In questo modo diminuiamo la possibilità da parte dell'altra squadra di recuperare bandierine in zone vicine ai loro bin. Questa tattica, significativamente da noi chiamata **Attack,** risulta essere conveniente dal momento che se il tentativo andasse a buon fine (e dato l'alto numero di bandierine presenti in campo il successo di Attack è abbastanza probabile) noi ci troveremmo ad essere in vantaggio rispetto alla squadra concorrente. Infatti oltre ad averle sottratto due bandierine nella "sua" zona, abbiamo anche avuto la possibilità di sondare l'eventuale presenza di bandierine nel percorso effettuato durante il suddetto recupero; questa seconda funzionalità di Attack è di vitale importanza in funzione del resto della strategia di gara. Da notare inoltre che Attack ci consente di essere in vantaggio rispetto all'altra squadra o, al più, di partire in una situazione di parità nel caso in cui anche l'altra squadra adottasse una strategia analoga.
La nostra strategia si basa su un meccanismo di memorizzazione delle coordinate cartesiane che individuano le bandierine rilevate e sullo scambio di informazioni riguardo queste ultime tra i robot. Questo avviene in qualsiasi momento della gara (Attack, esplorazione, consegna bandiere…). Inoltre il robot è dotato di un sistema che gli permette di riconoscere, date la sua posizione istantanea, le posizioni (con riferimento agli assi cartesiani) e le rispettive classi visuali di tutte le bandierine presenti in memoria, la bandierina più vicina contemporaneamente a se stesso e al bin di appartenenza: essa è dunque quella da recuperare immediatamente. È chiaro che il robot è dotato di opportuni check systems che gli consentono di:

1. eliminare dalla memoria una bandierina già recuperata;
2. aggiornare la posizione di una bandierina che a seguito di urti è stata spostata (e che deve dunque trovarsi entro un raggio limitato rispetto alla vecchia posizione).

Inoltre i due robot facenti parte della nostra squadra, come accennato sopra, si scambiano informazioni riguardo la bandierina appena avvistata ovvero recuperata. In questo modo consentiamo ai due robot di avere una **visione globale** del panorama di targets presenti nel campo. Se dunque uno dei due robot si trova in esplorazione in parti di campo prive di bandierine mentre l'altro è localizzato in una zona dove viene rilevata la loro abbondante

presenza, tramite comunicazione si evitano i tempi morti di un'inutile esplorazione. Da notare che se uno dei due robot avvista e recupera una sola bandierina l'altro agente ricevendo il messaggio di avvenuto recupero evita di inoltrarsi in quella parte di campo verosimilmente ormai priva di oggetti.

Veniamo adesso ad uno dei punti più cruciali che una buona strategia di gara deve affrontare prendendo in considerazione la particolare tipologia di mappa usata per la competizione di quest' anno: infatti alcune bandierine sono posizionate in posti che rendono difficoltosa la fase di consegna al bin opportuno.

Cercheremo adesso di esemplificare quanto esposto sopra analizzando direttamente uno dei casi che possono presentarsi:

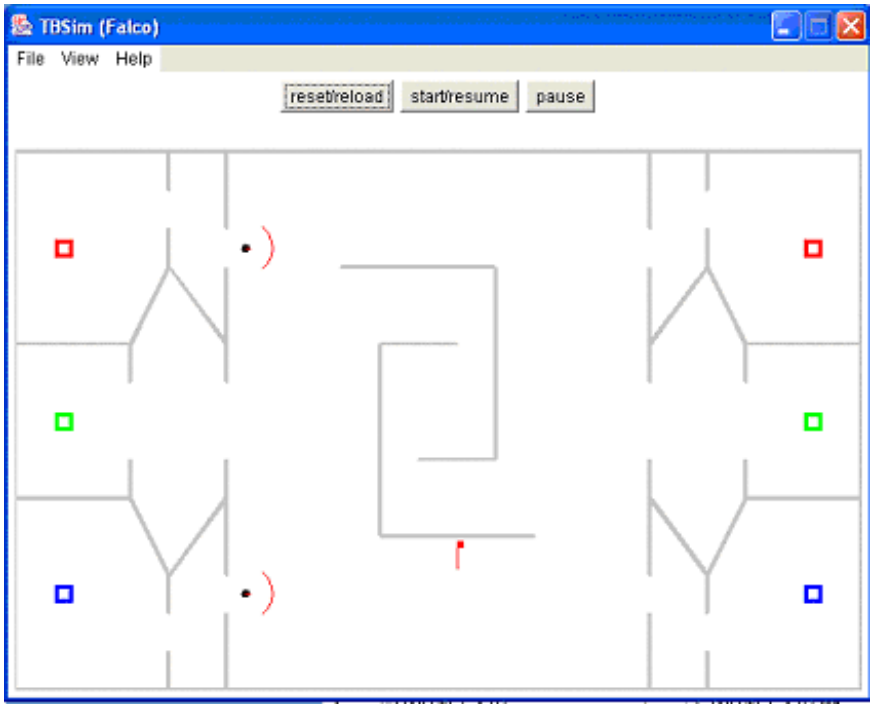


Figura 1.1

Come si vede la bandierina rossa si trova in una posizione troppo prossima alla parete inferiore della chiocciola centrale. È evidente già da una semplice osservazione visiva di Figura 1.1 che indipendentemente da quale robot possa recuperare la bandierina in questione, esso, sistemerà le impostazioni di steering e torretta per dirigersi al suo bin rosso,

che come si vede si trova nella parte superiore sinistra della mappa. Ma così facendo, è chiaro che trovandosi di fronte la parete si bloccherà. Dal momento che stiamo adoperando il ControlSystemMFN150Explore, ovvero la versione del robot dotato di sonar e laser, è lecito supporre che l'unità robotica riuscirà lentamente a superare la situazione di stallo tentando di ricostruire la pianta degli ostacoli che lo circondano. Tuttavia non è dato sapere se e quando ci riuscirà. È evidente che questa situazione, in fase di gara, è inaccettabile.
Noi abbiamo adottato una soluzione che si basa sulla divisione della mappa in zone. In questo modo il robot, riconoscendo la propria locazione all'interno dello spazio e valutando il colore della bandierina in gripper adotta la scelta opportuna per la fase di consegna. Vediamo di chiarire anche questo punto graficamente:

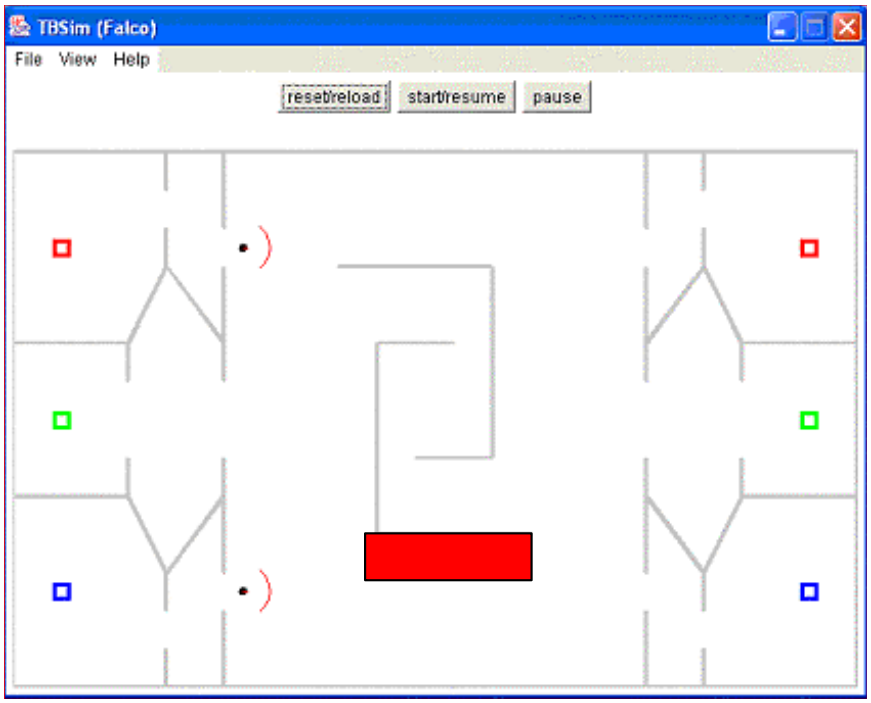


Figura 1.2

Ricollegandoci all'esempio di Figura 1.1, nel momento in cui il robot sa di trovarsi nella zona che in Figura 1.2 abbiamo evidenziato e sa di dover andare a depositare una bandierina di colore rosso, regola l'angolazione del suo movimento verso la zona del bin di un valore θ tale da riuscire a evitare lo scontro con la parete. Questo tipo di approccio merita un

opportuno approfondimento. In effetti la taratura dell'angolo che presiede al movimento del robot non è altro che un'applicazione pratica della teoria dei **campi potenziali,** che del resto già trova applicazione nel TeamBots. Tecnicamente quindi, la scelta verso cui abbiamo optato e che abbiamo delucidato in Figura 1.2, rappresenta un incremento del campo repulsivo esercitato dalla parete sul robot. Infatti, per valori di θ che deviano sufficientemente la traiettoria che il robot tenderebbe a seguire, non facciamo altro che sommare un campo potenziale a quello già esistente le cui linee di forza sono uscenti dalla parete. Secondo questo modello rappresentativo il robot si trova immerso in un campo potenziale dato dalla somma vettoriale dei due campi presenti nello spazio; l'intensità del vettore generato dal campo in ogni punto sarà determinata dalla distanza **d** della parete dal robot, in particolare aumenterà in valore assoluto al diminuire di **d.**

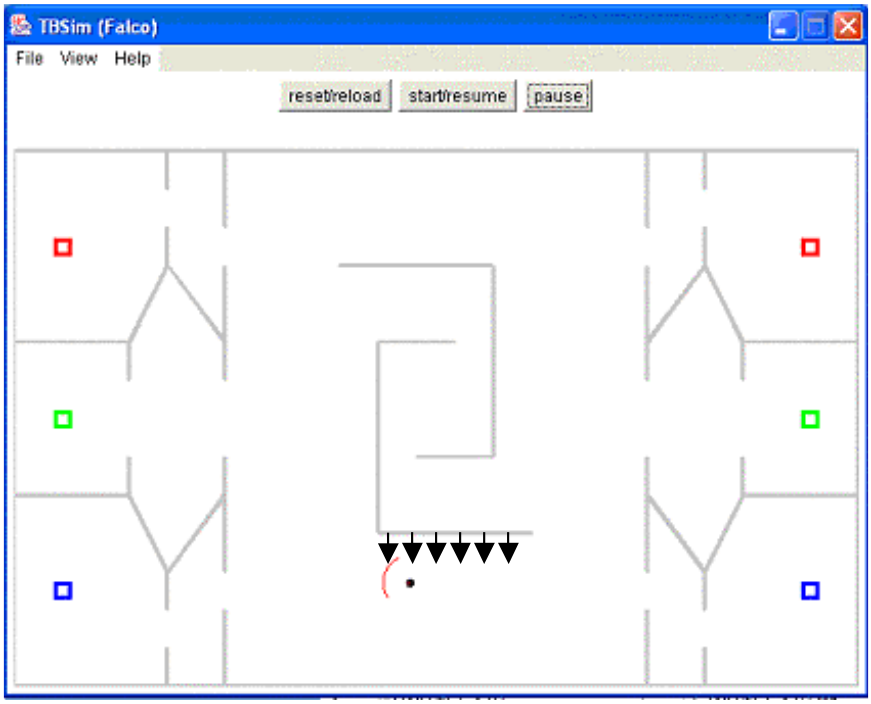
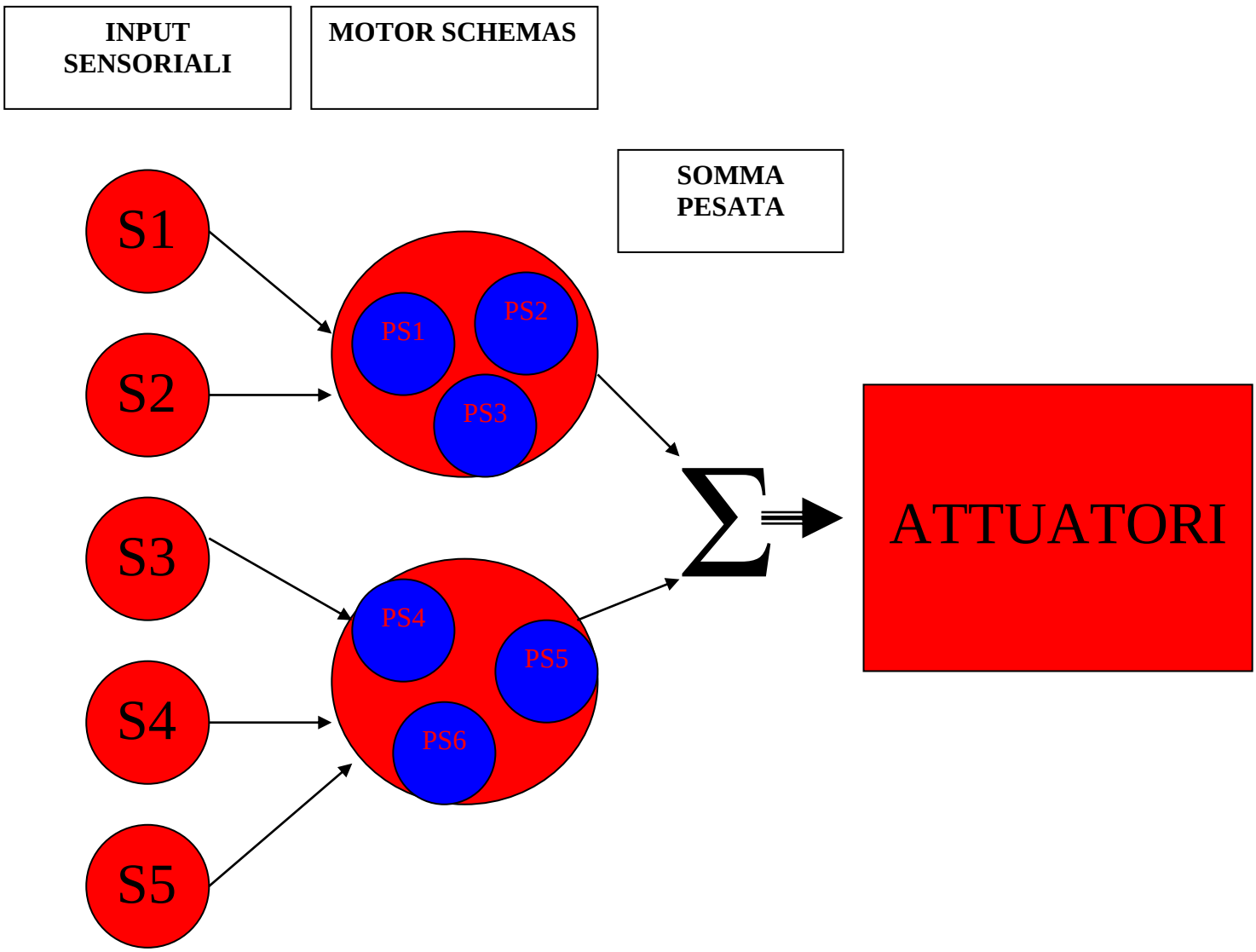


Figura 1.3

Il campo potenziale repulsivo generato dalla parete inferiore della stanza centrale

Chiaramente questa è solo una delle possibili situazioni di stallo di questo genere che possono verificarsi. Noi abbiamo cercato di implementare un sistema che prevenga ogni circostanza analoga. Per ogni eventuale chiarimento e/o precisazione consultare il commento al codice del robot. Questo approccio, come ormai a questo punto risulta chiaro,

segue la teoria dei Motor Schemas elaborata da Ronald Arkin e che Tucker Balch stesso, autore del Team Bots e ricercatore facente parte del gruppo di lavoro di Ronald Arkin, ha eletto a fondamento teorico del suo SW. Infatti alla base dei movimenti simulati dei robot, nel Team Bots, definiamo appunto dei Motor Schemas (**MS**) che non sono altro che istanze di classi Java che "vettorizzano" dei Perceptual Schemas (**PS**). Questi ultimi rappresentano gli input sensoriali che il robot elabora ad ogni istante di tempo. I vettori forniti in output da ogni singolo MS vengono legati per fornire una risposta unica ai motori che renderanno effettiva l'azione da compiere: questo avviene tramite una **somma pesata** dei diversi MS.



Una delle classi, presenti nel package EDU.gatech.cc.is.clay, più usate per istanziare oggetti che nel simulatore rappresentano MS è *v_LinearAttraction_v (double czr, double dzr, NodeVec2 im1)*. Gli oggetti di questa classe sono dei vettori che puntano alla posizione della

mappa, univocamente individuata dalle due coordinate del piano e precisata attraverso il PS im1, che indica la posizione della bandierina da prelevare e la cui intensità dipende dai valori di czr e dzr che vengono opportunamente valutati per determinare se il robot si trova in una zona soggetta all'influenza di un particolare campo potenziale o meno. Il nostro robot fa uso di tale classe ma anche di una sua estensione (vedi codice).

Una situazione analoga alla fase di consegna si presenta nel momento in cui il robot, trovandosi da un lato della chiocciola, ricava dalla memoria una bandierina, vista come più vicina, che si trova dalla parte opposta della chiocciola. Il recupero di quest'ultima in questo caso può rendersi alquanto difficoltoso dato che il robot potrebbe bloccarsi contro una parete nel tentativo di "lasciarsi guidare" dal vettore di attrazione lineare che viene generato. Per questa ragione abbiamo suddiviso la mappa in zone; il robot, riconoscendo la zona in cui si trova esso stesso e calcolando la zona in cui si trova la bandierina sceglie la strategia migliore per l'acquisizione. Le zone di cui abbiamo accennato sono indicate nelle figure 1.4.a -1.4.b:

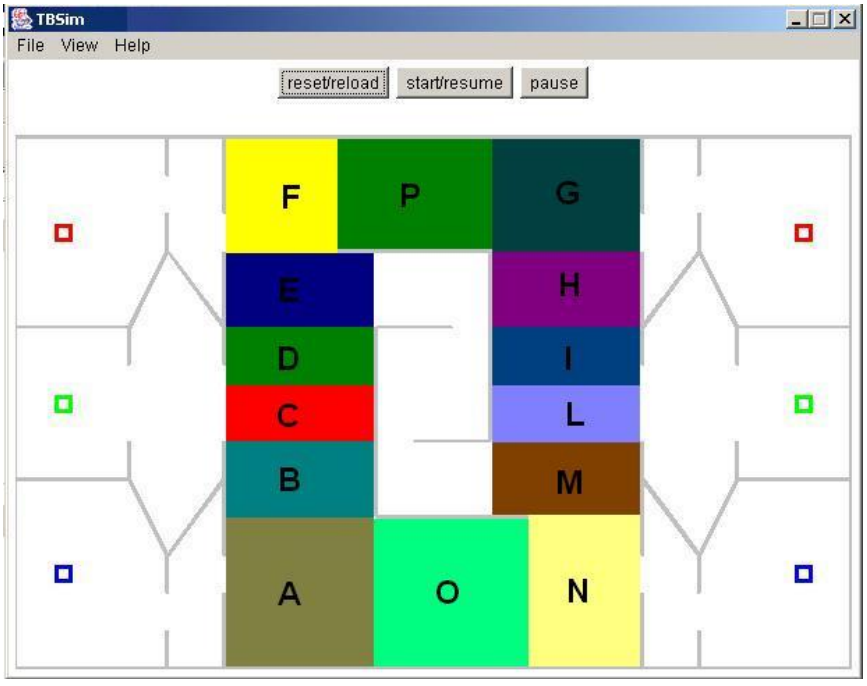


Figura 1.4.a

Zone definite per individuare la posizione corrente del robot

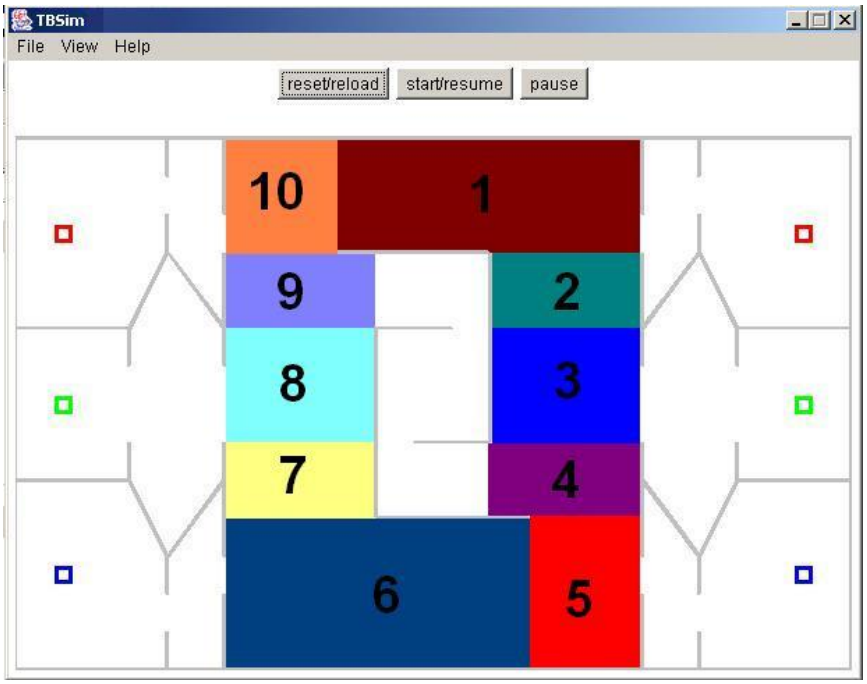


Figura 1.4.b

Zone definite per individuare la posizione della bandiera da recuperare

Quanto detto significa che il robot sceglierà di seguire una traiettoria su cui influirà notevolmente un campo potenziale (nel caso particolare trattato esso sarà di tipo repulsivo) che viene sommato a quello preesistente punto per punto. La metodologia operativa adoperata fa riferimento alla stessa definita sopra. Vediamo un esempio pratico:

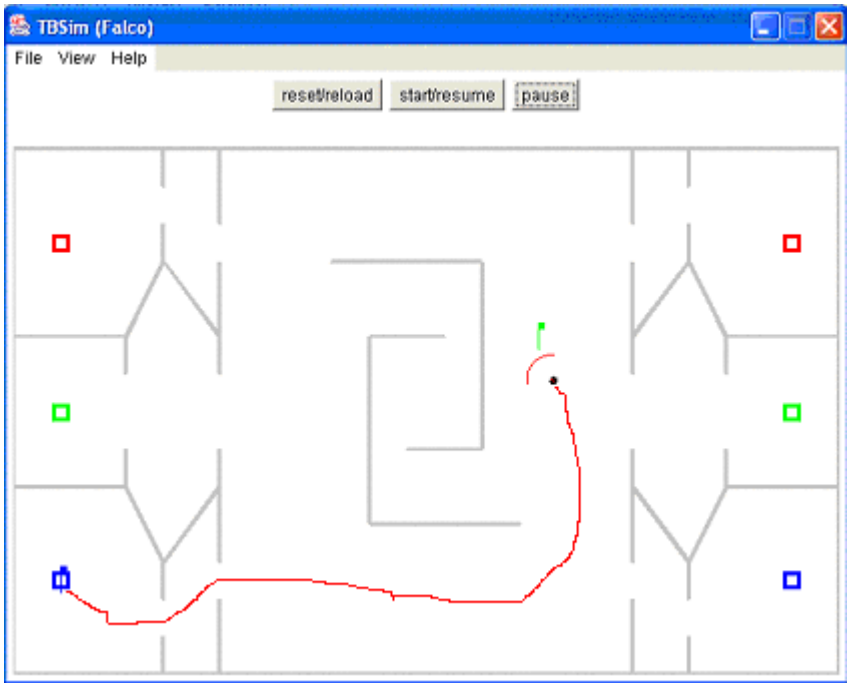


Figura 1.5

Da quanto si vede, il robot, dopo aver navigato nella stanza ed aver individuato la presenza di due bandiere di colore rispettivamente verde e blu, ne consegna una (quella blu) e successivamente tenta di recuperare quella verde. Il robot segue un percorso che gli consente agevolmente di giungere al punto dove è localizzata la bandiera, le cui coordinate si ritrovano in memoria. Se non avessimo fatto ricorso alla nostra strategia il robot avrebbe tentato, non appena fuori dalla stanza del bin blu, di attuare il percorso indicato dalla freccetta in Figura 1.6:

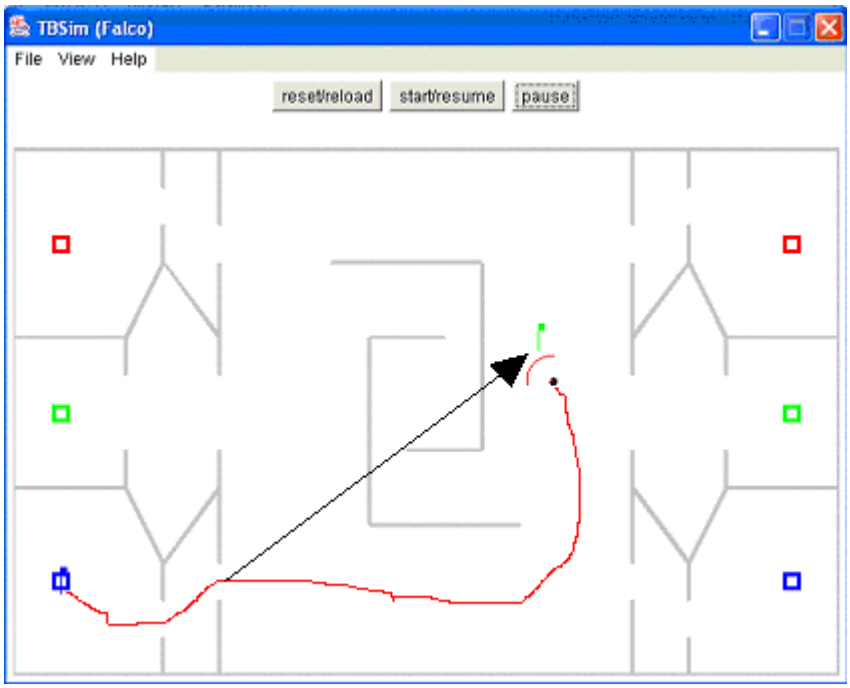


Figura 1.6

Il percorso che la linear_attraction tenderebbe a impostare (indicato dalla freccia), e il percorso effettivamente prodotto dal nostro sistema di controllo (traccia in rosso).

La suddivisione della mappa in stanze non ha la pretesa di essere la migliore soluzione all'evidente problema di recupero bandiere, tuttavia la sua efficienza è stato uno sprono alla sua realizzazione. La localizzazione di una bandiera e del robot stesso all'interno della mappa, nonché la generazione del campo potenziale opportuno per il recupero vengono effettuate dalla classe Location (vedi codice).

Una filosofia analoga a quella appena descritta è stata adoperata anche per quanto concerne la localizzazione della posizione del robot e delle bandierine in particolari spazi del campo. Questo è stato implementato nella classe RoomLocation, in cui tramite equazioni cartesiane ed opportuni vincoli abbiamo rappresentato le anticamere e le tre stanze in cui è stato suddiviso il labirinto. Per scendere nel tecnico rimandiamo ai commenti al codice ed al codice stesso.

Per quanto concerne la migliore strategia di esplorazione (da applicare, ad esempio, nel momento in cui il robot si ritrovi senza bandierine in memoria) ci siamo avvantaggiati di alcune delle classi presenti nei packages presenti in CSAI.unipa. In modo particolare nel sistema di controllo del nostro robot abbiamo inserito un MS a cui passiamo un Perceptual Schema che è un oggetto *v_ChooseZone_mv.* Le basi teoriche che ci hanno guidato nella realizzazione di questo particolare aspetto della nostra strategia sono senza dubbio i **sistemi a produzioni.**

Sarà bene a questo punto effettuare una digressione per chiarire cosa si intende per sistemi a produzione. Un sistema a produzioni è definito come una lista di regole (o produzioni) composte da una prima parte detta *condizione* e una seconda detta *parte azione.*

Definito il sistema a produzioni:

$$c_1 \longrightarrow a_1$$

$$c_2 \longrightarrow a_2$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$c_i \longrightarrow a_i$$

dove la parte condizione può essere una qualsiasi funzione a valori booleani, selezionare un'azione vuol dire valutare la parte condizione di ogni regola nell'ordine dato finché non viene trovata una parte condizione il cui valore sia 1. La parte azione corrispondente può essere un'azione primitiva, un insieme di azioni da eseguire contemporaneamente ovvero una chiamata ad un altro sistema a produzioni. Solitamente l'ultima regola ha la parte condizione uguale ad 1. E' la classica condizione di default. Dal momento che abbiamo a

che fare con agenti robotici, a cui è stato esplicitamente demandato di interagire con la realtà che li circonda, è chiaro che i valori delle caratteristiche, che discendono dall'elaborazione percettiva degli ingressi sensoriali, e che a loro volta controllano le funzioni condizione, devono variare nel tempo come diretta e tangibile conseguenza delle azioni del robot.

Una delle parti più problematiche nella navigazione all'interno del campo è senza dubbio la presenza di anticamere poste immediatamente prima della zona bin e la presenza di un labirinto nella parte centrale.

Nel codice del nostro robot abbiamo effettuato delle scelte per la risoluzione di questi aspetti implementativi che si differenziano l'una dall'altra. A fondamento delle scelte vi sono tuttavia delle analoghe matrici di ragionamento legate alla natura delle stanze e alle esigenze di gara.

Il nostro robot è stato programmato per esplorare le anticamere nel momento in cui, non avendo bandierine in memoria e dunque trovandosi in fase di explore, vada a posizionarsi in una fascia (larga 3x) di fronte le pareti di accesso alle stanze. Evitando di visitare le stanze all'avvio della gara facciamo in modo che il robot non consenta alla squadra avversaria di avere spazio libero nel recupero delle bandierine sparse per il campo. Infatti è abbastanza improbabile che la squadra avversaria abbia una strategia che porti il proprio robot a esplorare in primis le nostre stanze. Ciononostante appena il robot effettuerà la prima consegna, se nell'attraversare l'anticamera individuerà delle bandierine e se queste soddisferanno il computo della bandiera più vicina, l'agente provvederà al recupero. La visita della stanza è programmata per una sola volta, ma a tale limite non è sottoposta l'entrata per il recupero di eventuali bandiere. È importante sottolineare che il robot visita **tutte** le anticamere presenti, sia le proprie che le avversarie. Infine la visita delle anticamere, date le loro dimensioni, è stata temporizzata.

Vediamo adesso un esempio di funzionamento. In Figura 1.7 i 2 robot Try_it hanno esplorato il campo. Le 2 bandiere, l'una blu e l'altra rossa, ancora presenti nella zona avversaria non sono state rilevate. Quindi i nostri robot, trovandosi ancora in stato di **Explore** ed essendosi venuti a trovare in prossimità delle pareti delle proprie stanze hanno subito un cambiamento di stato. L'uno, quello più in alto è già passato allo stato **Explore stanza.** L'altro, appena giunto in prossimità della parete, si sta dirigendo verso l'entrata.

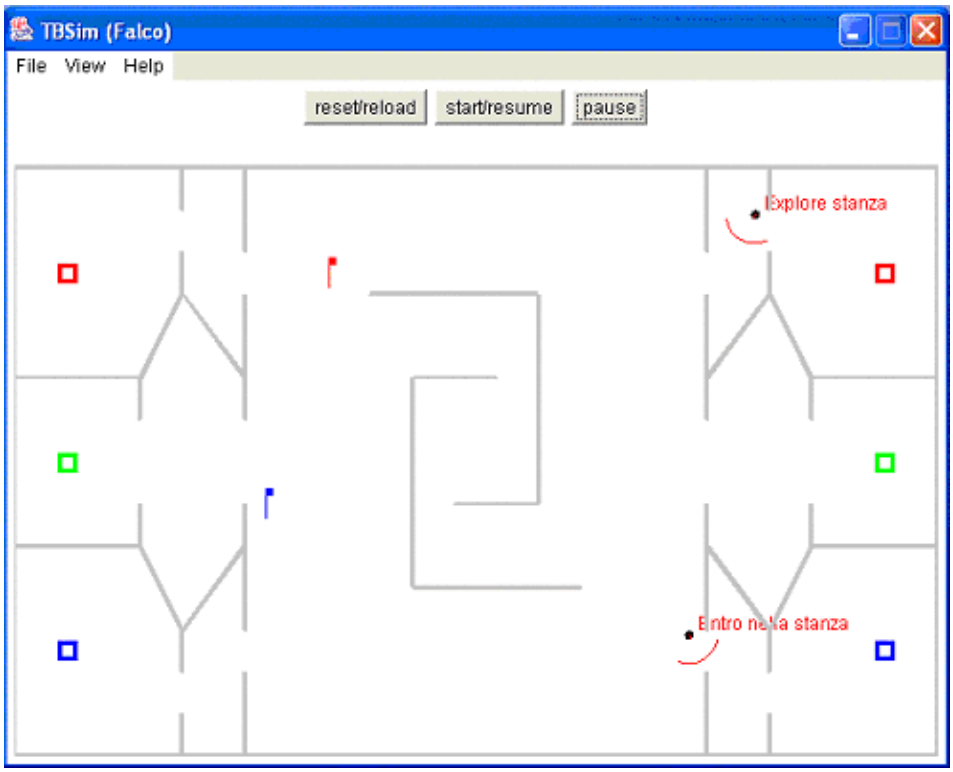


Figura 1.7

Adesso incentriamo la nostra analisi sul criterio di esplorazione e più in generale di visita del labirinto. Diversamente dalla strategia di esplorazione delle stanze la visita del labirinto non è stata limitata ad una volta, sebbene sia la fase di entrata (vicinanza ad uno dei 2 ingressi della chiocciola) che la fase di recupero di eventuali bandiere rilevate siano state gestite similmente a quanto fatto per le anticamere. Ad ogni modo ciò che più di ogni altra cosa differenzia le due tattiche di esplorazione è che la visita della chiocciola è stata implementata in SW ricorrendo al linguaggio sGolog. Il piano che si occupa di tale fase è stato significativamente chiamato lab_plan e costituisce lo stato 8 dell'FSA.

Nelle figure 1.8 e 1.9 esemplifichiamo visivamente alcune delle principali operazioni svolte durante l'esecuzione del piano allo stato 8:

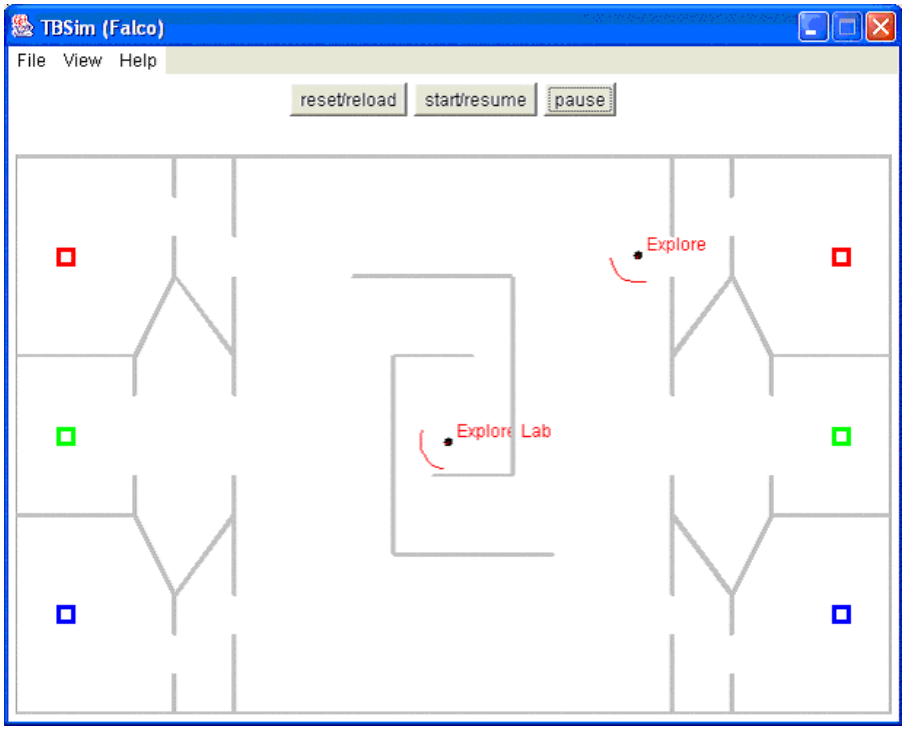


Figura 1.8

In Figura 1.8 osserviamo una semplice visita da parte di uno dei 2 robot all'interno del labirinto. Come visibile viene mostrata la display string Explore Lab. In Figura 1.9, invece, il robot all'interno del labirinto ha rilevato una bandierina e dopo averla afferrata si sta preparando alla fase di consegna.

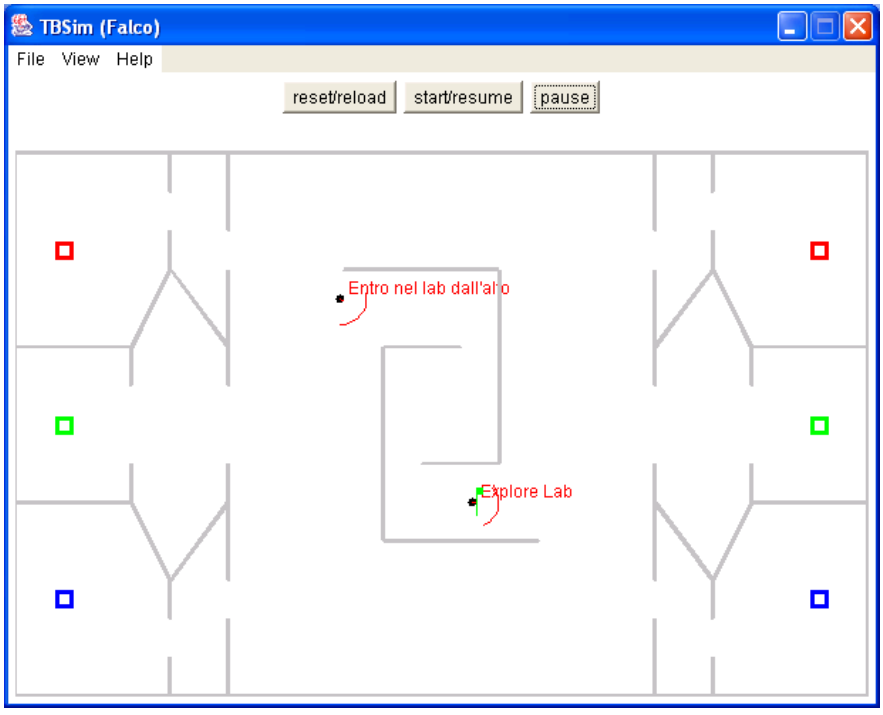


Figura 1.9

La fase di consegna della bandiera, di qualsiasi colore essa sia, ed in qualsiasi posto essa venga rilevata e acquisita (ad esclusione che si trovi dentro il labirinto), è gestita da un piano di consegna (bring plan) che abbiamo implementato ricorrendo alle potenzialità del linguaggio sGolog. Il robot effettua un sense sulla classe visuale della bandierina che si ritrova nel gripper come risultato della fase di recupero di tipo reattivo, ed effettua autonomamente la scelta appropriata per la consegna nel bin corretto. Nel pieno rispetto del regolamento, una volta giunto sulla soglia della camera dove si trova il punto di consegna, l'agente resta in stato di wait in attesa del messaggio dell'altro robot compagno di squadra che gli confermi che anche esso si trova in stato di wait. Solo a quel punto entrambi procedono alla consegna.

Alla luce di questa analisi è possibile dunque rendersi conto della natura **ibrida** del robot documentato. Riepilogando brevemente infatti il robot è un esempio di **architettura deliberativa** in quanto:

1. La fase di esplorazione fa uso di un **sistema a produzioni** che fa muovere il robot in maniera intelligente nel campo di gioco. In particolare il robot, tenderà a visitare zone della mappa che non ha ancora visitato;
2. I due robot mantengono una **memoria globale** delle bandierine presenti sul campo di gioco grazie al continuo scambio di messaggi: questo offre loro il vantaggio di spostamenti intelligenti poiché l'uno "sa" della presenza di bandiere in zone della mappa che l'altro ha già esplorato;
3. La consegna delle bandiere e la visita del labirinto sono gestite tramite l'elaborazione di un **piano sGolog** che lascia dunque al robot il compito di decidere cosa fare.

Ma al contempo lampante è l'influenza di modelli di **architetture reattive basate sul comportamento:**

1. Il robot basa la sua capacità motoria sulla possibilità di generare dei vettori specificati in modulo, verso e direzione che causano attrazione verso determinati punti (come ad esempio le bandiere da recuperare): è la teoria dei **Motor Schemas.**
2. La presenza di ostacoli e bandierine che rispettivamente generano forze repulsive ed attrattive fa si che l'agente si trovi immerso in un **campo potenziale** in cui esso si muove a seconda del modulo della somma vettoriale dei vettori di campo che insistono sulla sua posizione istantanea.

È evidente che una semplice lettura del codice che abbiamo presentato per il sistema di controllo del nostro robot non potrebbe costituire un valido aiuto per la piena comprensione degli aspetti delle nostre scelte progettuali. Per questo motivo la sezione seguente fornisce chiarimenti e commenti riguardo il codice Java ed sGolog da noi presentato.
Alcune delle classi java da noi presentate sono una modifica di classi preesistenti all'interno del simulatore. La procedura di estensione del codice che elegantemente caratterizza la programmazione ad oggetti non è stata da noi applicata in ragione delle scelte implementative adottate nelle classi da noi riusate.

✓ NodeMemory_Try_it

Questa classe è un ampliamento della classe NodeMemory presente all'interno della cartella CSAI.unipa.knowledgement. NodeMemory_Try_it ritorna un array di Vec2 che rappresentano le coordinate delle bandiere la cui presenza il robot ha rilevato durante la navigazione sul campo di gara. Innanzi tutto noi abbiamo istanziato al suo interno un nuovo oggetto *MultiForageN150ExploreSim* piuttosto del *MultiForageN150Sim* presente nel codice originale. Il nostro robot è difatti un'implementazione della versione del Multiforage dotato di sonar e laser.

```
        private VisualObjectSensor visual_robot;
        private SimpleInterface abstract_robot;
        private int attractorType1, attractorType2, attractorType3;
        public static final boolean DEBUG = Node.DEBUG;
        private boolean control;
        private MultiForageN150ExploreSim multi_robot;
        private int length=0;
```

Abbiamo inoltre inserito una variabile intera length per conoscere il numero delle bandierine che il robot vede durante la navigazione e di cui dunque ha memorizzato le posizioni cartesiane. Tale numero di bandierine viene settato a zero nel caso in cui, nel metodo Value, il *catch(NullPointerException e)* di riga 217 raccoglie l'eccezione lanciata a riga 160.

Il metodo getLength, che ritornava un double, è stato ridefinito e adesso ritorna un int che è pari a 0 se non ci sono bandiere in memoria (nel codice originale in questo caso avrebbe ritornato 1).
Nel metodo pop è stato aggiunto l'invio di un messaggio che fa eliminare al robot destinatario la bandiera specificata. Il nostro robot, infatti, come già detto, fa un uso della memoria basato sullo scambio dei messaggi tra i due robot della squadra.
Alla riga 344 viene introdotta una variabile contatore intera count inizializzata a zero che mi consente di tenere traccia delle bandierine da cancellare. Questo check è indispensabile in quanto altrimenti la memoria rischierebbe di essere aggiornata con una dimensione errata.

✓ Stalling

Questa classe è stata pensata per gestire opportunamente le situazioni di stallo che si possono presentare durante lo svolgimento della competizione. L'utilizzo di variabili contatore si è da subito dimostrato inefficiente da un punto di vista tecnico (il metodo take step, continuamente richiamato in run-time, è sensibile alla piattaforma HW su cui gira il programma) e da un punto di vista di buono stile di programmazione. Quindi abbiamo preferito basare il conteggio sul clock di sistema richiamando il metodo `System.currentTimeMillis()`. Questo metodo ritorna il tempo espresso in millisecondi. La classe java ha quindi il compito di valutare che il robot non si trovi a compiere la stessa azione per più di cut millisecondi:

```
System.currentTimeMillis()- time>cut;
```

dove time corrisponde all'istante di tempo cui si fa riferimento e cut è l'intervallo di tempo che viene passato come parametro al metodo isStall e che costituisce il valore discriminante in base al quale se la differenza tra il tempo di sistema e la variabile time risulta true il robot è in stallo.

✓ RoomLocation

La classe RoomLocation è stata pensata, in un'ottica di programmazione ad oggetti, per concentrare in un'unica prospettiva gli sforzi di localizzazione del robot e della posizione delle bandierine. Vengono dichiarati dei metodi che ritornano true o false a seconda che il robot e/o le bandierine si trovino nella zona indicata o meno. Di seguito riportiamo una parte del codice in cui sono presentati i tre metodi a cui il robot fa riferimento per calcolare in quale delle tre anticamere si ritrova:

```
public boolean in_red_Right(){
      return zone_red_Right;
}
public boolean in_green_Right(){
      return zone_green_Right;
}
public boolean in_blue_Right(){
      return zone_blue_Right;
}
```

Come è già chiaro dal codice stesso, se il robot si trova nell'anticamera del bin rosso il primo dei tre metodi tornerà true. Stesso ragionamento vale per gli altri metodi che seguono lo stesso principio.

```
public boolean flag_in_blue(Vec2 flag){
      double x_flag=flag.x;
      double y_flag=flag.y;
    boolean flag_blue=x_flag>33 && x_flag<36 && y_flag>0 && y_flag<10&&
                        y_flag<((-4*x_flag+162)/3);
      return  flag_blue;
           }
```

Qui sopra è stato riportato il codice corrispondente ad uno dei metodi flag_in_*, i quali consentono di localizzare la posizione le bandierine all'interno del campo.

✓ v_Closest_var_Try_it

Questa classe è un' estensione della classe v_Closest_var presente all'interno della cartella CSAI.unipa.knowledgement. v_Closest_var_Try_it ritorna il più vicino in una lista di Vec2. In modo particolare la funzionalità che noi abbiamo sfruttato riguarda il calcolo della prossima bandierina (le cui coordinate sono presenti in memoria) da recuperare. Per questo motivo alla riga 35 è stata dichiarata:

```
private NodeMemory_Try_it embedded1;
```

Analiticamente la classe in esame, nel metodo Value, valuta la classe visuale di ogni bandiera memorizzata, e calcola la più vicina contemporaneamente al robot (all'istante di tempo in cui viene effettuata la computazione) e al bin di consegna. In questo modo si evitano inutili lunghi percorsi per recuperare una bandiera lontana quando è invece possibile acquisirne una vicina, oltretutto evitando di correre il rischio sia di tentare di acquisire una bandiera che essendo in zona avversaria magari verrà prima recuperata da un robot concorrente, sia di memorizzare eventuali altre bandiere incontrate in zone lontane piuttosto che dare la precedenza alle bandiere poste in prossimità dei propri bin.

✓ v_LinearAttraction_v_Try_it

Questa classe è un' estensione della classe v_LinearAttraction_v presente all'interno della cartella EDU.gatech.cc.is.clay. v_LinearAttraction_v_Try_it consente la generazione di un vettore che punta alla posizione del punto oggetto dell'attrazione. Il vettore è dotato dunque di una sua direzione, ma anche di una sua intensità. In modo particolare l'intensità del vettore è calcolata ad ogni takestep (al metodo Value della classe in esame passiamo il parametro *long timestamp*) e tiene conto del campo potenziale in cui il robot si trova immerso: al crescere della distanza aumenta la forza attrattiva (e di conseguenza decresce proporzionalmente la forza repulsiva).

Per le nostre esigenze di strategia di cui abbiamo discusso sopra, abbiamo definito un metodo setRot_teta che setta la variabile di classe rot_teta col valore t che viene passato al metodo (vedere commento alla classe Location).

```
public void setRot_teta(double t){
                if(t>=0)
                      rot_teta=t;
                else
                    rot_teta=(2.0 * Math.PI)+t;
          }
```

La singola istruzione else è stata inserita per rendere positivi eventuali valori negativi dell'angolo (in radianti) t passato al metodo.
Alla riga 110 ruotiamo il vettore goal (la cui copia last_val è il valore ritornato dal metodo Value nelle sue componenti last_val.x e last_val.y) del valore di rot_teta impostato dal metodo setRot_teta mostrato nel riquadro precedente.
Da notare che le rotazioni avvengono in senso antiorario per valori positivi di teta.

✓ Location

Questa classe si basa sul concetto che ogni zona ha un differente campo potenziale e il vettore risultante di attrazione verso il punto attrattore (bandierina o punto che sia) è ottenuto come interazione (somma pesata) del vettore di attrazione lineare, del vettore di

noise(rumore), del vettore (repulsivo) degli ostacoli e, in più, del contributo di un vettore fittizio applicato dalla zona in cui il robot si trova istante per istante.
Questo vettore fittizio è implementato semplicemente agendo sull'angolo teta del vettore risultante con il metodo *setRot_teta* della classe *v_LinearAttraction_v_Try_it* .
Ritornando alla classe *Location*, possiamo dire che, innanzitutto, essa adotta due layer di suddivisione del campo di gara: il primo suddivide il campo in zone riferite alla posizione del robot,chiamate *zone_A, zoneB,*ecc.. mentre il secondo lo suddivide in zone riferite alla posizione degli attrattori chiamate *flag_1 ,flag_2,* etc.
Dopodichè con il metodo *in_special_case()* individua se la situazione in cui si trova il robot che si appresta a dirigersi verso la bandierina più vicina è qualcuna di quelle previste, cioè di quelle in cui la zona deve "fare sentire" il suo peso nella composizione del vettore risultante.
Inoltre la classe contiene il metodo *getTeta()* che ritorna di quanto il vettore risultante deve ruotare, (a seconda della zona in cui si trova il robot e la bandierina), il metodo *getStopper()* che ritorna il tempo per cui la zona deve farsi sentire e il metodo *getString()* che ritorna semplicemente la stringa che indica in quale situazione si trova il robot e che viene visualizzata dal simulatore.

✓ Competition_Try_it.pl

Questo è il codice sGolog che utilizziamo per realizzare una consistente parte del lato deliberativo del nostro robot. Nella parte iniziale abbiamo dichiarato le azioni primitive che il robot deve essere in grado di riconoscere come tali. Nelle righe successive esprimiamo con i *poss(a, s)* che è lecito compiere qualsiasi azione primitiva in qualsiasi situazione s. A questo punto vengono dichiarati gli assiomi di stato successore: ricordiamo che deve esistere un assioma di stato successore per ogni fluente F, ed esso mi specifica quali azioni hanno effetto ed in quale modo.
A questo punto si passa alla definizione delle procedure: procediamo all'esame della *bring_flag*. Questa procedura al suo interno richiama la procedura *deliver_red_flag* la quale a sua volta richiama a cascata la *deliver_green_flag* e la *deliver_blue_flag.* La nomenclatura stessa è già indicativa della funzione di queste procedure. La *deliver_red_flag* effettua un sense sulla bandierina in gripper, se essa è rossa viene richiamata la procedura

go_bin(Z,X,Y) a cui vengono passati il red_attractor1, red_attractor2 e red_bin (rispettivamente Z, X, Y). Se invece la bandierina non dovesse risultare rossa viene richiamata la *deliver_green_flag* la cui definizione è analoga alla precedente. Infine se la bandierina non dovesse essere neppure verde viene richiamata *deliver_blue_flag.* A questo punto, se la bandierina è blu viene invocato *go_bin(Z,X,Y)* altrimenti si passa allo stato di *wander*. Proviamo a commentare brevemente la procedura *go_bin(Z,X,Y)*:

```
proc(go_bin(Z,X,Y),
        goto(Z) :
        (sense(in_lab):
         branch_on(in_lab):
           if(in_lab,
                exit_lab:goto(Z),
                goto(Z))
        ):
            goto(X) : send_message(am_at_attractor) :
           (sense(can_go):
                branch_on(can_go):
                      if(can_go,
                      goto(Y) : open_gripper : goto(X) : goto(Z),
                      wait: goto(Y) : open_gripper : goto(X) : goto(Z)))
).
```

Ricordando che i due punti (:) in sGolog indicano una sequenza di azioni, vediamo che nell'ordine il robot dovrà portarsi all'attrattore Z; a questo punto dovrà controllare se si trova dentro il labirinto (dal momento che non è infrequente che nel calcolo del miglior percorso si introduca in esso). Se si trova dentro il labirinto dovrà richiamare la procedura exit_lab e quindi procedere nuovamente verso Z e successivamente verso l'attrattore X. Giunto a questo punto dovrà inviare il messaggio contenente la stringa "am at attractor" all'altro robot e valutare se risulta vero il fluente can_go. In caso affermativo procederà al deposito della bandierina, altrimenti si andrà in stato di wait che terminerà con l'invio anche da parte del compagno di squadra della messaggio che conferma l'arrivo all'attrattore X.

```
proc(explore_lab,

           explore(entry):
           (sense(something_visible_lab):
           branch_on(something_visible_lab):
                 if(something_visible_lab,
                       go_flag:exit_lab,
                       goto_lab(centre):explore_centre

                       )
           )
).
```

ba

La parte di codice soprastante è relativa alla procedura explore_lab. Questa è la procedura principale che gestisce la fase di esplorazione del labirinto; nel momento in cui si trova al suo interno il robot deve valutare se c'è qualcosa di visibile (in termini di bandierine chiaramente). Se questo test dà esito positivo in risposta l'agente si dedica al recupero della stessa e all'uscita del labirinto; nel caso opposto allora si recherà all'interno della stanza centrale della chiocciola e da qui darà avvio alla nuova fase di visita. Tale fase è espressamente prevista dalla procedura explore_centre nella quale sono presenti una serie di sense nidificati volti a gestire ogni eventuale situazione. In ognuno di essi la immancabile condizione di uscita dalla procedura è fornita dal richiamo alla exit_lab.

✓ Try_it_Robot

Questa è la classe del nostro robot. Il codice si compone fondamentalmente di due parti: il metodo configure( ) ed il metodo takestep( ). Il primo viene richiamato solo una volta quando viene istanziato l'oggetto robot e si preoccupa di impostare il sistema di controllo che fa muovere l'agente. Il metodo takestep( ) viene richiamato ogni istante di tempo in quanto contiene le istruzioni che consentono al robot di sapere cosa fare. In questo secondo metodo sono difatti contenuti gli stati che determinano il comportamento del robot: istante per istante esso si trova in uno degli 8 stati da noi usati (l'FSA è rappresentato alla fine del commento).

Metodo Configure( )

In questo metodo oltre a configurare le impostazioni hardware del robot si è creato un oggetto miopunteggio che è un segnapunti per tenere conto delle bandierine recuperate. Ma incentrando l'attenzione sugli aspetti che più da vicino ci interessano dal punto di vista implementativo iniziamo a scorrere le righe di codice: inizialmente memorizziamo in una variabile id l'identificativo che in fase di gara il simulatore ha assegnato al robot e con un costrutto switch lo rendiamo consapevole anche dell'identificativo assegnato al suo

compagno di squadra. Questa parte è essenziale per garantire il corretto funzionamento della parte di messaggistica in unicast.

In seguito creiamo un oggetto File eclipseProgram localizzato in una cartella Database nella nostra attuale directory di lavoro. Questo file viene passato come parametro per la creazione di un oggetto sGologClient che verrà consultato per l'esecuzione del piano sGologPlan che abbiamo dichiarato come bring_plan. Seguono la creazione di oggetti NodeMap, NodeMemory_Try_it, Location e v_Closest_var_Try_it.

Passiamo a questo punto ai Perceptual Schemas:

```
PS_OBS = new va_Obstacles_r(abstract_robot);

PS_SONAR = new va_SonarSensed_r(abstract_robot);

PS_LASER = new va_LaserSensed_r(abstract_robot);

NodeVec2Array
PS_TEMP = new va_Merge_vava(PS_LASER, PS_SONAR);

NodeVec2Array
PS_ALL_OBS = new va_Merge_vava(PS_TEMP, PS_OBS);
```

I PS elencati nel riquadro sono d'obbligo nel nostro robot dal momento che il MultiForageN150Explore è dotato di laser e sonar le cui rilevazioni devono essere elaborate in modo opportuno. Infatti la `PS_ALL_OBS` contiene tutti gli ostacoli rilevati dal robot durante la navigazione (tecnicamente essa è un merge tra gli ostacoli rilevati dal normale sistema di visione del robot e gli ostacoli rilevati sia dal sonar che dal laser).

La `PS_CHOOSE_ZONE` è un oggetto v_ChooseZone_mv che consente al robot di calcolare il prossimo punto della mappa verso cui muoversi ottimizzando la fase di esplorazione (sistema a produzioni di cui parlato sopra). Altri PS riguardano la definizione di particolari punti attrattori dinamici (ad esempio `PS_ATTRACTOR)`.

Passiamo a questo punto alle Percpetual Features. Il robot si avvale di PF per quanto concerne la possibilità di comprendere di trovarsi in una particolare situazione. Un esempio illuminante può essere fornito da `PF_SOMETHING_IN_GRIPPER` che è un oggetto di tipo NodeBoolean, e dunque ritornerà valore TRUE se il robot ha qualcosa in gripper, ovvero se ha afferrato la bandierina. Alcune particolari PF rivestono una certa rilevanza all'interno del nostro codice: ad esempio `PF_OTHER_ROBOT` è un NodeBooleanDinamic, il cui valore muta dunque di continuo in run-time, che indica ad un robot se l'altro si trova in fase di wait o

meno dal momento che alla ricezione del messaggio “am at attractor” questa variabile viene aggiornata. Alcune altre variabili di questo tipo come `PF_EXIT_0` (anch’essa

```
linear_attraction= new
v_LinearAttraction_v_Try_it(0.4,0.0,PS_SPECIAL_ATTRACTOR);

   NodeVec2
   MS_MOVE_TO_SPECIAL_ZONE=linear_attraction;

   // AS_GOTO_SPECIAL_ZONE
   //======
   v_StaticWeightedSum_va
   AS_GOTO_SPECIAL_ZONE = new v_StaticWeightedSum_va();

   AS_GOTO_SPECIAL_ZONE.weights[0]  = avoidgain;
   AS_GOTO_SPECIAL_ZONE.embedded[0] = MS_AVOID_OBSTACLES;

   AS_GOTO_SPECIAL_ZONE.weights[1]  = noisegain;
   AS_GOTO_SPECIAL_ZONE.embedded[1] = MS_NOISE_VECTOR;

   AS_GOTO_SPECIAL_ZONE.weights[2]  = mtggain;
   AS_GOTO_SPECIAL_ZONE.embedded[2] = MS_MOVE_TO_SPECIAL_ZONE;
```

Successivamente richiamiamo il metodo addAction di bring_plan per aggiungere le azioni che ci servono per la corretta esecuzione del piano. Da notare inoltre che bisogna precisare lo stato in cui l’interprete deve eseguire le azioni del piano (nel nostro caso allo stato 2).
Quindi segue la State Machine, e le impostazioni associate al comportamento dello stato corrispondente di torretta, steering, gripper fingers (dove 0=chiuso, 1=aperto, -1=pronto), laser e sonar.

Metodo takestep( )

A questo punto avviamo la discussione sul takestep. Nella parte iniziale dichiariamo le variabili che utilizzeremo nel corpo del metodo e particolarmente degna di nota è la variabile intera state che viene aggiornata correntemente col valore attuale della macchina a stati (ovvero questa variabile dice al robot in che stato si trova).

Adesso analizziamo i vari stati in cui il robot può venirsi a trovare (per una visuale completa guardare l'FSA riportato alla fine del commento).

```
if(bufferMessage.hasMoreElements())
       {
           try
            {
     LongMessage idMessage = (LongMessage) bufferMessage.nextElement();
        if(idMessage.val == other_id && bufferMessage.hasMoreElements())
                 {
     StringMessage message = (StringMessage)bufferMessage.nextElement();
                    if(message.val=="Bandiera nuova")
                    {
    PointMessage flagPosition=(PointMessage)bufferMessage.nextElement();
    Vec2 pos=new Vec2 (flagPosition.xValue,flagPosition.yValue);

    LongMessage visualMessage=(LongMessage)bufferMessage.nextElement();
    int vis=(int)visualMessage.val;
    flag_memory.push(pos,vis,curr_time);
                    }
                    else if(message.val=="Delete Flag")
                    {
    PointMessage flagPosition=(PointMessage)bufferMessage.nextElement();
    Vec2 pos=new Vec2 (flagPosition.xValue,flagPosition.yValue);
    flag_memory.pop_only(pos,curr_time);
                    }
                    else
                     if(message.val== "Pronto")
                    {

    LongMessage idFlag=(LongMessage) bufferMessage.nextElement();
                      if(idFlag.val==PS_IN_GRIPPER.intValue(curr_time)){
                            PF_EQUAL_FLAG.setValue(true);
                      }
                      else PF_EQUAL_FLAG.setValue(false);
                      PF_OTHER_ROBOT.setValue(true);
                    }
                 }
            }
             catch(ClassCastException e){}
        }
```

Nella parte di codice soprastante è stata inserita la parte principale concernente la messagistica del nostro robot (l'altro grande blocco si trova nello stato 2). Come si evince già da una prima lettura il robot controlla se sono presenti nuovi messaggi in memoria, se il controllo ha esito positivo si procede ad un'identificazione del messaggio contenuto. I casi possibili sono 3: "Bandiera nuova"-"Delete flag"-"Pronto". Nel primo caso il robot viene avvertito dall'altro che è stata rilevata una nuova bandiera, l'azione da compiere è dunque di inserire in memoria le coordinate di tale bandiera. Nel secondo caso una bandiera precedentemente rilevata è stata rimossa, l'azione da compiere è di eliminarla dalla

memoria. Nel terzo caso uno dei due robot compagni ha recuperato una bandiera e si trova già posizionato davanti all'accesso all'homebase, comunica dunque all'altro che, se quest'ultimo dovesse recuperare una bandiera dello stesso colore, al fine di evitare inutili intralci deve andarsi a posizionare nel punto secondario di attesa all'accesso.

Allo **stato 0** il robot valuta la sua posizione (al momento della partenza) e conseguentemente setta un attrattore che gli consente di eseguire quella che noi abbiamo definito fase di Attack (vedi la sezione riguardante la strategia).

Lo **stato 1** è quello in cui il robot viene a trovarsi quando, avendo in memoria le posizioni delle bandiere viste, calcola quale bandiera sia più vicina e procede con la fase di acquisizione. Da questo stato scattano diversi trigger. Se la bandiera da recuperare si trova in una special_zone (ovvero una zona il cui raggiungimento è reso difficoltoso dalla presenza del labirinto, individuata con l'ausilio della classe Location) da questo si passa allo stato 3. Se la memoria è vuota si passa allo stato 4 di Explore. Una situazione particolare si presenta nel momento in cui la bandiera da recuperare si trova in una delle anticamere ovvero nel labirinto:

```
if((roomLocation.flag_in_red(flag)&&roomLocation.closed_red_Right())||(roo
mLocation.flag_in_green(flag)&&roomLocation.closed_green_Right())||(roomLo
cation.flag_in_blue(flag)&&roomLocation.closed_blue_Right())||(roomLocatio
n.flag_in_lab(flag)&&(roomLocation.closed_lab_Left()||roomLocation.closed_
lab_Right())))
           {
                PF_ENTRY.setValue(true);
           }
           else PF_ENTRY.setValue(false);
```

In questo caso una delle condizioni del costrutto if del codice sopra risulterà vera e la `PF_ENTRY` portata a true consentirà il passaggio allo stato 7.

Chiaramente in questo stato (così come nello stato 4 come vedremo) viene effettuato un controllo sulle bandiere eventualmente ancora presenti in memoria.

```
     if(localizer.in_special_case(x_robot,y_robot,flag.x,flag.y)){
                    PF_GOTO_SPECIAL_ZONE.setValue(true);
                    }
                    else PF_GOTO_SPECIAL_ZONE.setValue(false);
```

Particolarmente importante risulta essere la parte di codice riportata sopra. Il robot è stato programmato per riconoscere, date le proprie coordinate attuali e quelle della bandiera da recuperare, se si trova in una `SPECIAL_ZONE`. In caso affermativo, passerà allo stato 3 (vedasi) dove prenderà le decisioni dettate caso per caso.

Lo **stato 2** è come già detto lo stato del bring_plan, ovvero di consegna della bandiera acquisita. Il normale funzionamento del robot prevede che da qui si passi nuovamente allo stato 1. All'interno della struttura di if annidati che ha come condizione di entrata *if(bring_plan.isChangedState())* abbiamo isolato caso per caso ogni singola possibile situazione di consegna, e abbiamo applicato quanto già detto riguardo ai campi potenziali.

```
if(bring_plan.getParameter()[0].equals("red_attractor1")){
                        PS_ATTRACTOR.setValue(curr_time, 32.0, 23.2);
                        if(x_robot>11 && x_robot<19&& y_robot<22){
                              linear_attraction2.setRot_teta(1.2);
                        }
           else if(x_robot>19 && x_robot<25 && y_robot<12 && y_robot>8){
                              linear_attraction2.setRot_teta(-0.4);
                        }
```

recuperato una bandiera rossa e dunque deve andarla a consegnare, il piano sGolog setterà l'attrattore in red_attractor1 (32.0, 23.2) solo che, in dipendenza dalla posizione in cui il robot viene a trovarsi il vettore di attrazione lineare verso questo punto verrà ruotato dell'angolo specificato. Inoltre può verificarsi che in fase di consegna il robot vada ad introdursi in una delle stanze di ingresso al labirinto. In questa condizione il piano sGolog prevede di avviare la procedura exit_lab e continuare normalmente la fase di consegna (vedi commento al file prolog). All'interno dello stato 2 è inoltre presente la parte relativa all'invio del messaggio che consente ai due robot di aspettarsi davanti la camera dei bin. Nel messaggio i due robot inviano anche un identificativo della classe visuale della bandiera in gripper. La motivazione di quest'aggiunta è la seguente: se i due robot devono consegnare due bandierine dello stesso colore, piuttosto che ostacolarsi tra loro dal momento che il piano sGolog li porta in stato di wait nello stesso punto, essi si posizionano in due punti diversi entrambi sulla soglia d'ingresso alla stanza dei bin.

Lo **stato 3** presiede all'impostazione del valore corretto del teta del vettore di attrazione lineare. Questo valore è ritornato dalla chiamata al metodo getTeta (x_robot, y_robot, flag.x, flag.y) della classe Location:

```
linear_attraction.setRot_teta(localizer.getTeta(x_robot,y_robot,flag.x,flag.y));
```

Dal momento che nel passare da una zona all'altra deve variare il valore del teta , abbiamo da un lato introdotto la condizione che l'oggetto antistallo debba essere aggiornato ogni volta che si passa di zona in zona, e dall'altro inserito un'istruzione setRot_teta. Da notare che ogni volta che al robot scade l'intervallo di tempo entro cui dovrebbe essere uscito dalla zona momentanea settiamo a true il `PF_STOP_GOTO_SPECIAL_ZONE` che consente il passaggio allo stato 1.

Lo **stato 4** è lo stato di esplorazione. Da questo stato esce se vede qualcosa, quindi la memoria non è più vuota e può iniziare la fase di recupero.

Iniziamo ad analizzare le parti salienti di questo stato estrapolate dal codice:

```
if((!red_room_Right&&roomLocation.closed_red_Right())||(!green_room_Right&
&roomLocation.closed_green_Right())||(!blue_room_Right&&roomLocation.close
d_blue_Right())||(!lab_room&&roomLocation.closed_lab_()) )
          {
               PF_ENTRY.setValue(true);
          }
          else PF_ENTRY.setValue(false);
```

In questa parte di codice si presenta il caso che il robot, essendosi venuto a trovare in prossimità di una delle anticamere o del labirinto, se non ha ancora visitato la zona in questione cambia stato attraverso il settaggio a `TRUE` della `PF_ENTRY`.

```
     if(flag_memory.getLength(curr_time)!=0){
               PF_EMPTY_MEMORY.setValue(false);
               PF_NOT_EMPTY_MEMORY.setValue(true);
               }
          else{
               PF_EMPTY_MEMORY.setValue(true);
```

Nella parte di codice soprastante il robot effettua un controllo sulla memoria. Se la lunghezza della memoria all'istante corrente non è nulla significa che essa non è vuota; viceversa non essendo stata avvistata nessuna bandiera il robot deve permanere in stato di explore. Il settaggio di queste PF sarà usato nello stato 1.

Lo **stato 5** è uno stato intermedio tra la localizzazione e direzione immediata verso la bandiera avvistata (stato 1) e la pronta consegna (stato 2). Infatti la condizione principale di uscita da questo stato è determinata dal valore alto della `PF_SOMETHING_IN_GRIPPER`.

Tuttavia, dal momento che il robot potrebbe non riuscire ad afferrare la bandiera, ecco che nasce l'esigenza di una condizione di uscita alternativa dallo stato in esame.

```
if(delete==true){
            delete=false;
            antistallo.update();
      }
            if(antistallo.isStall(3500)){
                flag_memory.pop(flag,curr_time);
                PF_STALL.setValue(true);
                  }
                        else PF_STALL.setValue(false);
```

Come si evince chiaramente dal codice è stato inserito un antistallo. Nel momento in cui il robot si troverà fermo a cercare di recuperare il flag per un intervallo di tempo superiore ai 2.5 secondi allora scatterà la `PF_STALL` che riporterà allo stato 1 in cui verrà calcolata una nuova bandiera verso cui dirigersi.

Lo **stato 6** è stato pensato come punto di "sbocco" per tutti quegli stati che come condizione di uscita da una particolare zona (si pensi allo scadere del tempo nella visita di una delle anticamere) necessitano della presenza di un punto attrattore verso cui il robot possa tranquillamente dirigersi.

```
if(roomLocation.in_red_Right()){
            PS_ATTRACTOR.setValue(curr_time,31.0,23.0);
            PF_EXIT.setValue(false);

            }
```

Quanto detto sopra è espresso in modo cristallino dal codice di cui sopra: se il robot si trova all'interno dell'anticamera rossa di destra ed è scaduto il tempo prestabilito di esplorazione, la condizione di uscita dalla zona prevede che il robot si diriga verso il punto di coordinate (31.0, 23.0).

Allo **stato 7** si passa dallo stato 1 e dallo stato 4 nel momento in cui è stata programmata la visita delle anticamere e del labirinto. I due casi sono gestititi separatamente in quanto l'accesso del labirinto successivamente al passaggio al robot di un punto attrattore d'accesso, come già detto è affidato al piano golog di cui allo stato 8, mentre per le anticamere viene passato un attrattore dinamico di comodo al robot e una volta raggiuntolo si passa allo stato 4 di esplorazione della stanza. Chiaramente nel caso delle anticamere il

cambio di stato è garantito dal `PF_CLOSE` che in automatico viene settato dal robot quando raggiunge le coordinate impostate, nel caso invece del labirinto occorrerà gestire il settaggio "manualmente":

```
if(roomLocation.in_lab()){
                PF_LAB.setValue(true);
          }
          else PF_LAB.setValue(false);
```

Per quanto concerne lo **stato 8** esso implementa l' esplorazione e più in generale l'uso del labirinto. Analizziamo la seguente riga di codice:

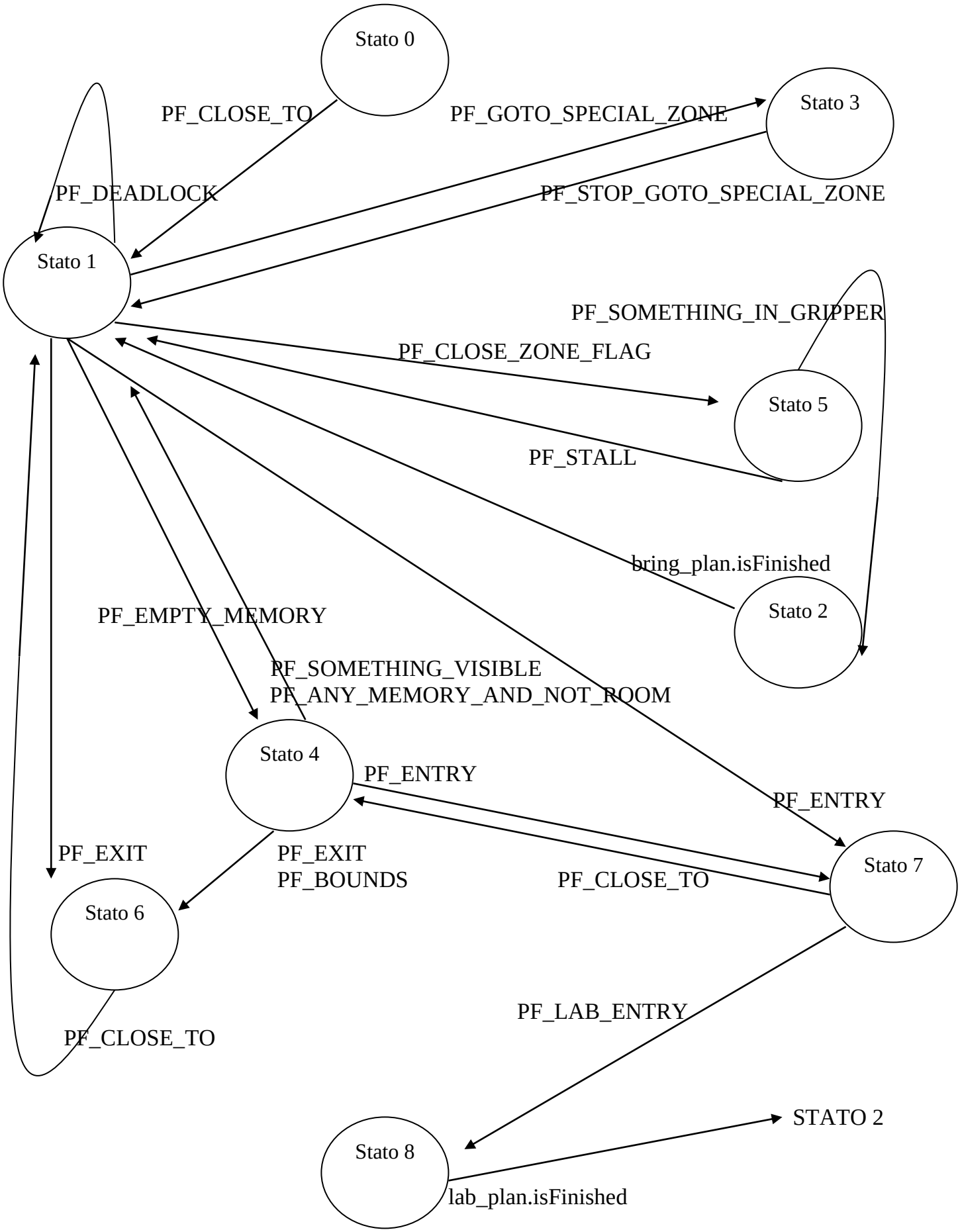
```
int y_rand=(Math.abs(random.nextInt())%4)+13;
```

Questa rigasi trova in uno degli if annidati del costrutto if principale che presiede alle attività gestite dall'azione di explore del labirinto. Facendo generare al java un valore casuale compreso tra 0 e 4 (lunghezza della stanza centrale) e incrementato di 13 unità per far rientrare precisamente tale valore all'interno del labirinto il robot è in grado di generare di volta in volta una nuova coordinata per il punto attrattore usato per navigare. Tale scelta vede la sua importanza dal momento che variando tale punto la zona esplorata risulta notevolmente accresciuta e quindi, date le dimensioni dell'ambiente, possibilmente sarà totale.

```
if(action.startsWith("go_flag")){
                            begin_flag=false;
                            antistallo.update();
                            lab_flag=new
Vec2(PS_CLOSEST_FLAG.Value(curr_time).x+x_robot,PS_CLOSEST_FLAG.Value(c
urr_time).y+y_robot);
                            flag_memory.pop(lab_flag,curr_time);
                            PF_HAVE_FLAG.setValue(true);
                            PF_HAVE_FLAG_LAB.setValue(true);
                                    }
```

La parte soprastante di codice è relativa all'antistallo opportunamente inserito per l'azione di go_flag. Infatti può verificarsi che all'interno del labirinto, nel modo di recuperare una bandiera, il robot si ritrovi a tentare indefinitamente di prendere una bandiera che è già stata afferrata possibilmente da un robot avversario. Se il robot si ritrovasse in questa azione di go_flag per più del tempo fissato per lo stallo allora si sblocca. Chiaramente, trovandoci in run-time all'interno del piano gestito dall'interprete golog, le scelte da adottare in caso di stallo sono state codificate nel file.pl e successivamente mappate anche in java (cfr. codice

analizzato sopra). Queste sono le parti fondamentali dello stato in esame, per ulteriori approfondimenti riferirsi al codice.


Stato 0
PF_CLOSE_TO
PF_GOTO_SPECIAL_ZONE
Stato 3
PF_DEADLOCK
PF_STOP_GOTO_SPECIAL_ZONE
Stato 1
PF_SOMETHING_IN_GRIPPER
PF_CLOSE_ZONE_FLAG
Stato 5
PF_STALL
bring_plan.isFinished
Stato 2
PF_EMPTY_MEMORY
PF_SOMETHING_VISIBLE
PF_ANY_MEMORY_AND_NOT_ROOM
Stato 4
PF_ENTRY
PF_ENTRY
PF_EXIT
PF_EXIT
PF_BOUNDS
PF_CLOSE_TO
Stato 7
Stato 6
PF_CLOSE_TO
PF_LAB_ENTRY
STATO 2
Stato 8
lab_plan.isFinished

Classe Robot.java

```
import EDU.gatech.cc.is.abstractrobot.*;
import EDU.gatech.cc.is.util.Vec2;
import EDU.gatech.cc.is.clay.*;
import EDU.gatech.cc.is.communication.*;

import CSAI.unipa.clay.*;
import CSAI.unipa.abstractrobot.*;
import CSAI.unipa.SGolog.*;
import CSAI.unipa.knowledgment.*;
import CSAI.unipa.communication.PointMessage;

import java.awt.*;
import java.io.*;
import java.util.Enumeration;
import java.util.Date;
import java.util.Random;

public class Robot extends ControlSystemMFN150Explore{
   public final static boolean DEBUG = false;
   private NodeVec2           turret_configuration;
   private NodeVec2           steering_configuration;
   private NodeBoolean        sonar_configuration;
   private NodeDouble         gripper_fingers_configuration;
   private NodeBoolean        laser_configuration;
   private NodeGain           mtggain = new NodeGain(1.0);
   private NodeGain           swirlgain = new NodeGain(0.7);
   private NodeGain           avoidgain = new NodeGain(0.8);
   private NodeGain           noisegain = new NodeGain(0.3);
   private NodeInt            state_monitor;
   private NodeGain           gripper=new NodeGain(1.0);
   private i_FSA_ba           STATE_MACHINE;
   private NodeVec2Array      PS_SONAR;
   private NodeVec2           PS_RED_BIN;
   private NodeVec2           PS_GREEN_BIN;
   private NodeVec2           PS_BLUE_BIN;
   private NodeVec2           PS_RED_ATTRACTOR;
   private NodeVec2           PS_GREEN_ATTRACTOR;
   private NodeVec2           PS_BLUE_ATTRACTOR;
   private NodeVec2           PS_CLOSEST_FLAG;
   private NodeInt            PS_IN_GRIPPER;
   private v_DinamicPoint_    PS_ATTRACTOR,PS_SPECIAL_ATTRACTOR,PS_ATTRACTOR_FLAG;
   private NodeVec2           PS_GLOBAL_POS;
   private v_ChooseZone_mv_Try_it CHOOSE_ZONE;
   private Random             random;
   private NodeBooleanDinamic
PF_OTHER_ROBOT,PF_STALL,PF_ROOM,PF_ENTRY,PF_EXIT_CENTRE,PF_CENTRE;
   private NodeBoolean        PF_CLOSE_TO,PF_SOMETHING_IN_GRIPPER;
   private NodeBoolean        PF_CLOSE_ZONE_FLAG;
   private NodeBooleanDinamic  PF_DEADLOCK,PF_EMPTY_MEMORY,PF_NOT_EMPTY_MEMORY,PF_EXIT;
   private NodeBooleanDinamic
PF_GOTO_SPECIAL_ZONE,PF_STOP_GOTO_SPECIAL_ZONE,PF_EQUAL_FLAG;
   private NodeBooleanDinamic  PF_TIME_OUT,PF_LAB,PF_COME_FROM_LOW,PF_BOUNDS;
   private NodeBooleanDinamic
PF_IN_LOW_LAB,PF_IN_CENTRE_LAB,PF_HAVE_FLAG,PF_HAVE_FLAG_LAB,PF_IN_LAB;
   private NodeBooleanDinamic PF_IN_HIGH_LAB,PF_STALL_FLAG,PF_USE_LAB,PF_USE_LAB_RED_BLUE;
   private Enumeration bufferMessage;
   private sGologClient       client;
   private SGologPlan         bring_plan,lab_plan;
   private NodeMap            MAP;
```

```
private NodeMemory_Try_it   flag_memory;
private v_Closest_var_Try_it flag_closest;
private NodeInt MIOPUNTEGGIO;
private static Res MOSTRA =new Res("Try_it");
private Date       stall_ms;
private Date       stall1_ms;
private int        id;
private int        other_id;
private int        count_3=0,count_7=0;
private int        count_in_room=0,count_in_centre=0;
private int        stall=0,stall9=0;
private int        tempo=0;
private double old_teta=0.0;
private Vec2  flag;
private v_LinearAttraction_v_Try_it linear_attraction,linear_attraction2;
private Location localizer;
private RoomLocation roomLocation;
private boolean begin_flag=true;
private static boolean red_room_Left=false,green_room_Left=false,blue_room_Left=false;
private static boolean red_room_Right=false,green_room_Right=false,blue_room_Right=false,lab_room=false;
private boolean    come_from_low=true;
private static boolean lab=false;
private boolean delete =true;  // Variabile utilizzata per eliminare una bandiera dalla memoria
private boolean start1 =true;
private boolean start =true;
private Date   time_start9_ms;
private boolean begin8=true;
private boolean begin1=true;
private boolean begin4=true;
private Stalling antistallo;
private Vec2 lab_flag=new Vec2(0,0);


/**
 * Configure the control system using Clay.
 */


public void configure(){

   SimpleInterface ab=abstract_robot;
   stall_ms=new Date();
   MIOPUNTEGGIO = new i_myscore_r(abstract_robot,"Try_it");

   //======
   // Set some initial hardware configurations.
   //======

   id = abstract_robot.getPlayerNumber(abstract_robot.getTime());
   random=new Random();
   switch(id){
      case 0:
         other_id=1;
         break;
      case 1:
         other_id=0;
         break;
      case 2:
         other_id=3;
         break;
      case 3:
         other_id=2;
```

```
        break;
    }

    abstract_robot.setObstacleMaxRange(3.0);
    abstract_robot.setBaseSpeed(1.0*abstract_robot.MAX_TRANSLATION);
    flag=new Vec2();

    //abbassamento dell'artiglio
    abstract_robot.setGripperHeight(-1,0);

    //modalità trigger
    abstract_robot.setGripperFingers(-1,-1);

    File eclipseProgram = new File(".\\Database_Try_it\\sGolog_Try_it.pl");

    client = new sGologClient(eclipseProgram);

    eclipseProgram = new File(".\\Database_Try_it\\Competition_Try_it.pl");

    client.consultFile(eclipseProgram);

    bring_plan = new SGologPlan(abstract_robot);

    lab_plan = new SGologPlan(abstract_robot);

    MAP = new NodeMap(0.0, 44.0,30.0, 0.0,0.8,0.7,"Mappa",false,abstract_robot);


    flag_memory=new NodeMemory_Try_it(false,2,1,0,ab);

    localizer=new Location();

    antistallo= new Stalling();

    bufferMessage = abstract_robot.getReceiveChannel();



    //======
    // perceptual schemas
    //======
    //--- robot's global position

    PS_GLOBAL_POS = new v_GlobalPosition_r(abstract_robot);

    roomLocation=new RoomLocation(PS_GLOBAL_POS);

    //   NodeVec2
    CHOOSE_ZONE=new v_ChooseZone_mv_Try_it(MAP,PS_GLOBAL_POS,0,2,1,3);
    NodeVec2 PS_CHOOSE_ZONE=CHOOSE_ZONE;

    NodeVec2 bin_red=new v_FixedPoint_(2.5, 23);
    NodeVec2 bin_green=new v_FixedPoint_(2.5, 14);
    NodeVec2 bin_blue=new v_FixedPoint_(2.5,5);

    flag_closest=new v_Closest_var_Try_it(flag_memory,ab,bin_red,bin_green,bin_blue);

    //--- obstacles
    NodeVec2Array // the sonar readings
    PS_OBS = new va_Obstacles_r(abstract_robot);

    PS_SONAR = new va_SonarSensed_r(abstract_robot);
```

```
NodeVec2Array
PS_ALL_OBS = new va_Merge_vava(PS_SONAR, PS_OBS);

PS_ATTRACTOR = new v_DinamicPoint_(PS_GLOBAL_POS, 0.0, 0.0);

PS_ATTRACTOR_FLAG= new v_DinamicPoint_(PS_GLOBAL_POS,27.0,30.0);

PS_SPECIAL_ATTRACTOR= new v_DinamicPoint_(PS_GLOBAL_POS,0.0,0.0);

NodeVec2
PS_CLOSEST_ZONE = new v_Closest_mr(MAP, abstract_robot);



//--- targets of visual class 0
NodeVec2Array
PS_RED_FLAG_EGO =
new va_VisualObjects_r(2,abstract_robot);

//--- targets of visual class 1
NodeVec2Array
PS_GREEN_FLAG_EGO =
new va_VisualObjects_r(1,abstract_robot);

//--- targets of visual class 1
NodeVec2Array
PS_BLUE_FLAG_EGO =
new va_VisualObjects_r(0,abstract_robot);

NodeVec2Array
PS_TEMP = new va_Merge_vava(PS_GREEN_FLAG_EGO, PS_RED_FLAG_EGO);

NodeVec2Array
PS_TEMP2 = new va_Merge_vav(PS_TEMP, PS_ATTRACTOR_FLAG);

NodeVec2Array
PS_ALL_TARGET = new va_Merge_vava(PS_TEMP, PS_BLUE_FLAG_EGO);

//     NodeVec2
PS_CLOSEST_FLAG = new v_Closest_va(PS_ALL_TARGET);


//--- type of object in the gripper
//    NodeInt
PS_IN_GRIPPER = new i_InGripper_r(abstract_robot);




//======
// Perceptual Features
//======


PF_CLOSE_TO = new b_CloseDinamic_vv(0.4, PS_GLOBAL_POS, PS_ATTRACTOR);

NodeBoolean
PF_SOMETHING_VISIBLE=new b_NonZero_v(PS_CLOSEST_FLAG);

NodeBoolean
```

```
PF_SOMETHING_VISIBLE_MEMORY=new b_NonZero_v(PS_ATTRACTOR_FLAG);

NodeBoolean
PF_FLAG_VISIBLE=new b_NonZero_v(PS_ATTRACTOR_FLAG);

PF_OTHER_ROBOT = new NodeBooleanDinamic(false);

NodeBoolean
PF_RED_FLAG_IN_GRIPPER = new b_Equal_i(2, PS_IN_GRIPPER);

NodeBoolean
PF_GREEN_FLAG_IN_GRIPPER = new b_Equal_i(1, PS_IN_GRIPPER);

NodeBoolean
PF_BLUE_FLAG_IN_GRIPPER = new b_Equal_i(0, PS_IN_GRIPPER);


PF_SOMETHING_IN_GRIPPER = new b_Persist_s(4.0,new b_NonNegative_s(PS_IN_GRIPPER));

NodeBoolean
PF_NOT_SOMETHING_IN_GRIPPER = new b_Not_s(PF_SOMETHING_IN_GRIPPER);

NodeBoolean
PF_NOT_SOMETHING_VISIBLE= new b_Not_s(PF_SOMETHING_VISIBLE);

NodeBoolean
PF_NOT_SOMETHING_VISIBLE_MEMORY= new b_Not_s(PF_SOMETHING_VISIBLE_MEMORY);

PF_CLOSE_ZONE_FLAG = new b_CloseDinamic_vv(0.6, PS_GLOBAL_POS, PS_ATTRACTOR_FLAG);

PF_DEADLOCK=new NodeBooleanDinamic(false);

PF_STALL=new NodeBooleanDinamic(false);

PF_STALL_FLAG=new NodeBooleanDinamic(false);

PF_TIME_OUT=new NodeBooleanDinamic(false);

PF_EMPTY_MEMORY=new NodeBooleanDinamic(false);

PF_ROOM=new NodeBooleanDinamic(false);

PF_LAB=new NodeBooleanDinamic(false);

PF_USE_LAB=new NodeBooleanDinamic(false);

PF_USE_LAB_RED_BLUE=new NodeBooleanDinamic(false);

PF_COME_FROM_LOW=new NodeBooleanDinamic(false);

PF_IN_LOW_LAB=new NodeBooleanDinamic(false);

PF_IN_HIGH_LAB=new NodeBooleanDinamic(false);

PF_IN_CENTRE_LAB=new NodeBooleanDinamic(false);

PF_HAVE_FLAG=new NodeBooleanDinamic(false);

PF_HAVE_FLAG_LAB=new NodeBooleanDinamic(false);

PF_IN_LAB=new NodeBooleanDinamic(false);
```

```
PF_BOUNDS=new NodeBooleanDinamic(false);



NodeBoolean
PF_NOT_ROOM= new b_Not_s(PF_ROOM);

PF_CENTRE=new NodeBooleanDinamic(false);

NodeBoolean
PF_NOT_CENTRE= new b_Not_s(PF_CENTRE);


PF_ENTRY=new NodeBooleanDinamic(false);

PF_NOT_EMPTY_MEMORY = new NodeBooleanDinamic(true);

PF_EXIT=new NodeBooleanDinamic(false);

PF_EXIT_CENTRE=new NodeBooleanDinamic(false);


PF_GOTO_SPECIAL_ZONE=new NodeBooleanDinamic(false);

PF_STOP_GOTO_SPECIAL_ZONE=new NodeBooleanDinamic(false);

PF_EQUAL_FLAG=new NodeBooleanDinamic(false);

NodeBoolean
PF_ANY_MEMORY_AND_NOT_ROOM=new b_And_bb(PF_NOT_ROOM,PF_NOT_EMPTY_MEMORY);

NodeBoolean
PF_LAB_ENTRY=new b_And_bb(PF_LAB,PF_CLOSE_TO);

NodeBoolean
PF_EXIT_EXPLORE=new b_Or_bb(PF_SOMETHING_VISIBLE,PF_CLOSE_TO);

NodeBoolean
PF_CLOSE_TO_OR_IN_LAB=new b_Or_bb(PF_IN_LAB,PF_CLOSE_TO);

NodeBoolean
PF_FLAG_OR_STALL=new b_Or_bb(PF_SOMETHING_IN_GRIPPER,PF_STALL_FLAG);
//======
// motor schemas
//======

// avoid obstacles
NodeVec2
MS_AVOID_OBSTACLES = new v_AvoidSonar_va(3.0, abstract_robot.RADIUS + 0.1,
PS_OBS, PS_GLOBAL_POS);

NodeVec2
MS_MOVE_TO_FLAG = new v_LinearAttraction_v(0.4,0.0,PS_CLOSEST_FLAG);

NodeVec2
MS_MOVE_TO_ZONE_FLAG = new v_LinearAttraction_v( 0.4, 0.0, PS_ATTRACTOR_FLAG);

// noise vector
NodeVec2
MS_NOISE_VECTOR = new v_Noise_(10,5);

NodeVec2
```

```
MS_SWIRL_CLOSEST_ZONE = new v_Swirl_vav(1.5, abstract_robot.RADIUS + 0.18,
PS_OBS, PS_CLOSEST_ZONE);

NodeVec2
MS_SWIRL_TO = new v_Swirl_vav(1.5, abstract_robot.RADIUS + 0.18,
PS_OBS, PS_ATTRACTOR);

//   v_LinearAttraction_v_Try_it
linear_attraction=new v_LinearAttraction_v_Try_it(0.4,0.0,PS_SPECIAL_ATTRACTOR);

NodeVec2
MS_MOVE_TO_SPECIAL_ZONE=linear_attraction;

linear_attraction2=new
                                                    v_LinearAttraction_v_Try_it(0.4,0.0,P
S_ATTRACTOR);

NodeVec2
MS_MOVE_TO = linear_attraction2;

NodeVec2
MS_MOVE_TO_CHOOSE_ZONE=new v_LinearAttraction_v(0.4,0.0,PS_CHOOSE_ZONE);


//======
// AS_GOTO
//======
v_StaticWeightedSum_va
AS_GOTO = new v_StaticWeightedSum_va();


AS_GOTO.weights[0]  = avoidgain.Value(abstract_robot.getTime());
AS_GOTO.embedded[0] = MS_AVOID_OBSTACLES;

AS_GOTO.weights[1]  = noisegain.Value(abstract_robot.getTime());
AS_GOTO.embedded[1] = MS_NOISE_VECTOR;

AS_GOTO.weights[2]  = mtggain.Value(abstract_robot.getTime());
AS_GOTO.embedded[2] = MS_MOVE_TO;

AS_GOTO.embedded[3] = MS_SWIRL_TO;
AS_GOTO.weights[3] = swirlgain.Value(abstract_robot.getTime());


//======
// AS_GOTO_ZONE_FLAG
//======
v_StaticWeightedSum_va
AS_GO_ZONE_FLAG = new v_StaticWeightedSum_va();


AS_GO_ZONE_FLAG.weights[0]  = avoidgain.Value(abstract_robot.getTime());
AS_GO_ZONE_FLAG.embedded[0] = MS_AVOID_OBSTACLES;

AS_GO_ZONE_FLAG.weights[1]  = noisegain.Value(abstract_robot.getTime());
AS_GO_ZONE_FLAG.embedded[1] = MS_NOISE_VECTOR;

AS_GO_ZONE_FLAG.weights[2]  = mtggain.Value(abstract_robot.getTime());
AS_GO_ZONE_FLAG.embedded[2] = MS_MOVE_TO_ZONE_FLAG;


//======
```

```
    // AS_GOTO_ZONE_FLAG
    //======
    v_StaticWeightedSum_va
    AS_GO_FLAG = new v_StaticWeightedSum_va();



/*    AS_GO_FLAG.weights[0]  = avoidgain.Value(abstract_robot.getTime());
    AS_GO_FLAG.embedded[0] = MS_AVOID_OBSTACLES;

    AS_GO_FLAG.weights[1]  = noisegain.Value(abstract_robot.getTime());
    AS_GO_FLAG.embedded[1] = MS_NOISE_VECTOR;
 */
    AS_GO_FLAG.weights[0]  = mtggain.Value(abstract_robot.getTime());
    AS_GO_FLAG.embedded[0] = MS_MOVE_TO_FLAG;


    //======
    // AS_GOTO_SPECIAL_ZONE
    //======
    v_StaticWeightedSum_va
    AS_GOTO_SPECIAL_ZONE = new v_StaticWeightedSum_va();


    AS_GOTO_SPECIAL_ZONE.weights[0]  = avoidgain.Value(abstract_robot.getTime());
    AS_GOTO_SPECIAL_ZONE.embedded[0] = MS_AVOID_OBSTACLES;

    AS_GOTO_SPECIAL_ZONE.weights[1]  = noisegain.Value(abstract_robot.getTime());
    AS_GOTO_SPECIAL_ZONE.embedded[1] = MS_NOISE_VECTOR;

    AS_GOTO_SPECIAL_ZONE.weights[2]  = mtggain.Value(abstract_robot.getTime());
    AS_GOTO_SPECIAL_ZONE.embedded[2] = MS_MOVE_TO_SPECIAL_ZONE;


    /************
     * AS GO CHOOSE ZONE
     *************/

    v_StaticWeightedSum_va
    AS_GO_CHOOSE_ZONE =  new v_StaticWeightedSum_va();

    AS_GO_CHOOSE_ZONE.embedded[0] = MS_MOVE_TO_CHOOSE_ZONE;
    AS_GO_CHOOSE_ZONE.weights[0] = mtggain.Value(abstract_robot.getTime());

    AS_GO_CHOOSE_ZONE.embedded[1] = MS_AVOID_OBSTACLES;
    AS_GO_CHOOSE_ZONE.weights[1] = avoidgain.Value(abstract_robot.getTime());

    AS_GO_CHOOSE_ZONE.embedded[2] = MS_NOISE_VECTOR;
    AS_GO_CHOOSE_ZONE.weights[2] = 0.5;


    AS_GO_CHOOSE_ZONE.embedded[3] = MS_SWIRL_CLOSEST_ZONE;
    AS_GO_CHOOSE_ZONE.weights[3] = swirlgain.Value(abstract_robot.getTime());



    bring_plan.debugging(false);

    bring_plan.addAction("exit",
    AS_GOTO,
    AS_GOTO,
```

```
PF_CLOSE_TO,
0,
new int[]{0,4,8,12},
NodeAction.LASER_ON);


bring_plan.addAction("goto",
AS_GOTO,
AS_GOTO,
PF_CLOSE_TO_OR_IN_LAB,
0,
new int[]{0,2,4,6,8,10,12,14},
NodeAction.LASER_ON);

bring_plan.addAction("goto_lab",
AS_GOTO,
AS_GOTO,
PF_CLOSE_TO,
0,
new int[]{0,4,8,12},
NodeAction.LASER_ON);

bring_plan.addAction("open_gripper",
NodeAction.NO_STEERING,
NodeAction.NO_TURRET,
NodeAction.IMMEDIATE_TRIGGER,
1,
NodeAction.NO_SONAR,
NodeAction.NO_LASER);

bring_plan.addAction("send_message",
NodeAction.NO_STEERING,
NodeAction.NO_TURRET,
NodeAction.IMMEDIATE_TRIGGER,
NodeAction.NO_GRIPPER,
NodeAction.NO_SONAR,
NodeAction.NO_LASER);

bring_plan.addAction("wait",
AS_GOTO,
AS_GOTO,
PF_OTHER_ROBOT,
NodeAction.NO_GRIPPER,
NodeAction.NO_SONAR,
NodeAction.NO_LASER);


bring_plan.addAction("room", PF_ROOM);

bring_plan.addAction("red_flag", PF_RED_FLAG_IN_GRIPPER);

bring_plan.addAction("green_flag", PF_GREEN_FLAG_IN_GRIPPER);

bring_plan.addAction("blue_flag", PF_BLUE_FLAG_IN_GRIPPER);

bring_plan.addAction("can_go", PF_OTHER_ROBOT);

bring_plan.addAction("in_lab", PF_IN_LAB);

bring_plan.addAction("in_low_lab", PF_IN_LOW_LAB);

bring_plan.addAction("in_high_lab", PF_IN_HIGH_LAB);
```

```
bring_plan.addAction("in_centre_lab", PF_IN_CENTRE_LAB);

bring_plan.addAction("have_flag", PF_HAVE_FLAG);

bring_plan.addAction("use_lab", PF_USE_LAB);

bring_plan.addAction("use_lab_red_blue", PF_USE_LAB_RED_BLUE);

bring_plan.setPlan(client.invoke("do(bring_flag,s0,S)"),2);


lab_plan.debugging(false);

lab_plan.addAction("goto_lab",
AS_GOTO,
AS_GOTO,
PF_CLOSE_TO,
0,
new int[]{0,4,8,12},
NodeAction.LASER_ON);

lab_plan.addAction("go_flag",
AS_GO_FLAG,
AS_GO_FLAG,
PF_FLAG_OR_STALL,
-1,
NodeAction.NO_SONAR,
NodeAction.NO_LASER);

lab_plan.addAction("explore",
AS_GOTO,
AS_GOTO,
PF_EXIT_EXPLORE,
NodeAction.NO_GRIPPER,
new int[]{0,2,4,6,8,10,12,14},
NodeAction.NO_LASER);

lab_plan.addAction("something_visible_lab", PF_SOMETHING_VISIBLE);
lab_plan.addAction("come_from_low", PF_COME_FROM_LOW);
lab_plan.addAction("in_low_lab", PF_IN_LOW_LAB);
lab_plan.addAction("in_high_lab", PF_IN_HIGH_LAB);
lab_plan.addAction("in_centre_lab", PF_IN_CENTRE_LAB);
lab_plan.addAction("have_flag", PF_HAVE_FLAG);
lab_plan.setPlan(client.invoke("do(explore_lab,s0,S)"),8);



//======
// STATE_MACHINE
//======
STATE_MACHINE = new i_FSA_ba();


STATE_MACHINE.state = 0;


STATE_MACHINE.triggers[0][0]  = PF_CLOSE_TO;     //Stato 0:ATTACK
STATE_MACHINE.follow_on[0][0] = 1;
```

```
// Stato 1:
STATE_MACHINE.triggers[1][0]  = PF_CLOSE_ZONE_FLAG ;
STATE_MACHINE.follow_on[1][0] = 5;

STATE_MACHINE.triggers[1][1]=PF_DEADLOCK;
STATE_MACHINE.follow_on[1][1]=1;

STATE_MACHINE.triggers[1][2]=PF_EMPTY_MEMORY;
STATE_MACHINE.follow_on[1][2]=4;

STATE_MACHINE.triggers[1][3]=PF_GOTO_SPECIAL_ZONE;
STATE_MACHINE.follow_on[1][3]=3;

STATE_MACHINE.triggers[1][4]=PF_EXIT;
STATE_MACHINE.follow_on[1][4]=6;

STATE_MACHINE.triggers[1][5]=PF_ENTRY;
STATE_MACHINE.follow_on[1][5]=7;



// Stato 2:
STATE_MACHINE.triggers[2][0]  = bring_plan.isFinished(abstract_robot.getTime());
STATE_MACHINE.follow_on[2][0] = 1;



// Stato 3:
STATE_MACHINE.triggers[3][0]  =  PF_CLOSE_ZONE_FLAG;
STATE_MACHINE.follow_on[3][0] = 5;

STATE_MACHINE.triggers[3][1]  = PF_STOP_GOTO_SPECIAL_ZONE;
STATE_MACHINE.follow_on[3][1] = 1;



// Stato 4:
STATE_MACHINE.triggers[4][0]  =  PF_SOMETHING_VISIBLE;
STATE_MACHINE.follow_on[4][0] = 1;

STATE_MACHINE.triggers[4][1]  =  PF_EXIT;
STATE_MACHINE.follow_on[4][1] = 6;

STATE_MACHINE.triggers[4][2]  =  PF_ANY_MEMORY_AND_NOT_ROOM;
STATE_MACHINE.follow_on[4][2] = 1;

STATE_MACHINE.triggers[4][3]  =  PF_ENTRY;
STATE_MACHINE.follow_on[4][3] = 7;

STATE_MACHINE.triggers[4][4]  =  PF_BOUNDS;
STATE_MACHINE.follow_on[4][4] = 6;



// Stato 5:
STATE_MACHINE.triggers[5][0]  =  PF_SOMETHING_IN_GRIPPER;
STATE_MACHINE.follow_on[5][0] = 2;

STATE_MACHINE.triggers[5][1]  =  PF_STALL;
STATE_MACHINE.follow_on[5][1] = 1;
```

```
// Stato 6:
STATE_MACHINE.triggers[6][0]=PF_CLOSE_TO;
STATE_MACHINE.follow_on[6][0]=1;



// Stato 7:
STATE_MACHINE.triggers[7][0]  =  PF_LAB_ENTRY;
STATE_MACHINE.follow_on[7][0] = 8;

STATE_MACHINE.triggers[7][1]=PF_CLOSE_TO;
STATE_MACHINE.follow_on[7][1]=4;



// Stato 8:
STATE_MACHINE.triggers[8][0]  = lab_plan.isFinished(abstract_robot.getTime());
STATE_MACHINE.follow_on[8][0] = 2;


state_monitor = STATE_MACHINE;


//======
// STEERING
//======
v_Select_vai
STEERING = new v_Select_vai((NodeInt)STATE_MACHINE);


STEERING.embedded[0] = AS_GOTO;
STEERING.embedded[1] = AS_GO_ZONE_FLAG;
STEERING.embedded[2] = bring_plan.getSteering(abstract_robot.getTime());
STEERING.embedded[3] = AS_GOTO_SPECIAL_ZONE;
STEERING.embedded[4] = AS_GO_CHOOSE_ZONE;
STEERING.embedded[5] = AS_GO_ZONE_FLAG;
STEERING.embedded[6] = AS_GOTO;
STEERING.embedded[7] = AS_GOTO;
STEERING.embedded[8] = lab_plan.getSteering(abstract_robot.getTime());

//======
// TURRET
//======
v_Select_vai
TURRET = new v_Select_vai((NodeInt)STATE_MACHINE);

TURRET.embedded[0] = AS_GOTO;
TURRET.embedded[1] = AS_GO_ZONE_FLAG;
TURRET.embedded[2] = bring_plan.getTurret(abstract_robot.getTime());
TURRET.embedded[3] = AS_GOTO_SPECIAL_ZONE;
TURRET.embedded[4] = AS_GO_CHOOSE_ZONE;
TURRET.embedded[5] = AS_GO_ZONE_FLAG;
TURRET.embedded[6] = AS_GOTO;
TURRET.embedded[7] = AS_GOTO;
TURRET.embedded[8] = lab_plan.getTurret(abstract_robot.getTime());

//======
// GRIPPER_FINGERS
//======
d_Select_i
```

```
GRIPPER_FINGERS = new d_Select_i(STATE_MACHINE);

GRIPPER_FINGERS.embedded[0] = 1;
GRIPPER_FINGERS.embedded[1] = 1;
GRIPPER_FINGERS.embedded[2] = bring_plan.getGripperFingers(abstract_robot.getTime());
GRIPPER_FINGERS.embedded[3] = 1;
GRIPPER_FINGERS.embedded[4] =(int)gripper.Value(abstract_robot.getTime());
GRIPPER_FINGERS.embedded[5] = -1;
GRIPPER_FINGERS.embedded[6] = 0;
GRIPPER_FINGERS.embedded[7] = 0;
GRIPPER_FINGERS.embedded[8] = lab_plan.getGripperFingers(abstract_robot.getTime());

//======
// SONAR_CONFIGURATION
//======
d_SonarControl_ir
SONAR_CONFIGURATION = new d_SonarControl_ir(STATE_MACHINE, abstract_robot);

SONAR_CONFIGURATION.sonarActived[0] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[0] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[1] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[1] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[2] = bring_plan.getSonar(abstract_robot.getTime());
SONAR_CONFIGURATION.sonarPrecision[2] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[3] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[3] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[4] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[4] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[5] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[5] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[6] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[6] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[7] = new int[]{0,4,8,12};
SONAR_CONFIGURATION.sonarPrecision[7] = SonarObjectSensor.MEDIUM;

SONAR_CONFIGURATION.sonarActived[8] = lab_plan.getSonar(abstract_robot.getTime());
SONAR_CONFIGURATION.sonarPrecision[8] = SonarObjectSensor.MEDIUM;


//======
// LASER_CONFIGURATION
//======
b_Select_ir
LASER_CONFIGURATION = new b_Select_ir(STATE_MACHINE, abstract_robot);

LASER_CONFIGURATION.embedded[0] = false;
LASER_CONFIGURATION.embedded[1] = false;
LASER_CONFIGURATION.embedded[2] = bring_plan.getLaser(abstract_robot.getTime());
LASER_CONFIGURATION.embedded[3] = false;
LASER_CONFIGURATION.embedded[4] = false;
LASER_CONFIGURATION.embedded[5] = false;
LASER_CONFIGURATION.embedded[6] = false;
LASER_CONFIGURATION.embedded[7] = false;
LASER_CONFIGURATION.embedded[8] = lab_plan.getLaser(abstract_robot.getTime());
```

```
    laser_configuration = LASER_CONFIGURATION;
    sonar_configuration = SONAR_CONFIGURATION;
    turret_configuration = TURRET;
    steering_configuration = STEERING;
    gripper_fingers_configuration = GRIPPER_FINGERS;

  }


  /**
   * Called every timestep to allow the control system to
   * run.
   */
  public int takeStep(){

    Vec2                                                    result;
    double                                                  dresult;
    boolean bresult;
    long                                                    curr_time = abstract_robot.getTime();
    String  action;
    double x,y;
    double x_robot=PS_GLOBAL_POS.Value(curr_time).x;
    double y_robot=PS_GLOBAL_POS.Value(curr_time).y;

    swirlgain.setValue(0.2);
    avoidgain.setValue(0.8);
    noisegain.setValue(0.2);


    MOSTRA.setScore(MIOPUNTEGGIO.Value(curr_time));

    int state=STATE_MACHINE.Value(curr_time);

    flag_memory.setValue(curr_time);

    MAP.setCellVisited(curr_time);

    MAP.setObstacle(PS_SONAR.Value(curr_time), abstract_robot.getPrecisionRivelation(), curr_time);

    Vec2 position=MAP.getPosition(curr_time);
    roomLocation.Value(curr_time);

    // PF usati per mappare fluenti golog
    if(roomLocation.in_lab_down()){
       PF_IN_LOW_LAB.setValue(true);
    }else PF_IN_LOW_LAB.setValue(false);

    if(roomLocation.in_lab_top()){
       PF_IN_HIGH_LAB.setValue(true);
    }else PF_IN_HIGH_LAB.setValue(false);

    if(roomLocation.in_lab_middle()){
       PF_IN_CENTRE_LAB.setValue(true);
    }else PF_IN_CENTRE_LAB.setValue(false);



    if(roomLocation.in_lab()){
       PF_IN_LAB.setValue(true);
    }else PF_IN_LAB.setValue(false);
```

```
// Messagistica

if(bufferMessage.hasMoreElements()) {

   try {
      LongMessage idMessage = (LongMessage) bufferMessage.nextElement();
      if(idMessage.val == other_id && bufferMessage.hasMoreElements()) {
         StringMessage message = (StringMessage)bufferMessage.nextElement();
         if(message.val=="Bandiera nuova") {
            PointMessage flagPosition=(PointMessage) bufferMessage.nextElement();
            Vec2 pos=new Vec2(flagPosition.xValue,flagPosition.yValue);
            LongMessage visualMessage =(LongMessage) bufferMessage.nextElement();
            int vis=(int)visualMessage.val;
            flag_memory.push(pos,vis,curr_time);
         }
         else if(message.val=="Delete Flag") {
            PointMessage flagPosition=(PointMessage) bufferMessage.nextElement();
            Vec2 pos=new Vec2(flagPosition.xValue,flagPosition.yValue);
            flag_memory.pop_only(pos,curr_time);
         }
         else
            if(message.val== "Pronto") {

               LongMessage idFlag=(LongMessage) bufferMessage.nextElement();
               if(idFlag.val==PS_IN_GRIPPER.intValue(curr_time)){
                  PF_EQUAL_FLAG.setValue(true);
               }
               else PF_EQUAL_FLAG.setValue(false);
               PF_OTHER_ROBOT.setValue(true);

            }
      }
   }
   catch(ClassCastException e){}

}
if(roomLocation.in_room()){
   PF_ROOM.setValue(true);
}
else PF_ROOM.setValue(false);


// Stato 0: Il Robot si porta verso una zona che gli consente di osservare
//         la dislocazione delle bandierine nella zona avversaria.
if(state==0){
   if(y_robot<15.0)
      PS_ATTRACTOR.setValue(curr_time,24.0,7.0);
   else
      PS_ATTRACTOR.setValue(curr_time,26.0,17.0);

   // STEER
   result = steering_configuration.Value(curr_time);
   abstract_robot.setSteerHeading(curr_time, result.t);
   abstract_robot.setSpeed(curr_time, result.r);

   // TURRET
   result = turret_configuration.Value(curr_time);
   abstract_robot.setTurretHeading(curr_time, abstract_robot.getTurretHeading(curr_time)+0.4);
   abstract_robot.setDisplayString("Attack");

}
```

```
        // Stato 1: Calcola la bandierina più vicina ad ogni take step,
        //         si dirige verso di essa, affronta le problematiche relative al recupero
        //         delle bandiere
        if(state==1){
            delete=true;

            begin4=true;
            PF_HAVE_FLAG.setValue(false);
            count_3=0;
            delete=false;  // Utilizzata per eliminare la bandierina nello stato 5
            flag=flag_closest.Value(curr_time);

if((roomLocation.flag_in_red(flag)&&roomLocation.closed_red())||(roomLocation.flag_in_green(flag)&&roomLocatio
n.closed_green())||(roomLocation.flag_in_blue(flag)&&roomLocation.closed_blue())||(roomLocation.flag_in_lab(flag)
&&(roomLocation.closed_lab_Left()||roomLocation.closed_lab_Right()))) {
                PF_ENTRY.setValue(true);
            }
            else PF_ENTRY.setValue(false);

            if(roomLocation.in_lab()) {
                PF_EXIT.setValue(true);
            }else PF_EXIT.setValue(false);

            //Se la bandiera è nel lab faccio credere al robot che si trova
            //in un altro punto che risulta  utile per farlo entrare nel lab
            if(roomLocation.flag_in_lab(flag)){
                if(x_robot<22){
                    flag.x=16;
                    flag.y=20;
                }
                else{
                    flag.x=28;
                    flag.y=10;
                }
            }

            PS_ATTRACTOR_FLAG.setValue(curr_time,flag.x,flag.y);
            PS_ATTRACTOR.setValue(curr_time,flag.x,flag.y);


            if(roomLocation.in_room()){
                count_in_room++;

                abstract_robot.setDisplayString("Bandierina nella stanza");
                if(count_in_room>120){
                    PF_EXIT.setValue(true);
                }
                else PF_EXIT.setValue(false);
            }
            else{
                abstract_robot.setDisplayString("Vado alla bandierina vista più vicina");
                count_in_room=0;
                if((flag_memory.getLength(curr_time)==0)){
                    PF_NOT_EMPTY_MEMORY.setValue(false);
                    PF_EMPTY_MEMORY.setValue(true);
                    //System.out.println("Memoria Vuota"+PF_EMPTY_MEMORY.Value(curr_time));
                }
                else {
                    PF_NOT_EMPTY_MEMORY.setValue(true);
                    PF_EMPTY_MEMORY.setValue(false);
```

```
        }

    }

    if(begin1==true){
        begin1=false;
        antistallo.update();
    }
    if(antistallo.isStall(6000)){
        begin1=true;
        flag_memory.pop(flag,curr_time);
        PF_DEADLOCK.setValue(true);
    }
    else PF_DEADLOCK.setValue(false);


    if(localizer.in_special_case(x_robot,y_robot,flag.x,flag.y)){
        PF_GOTO_SPECIAL_ZONE.setValue(true);
    }
    else PF_GOTO_SPECIAL_ZONE.setValue(false);



    // STEER
    result = steering_configuration.Value(curr_time);
    abstract_robot.setSteerHeading(curr_time, result.t);
    abstract_robot.setSpeed(curr_time, result.r);

    // TURRET
    result = turret_configuration.Value(curr_time);
    abstract_robot.setTurretHeading(curr_time,result.t); //abstract_robot.getTurretHeading(curr_time)+0.4);
    //  abstract_robot.setTurretHeading(curr_time, result.t);
    dresult=gripper_fingers_configuration.Value(curr_time);
    abstract_robot.setGripperFingers(curr_time,dresult);

}



// Stato 2:  Consegna delle bandiere.Sono previsti 3 casi:
//                  a. Bandierina nella stanza;
//                   b. Bandierina in una posizione indefinita al di fuori delle
//                      stanze o del labirinto;
//                   c. Il Robot recupera una bandiera e sceglie di usare il
//                      labirinto come porta di scambio tra zona sinistra e destra.
if(state==2){
    delete=true;
    begin8=true;
    stall=0;
    tempo=0;
    PF_HAVE_FLAG.setValue(true);
    if(PF_HAVE_FLAG_LAB.Value(curr_time)==true){
        PF_HAVE_FLAG_LAB.setValue(false);
        flag_memory.pop(lab_flag,curr_time);
        //System.out.println("CAncella flag lab ROBOT:"+abstract_robot.getPlayerNumber(curr_time));
        //System.out.println("Flag:"+lab_flag.x+","+lab_flag.y);
    }
    else {
        flag_memory.pop(flag,curr_time);
    }

    if(roomLocation.use_lab_right()){
```

```
   PF_USE_LAB.setValue(true);
}
else PF_USE_LAB.setValue(false);

if(roomLocation.use_lab_right_redBlue()){
   PF_USE_LAB_RED_BLUE.setValue(true);
}
else PF_USE_LAB_RED_BLUE.setValue(false);

bring_plan.update(curr_time, state);
if(bring_plan.isChangedState()){
   action = bring_plan.getActionName();
   if(action.startsWith("exit")){
      if(roomLocation.in_red_Right()){
         PS_ATTRACTOR.setValue(curr_time,32.0,23.0);
      }
      if(roomLocation.in_green_Right()){
         PS_ATTRACTOR.setValue(curr_time,32.0,14.0);
      }
      if(roomLocation.in_blue_Right()){
         PS_ATTRACTOR.setValue(curr_time,32.0,5.0);
      }
      if(roomLocation.in_red_Left()){
         PS_ATTRACTOR.setValue(curr_time,12.0,23.0);
      }
      if(roomLocation.in_green_Left()){
         PS_ATTRACTOR.setValue(curr_time,12.0,14.0);
      }

      if(roomLocation.in_blue_Left()){
         PS_ATTRACTOR.setValue(curr_time,12.0,5.0);
      }
   }
   else
      if(action.startsWith("goto_lab")){
         if(bring_plan.getParameter()[0].equals("centre")){
            linear_attraction2.setRot_teta(0.0);
            if(y_robot<15.0)
               PS_ATTRACTOR.setValue(curr_time,20.0,12.0);
            else
               PS_ATTRACTOR.setValue(curr_time,24.0,18.0);

         }
         else
            linear_attraction2.setRot_teta(0.0);
         if(bring_plan.getParameter()[0].equals("door1_low")){
            PS_ATTRACTOR.setValue(curr_time,30.5,10.0);
         }
         else
            if(bring_plan.getParameter()[0].equals("door2_low")){
               PS_ATTRACTOR.setValue(curr_time,20.0,12.0);
            }
            else
               if(bring_plan.getParameter()[0].equals("door1_high")){
                  PS_ATTRACTOR.setValue(curr_time,13.5,20.0);
               }
               else
                  if(bring_plan.getParameter()[0].equals("door2_high")){
                     PS_ATTRACTOR.setValue(curr_time,24.0,18.0);
                  }
                  else
                     if(bring_plan.getParameter()[0].equals("door2")){
```

```
                    if(y_robot<15.0)
                        PS_ATTRACTOR.setValue(curr_time,20.0,11.4);
                    else
                        PS_ATTRACTOR.setValue(curr_time,24.0,18.6);
                }
}else
    if (action.startsWith("goto(")){
        if(bring_plan.getParameter()[0].equals("red_bin")){
            PS_ATTRACTOR.setValue(curr_time, 2.5, 23);
            PF_OTHER_ROBOT.setValue(false);
            PF_HAVE_FLAG.setValue(false);
        }
        else
            if(bring_plan.getParameter()[0].equals("green_bin")){
                PS_ATTRACTOR.setValue(curr_time, 2.5, 14);
                PF_OTHER_ROBOT.setValue(false);
                PF_HAVE_FLAG.setValue(false);
            }
            else
                if(bring_plan.getParameter()[0].equals("blue_bin")){
                    PS_ATTRACTOR.setValue(curr_time, 2.5, 5);
                    PF_OTHER_ROBOT.setValue(false);
                    PF_HAVE_FLAG.setValue(false);
                }
                else
                    if(bring_plan.getParameter()[0].equals("red_attractor2")){
                        linear_attraction2.setRot_teta(0.0);
                        if (PF_OTHER_ROBOT.Value(curr_time)&& PF_EQUAL_FLAG.Value(curr_time))
                            PS_ATTRACTOR.setValue(curr_time, 8.0, 25.0);
                        else
                            PS_ATTRACTOR.setValue(curr_time, 8.0, 24.4);

                    }
                    else
                        if(bring_plan.getParameter()[0].equals("green_attractor2")){
                            linear_attraction2.setRot_teta(0.0);
                            if (PF_OTHER_ROBOT.Value(curr_time)&& PF_EQUAL_FLAG.Value(curr_time))
                                PS_ATTRACTOR.setValue(curr_time, 6.0, 13.2);
                            else
                                PS_ATTRACTOR.setValue(curr_time, 6.0, 14.8);
                        }
                        else
                            if(bring_plan.getParameter()[0].equals("blue_attractor2")){
                                linear_attraction2.setRot_teta(0.0);
                                if (PF_OTHER_ROBOT.Value(curr_time)&& PF_EQUAL_FLAG.Value(curr_time))
                                    PS_ATTRACTOR.setValue(curr_time, 8.0, 3.0);
                                else
                                    PS_ATTRACTOR.setValue(curr_time, 8.0, 3.6);
                            }
                            else
                                if(bring_plan.getParameter()[0].equals("red_attractor1")){
                                    PS_ATTRACTOR.setValue(curr_time, 11.5, 23.0);
                                    if(x_robot>=19 && x_robot<27 && y_robot>=3&& y_robot<=8){
                                        linear_attraction2.setRot_teta(0.35);
                                        //System.out.println("Sono nella stanza 5a");
                                    }
                                    else if(x_robot>=27 && x_robot<=33 && y_robot>=2&& y_robot<=8){
                                        linear_attraction2.setRot_teta(0.5);
                                        //System.out.println("Sono nella stanza 5b");
                                    }
                                    else if(x_robot>=25 && x_robot<=33 && y_robot>8 && y_robot <=22){
                                        linear_attraction2.setRot_teta(-1.1);
```

```
                        //System.out.println("Sono nella stanza 2-3-4");
                    }

                }
                else
                    if(bring_plan.getParameter()[0].equals("green_attractor1")){
                        PS_ATTRACTOR.setValue(curr_time, 12.4, 14);
                        if(x_robot>25 && x_robot<33 && y_robot>=15 && y_robot<22){
                            linear_attraction2.setRot_teta(-1.1);
                            //System.out.println("Sono nella stanza 2-3a");
                        }
                        else if(x_robot>=25 && x_robot<=33 && y_robot<15 && y_robot>=8){
                            linear_attraction2.setRot_teta(1.2);
                            //System.out.println("Sono nella stanza 3b-4");
                        }
                        else if(x_robot>=22 && x_robot<=33 && y_robot>22 && y_robot<28){
                            linear_attraction2.setRot_teta(-0.5);
                            //System.out.println("Sono nella stanza 1");
                        }
                        else if(x_robot>=25 && x_robot<=33 && y_robot<15 && y_robot<8){
                            linear_attraction2.setRot_teta(0.5);
                            //System.out.println("Sono nella stanza 5");
                        }
                    }
                    else
                        if(bring_plan.getParameter()[0].equals("blue_attractor1")){
                            PS_ATTRACTOR.setValue(curr_time, 11.5, 5.0);
                            if(x_robot>25 && x_robot<33 && y_robot>8){
                                linear_attraction2.setRot_teta(1.2);
                                //System.out.println("Sono nella stanza 1-2-3-4");
                            }
                            else{
                                linear_attraction2.setRot_teta(0.0);
                            }

                        }
                        else
                            if(bring_plan.getParameter()[0].equals("entry_lab")){
                                PS_ATTRACTOR.setValue(curr_time, 26, 9);
                                //System.out.println("use lab");
                            }
                            else
                                if(bring_plan.getParameter()[0].equals("here")){
                                    PS_ATTRACTOR.setValue(curr_time, x_robot, y_robot);
                                    //System.out.println("here");

                                }


            } else

                if(action.startsWith("send_message(am_at_attractor")){
                    try {
                        LongMessage idMessage = new LongMessage();
                        idMessage.val = abstract_robot.getPlayerNumber(curr_time);
                        abstract_robot.unicast(other_id, idMessage);
                        StringMessage message = new StringMessage();
                        message.val = "Pronto";
                        abstract_robot.unicast(other_id, message);
                        LongMessage idFlag=new LongMessage();
                        idFlag.val=PS_IN_GRIPPER.intValue(curr_time);
                        abstract_robot.unicast(other_id, idFlag);
```

```
                    }
                    catch(CommunicationException e){}

                }

        }



    }

    // Stato 3: Presiede all'impostazione corretta del valore del teta del vettore
    //        d'attrazione lineare
    if(state==3){
        tempo=0;
        begin1=true;
        if(old_teta!=localizer.getTeta(x_robot,y_robot,flag.x,flag.y)){  //gestisce il passaggio da una zona all'altra azzerando il count_3
            count_3=0;
        }

        linear_attraction.setRot_teta(localizer.getTeta(x_robot,y_robot,flag.x,flag.y));//1.0); metodo di Location che ritorna il teta in base alla osizione del robot e della bandierina
        count_3++;
        if(count_3>=localizer.getStopper(x_robot,y_robot,flag.x,flag.y)){//300){//metodo di Location che ritorna il contatore in base ecc,ecc
            count_3=0;
            PF_STOP_GOTO_SPECIAL_ZONE.setValue(true);
        }
        else
            PF_STOP_GOTO_SPECIAL_ZONE.setValue(false);



        PS_SPECIAL_ATTRACTOR.setValue(curr_time,flag.x,flag.y);
        abstract_robot.setDisplayString(localizer.getString(x_robot,y_robot,flag.x,flag.y));//"Sto andando nella zona alta"); ritorna la stringa in base.....


        // STEER
        result = steering_configuration.Value(curr_time);
        abstract_robot.setSteerHeading(curr_time, result.t);
        abstract_robot.setSpeed(curr_time, result.r);

        // TURRET
        result = turret_configuration.Value(curr_time);
        abstract_robot.setTurretHeading(curr_time, result.t);

        // FINGERS
        dresult=gripper_fingers_configuration.Value(curr_time);
        abstract_robot.setGripperFingers(curr_time,dresult);

        old_teta=localizer.getTeta(x_robot,y_robot,flag.x,flag.y);

        linear_attraction.setRot_teta(0.0);
    }


    // Stato 4: Esplorazione:sistema a produzioni
    //        Esce da questa stato perchè uno dei due robot ha avvistato qualcosa
```

```
    if(state==4){
       begin1=true;
       tempo=0;
       stall=0;
       abstract_robot.setDisplayString("Explore");
       noisegain.setValue(0.5);
       if(roomLocation.out_of_bounds()){
          PF_BOUNDS.setValue(true);
          //System.out.println("esco a causa del bounds"+abstract_robot.getPlayerNumber(curr_time));
       }
       else
          PF_BOUNDS.setValue(false);


if((!red_room_Left&&roomLocation.closed_red_Left())||(!green_room_Left&&roomLocation.closed_green_Left())||(!b
lue_room_Left&&roomLocation.closed_blue_Left())||(!lab_room&&roomLocation.closed_lab_())
||(!red_room_Right&&roomLocation.closed_red_Right())||(!green_room_Right&&roomLocation.closed_green_Right()
)||(!blue_room_Right&&roomLocation.closed_blue_Right())) {
          CHOOSE_ZONE.setPriority(1,2,0,3);
          PF_ENTRY.setValue(true);
       }
       else PF_ENTRY.setValue(false);

       if(roomLocation.in_room()){
          abstract_robot.setDisplayString("Explore stanza");
          swirlgain.setValue(0.5);
          avoidgain.setValue(0.0);
          noisegain.setValue(0.0);

          if(begin4==true){
             begin4=false;
             antistallo.update();
          }
          if(antistallo.isStall(6000)){
             begin4=true;
             PF_EXIT.setValue(true);

          }
          else PF_EXIT.setValue(false);

       }

       if(flag_memory.getLength(curr_time)!=0){
          PF_EMPTY_MEMORY.setValue(false);
          PF_NOT_EMPTY_MEMORY.setValue(true);
       }
       else{
          PF_EMPTY_MEMORY.setValue(true);
          PF_NOT_EMPTY_MEMORY.setValue(false);
       }


       // STEER
       result = steering_configuration.Value(curr_time);
       abstract_robot.setSteerHeading(curr_time, result.t);
       abstract_robot.setSpeed(curr_time, result.r);

       // TURRET
       result = turret_configuration.Value(curr_time);
       abstract_robot.setTurretHeading(curr_time, abstract_robot.getTurretHeading(curr_time)+0.4);

       // FINGERS
```

```
    dresult=gripper_fingers_configuration.Value(curr_time);
    abstract_robot.setGripperFingers(curr_time,dresult);
  }


  // Stato 5: E' uno stato intermedio tra la localizzazione e direzione immediata
  //          verso la bandiera avvistata (stato 1) e la pronta consegna (stato 2).
  if(state==5){
    begin1=true;
    tempo=0;
    flag=flag_closest.Value(curr_time);
    PS_ATTRACTOR_FLAG.setValue(curr_time,flag.x,flag.y);
    PS_ATTRACTOR.setValue(curr_time,flag.x,flag.y);

    if(delete==true){
      delete=false;
      antistallo.update();
    }
    if(antistallo.isStall(3500)){
      delete=true;
      flag_memory.pop(flag,curr_time);
      PF_STALL.setValue(true);
    }
    else PF_STALL.setValue(false);


    // STEER
    result = steering_configuration.Value(curr_time);
    abstract_robot.setSteerHeading(curr_time, result.t);
    abstract_robot.setSpeed(curr_time, result.r);


    // TURRET
    result = turret_configuration.Value(curr_time);
    abstract_robot.setTurretHeading(curr_time,result.t);



    dresult=gripper_fingers_configuration.Value(curr_time);
    abstract_robot.setGripperFingers(curr_time,dresult);
    abstract_robot.setDisplayString("Devo prendere la bandierina");
  }

  // Stato 6: Consente al robot di uscire dalle stanze
  if(state==6){
    begin1=true;
    begin4=true;
    PF_BOUNDS.setValue(false);

    CHOOSE_ZONE.setPriority(0,3,1,2);
    if(roomLocation.in_red_Right()){
      PS_ATTRACTOR.setValue(curr_time,31.0,23.0);
      PF_EXIT.setValue(false);
      //System.out.println("Sono nella stanza rossa.Sto uscendo...");
      red_room_Right=true;
    }

    if(roomLocation.in_green_Right()){
      PS_ATTRACTOR.setValue(curr_time,31.0,14.0);
      PF_EXIT.setValue(false);
      //System.out.println("Sono nella stanza verde.Sto uscendo...");
      green_room_Right=true;
```

```
    }

    if(roomLocation.in_blue_Right()){
       PS_ATTRACTOR.setValue(curr_time,31.0,5.0);
       PF_EXIT.setValue(false);
       //System.out.println("Sono nella stanza blue.Sto uscendo...");
       blue_room_Right=true;
    }

    if(roomLocation.in_red_Left()){
       PS_ATTRACTOR.setValue(curr_time,13.0,23.0);
       //System.out.println("Sono nella stanza rossa.Sto uscendo...");
       PF_EXIT.setValue(false);
       red_room_Left=true;
    }

    if(roomLocation.in_blue_Left()){
       PS_ATTRACTOR.setValue(curr_time,13.0,5.0);
       //System.out.println("Sono nella stanza blue.Sto uscendo...");
       PF_EXIT.setValue(false);
       blue_room_Left=true;
    }

    if(roomLocation.in_green_Left()){
       PS_ATTRACTOR.setValue(curr_time,13.0,14.0);
       //System.out.println("Sono nella stanza verde.Sto uscendo...");
       PF_EXIT.setValue(false);
       green_room_Left=true;
    }
    if(roomLocation.in_lab_down()){
       PF_EXIT.setValue(false);
       PS_ATTRACTOR.setValue(curr_time,30,7);
    }
    if(roomLocation.in_lab_top()){
       PF_EXIT.setValue(false);
       PS_ATTRACTOR.setValue(curr_time,14,23);
    }

    // STEER
    result = steering_configuration.Value(curr_time);
    abstract_robot.setSteerHeading(curr_time, result.t);
    abstract_robot.setSpeed(curr_time, result.r);

    // TURRET
    result = turret_configuration.Value(curr_time);
    abstract_robot.setTurretHeading(curr_time,result.t);
    dresult=gripper_fingers_configuration.Value(curr_time);
    abstract_robot.setGripperFingers(curr_time,dresult);
    abstract_robot.setDisplayString("EXIT");

  }

  // Stato 7: Questo stato scatta nel momento in cui il robot deve visitare le stanze o
  //         il labirinto
  if(state==7){
    begin1=true;
    begin4=true;
    PF_ENTRY.setValue(false);
    count_in_room=0;
    if(roomLocation.closed_red_Right()){
       PS_ATTRACTOR.setValue(curr_time,34.0,23.0);
       abstract_robot.setDisplayString("Entro nella stanza");
```

```
    }

    if(roomLocation.closed_green_Right()){
       PS_ATTRACTOR.setValue(curr_time,35.0,14.0);
       abstract_robot.setDisplayString("Entro nella stanza");
    }

    if(roomLocation.closed_blue_Right()){
       PS_ATTRACTOR.setValue(curr_time,34.0,5.0);
       abstract_robot.setDisplayString("Entro nella stanza");
    }
    if(roomLocation.closed_red_Left()){
       PS_ATTRACTOR.setValue(curr_time,10.0,23.0);
       abstract_robot.setDisplayString("Entro nella stanza");
    }

    if(roomLocation.closed_green_Left()){
       PS_ATTRACTOR.setValue(curr_time,9.0,14.0);
       abstract_robot.setDisplayString("Entro nella stanza");
    }

    if(roomLocation.closed_blue_Left()){
       PS_ATTRACTOR.setValue(curr_time,10.0,5.0);
       abstract_robot.setDisplayString("Entro nella stanza");
    }

    if(roomLocation.closed_lab_Left()){
       PS_ATTRACTOR.setValue(curr_time,21.0,20.0);
       abstract_robot.setDisplayString("Entro nel lab dall'alto");
    }

    if(roomLocation.closed_lab_Right()){
       PS_ATTRACTOR.setValue(curr_time,24.0,10.0);
       abstract_robot.setDisplayString("Entro nel lab dal basso");
    }

    if(roomLocation.in_lab()){
       PF_LAB.setValue(true);
    }
    else PF_LAB.setValue(false);

    // STEER
    result = steering_configuration.Value(curr_time);
    abstract_robot.setSteerHeading(curr_time, result.t);
    abstract_robot.setSpeed(curr_time, result.r);

    // TURRET
    result = turret_configuration.Value(curr_time);
    abstract_robot.setTurretHeading(curr_time,result.t);
    dresult=gripper_fingers_configuration.Value(curr_time);
    abstract_robot.setGripperFingers(curr_time,dresult);

  }



  // Stato 8: Implementa l'esplorazione e più in generale l'utilizzo del labirinto
  if(state==8){
    //                                                                PF_HAVE_FLAG.setValue(false);
    lab_plan.update(curr_time, state);
    if(!begin_flag){
       if(antistallo.isStall(3000)){
```

```
        begin_flag=true;
        flag_memory.pop(lab_flag,curr_time);
        PF_HAVE_FLAG.setValue(false);
        PF_HAVE_FLAG_LAB.setValue(false);
        PF_STALL_FLAG.setValue(true);
      }
    }
    else {
      PF_STALL_FLAG.setValue(false);
    }

    if(lab_plan.isChangedState()){

      action = lab_plan.getActionName();
      if(action.startsWith("goto_lab")){
        if(lab_plan.getParameter()[0].equals("centre")){
          if(y_robot<15.0)
            PS_ATTRACTOR.setValue(curr_time,20.0,12.0);
          else
            PS_ATTRACTOR.setValue(curr_time,24.0,18.0);

        }
        else
          if(lab_plan.getParameter()[0].equals("door1_low")){
            PS_ATTRACTOR.setValue(curr_time,30.5,10.0);
          }
          else
            if(lab_plan.getParameter()[0].equals("door2_low")){
              PS_ATTRACTOR.setValue(curr_time,20.0,12.0);
            }
            else
              if(lab_plan.getParameter()[0].equals("door1_high")){
                PS_ATTRACTOR.setValue(curr_time,13.5,20.0);
              }
              else
                if(lab_plan.getParameter()[0].equals("door2_high")){
                  PS_ATTRACTOR.setValue(curr_time,24.0,18.0);
                }
                else
                  if(lab_plan.getParameter()[0].equals("door2")){
                    if(y_robot<15.0)
                      PS_ATTRACTOR.setValue(curr_time,20.0,11.4);
                    else
                      PS_ATTRACTOR.setValue(curr_time,24.0,18.6);
                  }
      }
      if(action.startsWith("explore")){
        if(lab_plan.getParameter()[0].equals("entry")){
          if(y_robot<15.0){
            PF_COME_FROM_LOW.setValue(true);
            PS_ATTRACTOR.setValue(curr_time,20.0,10.0);
          }
          else {

            PF_COME_FROM_LOW.setValue(false);
            PS_ATTRACTOR.setValue(curr_time,24.0,20.0);
          }
        }
        else
          if(lab_plan.getParameter()[0].equals("exit")){
            if(y_robot>15.0){
              PS_ATTRACTOR.setValue(curr_time,17.0,20.0);
```

```
                }
                else {
                    PS_ATTRACTOR.setValue(curr_time,27.0,10.0);
                }
            }
            else
                if(lab_plan.getParameter()[0].equals("attractor1")){
                    if(y_robot<15.0){
                        int y_rand=(Math.abs(random.nextInt())%4)+13;
                        //System.out.println(y_rand);
                        PS_ATTRACTOR.setValue(curr_time,24.6,y_rand);
                        come_from_low=true;
                    }
                    else {
                        int y_rand=(Math.abs(random.nextInt())%4)+13;
                        //System.out.println(y_rand);
                        PS_ATTRACTOR.setValue(curr_time,19.4,y_rand);
                        come_from_low=false;
                    }

                }
                else
                    if(lab_plan.getParameter()[0].equals("attractor2")){
                        if(come_from_low){
                            int y_rand=(Math.abs(random.nextInt())%4)+13;
                            //System.out.println(y_rand);
                            PS_ATTRACTOR.setValue(curr_time,19.4,y_rand);
                        }
                        else {
                            int y_rand=(Math.abs(random.nextInt())%4)+13;
                            //System.out.println(y_rand);
                            PS_ATTRACTOR.setValue(curr_time,24.6,y_rand);
                        }
                    }else
                        if(lab_plan.getParameter()[0].equals("door2")){
                            if(come_from_low){
                                PS_ATTRACTOR.setValue(curr_time,24.0,17.8);
                            }
                            else {
                                PS_ATTRACTOR.setValue(curr_time,20.0,12.2);
                            }

                        }

        }
        if(action.startsWith("go_flag")){
            begin_flag=false;
            antistallo.update();
            lab_flag=new
Vec2(PS_CLOSEST_FLAG.Value(curr_time).x+x_robot,PS_CLOSEST_FLAG.Value(curr_time).y+y_robot);


            flag_memory.pop(lab_flag,curr_time);
            PF_HAVE_FLAG.setValue(true);
            PF_HAVE_FLAG_LAB.setValue(true);
            abstract_robot.setDisplayString("Vado al flag");
            //System.out.println("posiziona bandiera"+ PS_CLOSEST_FLAG.Value(curr_time).x+
PS_CLOSEST_FLAG.Value(curr_time).y);

        }
    }
    abstract_robot.setDisplayString("Labirinto");
```

```
        }


        return(CSSTAT_OK);
    }

    // Consente l'aggiornamento del contatore delle bandierine
    public void trialEnd(){

        if (MOSTRA!=null) MOSTRA.dispose();


    }

}
```

Classe Location.java

```
/*
 * Scompone la mappa in zone di differente campo potenziale settando l'angolo teta
 * della classe v_LinearAttraction_v_Try_it.
 * Il robot,quindi,è soggetto ad un campo potenziale differente a secondo della zona
 * in cui si trova e dell'ostacolo che si prepara ad affrontare per arrivare al goal. *
 *
 **/



public class Location {

    //Zone in cui si può trovare il robot
    private boolean zone_A;
    private boolean zone_B;
    private boolean zone_C;
    private boolean zone_D;
    private boolean zone_E;
    private boolean zone_F;
    private boolean zone_G;
    private boolean zone_H;
    private boolean zone_I;
    private boolean zone_L;
    private boolean zone_M;
    private boolean zone_N;
    private boolean zone_O;
    private boolean zone_P;

    //                                                                  Zone in cui si possono trovare le
bandierine
    private boolean flag_1;
    private boolean flag_2;
    private boolean flag_3;
    private boolean flag_4;
    private boolean flag_5;
    private boolean flag_6;
    private boolean flag_7;
    private boolean flag_8;
    private boolean flag_9;
    private boolean flag_10;

    private boolean any_zone_flag_sx,any_zone_flag_dx; //Indicano se le bandierine si trovano in qualche zona di quelle
precedentemente dichiarate



    public Location(){ }    //costruttore



       /*
        *  METODO in_special_case
        *
        *Indica se il robot si trova in una zona a cui bisogna applicare un particolare
        *angolo teta.Questo,ovviamente,è dipendente anche dalla posizione della bandierina
        *
```

```
     */
   public boolean in_special_case(double x_rob,double y_rob,double flag_x,double flag_y){

      zone_A = x_rob<19 && x_rob>11 && y_rob<8;               //robot in zone A?
      zone_B = x_rob<19 && x_rob>11 && y_rob>=8 && y_rob<12;   //robot in zone B?
      zone_C = x_rob<19 && x_rob>13 && y_rob>=12 && y_rob<15;  //robot in zone C?
      zone_D = x_rob<19 && x_rob>13 && y_rob>=15 && y_rob<18;  //robot in zone D?
      zone_E = x_rob<19 && x_rob>11 && y_rob>=18 && y_rob<22;  //robot in zone E?
      zone_F = x_rob<19 && x_rob>11 && y_rob>=22 && y_rob<28;  //robot in zone F?

      zone_G = x_rob>25 && x_rob<33 && y_rob>22;               //robot in zone G?
      zone_H = x_rob>25 && x_rob<33 && y_rob<=22 && y_rob>18;  //robot in zone H?
      zone_I = x_rob>25 && x_rob<31 && y_rob<=18 && y_rob>15;  //robot in zone I?
      zone_L = x_rob>25 && x_rob<31 && y_rob<=15 && y_rob>12 ;  //robot in zone L?
      zone_M = x_rob>27 && x_rob<33 && y_rob<=12 && y_rob>8;    //robot in zone M?
      zone_N = x_rob>27 && x_rob<33 && y_rob<=8 && y_rob>0;    //robot in zone N?

      zone_O = x_rob>=19 && x_rob<=29 && y_rob<8 && y_rob>0;    //robot in zone O?
      zone_P = x_rob>=16 && x_rob<=25 && y_rob<28 && y_rob>22;    //robot in zone P?


      flag_1 = flag_x>17 && flag_x<33 && flag_y<28 && flag_y>22;  //flag in zone 1?
      flag_2 = flag_x>25 && flag_x<33 && flag_y<=22 && flag_y>18; //flag in zone 2?
      flag_3 = flag_x>25 && flag_x<33 && flag_y<=18 && flag_y>12; //flag in zone 3?
      flag_4 = flag_x>25 && flag_x<33 && flag_y<=12 && flag_y>8;  //flag in zone 4?
      flag_5 = flag_x>19 && flag_x<33 && flag_y<=8 && flag_y>0;   //flag in zone 5?

      flag_6 = flag_x>11 && flag_x<27 && flag_y<=8 && flag_y>0;  //flag in zone 6?
      flag_7 = flag_x>11 && flag_x<19 && flag_y<=12 && flag_y>8; //flag in zone 7?
      flag_8 = flag_x>11 && flag_x<19 && flag_y<=18 && flag_y>12; //flag in zone 8?
      flag_9 = flag_x>11 && flag_x<19 && flag_y<=22 && flag_y>18;  //flag in zone 9?
      flag_10 = flag_x>11 && flag_x<25 && flag_y<28 && flag_y>22;   //flag in zone 10?



      any_zone_flag_dx=flag_1 || flag_2 || flag_3 || flag_4 || flag_5;  //la bandierina si trova in almeno una delle zone
1,2,3,4,5?

      any_zone_flag_sx=flag_6 || flag_7 || flag_8 || flag_9 || flag_10;  //la bandierina si trova in almeno una delle zone
6,7,8,9,10?



      //Condizioni per valutare se il robot e la bandierina si trovano in un caso particolare,cioè se si trovano in
determinate zone
      if((zone_A && (flag_1 || flag_2 || flag_3 || flag_4)) || (zone_B && any_zone_flag_dx) || (zone_C &&
any_zone_flag_dx) || (zone_D && any_zone_flag_dx) || (zone_E && any_zone_flag_dx) || (zone_F && (flag_2 ||
flag_3 || flag_4 || flag_5)))

         return true;

      if((zone_G && (flag_6 || flag_7 || flag_8 || flag_9)) || (zone_H && any_zone_flag_sx) || (zone_I &&
any_zone_flag_sx) || (zone_L && any_zone_flag_sx) || (zone_M && any_zone_flag_sx) || (zone_N && (flag_7 ||
flag_8 || flag_9 || flag_10)))

         return true;

      if(zone_O && (flag_1 || flag_2 || flag_3 || flag_4 || flag_7 || flag_8 || flag_9 || flag_10))

         return true;
```

```
    if(zone_P && (flag_2 || flag_3 || flag_4 || flag_5 || flag_6 || flag_7 || flag_8 || flag_9))

        return true;

    else

        return false;


}


    /*
     *  METODO getTeta
     *
     * Ritorna l'angolo di rotazione teta del vettore di attrazione lineare in base
     * alla zona in cui si trova il robot,agli ostacoli che lo circondano e alla
     * posizione della bandierina.
     *
     */

public double getTeta(double x_rob,double y_rob,double flag_x,double flag_y){

    zone_A = x_rob<19 && x_rob>11 && y_rob<8;                //robot in zone A?
    zone_B = x_rob<19 && x_rob>11 && y_rob>=8 && y_rob<12;   //robot in zone B?
    zone_C = x_rob<19 && x_rob>13 && y_rob>=12 && y_rob<15;  //robot in zone C?
    zone_D = x_rob<19 && x_rob>13 && y_rob>=15 && y_rob<18;  //robot in zone D?
    zone_E = x_rob<19 && x_rob>11 && y_rob>=18 && y_rob<22;  //robot in zone E?
    zone_F = x_rob<16 && x_rob>11 && y_rob>=22 && y_rob<28;  //robot in zone F?

    zone_G = x_rob>25 && x_rob<33 && y_rob>22;               //robot in zone G?
    zone_H = x_rob>25 && x_rob<33 && y_rob<=22 && y_rob>18;  //robot in zone H?
    zone_I = x_rob>25 && x_rob<31 && y_rob<=18 && y_rob>15;  //robot in zone I?
    zone_L = x_rob>25 && x_rob<31 && y_rob<=15 && y_rob>12 ;  //robot in zone L?
    zone_M = x_rob>27 && x_rob<33 && y_rob<=12 && y_rob>8;    //robot in zone M?
    zone_N = x_rob>27 && x_rob<33 && y_rob<=8 && y_rob>0;    //robot in zone N?

    zone_O = x_rob>=19 && x_rob<=29 && y_rob<8 && y_rob>0;    //robot in zone O?
    zone_P = x_rob>=15 && x_rob<=25 && y_rob<28 && y_rob>22;    //robot in zone P?


    flag_1 = flag_x>17 && flag_x<33 && flag_y<28 && flag_y>22;  //flag in zone 1?
    flag_2 = flag_x>25 && flag_x<33 && flag_y<=22 && flag_y>18; //flag in zone 2?
    flag_3 = flag_x>25 && flag_x<33 && flag_y<=18 && flag_y>12; //flag in zone 3?
    flag_4 = flag_x>25 && flag_x<33 && flag_y<=12 && flag_y>8;  //flag in zone 4?
    flag_5 = flag_x>19 && flag_x<33 && flag_y<=8 && flag_y>0;   //flag in zone 5?

    flag_6 = flag_x>11 && flag_x<27 && flag_y<=8 && flag_y>0;  //flag in zone 6?
    flag_7 = flag_x>11 && flag_x<19 && flag_y<=12 && flag_y>8; //flag in zone 7?
    flag_8 = flag_x>11 && flag_x<19 && flag_y<=18 && flag_y>12; //flag in zone 8?
    flag_9 = flag_x>11 && flag_x<19 && flag_y<=22 && flag_y>18;  //flag in zone 9?
    flag_10 = flag_x>11 && flag_x<25 && flag_y<28 && flag_y>22;   //flag in zone 10?



    if(x_rob<20){ //robot in zona sinistra

        if((zone_A || zone_B) && (flag_1 || flag_2)){
            return 1.2;
        }
        if((zone_C || zone_D || zone_E) && (flag_1 || flag_2)){
            return 1.2;
```

```
    }
    if((zone_F) && (flag_2)){
       return 0.4;
    }


    if((zone_A || zone_B) && ( flag_3)){
       return -1.15;
    }
    if((zone_C) && (flag_3)){
       return -1.5;
    }
    if((zone_D) && (flag_3)){
       return 1.5;
    }
    if((zone_E || zone_F) && ( flag_3)){
       return 0.8;
    }


    if((zone_A || zone_B) && ( flag_4)){
       return -1.15;
    }
    if((zone_B) && (flag_5)){
       return -0.45;
    }
    if((zone_C || zone_D) && (flag_4 || flag_5)){
       return -0.75;
    }
    if((zone_E || zone_F) && (flag_4)){
       return 1.0;
    }
    if((zone_E || zone_F) && ( flag_5)){
       return -0.45;
    }


}

else{  //robot in zona destra

    if((zone_G || zone_H) && (flag_6 || flag_7)){
       return 1.2;
    }
    if((zone_I || zone_L || zone_M) && (flag_6 || flag_7)){
       return 1.2;
    }
    if((zone_N) && (flag_7)){
       return 0.4;
    }


    if((zone_G || zone_H) && ( flag_8)){
       return -1.15;
    }
    if((zone_I) && (flag_8)){
       return -1.5;
    }
    if((zone_L) && (flag_8)){
       return 1.5;
    }
    if((zone_M) && ( flag_8)){
```

```
            return 1.2;
        }
        if((zone_N) && ( flag_8)){
            return 1.0;
        }


        if((zone_G || zone_H) && ( flag_9)){
            return -0.45;
        }
        if((zone_H) && (flag_10)){
            return -0.45;
        }
        if((zone_I || zone_L) && (flag_9 || flag_10)){
            return -0.75;
        }
        if((zone_M || zone_N) && (flag_9)){
            return 1.0;
        }
        if((zone_M || zone_N) && ( flag_10)){
            return -0.45;
        }

    }

    if((zone_O) && ( flag_1 || flag_2 || flag_3 || flag_4 )){
        return -1.2;
    }
    if((zone_O) && ( flag_7 || flag_8 || flag_9 || flag_10 )){
        return 1.2;
    }
    if((zone_P) && ( flag_2 || flag_3 || flag_4 || flag_5)){
        return 1.2;
    }
    if((zone_P) && ( flag_6 || flag_7 || flag_8 || flag_9)){
        return -1.2;
    }


    return 0.0;

}


    /*
     *  METODO getStopper
     *
     * Ritorna il tempo per cui il robot deve tener conto del campo potenziale
     * della zona in cui si trova.
     *
     */


  public int getStopper(double x_rob,double y_rob,double flag_x,double flag_y){

    zone_A = x_rob<19 && x_rob>11 && y_rob<8;                //robot in zone A?
    zone_B = x_rob<19 && x_rob>11 && y_rob>=8 && y_rob<12;   //robot in zone B?
    zone_C = x_rob<19 && x_rob>13 && y_rob>=12 && y_rob<15;  //robot in zone C?
    zone_D = x_rob<19 && x_rob>13 && y_rob>=15 && y_rob<18;  //robot in zone D?
    zone_E = x_rob<19 && x_rob>11 && y_rob>=18 && y_rob<22;  //robot in zone E?
    zone_F = x_rob<16 && x_rob>11 && y_rob>=22 && y_rob<28;  //robot in zone F?
```

```
zone_G = x_rob>25 && x_rob<33 && y_rob>22;              //robot in zone G?
zone_H = x_rob>25 && x_rob<33 && y_rob<=22 && y_rob>18;  //robot in zone H?
zone_I = x_rob>25 && x_rob<31 && y_rob<=18 && y_rob>15;  //robot in zone I?
zone_L = x_rob>25 && x_rob<31 && y_rob<=15 && y_rob>12 ;  //robot in zone L?
zone_M = x_rob>27 && x_rob<33 && y_rob<=12 && y_rob>8;    //robot in zone M?
zone_N = x_rob>27 && x_rob<33 && y_rob<=8 && y_rob>0;    //robot in zone N?

zone_O = x_rob>=19 && x_rob<=29 && y_rob<8 && y_rob>0;    //robot in zone O?
zone_P = x_rob>=15 && x_rob<=25 && y_rob<28 && y_rob>22;    //robot in zone P?


flag_1 = flag_x>17 && flag_x<33 && flag_y<28 && flag_y>22;  //flag in zone 1?
flag_2 = flag_x>25 && flag_x<33 && flag_y<=22 && flag_y>18; //flag in zone 2?
flag_3 = flag_x>25 && flag_x<33 && flag_y<=18 && flag_y>12; //flag in zone 3?
flag_4 = flag_x>25 && flag_x<33 && flag_y<=12 && flag_y>8;  //flag in zone 4?
flag_5 = flag_x>19 && flag_x<33 && flag_y<=8 && flag_y>0;   //flag in zone 5?

flag_6 = flag_x>11 && flag_x<27 && flag_y<=8 && flag_y>0;  //flag in zone 6?
flag_7 = flag_x>11 && flag_x<19 && flag_y<=12 && flag_y>8; //flag in zone 7?
flag_8 = flag_x>11 && flag_x<19 && flag_y<=18 && flag_y>12; //flag in zone 8?
flag_9 = flag_x>11 && flag_x<19 && flag_y<=22 && flag_y>18;  //flag in zone 9?
flag_10 = flag_x>11 && flag_x<25 && flag_y<28 && flag_y>22;   //flag in zone 10?

if(x_rob<20){                                                    //robot in zona sinistra

   if((zone_A || zone_B) && (flag_1 || flag_2)){
      return 250;
   }
   if((zone_C || zone_D) && (flag_1 || flag_2)){
      return 150;
   }
   if((zone_E) && (flag_1)){
      return 75;
   }
   if((zone_E || zone_F) && (flag_2)){
      return 100;
   }


   if((zone_A || zone_B) && ( flag_3)){
      return 400;
   }
   if((zone_C) && (flag_3)){
      return 250;
   }
   if((zone_D) && (flag_3)){
      return 250;
   }
   if((zone_E || zone_F) && ( flag_3)){
      return 230;
   }


   if((zone_A || zone_B) && ( flag_4)){
      return 350;
   }
   if((zone_B) && (flag_5)){
      return 40;
   }
   if((zone_C || zone_D) && (flag_4 || flag_5)){
      return 150;
   }
```

```
    if((zone_E || zone_F) && (flag_4)){
      return 75;
    }
    if((zone_E || zone_F) && ( flag_5)){
      return 200;
    }

  }

  else{  //robot in zona destra

    if((zone_G || zone_H) && (flag_6 || flag_7)){
      return 250;
    }
    if((zone_I || zone_L) && (flag_6 || flag_7)){
      return 150;
    }
    if((zone_M) && (flag_6)){
      return 75;
    }
    if((zone_M || zone_N) && (flag_7)){
      return 100;
    }


    if((zone_G || zone_H) && ( flag_8)){
      return 400;
    }
    if((zone_I) && (flag_8)){
      return 250;
    }
    if((zone_L) && (flag_8)){
      return 250;
    }
    if((zone_M || zone_N) && ( flag_8)){
      return 230;
    }


    if((zone_G|| zone_H) && ( flag_9)){
      return 350;
    }
    if((zone_H) && (flag_10)){
      return 40;
    }
    if((zone_I || zone_L) && (flag_9 || flag_10)){
      return 150;
    }
    if((zone_M || zone_N) && (flag_9)){
      return 75;
    }
    if((zone_M || zone_N) && ( flag_10)){
      return 200;
    }


  }

  if((zone_O) && ( flag_1 || flag_2 || flag_3 || flag_4 || flag_7 || flag_8 || flag_9 || flag_10)){
    return 75;
  }
  if((zone_P) && ( flag_2 || flag_3 || flag_4 || flag_5 || flag_6 || flag_7 || flag_8 || flag_9)){
```

```
        return 75;
    }


    return 0;

}


    /*
     *  METODO getString
     *
     * Ritorna la stringa che indica in quale particolare zona della mappa il robot
     * si trova e che verrà visualizzata nella simulazione.
     *
     */


public String getString(double x_rob,double y_rob,double flag_x,double flag_y){

    zone_A = x_rob<19 && x_rob>11 && y_rob<8;                //robot in zone A?
    zone_B = x_rob<19 && x_rob>11 && y_rob>=8 && y_rob<12;   //robot in zone B?
    zone_C = x_rob<19 && x_rob>13 && y_rob>=12 && y_rob<15;  //robot in zone C?
    zone_D = x_rob<19 && x_rob>13 && y_rob>=15 && y_rob<18;  //robot in zone D?
    zone_E = x_rob<19 && x_rob>11 && y_rob>=18 && y_rob<22;  //robot in zone E?
    zone_F = x_rob<16 && x_rob>11 && y_rob>=22 && y_rob<28;  //robot in zone F?

    zone_G = x_rob>25 && x_rob<33 && y_rob>22;               //robot in zone G?
    zone_H = x_rob>25 && x_rob<33 && y_rob<=22 && y_rob>18;  //robot in zone H?
    zone_I = x_rob>25 && x_rob<31 && y_rob<=18 && y_rob>15;  //robot in zone I?
    zone_L = x_rob>25 && x_rob<31 && y_rob<=15 && y_rob>12 ;  //robot in zone L?
    zone_M = x_rob>27 && x_rob<33 && y_rob<=12 && y_rob>8;    //robot in zone M?
    zone_N = x_rob>27 && x_rob<33 && y_rob<=8 && y_rob>0;    //robot in zone N?

    zone_O = x_rob>=19 && x_rob<=29 && y_rob<8 && y_rob>0;    //robot in zone O?
    zone_P = x_rob>=15 && x_rob<=25 && y_rob<28 && y_rob>22;    //robot in zone P?


    flag_1 = flag_x>17 && flag_x<33 && flag_y<28 && flag_y>22;  //flag in zone 1?
    flag_2 = flag_x>25 && flag_x<33 && flag_y<=22 && flag_y>18; //flag in zone 2?
    flag_3 = flag_x>25 && flag_x<33 && flag_y<=18 && flag_y>12; //flag in zone 3?
    flag_4 = flag_x>25 && flag_x<33 && flag_y<=12 && flag_y>8;  //flag in zone 4?
    flag_5 = flag_x>19 && flag_x<33 && flag_y<=8 && flag_y>0;   //flag in zone 5?

    flag_6 = flag_x>11 && flag_x<27 && flag_y<=8 && flag_y>0;  //flag in zone 6?
    flag_7 = flag_x>11 && flag_x<19 && flag_y<=12 && flag_y>8; //flag in zone 7?
    flag_8 = flag_x>11 && flag_x<19 && flag_y<=18 && flag_y>12; //flag in zone 8?
    flag_9 = flag_x>11 && flag_x<19 && flag_y<=22 && flag_y>18;  //flag in zone 9?
    flag_10 = flag_x>11 && flag_x<25 && flag_y<28 && flag_y>22;   //flag in zone 10?

    if(x_rob<20){

        if((zone_A || zone_B) && (flag_1 || flag_2)){
            return "Zone A/B to ZoneFlag 1/2";
        }
        if((zone_C || zone_D) && (flag_1 || flag_2)){
            return "Zone C/D to ZoneFlag 1/2";
        }
        if((zone_E) && (flag_1)){
            return "Zone E to ZoneFlag 1";
        }
        if((zone_E || zone_F) && (flag_2)){
```

```
        return "Zone E/F to ZoneFlag 2";
    }


    if((zone_A || zone_B) && ( flag_3)){
        return "Zone A/B to ZoneFlag 3";
    }
    if((zone_C) && (flag_3)){
        return "Zone C to ZoneFlag 3";
    }
    if((zone_D) && (flag_3)){
        return "Zone D to ZoneFlag 3";
    }
    if((zone_E || zone_F) && ( flag_3)){
        return "Zone E/F to ZoneFlag 3";
    }


    if((zone_A || zone_B) && ( flag_4)){
        return "Zone A/B to ZoneFlag 4";
    }
    if((zone_B) && (flag_5)){
        return "Zone B to ZoneFlag 5";
    }
    if((zone_C || zone_D) && (flag_4 || flag_5)){
        return "Zone C/D to ZoneFlag 4/5";
    }
    if((zone_E || zone_F) && (flag_4)){
        return "Zone E/F to ZoneFlag 4";
    }
    if((zone_E || zone_F) && ( flag_5)){
        return "Zone E/F to ZoneFlag 5";
    }

}

else{

    if((zone_G || zone_H) && (flag_6 || flag_7)){
        return "Zone G/H to ZoneFlag 6/7";
    }
    if((zone_I || zone_L) && (flag_6 || flag_7)){
        return "Zone I/L to ZoneFlag 6/7";
    }
    if((zone_M) && (flag_6)){
        return "Zone M to ZoneFlag 6";
    }
    if((zone_M || zone_N) && (flag_7)){
        return "Zone M/N to ZoneFlag 7";
    }


    if((zone_G || zone_H) && ( flag_8)){
        return "Zone G/H to ZoneFlag 8";
    }
    if((zone_I) && (flag_8)){
        return "Zone I to ZoneFlag 8";
    }
    if((zone_L) && (flag_8)){
        return "Zone L to ZoneFlag 8";
    }
    if((zone_M || zone_N) && ( flag_8)){
```

```
            return "Zone M/N to ZoneFlag 8";
        }


        if((zone_G || zone_H) && ( flag_9)){
            return "Zone G/H to ZoneFlag 9";
        }
        if((zone_H) && (flag_10)){
            return "Zone H to ZoneFlag 10";
        }
        if((zone_I || zone_L) && (flag_9 || flag_10)){
            return "Zone I/L to ZoneFlag 9/10";
        }
        if((zone_M || zone_N) && (flag_9)){
            return "Zone M/N to ZoneFlag 9";
        }
        if((zone_M || zone_N) && ( flag_10)){
            return "Zone M/N to ZoneFlag 10";
        }

    }

    if((zone_O) && ( flag_1 || flag_2 || flag_3 || flag_4 || flag_7 || flag_8 || flag_9 || flag_10)){
        return "Zone O to ZoneFlag";
    }
    if((zone_P) && ( flag_2 || flag_3 || flag_4 || flag_5 || flag_6 || flag_7 || flag_8 || flag_9)){
        return "Zone P to ZoneFlag";
    }


    return "Vado alla bandierina Location";  // quando il robot non si trova in nessuna delle altre zone
  }


}
```

Classe NodeMemory_Try_it.java

```
/*
 * NodeVec2Array.java
 */


import EDU.gatech.cc.is.clay.*;
import EDU.gatech.cc.is.util.Vec2;
import EDU.gatech.cc.is.abstractrobot.*;

import EDU.gatech.cc.is.communication.*;
import CSAI.unipa.communication.*;
import CSAI.unipa.abstractrobot.*;

/**
 * A Node that returns an array of Vec2 values.
 * <P>
 * For detailed information on how to configure behaviors, see the
 * <A HREF="../clay/docs/index.html">Clay page</A>.
 * <P>
 * <A HREF="../COPYRIGHT.html">Copyright</A>
 * (c)1997, 1998 Tucker Balch
 *
 * @author Tucker Balch
 * @version $Revision: 1.1 $
 */


public class NodeMemory_Try_it extends NodeVec2Array {

    private VisualObjectSensor visual_robot;
    private SimpleInterface abstract_robot;
    private int attractorType1, attractorType2, attractorType3;
    public static final boolean DEBUG = Node.DEBUG;
    private boolean control;
    private MultiForageN150ExploreSim multi_robot;
    private int length=0;


    /** Creates a new instance of va_Memory_va */
    public NodeMemory_Try_it(boolean memory_control, int t1, int t2, int t3, SimpleInterface ar) {
        visual_robot = (VisualObjectSensor)ar;
        abstract_robot = ar;
        attractorType1 = t1;
        attractorType2 = t2;
        attractorType3 = t3;
        control = memory_control;
        multi_robot = (MultiForageN150ExploreSim)ar;
    }


    long                                                    lasttime = 0;
    Vec2[]  last_val = new Vec2[0];
    int[]   visual_number = new int[0];

    /**
     * Return a Vec2Array that is the merge of two others.
     * @param timestamp long, only get new information
     *                                                      if timestamp > than last call or
timestamp == -1.
```

```
 * @return the merged list.
 */
public Vec2[] Value(long timestamp) {
   if (DEBUG) System.out.println("va_Obstacles_r: Value()");
   try {
      //The vector with all the obstacle seen by robot
      Vec2[] im0_1 = visual_robot.getVisualObjects(timestamp, attractorType1);
      Vec2[] im0_2 = visual_robot.getVisualObjects(timestamp, attractorType2);
      Vec2[] im0_3 = visual_robot.getVisualObjects(timestamp, attractorType3);

      //the array with the visual number of the new flags
      int[] visual_new = new int[im0_1.length+im0_2.length+im0_3.length];

      //the array of the new flags merging the three
      Vec2[] im0 = new Vec2[im0_1.length+im0_2.length+im0_3.length];
      int count = 0;
      for(int i=0; i<im0_1.length; i++) {
         im0[count] = new Vec2(im0_1[i].x, im0_1[i].y);
         visual_new[count++] = attractorType1;
      }
      for(int i=0; i<im0_2.length; i++) {
         im0[count] = new Vec2(im0_2[i].x, im0_2[i].y);
         visual_new[count++] = attractorType2;
      }
      for(int i=0; i<im0_3.length; i++) {
         im0[count] = new Vec2(im0_3[i].x, im0_3[i].y);
         visual_new[count++] = attractorType3;
      }


      //the position of the robot
      Vec2 position = abstract_robot.getPosition(timestamp);

      //      if(((timestamp > lasttime)||(timestamp == -1)) && im0.length!=0)
      //      {
      /*--- reset the timestamp ---*/
      //        if (timestamp > 0) lasttime = timestamp;
      if(im0.length !=0) {
         //make it global
         for(int i=0; i<im0.length; i++)
            im0[i].add(position);

         //im1 is global and is rounded
         Vec2[] im1 = roundVector(im0);
         //im2 is global
         Vec2[] im2 = last_val;

         int[] visual_old = visual_number;

         //is a temp vector array in which save only new obstacles and after merge with old
         int k = im1.length;

         //Initialization of im3, is 1 if is a new obstacle, -1 if is old
         int[] im3 = new int[im1.length];
         for(int i=0; i<im3.length; i++)
            im3[i] = 1;


         //control the objects and decide what are to be saved
         for(int i=0; i<im1.length; i++) {
            for(int j=0; j<im2.length; j++) {
               //if condition certainly is a flag that has been moved by robot
```

```
        if((Math.abs(im1[i].x-im2[j].x)<=0.3) &&
        (Math.abs(im1[i].y-im2[j].y)<=0.3)) {
            //-1 because i must not insert it as a new flag
            im3[i] = -1;
            //substitute the old flag with this that's the same moved
            im2[j] = new Vec2(im1[i].x, im1[i].y);
            k--;
            break;
        }
        //NON SERVE PIù PERCHè HO AGGIUNTO LA CONDIZIONE VEDI SE C'è LA BANDIERINA
        /*else
        if((Math.abs(im1[i].x-im2[j].x)<noise_factor+0.2) &&
          (Math.abs(im1[i].y-im2[j].y)<noise_factor+0.2))
        {
            //set to -1 the obstacles that are old
            im3[i] = new Vec2(-1,0);
            //im2[j] = new Vec2(im1[i].x, im1[i].y);
            k--;
            break;
        }*/
    }
}

//create the vector for the flag and the visual tpe
last_val= new Vec2[im2.length+k];
visual_number = new int[last_val.length];

//merge the vector arrays of new and old flags
int pos = 0;

//this flag says if has been modified the vector array
boolean flag = false;

//before insert all new obstacles
for(int i=0; i<im3.length; i++) {
    if(im3[i]==1) {
        last_val[pos] = new Vec2(im1[i].x, im1[i].y);
        visual_number[pos] = visual_new[i];
        flag = true;

        try {
            //Message----------new flag
            LongMessage idMessage = new LongMessage();
            idMessage.val = multi_robot.getPlayerNumber(timestamp);
            int id = 0;
            if(idMessage.val == 0)
                id = 1;
            if(idMessage.val == 1)
                id = 0;
            if(idMessage.val == 2)
                id = 3;
            if(idMessage.val == 3)
                id = 2;
            multi_robot.unicast(id, idMessage);

            StringMessage message = new StringMessage();
            message.val = "Bandiera nuova";
            multi_robot.unicast(id, message);

            PointMessage flagPosition = new PointMessage(last_val[pos].x, last_val[pos].y);
            multi_robot.unicast(id, flagPosition);
```

```
                    LongMessage visualMessage = new LongMessage();
                    visualMessage.val = visual_number[pos];
                    multi_robot.unicast(id, visualMessage);
                }
                catch(CommunicationException e){}
                pos++;
            }
        }

        //and after insert the old obstacles
        for(int i=k; i<last_val.length; i++){
            last_val[i] = new Vec2(im2[i-k].x, im2[i-k].y);
            visual_number[i] = visual_old[i-k];
        }

        if (flag && control) {
            System.out.println("******************Memoria ROBOT
"+multi_robot.getPlayerNumber(timestamp)+" aggiornata******************");
            for(int i=0; i<last_val.length; i++) {
                System.out.println(visual_number[i] +" "+last_val[i].toString());
            }
            System.out.println();
        }
      }
      length=last_val.length;
      Vec2[] ret_val = new Vec2[last_val.length];

      for(int i = 0; i<ret_val.length; i++)
          ret_val[i] = new Vec2(last_val[i].x, last_val[i].y);
      return(ret_val);

    }
    catch(NullPointerException e){System.out.println("Exception in NodeMemory_Try_it");}
    length=0;
    return(new Vec2[0]);
  }

  ;

  ;

  ;

  ;

  /**
   * Set the value of the node.
   * If you implement a NodeVec2Array_cb, you need to define
   * this method.
   *
   * @param value NodeVec2Array indicates the value to set
   * @param timestamp long indicates time of the request
   */
  public int getAttrType1(){return(attractorType1);};

  /**
   * Set the value of the node.
   * If you implement a NodeVec2Array_cb, you need to define
   * this method.
   *
   * @param value NodeVec2Array indicates the value to set
   * @param timestamp long indicates time of the request
```

```
     */
    public int getAttrType2(){return(attractorType2);};

    /**
     * Set the value of the node.
     * If you implement a NodeVec2Array_cb, you need to define
     * this method.
     *
     * @param value NodeVec2Array indicates the value to set
     * @param timestamp long indicates time of the request
     */
    public int getAttrType3(){return(attractorType3);};


    ;

    /**
     * Set the value of the node.
     * If you implement a NodeVec2Array_cb, you need to define
     * this method.
     *
     * @param value NodeVec2Array indicates the value to set
     * @param timestamp long indicates time of the request
     */
    public void setValue(long timestamp) {
       Value(timestamp);
    }



    public double getX(int item, long timestamp) {

       return(last_val[item].x);
    }

    public double getY(int item, long timestamp) {

       return(last_val[item].y);
    }

    public double getr(int item, long timestamp) {

       return(last_val[item].r);
    }

  /*      public double getLength(long timestamp) {

           return(last_val.length);
        }  */

    public int getLength(long timestamp) {

       return(length);
    }

    /**
     * Pop up a NodeVec2 from the embedded
     * If you implement a NodeVec2Array_cb, you need to define
     * this method
     *
     * @param im1 NodeVec2 indicates the Vector to pop
     * @param timestamp long indicates time of request
```

```
 */
public void pop(Vec2 im1, long timestamp) {

   boolean flag=false;
   try {
      //Message----------new flag
      LongMessage idMessage = new LongMessage();
      idMessage.val = multi_robot.getPlayerNumber(timestamp);
      int id = 0;
      if(idMessage.val == 0)
         id = 1;
      if(idMessage.val == 1)
         id = 0;
      if(idMessage.val == 2)
         id = 3;
      if(idMessage.val == 3)
         id = 2;
      multi_robot.unicast(id, idMessage);

      StringMessage message = new StringMessage();
      message.val = "Delete Flag";
      multi_robot.unicast(id, message);

      PointMessage flagPosition = new PointMessage(im1.x, im1.y);
      multi_robot.unicast(id, flagPosition);
   }
   catch(CommunicationException e){}

   try {
      int count=0;
      for(int i=0; i<last_val.length; i++)
         if((Math.abs(im1.x - last_val[i].x)<=0.4)    //modificato!!!!!era 0.3
         && (Math.abs(im1.y - last_val[i].y)<=0.4)) {
            count++;
            flag = true;

         }
      if (flag) {
         int pos = 0;
         Vec2[] temp = new Vec2[last_val.length-count];
         for(int i=0; i<last_val.length; i++)
            if((Math.abs(im1.x - last_val[i].x)>0.4) ||
            (Math.abs(im1.y - last_val[i].y)>0.4)) {
               temp[pos++] = new Vec2(last_val[i].x, last_val[i].y);
            }
         //else
         //    System.out.println("Tolta bandierina "+last_val[i].toString());
         last_val = temp;


         if(control) {
            System.out.println("******************Memoria ROBOT
"+multi_robot.getPlayerNumber(timestamp)+" decrementata******************");
            for(int i=0; i<last_val.length; i++)
               System.out.println(last_val[i].toString());
            System.out.println("Estratta bandierina "+im1.toString()+"\n");
         }
      }
   }
   catch(NullPointerException e){System.out.println("Exception in NodeMemory.Pop");}

}
```

```
    public void pop_only(Vec2 im1, long timestamp) {

        boolean flag=false;

        try {
            int count=0;
            for(int i=0; i<last_val.length; i++)
                if((Math.abs(im1.x - last_val[i].x)<=0.4)    //modificato!!!!!era 0.3 modificato
                && (Math.abs(im1.y - last_val[i].y)<=0.4))   //anche  in pop().
                {
                    count++;
                    flag = true;
                }
            if (flag) {
                int pos = 0;
                Vec2[] temp = new Vec2[last_val.length-count];
                for(int i=0; i<last_val.length; i++)
                    if((Math.abs(im1.x - last_val[i].x)>0.4) ||
                    (Math.abs(im1.y - last_val[i].y)>0.4)) {
                        temp[pos++] = new Vec2(last_val[i].x, last_val[i].y);
                    }
                //else
                //    System.out.println("Tolta bandierina "+last_val[i].toString());
                last_val = temp;


                if(control) {
                    System.out.println("******************Memoria ROBOT
"+multi_robot.getPlayerNumber(timestamp)+" decrementata******************");
                    for(int i=0; i<last_val.length; i++)
                        System.out.println(last_val[i].toString());
                    System.out.println("Estratta bandierina "+im1.toString()+"\n");
                }
            }
        }
        catch(NullPointerException e){System.out.println("Exception in NodeMemory.pop_only");}

    }

    public double roundDecimal(double val) {

        double temp = val * 10;
        long round = Math.round(temp);
        temp = (double)round / 10;
        //System.out.println(val+" "+round+" "+temp);
        return (temp);

    }

    public Vec2[] roundVector(Vec2[] im1) {

        Vec2[] ret_val = new Vec2[im1.length];

        for(int i=0; i<ret_val.length; i++)
            ret_val[i] = new Vec2(roundDecimal(im1[i].x), roundDecimal(im1[i].y));

        return(ret_val);
    }
```

```
/**
 * Pop up a NodeVec2 from the embedded
 * If you implement a NodeVec2Array_cb, you need to define
 * this method
 *
 * @param im1 NodeVec2 indicates the Vector to pop
 * @param timestamp long indicates time of request
 */
public int[] getVisual() {

   int[] ret_val = new int[visual_number.length];
   for(int i=0; i<ret_val.length; i++)
      ret_val[i] = visual_number[i];
   return(ret_val);

}

public boolean inMap(long timestamp, NodeVec2 im0){

   return(false);
}

/**
 * Pop up a NodeVec2 from the embedded
 * If you implement a NodeVec2Array_cb, you need to define
 * this method
 *
 *@param im1 NodeVec2 indicates the Vector to pop
 *@param timestamp long indicates time of request
 */
public void push(Vec2 im1, int visual, long timestamp) {
   boolean canPush=true;

   try {
      for(int i=0; i<last_val.length; i++)
         if((Math.abs(im1.x - last_val[i].x)<=0.3)
         && (Math.abs(im1.y - last_val[i].y)<=0.3)) {
            canPush = false;
         }
      if (canPush) {
         //create the vector for the flag and the visual tpe
         Vec2[] temp= new Vec2[last_val.length+1];
         int[] temp_visual = new int[last_val.length+1];

         int pos=0;
         for(int i=0; i<last_val.length; i++) {
            temp[i] = new Vec2(last_val[i].x, last_val[i].y);
            temp_visual[i] = visual_number[i];
            pos++;
         }
         temp[pos] = new Vec2(im1);
         temp_visual[pos] = visual;

         last_val = temp;//new Vec2[temp.length];
         visual_number = temp_visual;//new int[temp_visual.length];

         for(int i = 0; i<last_val.length; i++) {
            last_val[i] = new Vec2(temp[i]);
            visual_number[i] = temp_visual[i];
            //System.out.println("temp "+last_val[i].toString());
         }
```

```
            }
            if(control) {
                System.out.println("********************Memoria ROBOT "+multi_robot.getPlayerNumber(timestamp)+"
incrementata********************");
                for(int i=0; i<last_val.length; i++)
                    System.out.println(last_val[i].toString());
            }

        }
        catch(NullPointerException e){}
    }
}
```

Classe Stalling.java

```
import java.io.*;

/**
 * Controlla se il robot si trova in situazione di stallo basandosi sul tempo di sistema
 *
 */

public class Stalling {


    private long time;

    public Stalling(){                                                    }

    public void update(){
        time=System.currentTimeMillis();
    }


    public boolean isStall(double cut){
        return ((System.currentTimeMillis()-time)>cut);
    }

    public long getTime(){
        return time;
    }

}
```

Classe v_ChooseZone_mv_Try_it.java

```
import EDU.gatech.cc.is.clay.*;
import EDU.gatech.cc.is.util.Vec2;

import CSAI.unipa.knowledgment.NodeMap;
import CSAI.unipa.util.mapAttributes;


// Abbiamo aggiunto un metodo per settare il vettore priority
public class v_ChooseZone_mv_Try_it extends NodeVec2 {

    public static final boolean DEBUG = Node.DEBUG;
    private int[]               priority = new int[4];
    private NodeMap             map;
    private NodeVec2            embedded1;

    /** Creates a new instance of d_ChooseZone_m */
    public v_ChooseZone_mv_Try_it(NodeMap m, NodeVec2 pos, int f, int s, int t, int q) {
        priority[0] = f;
        priority[1] = s;
        priority[2] = t;
        priority[3] = q;
        map = m;
        embedded1 = pos;
    }

    Vec2                                                    last_val = new Vec2();
    long                                                    lasttime = 0;

    /**
     * Provides the value of the node.
     * If you implement a NodeDouble, you need to define
     * this method.
     *
     * @param timestamp long indicates time of the request
     * @return the value
     */
    public Vec2 Value(long timestamp) {
        if (DEBUG) System.out.println("d_ChooseZone_mv: Value()");

        if ((timestamp > lasttime)||(timestamp == -1)) {
            /*--- reset the timestamp ---*/
            if (timestamp > 0) lasttime = timestamp;

            Vec2 position = embedded1.Value(timestamp);

            int column = map.getCellColumn(position.x);
            int row = map.getCellRow(position.y);


            for(int k=0; k<4;k++){
                if((priority[k]==mapAttributes.NORD) &&
                (map.isFree(column, row-1))){
                    last_val = new Vec2(position.x, 10000);
                    priority[0] = mapAttributes.NORD;
                    priority[1] = mapAttributes.EST;
                    priority[2] = mapAttributes.SUD;
                    priority[3] = mapAttributes.OVEST;
                    row = row -1;
                    break;
```

```
            } else
               if((priority[k]==mapAttributes.SUD) &&
               (map.isFree(column, row+1))){
                  last_val = new Vec2(position.x, -10000);
                  row = row +1;
                  priority[0] = mapAttributes.SUD;
                  priority[1] = mapAttributes.OVEST;
                  priority[2] = mapAttributes.NORD;
                  priority[3] = mapAttributes.EST;
                  break;
               } else
                  if((priority[k]==mapAttributes.OVEST) &&
                  (map.isFree(column-1, row))){
                     last_val = new Vec2(-10000, position.y);
                     column=column-1;
                     priority[0] = mapAttributes.OVEST;
                     priority[1] = mapAttributes.NORD;
                     priority[2] = mapAttributes.EST;
                     priority[3] = mapAttributes.SUD;
                     break;
                  } else
                     if((priority[k]==mapAttributes.EST) &&
                     (map.isFree(column+1, row))){
                        column= column+1;
                        last_val = new Vec2(10000, position.y);
                        priority[0] = mapAttributes.EST;
                        priority[1] = mapAttributes.SUD;
                        priority[2] = mapAttributes.OVEST;
                        priority[3] = mapAttributes.NORD;
                        break;
                     }
            }
            //last_val = new Vec2(map.getXCenter(column),map.getYCenter(row));
         }
         return(last_val);
      }

      public void setPriority(int f,int s,int t,int q){
         priority[0] = f;
         priority[1] = s;
         priority[2] = t;
         priority[3] = q;
      }
   }
```

Classe v_Closest_var_Try_it.java

```
import java.lang.*;
import EDU.gatech.cc.is.util.Vec2;
import EDU.gatech.cc.is.util.Units;
import EDU.gatech.cc.is.clay.*;
import EDU.gatech.cc.is.abstractrobot.SimpleInterface;

/**
 * Finds the closest in a list of Vec2s.  Assumes the
 * vectors point egocentrically to the objects, so that
 * the closest one has the shortest r value.
 * <P>
 * For detailed information on how to configure behaviors, see the
 * <A HREF="../clay/docs/index.html">Clay page</A>.
 * <P>
 * <A HREF="../COPYRIGHT.html">Copyright</A>
 * (c)1997, 1998 Tucker Balch
 *
 * @author Tucker Balch
 * @version $Revision: 1.1 $
 */


public class v_Closest_var_Try_it extends NodeVec2 {
    /**
     * Turns debugging on or off.
     */
    public static final boolean DEBUG = Node.DEBUG;
    private NodeMemory_Try_it                                                    embedded1;
    private SimpleInterface abstract_robot;
    private Vec2            closestFlag = new Vec2(0,0);
    private boolean         filter = false;
    private Vec2[] Bin = new Vec2[3];
    private int             filterTime = 0;
    private int             time = 0;


    /**
     * Instantiate a v_Closest_va node.
     * @param im1 NodeVec2, the embedded node that generates a list
     * of items to scan.
     */

    public v_Closest_var_Try_it(NodeMemory_Try_it im1, SimpleInterface ar, NodeVec2 B1, NodeVec2 B2,
NodeVec2 B3) {
        if (DEBUG) System.out.println("v_Closest_va: instantiated.");
        embedded1 = im1;
        abstract_robot = ar;
        Bin[0] = B1.Value(ar.getTime());
        Bin[1] = B2.Value(ar.getTime());
        Bin[2] = B3.Value(ar.getTime());
    }


    Vec2  last_val = new Vec2();
    long lasttime = 0;

    /**
     * Return a Vec2 representing the closest object, or 0,0 if
     * none are visible.
```

```
 * @param timestamp long, only get new information if timestamp > than last call
 * or timestamp == -1.
 * @return the vector.
 */
public Vec2 Value(long timestamp) {

        lasttime = timestamp;

    Vec2 position = abstract_robot.getPosition(timestamp);

    /*--- reset output ---*/
    last_val.setr(99999999);

    /*--- get the list of obstacles ---*/
    Vec2[] objs = embedded1.Value(timestamp);

    int[] visual = embedded1.getVisual();

    /*--- consider each obstacle ---*/
    double closest = 99999999;
    //int k = 0;
    for(int i = 0; i<objs.length; i++) {
        Vec2 flagToRobot = new Vec2(objs[i].x, objs[i].y);
        flagToRobot.sub(position);
        Vec2 flagToBin = new Vec2(objs[i].x, objs[i].y);
        //System.out.println("inizio"+visual[1]);

        if(visual[i]==embedded1.getAttrType1())
            flagToBin.sub(Bin[0]);
        else if(visual[i]==embedded1.getAttrType2())
            flagToBin.sub(Bin[1]);
        else if(visual[i]==embedded1.getAttrType3())
            flagToBin.sub(Bin[2]);

        //Vec2 distance = new Vec2(flagToRobot.x, flagToRobot.y);
        Vec2 distance = new Vec2();
        distance.setr(flagToRobot.r+flagToBin.r);
        if (distance.r <= closest) {
            closest = distance.r;
            last_val = objs[i];
            //k = i;
        }
        // }
        //else System.out.println("non considero la bandierina "+closestFlag.toString());
    }
   /*               if(closest!=99999999)
                //System.out.println("r :"+closest+"classe: "+visual[k]+" "+last_val.toString());
                if (DEBUG) System.out.println("v_Closest_va.Value: "+
                        objs.length+" things to see "+
                        "output "+
                        last_val);*/


    return (new Vec2(last_val.x, last_val.y));
}

/**
 * Set the value of the node.
 * If you implement a NodeVec2_cb, you need to define
 * this method.
 *
 * @param value NodeVec2 indicates the value to set
```

```
     * @param timestamp long indicates time of the request
     */
    public void setValue(NodeMemory_Try_it im1, long timestamp) {
        embedded1 = im1;
        Value(timestamp);
    }

    /**
     * Return the length of the NodeVec2Array_cb
     * If you implement a NodeVec2Array_cb, you need to define
     * this method
     *
     * @param timestamp long indicates time of request
     */
    public double getLength(long timestamp) {
        if (last_val.r == 0)
            return(0);
        else
            return(1);
    }

    /**
     * Return the x value of an item
     * If you implement a NodeVec2Array_cb, you need to define
     * this method
     *
     * @param item int indicates the item of the array
     * @param timestamp long indicates time of request
     */
    public double getX(long timestamp) {
        return(last_val.x);
    }

    /**
     * Return the y value of an item
     * If you implement a NodeVec2Array_cb, you need to define
     * this method
     *
     * @param item int indicates the item of the array
     * @param timestamp long indicates time of request
     */
    public double getY(long timestamp) {
        return(last_val.y);
    }

    /**
     * Return the r value of an item
     * If you implement a NodeVec2Array_cb, you need to define
     * this method
     *
     * @param item int indicates the item of the array
     * @param timestamp long indicates time of request
     */
    public double getr(long timestamp) {
        return(last_val.r);
    }

    public void filterFlag(Vec2 im0, int time) {
        filter = true;
        closestFlag = im0;
        filterTime = time;
        time = 0;
```

```
        //System.out.println("Setto "+closestFlag.toString());
    }

    public void setFilter(boolean value){
        filter = value;
        if(!value) {
            filterTime = 0;
            time = 0;
        }
    }

}
```

Classe v_LinearAttraction_v_Try_it.java

```
import EDU.gatech.cc.is.clay.*;
import java.lang.*;
import EDU.gatech.cc.is.util.Vec2;
import EDU.gatech.cc.is.util.Units;

/**
 * Generates a vector towards a goal location
 * that varies with distance from the goal.  The attraction is
 * increased linearly at greater distances.  Based on
 * Arkin's formulation.
 * <P>
 * Arkin's original formulation is described in
 * "Motor Schema Based Mobile Robot
 * Navigation," <I>International Journal of Robotics Research</I>,
 * vol. 8, no 4, pp 92-112.
 * <P>
 * The source code in this module is based on "first principles"
 * (e.g. published papers) and is not derived from any previously
 * existing software.
 * <P>
 * For detailed information on how to configure behaviors, see the
 * <A HREF="../clay/docs/index.html">Clay page</A>.
 * <P>
 * <A HREF="../COPYRIGHT.html">Copyright</A>
 * (c)1997, 1998 Tucker Balch
 *
 * @author Tucker Balch
 * @version $Revision: 1.1 $
 */

public class v_LinearAttraction_v_Try_it extends NodeVec2 {
    /**
     * Turns debug printing on or off.
     */
    public static final boolean DEBUG = /*true;*/ Node.DEBUG;
    private NodeVec2 embedded1;
    private double controlled_zone = 1.0;
    private double dead_zone = 0.0;
    private double rot_teta=0.0;
    /**
     * Instantiate a v_LinearAttraction_v schema.
     * @param czr double, controlled zone radius.
     * @param dzr double, dead zone radius.
     * @param im1 double, the node that generates an
     * egocentric vector to the goal.
     * @param im2 NodeVec2, the embedded node that generates a vector
     * to be detected.
     */
    public v_LinearAttraction_v_Try_it(double czr, double dzr,
    NodeVec2 im1) {
        if (DEBUG) System.out.println("v_LinearAttraction_v: instantiated.");
        embedded1 = im1;
        if ((czr < dzr) || (czr<0) || (dzr<0)) {
            System.out.println("v_LinearAttraction_v: illegal parameters");
            return;
        }
        controlled_zone = czr;
        dead_zone = dzr;
```

```
    }


    Vec2 last_val = new Vec2();
    long lasttime = 0;
    /**
     * Return a Vec2 representing the direction to go towards the
     * goal.  Magnitude varies with distance.
     * @param timestamp long, only get new information if timestamp > than last call
     * or timestamp == -1.
     * @return the movement vector.
     */
    public Vec2 Value(long timestamp) {
        double mag;
        Vec2 goal = new Vec2();

        if ((timestamp > lasttime)||(timestamp == -1)) {
            if (DEBUG) System.out.println("v_LinearAttraction_v:");

            /*--- reset the timestamp ---*/
            if (timestamp > 0) lasttime = timestamp;

            /*--- get the goal ---*/
            goal = embedded1.Value(timestamp);

            /*--- consider the magnitude ---*/
            // inside dead zone?
            if (goal.r < dead_zone) {
                mag = 0;
            }
            // inside control zone?
            else if (goal.r < controlled_zone)
                mag = (goal.r - dead_zone)/
                (controlled_zone - dead_zone);
            // outside control zone
            else mag = 1.0;
            if (DEBUG) System.out.println(mag+" "+goal.r);

            /*--- set the vector ---*/
            goal.setr(mag);
            goal.rotate(rot_teta);
            last_val = goal;
        }
        if (DEBUG) System.out.println(last_val.r+" "+goal.r);
        return (new Vec2(last_val.x, last_val.y));
    }
    public void setRot_teta(double t){
        rot_teta=t;
    }
}
```

Classe RoomLocation.java

```
import EDU.gatech.cc.is.clay.*;
import EDU.gatech.cc.is.util.Vec2;


/**
 *  Permette la localizzazione del robot e delle bandiere all'interno delle stanze
 *
 */

public class RoomLocation {

   private double x_rob;
   private double y_rob;

   private NodeVec2 position;

   public RoomLocation(NodeVec2 pos){
      position=pos;
   }



   public void Value(long timestamp){
      x_rob=position.Value(timestamp).x;
      y_rob=position.Value(timestamp).y;

   }

   public boolean in_room(){

      return in_red_Right() || in_green_Right() || in_blue_Right() || in_red_Left() || in_green_Left() || in_blue_Left();

   }
   public boolean closed_To_Room_Right(){

      return closed_red_Right() || closed_green_Right() || closed_blue_Right();

   }

   public boolean in_red_Left(){
      return x_rob>7 && x_rob<11 && y_rob>18 && y_rob<28 && (y_rob>((-4*x_rob+98)/3));

   }

   public boolean in_green_Left(){
      return x_rob>5 && x_rob<11 &&(y_rob<((-4*x_rob+98)/3))&&((y_rob<(6+2*x_rob)))&&(y_rob>((4*x_rob-
14)/3))&&(y_rob>(-2*x_rob+22));

   }

   public boolean in_blue_Left(){
      return x_rob>7 && x_rob<11 && y_rob>0 && y_rob<10&& y_rob<((4*x_rob-14)/3);

   }


   public boolean in_red_Right(){
      return x_rob>33 && x_rob<37 && y_rob>18 && y_rob<28 && (y_rob>((4*x_rob-78)/3));
```

```
    }

    public boolean in_green_Right(){
        return x_rob>33 && x_rob<39 && (y_rob<((4/3)*x_rob-16))&&((y_rob<(94-2*x_rob)))&&(y_rob>((-
4*x_rob+162)/3))&&(y_rob>(2*x_rob-66));

    }

    public boolean in_blue_Right(){
        return x_rob>33 && x_rob<37 && y_rob>0 && y_rob<10&& y_rob<((-4*x_rob+162)/3);

    }



    public boolean in_lab_middle(){
        return x_rob>19&&x_rob<25&&y_rob>12&&y_rob<18;

    }

    public boolean in_lab_top(){
        return x_rob>17&&x_rob<25&&y_rob>18&&y_rob<22;

    }

    public boolean in_lab_down(){
        return x_rob>19&&x_rob<27&&y_rob>8&&y_rob<12;

    }

    public boolean in_lab(){
        return in_lab_middle()||in_lab_top()||in_lab_down();
    }

    public boolean use_lab_right(){
        return (x_rob>25&& x_rob<33 && y_rob>8 &&y_rob<22.5);
    }

    public boolean use_lab_right_redBlue(){
        return (x_rob>25&& x_rob<27.5 && y_rob>8 &&y_rob<22);
    }

    public boolean use_lab_left(){
        return (x_rob>11&& x_rob<19 && y_rob>7.5 &&y_rob<22);
    }

    public boolean use_lab_left_redBlue(){
        return (x_rob>16.5&& x_rob<19 && y_rob>8 &&y_rob<22);
    }

    public boolean closed_red_Left(){
        return x_rob>11 && x_rob<14 && y_rob>18.5 && y_rob<28;

    }

    public boolean closed_green_Left(){
        return x_rob>11 && x_rob<14 && y_rob>10 && y_rob<18;

    }

    public boolean closed_blue_Left(){
```

```
        return x_rob>11 && x_rob<14 && y_rob>0 && y_rob<9.5;

    }


    public boolean closed_red_Right(){
        return x_rob>30 && x_rob<33 && y_rob>18.5 && y_rob<28;
    }

    public boolean closed_green_Right(){
        return x_rob>30 && x_rob<33 && y_rob>10 && y_rob<18;

    }

    public boolean closed_blue_Right(){
        return x_rob>30 && x_rob<33 && y_rob>0 && y_rob<9.5;
    }

    public boolean closed_red(){
        return closed_red_Left() || closed_red_Right();
    }

    public boolean closed_green(){
        return closed_green_Left() || closed_green_Right();
    }

    public boolean closed_blue(){
        return closed_blue_Left() || closed_blue_Right();
    }

    public boolean closed_lab_(){
        return closed_lab_Left() || closed_lab_Right();

    }

    public boolean closed_lab_Left(){
        return x_rob>15&&x_rob<19&&y_rob>18&&y_rob<22;

    }

    public boolean closed_lab_Right(){
        return x_rob>25&&x_rob<29&&y_rob>8&&y_rob<12;

    }

    public boolean out_of_bounds(){
        return
(x_rob>36&&(y_rob>24||y_rob<4))||(x_rob>38&&y_rob<16.5&&y_rob>11.5)||(x_rob<8&&(y_rob<4||y_rob>24))||(x_
rob<6&&y_rob<16.5&&y_rob>11.5);

    }


    public boolean flag_in_red(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_red=(x_flag>33 && x_flag<36 && y_flag>20 && y_flag<28&& (y_flag>((4*x_flag-
78)/3)))&&(x_flag>7.5 && x_flag<11 && y_flag>18 && y_flag<28 && (y_flag>((-4*x_flag+98)/3)));
        return  flag_red;

    }
```

```
    public boolean flag_in_green(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_green=(x_flag>33 && x_flag<36 && y_flag>10 && y_flag<20&& (y_flag<((4*x_flag-
78)/3))&&((y_flag<(94-2*x_flag)))&&(y_flag>((-4*x_flag+162)/3))&&(y_flag>(2*x_flag-66)))&&(x_flag>5.5 &&
x_flag<11 && y_flag>5 && y_flag<22&& (y_flag<((-
4*x_flag+78)/3))&&((y_flag<(6+2*x_flag)))&&(y_flag>((4*x_flag-14)/3))&&(y_flag>(-2*x_flag+22)));
        return  flag_green;

    }


    public boolean flag_in_blue(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_blue=(x_flag>33 && x_flag<36 && y_flag>0 && y_flag<10&& y_flag<((-
4*x_flag+162)/3))&&(x_flag>7.5 && x_flag<11 && y_flag>0 && y_flag<10&& y_flag<((4*x_flag-14)/3));
        return  flag_blue;

    }

    public boolean flag_in_lab_centre(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_lab=x_flag>19&&x_flag<25&&y_flag>12&&y_flag<18;
        return  flag_lab;

    }

    public boolean flag_in_lab_top(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_lab=x_flag>19&&x_flag<25&&y_flag>18&&y_flag<22;
        return  flag_lab;

    }

    public boolean flag_in_lab_down(Vec2 flag){
        double x_flag=flag.x;
        double y_flag=flag.y;
        boolean flag_lab=x_flag>19&&x_flag<25&&y_flag>8&&y_flag<12;
        return  flag_lab;

    }

    public boolean flag_in_lab(Vec2 flag){
        return flag_in_lab_down(flag) || flag_in_lab_top(flag) || flag_in_lab_centre(flag);
    }

}
```