NG Café

Introduzione

Nell'anno 2013/2014 sono state effettuate massicce modifiche all'interfaccia grafica, da parte del gruppo Dell'Anna, Madeddu, Mensa. Questa guida si pone l'obiettivo di guidare i nuovi gruppi degli anni a venire nell'installazione e configurazione della GUI per il proprio progetto.

Durante lo sviluppo dell'interfaccia si è cercato di fornire uno strumento che fosse il più generale possibile (ergo che richiedesse il minor numero di modifiche da parte degli utenti futuri), ma parte del codice è strettamente legato al progetto e quindi sarà comunque necessario modificare alcuni metodi se si vuole configurare la GUI per progetti differenti da quello dell'anno 2013/2014.

È obiettivo di questo documento l'esegesi di quelle che sono le parti di codice project-related e soprattutto la definizione chiara di quali siano i requisiti per far sì che tutto il sistema funzioni adeguatamente.

La lettura di questo documento (seppur noiosa a tratti) è caldamente suggerita: si garantisce il risparmio di parecchie ore di lettura/studio del codice della GUI (che comunque è documentato al meglio delle nostre possibilità).

Al termine del documento è fornita una check-list che riassume tutte le dipendenze e ciò che è necessario avere (in termini di fatti in CLIPS) per poter adoperare agilmente il software.

Installazione dell'ambiente

Iniziamo con l'installare il progetto Netbeans.

- Testato su Mac OS (Mountain Lion) e su Windows 7, con Netbeans (versioni 8.0 RC1, 7.3 e 7.4) usando Java JDK 8u20 Build b15.
- Per Windows: Copiare il file `CLIPSJNI.dll` (che si trova nella cartella CLIPSJNI_dll) su Windows nella posizione **C:\Windows\System32**.
- Per Mac OS: copiare il file `libCLIPSJNI.jnilib` (che si trova nella cartella CLIPSJNI_dll) nelle cartelle **/System/Library/Java/Extensions**, **/Library/Java/Extensions/** e **/usr/bin**.

Nota: I path forniti sono quelli standard di Java. Qualora il motore inferenziale dovesse non funzionare, è possibile dare il comando `System.getProperty("java.library.path")` al fine di conoscere i propri path.

Nota 2: Se si usa una versione meno recente di quella usata da qualcun altro che ha lavorato o lavora al progetto è possibile avere l'errore di compilazione:

```
C:\<project root>\<project>\nbproject\build-impl.xml:834: copylibs  doesn't support the
"excludeFromCopy" attribute
```

Per risolvere tale issue è sufficiente:

1. Aprire il file `<project>\nbproject\build-impl.xml`.
2. Trovare la linea contenente `<copylibs ... excludeFromCopy= ... ></</copylibs>`
3. Rimuovere l'attributo `excludeFromCopy` da tali linee.

Dando il comando "Clean and build" e poi "Run" potrete verificare il funzionamento dell'ambiente.

Configurazione del proprio progetto

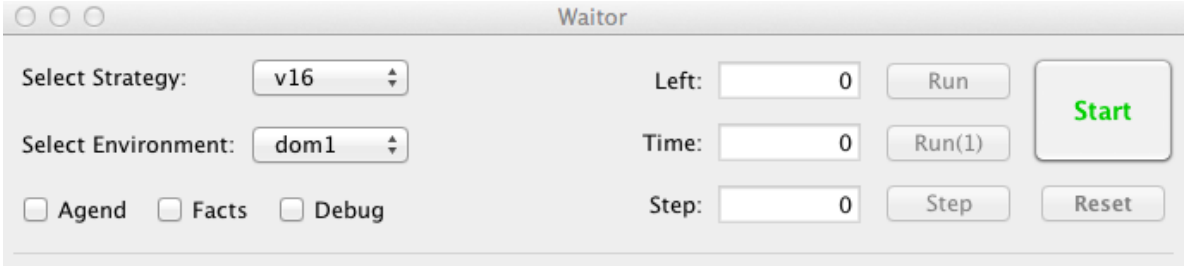
Cosa c'è nella GUI

La GUI è composta da alcuni semplici componenti:

- Un pannello di controllo (che esamineremo nel dettaglio in seguito).
- Una finestra di output: nella finestra di output sarà possibile leggere le stampe dichiarate all'interno dei file

clips (si veda la sezione dedicata più in fondo).

Il pannello di controllo



I selettori

Nel pannello di controllo abbiamo due selettori. Il primo seleziona la cartella dalla quale verranno prelevati tutti i file CLIPS. Si suppone che in ogni cartella vi sia una strategia differente (comunque ci devono essere tutti i file). Il secondo selettore riguarda invece i file che gestiscono l'ambiente: la configurazione della mappa e in caso di progetti multiagente anche la storia di tali agenti. Nel paragrafo a seguire è spiegato come configurare le due cartelle.

I checkbox

Sono a disposizione tre checkbox per attivare l'agenda, i fatti e una finestra di debug dove è possible inserire stampe direttamente via Java (maggiori dettagli sono forniti in seguito).

I pulsanti (regolare l'esecuzione)

Premendo il tasto **Start** il sistema carica i file del progetto (scelti tramite il primo selettore) e poi carica e visualizza la mappa (scelta usando il secondo selettore). Vengono anche eseguite alcune run(1), più precisamente finché non compare il fatto non ordinato `(init-agent (done yes))`, ovvero finché l'intero ambiente non è caricato (dal lato CLIPS). Si veda a fondo di questo documento per ulteriori informazioni in merito a tale fatto. A questo punto è possibile scegliere una modalità di esecuzione. Abbiamo tre scelte:

- **Step** effettua un numero di `run 1` tali da arrivare al completamento dello step attuale (e quindi all'inizio di quello successivo). Uno step è definito come l'intervallo di tempo fra una percezione e l'altra e dunque si basa sul fatto `last-perc` (che è stato introdotto da noi studenti: a fondo documento ci sono maggiori dettagli).
- **Run(1)** effettua l'equivalente di una `run 1` nella console CLIPS.
- **Run** effettua l'equivalente di una `run` nella console di CLIPS, con aggiornamento della mappa ogni step.

È presente anche un tasto di **Reset** che azzera l'interfaccia. Viene quindi effettuata una reset (nel senso di CLIPS) e poi vengono distrutti gli oggetti relativi alla GUI.

Shortcuts

Per una maggiore usabilità sono fornite anche delle shortcut per attivare i tasti:

Button	Shortcut
Start	Alt + A
Run	Alt + R
Run(1)	Alt + T
Step	Alt + S
Reset	Alt + X

Impostare i file CLIPS

La prima cosa da fare è inserire il proprio progetto. Nella cartella **CLP** si possono inserire delle cartelle che contengano le diverse strategie (ogni cartella deve contenere l'intero sistema per l'agente). La strategia è quindi selezionabile da GUI (la lista delle opzioni è la lista delle cartelle in **CLP**). All'interno di ogni cartella (strategia) è possibile inserire un numero qualsiasi di file .cpl, che verranno però letti in ordine lessicografico. Si suggerisce quindi di inserire un suffisso numerico onde evitare problemi di import/export.

La possibilità di mantenere diverse dei vostri file vi permetterà uno sviluppo incrementale delle feature (mantenendo le vecchie versioni funzionanti) e anche l'implementazione di versioni alternative dell'agente.

Nella cartella **envs** si possono inserire cartelle che contengono file (devono essere .txt o .clp) a piacere che verranno copiati nella cartella della strategia in esecuzione e cancellati al termine dell'esecuzione. Il gruppo di file è quindi selezionabile da GUI (la lista delle opzioni è la lista delle cartelle in **envs**).

Struttura della GUI

Il progetto Netbeans consta di pochi file, ma la correlazione fra questi può sembrare parecchio contorta di primo acchito. I file che l'utente dovrà modificare (poiché project-related) sono:

- **MonitorModel.java**: mantiene un modello (sotto forma di una matrice di stringhe chiamata `map`) del mondo: ad ogni cella della matrice `map` è assegnato un valore che è il corrispondente contenuto della cella all'interno del mondo. Tutto questo è reso possibile dai metodi `init()` e `updateMap()`.
- **MonitorView.java**: come una sorta di MVC, funge da view per la classe **MonitorModel.java**. I dati del modello vengono infatti interpretati e stampati sulla mappa (tramite il metodo `initializeMap()` ed `updateMap()`). L'aggiornamento della mappa (del modello e quindi della view) avviene ogni qualvolta si prema sul tasto **Run**, **Run(1)** o **Step**. Dalla classe **MonitorView** è anche possibile accedere alla finestra di debug (chiamata `DebugFrame`).

I metodi citati dovranno essere scritti basandosi sul retrieval dei fatti relativi al vostro progetto. I commenti sul codice dovrebbero guidare l'utente in questo task.

Si suggerisce anche di modificare il metodo `onDispose()` per modificare il pop-up di terminazione fornito all'utente quando l'agente termina la sua esecuzione.

Configurare le icone della mappa (MonitorView.java)

Per configurare le proprie icone della mappa è sufficiente editare all'interno del costruttore della classe **MonitorView.java** la HashMap `map_img`. Si deve associare ad ogni elemento della mappa (ad esempio `Empty`) una certa immagine contenuta nella cartella **img** del progetto. Le immagini devono essere 85x85 px.

Configurare le icone per gli altri agenti

Nell'anno 2013/2014 il sistema era multiagente. Gli agenti sono identificati da un fatto:

```
(deftemplate personstatus
    (slot step)
    (slot time)
    (slot ident)
    (slot pos-r)
    (slot pos-c)
    (slot activity)
    (slot move)
)
```

Lo slot `ident` contiene l'identificativo di una certa persona (che è utilizzato anche altrove, ad esempio nella definizione del path).

Nell'array `map_img` abbiamo la definizione di tutti i file che costituiscono le icone, fra cui quella relativa alle persone:

```
    map_img.put("person_Person", "person_empty.png"); //Persona in piedi
    map_img.put("person_Seat", "person_seat.png"); //Persona seduta
```

Qualora non si desideri avere icone personalizzate per ogni agente, sarà sufficiente lasciare intoccato il codice e fornire le due immagini **person_empty.png** e **person_seat.png** (che saranno quindi le immagini usate da ogni agente).

Se però si desidera personalizzare (per ogni person, quindi per ogni ident) l'immagine relativa, si possono fornire immagini differenziate secondo la forma:

```
<ident>-person_empty.png
<ident>-person_seat.png
```

In questo modo la persona con identificativo `<ident>` userà tali immagini. *Non è necessario modificare il codice della GUI*, ma solamente aggiungere tante coppie di immagini quanti sono gli identificativi delle persone. Qualora non ci sia matching fra le immagini e l'identificativo della persona, verranno usate le immagini di default **person_empty.png** e **person_seat.png**.

Facendo quindi un esempio, se si ha un fatto:

```
(personstatus
    (step 0)
    (time 0)
    (ident C2)
    (pos-r 8)
    (pos-c 10)
    (activity seated)
)
```

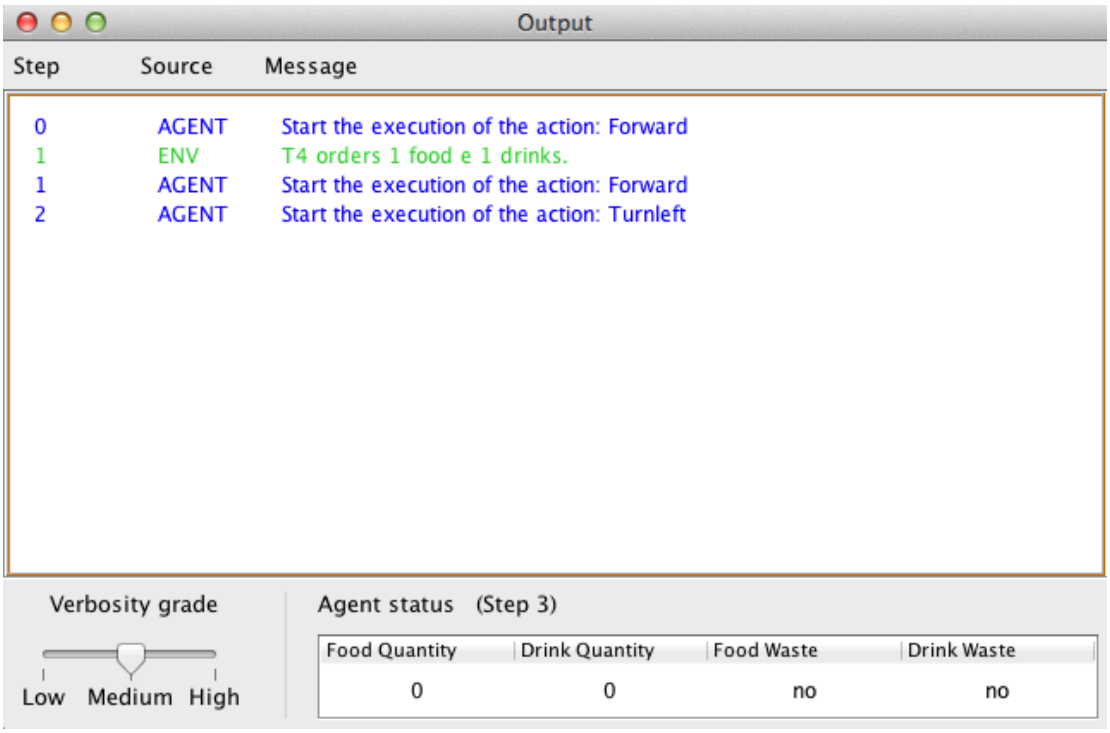
Si dovranno creare le due immagini:

```
C2-person_empty.png
C2-person_seat.png
```

Tutta la logica è implementata nei metodi `updateMap()` delle due classi **MonitorView.java** e **MonitorModel.java**.

Configurare le stampe in output

La finestra di output permette di visualizzare le varie stampe compiute dall'agente (regolandone le verbosità) e mostra anche una stampa relativa allo stato dell'agente (questa è chiaramente molto correlata al dominio che si sta trattando). Tutte e due le aree di testo sono aggiornate simultaneamente all'aggiornamento della mappa.



Finestra stampe

Per far funzionare le stampe si suggerisce di introdurre questo codice nel modulo *MAIN*:

```
(deftemplate printGUI
        (slot time (default ?NONE))
        (slot step (default ?NONE))
        (slot source (type STRING) (default ?NONE))
        (slot verbosity (type INTEGER) (allowed-integers 0 1 2) (default 0))
        (slot text (type STRING))
        (slot param1 (default ""))
        (slot param2 (default ""))
        (slot param3 (default ""))
        (slot param4 (default ""))
        (slot param5 (default ""))
)
```

La verbosity è uno slot che permette di fare stampe differenziate a tre livelli. Se il livello di verbosity è settato a 1 allora saranno visualizzate le stampe dei fatti printGUI di livello 0 e di livello 1. Le stampe di livello 0 sono sempre visualizzate. La verbosità è configurabile direttamente dalla finestra di output. Lo slot source invece è utile per identificare (sempre nella GUI) la fonte della stampa.

Giacché il campo `text` è una stringa e non può contenere variabili, ci sono cinque slot adibiti ad al salvataggio del valore delle variabili. Tali valori verranno poi sostituiti in fase di stampa del testo nella GUI: più precisamente è possibile usare l'escape `%p1` per indicare il contenuto della variabile nello slot `param1`. Quindi, ad esempio, la assert:

```
(assert (printGUI .... (text "Numero di ordini: %p1") (param1 ?orders)))
```

produrrà un testo dove il valore della variabile `?orders` sarà sostituito alla stringa `%p1`. È possibile avere al massimo 5 parametri, ognuno identificato chiaramente da un escape corrispondente (`%p2` per lo slot `param2` ecc.). I parametri hanno valore di default dunque non si è costretti ad esprimerli. L'uso di questa architettura per le stampe

vede però necessaria l'assegnazione dei valori di `step` e di `time`.

La verbosity è impostata tramite GUI in maniera dinamica, quindi è possibile in qualsiasi istante modificarla e vedere le stampe precedenti. Le stampe possono essere trattate dalla GUI, assegnando ad ogni source un particolare colore. Nella classe `PrintOutWindow.java` è definita una HashMap chiamata `sources` che assegna ad ogni source (una stringa, la chiave della map) un colore (oggetto `Color` di Java). Le stampe vengono visualizzate esplicitando la `source`, seguite dal campo `text` (debitamente modificato). È quindi evidente che ciò che viene inserito nello slot `source` debba anche apparire come chiave della HashMap `sources`.

Il retrieval dei fatti `printGUI` avviene nella classe **MonitorView**, precisamente nel metodo `updateOutput()`. Volendo è possibile configurare le stampe ed i formati a piacere lavorando sul suddetto metodo.

Finestra Agent Status

La finestra che illustra lo stato dell'agente è molto semplice e parte del suo contenuto è definito nel metodo `updateOutput()` che richiama `updateAgentStatusWindow()` (ovviamente nella classe **PrintOutWindow**). Parte dei dati che vengono stampati vengono presi da variabili contenute nel modello (quindi nella classe **MonitorModel**), come accade per la variabile `map` della `updateMap()`. In caso di personalizzazione di questa tabella (cosa molto probabile, essendo una parte fortemente legata al dominio) è necessario modificare la GUI (ma si suppone sia sufficente cambiare i campi della tabella).

In breve: fatti necessari

Quindi, riassumendo, per poter configurare il proprio progetto è necessario che vi siano diversi fatti:

- `(init-agent (done yes))`, che regola il funzionamento dello `Start` e indica quando l'ambiente CLIPS è inizializzato (cioè sono state eseguite le regole che inizializzano lo stato dell'agente).
- `(last-perc (step ?x))`, che regola il funzionamento degli step e della run. Ogni qualvolta avviene una percezione, tale fatto viene modificato (viene aggiornato lo slot `step`): funge dunque da clock: quando lo step è incrementato di uno, sappiamo che è stata eseguito un intero ciclo e l'agente ha nuovamente effettuato delle percezioni. È dunque necessario configurare questo fatto in modo che venga aggiornato subito dopo che l'agente ha interpretato le percezioni (va quindi trattato nel modulo *AGENT*).
- `(status ... (result [done/no]))`, che regola la terminazione dell'esecuzione. Più precisamente, l'esecuzione prosegue fintanto che c'è il fatto `(status ... (result no))`: è quindi necessario porre attenzione a quali regole impostino lo slot `result` a `done` (provocando così la terminazione dell'esecuzione).

Se questi due fatti sono presenti, il pannello di controllo dovrebbe funzionare correttamente e permettervi di eseguire i vostri file CLIPS. Se poi aggiungete i fatti `printGUI` (verificando la coerenza fra le source scritte in CLIPS e le chiavi della map `sources`) allora anche la finestra di output funzionerà correttamente.

Nessun altro file necessita di modifica alcuna.

In breve: cosa configurare

I passi di configurazione da effettuare sono quindi:

- Aggiungere i file .clp nelle cartella CLP
- File **MonitorModel.java** e **MonitorView.java** per modificare il retrieval dei fatti (che definiscono la mappa).
- Modificare le icone della mappa (configurando correttamente la `map_img` nella classe **MonitorView**).
- Modificare la tabella che determina lo stato dell'agente nella OutputWindow (sia nella GUI che nel metodo `updateAgentStatusWindow()`).