Relazione progetto IALAB

Maurizio Dominici, Alessandro Serra, Andrea Aloi

6 maggio 2014

Sommario

Relazione del progetto Intelligenza Artificiale e Laboratorio anno accademico 2012-2013. Illustrazione delle diverse strategie proposte, con una disamina dettagliata dei pregi e difetti di ognuna di esse, approfondendo anche i motivi delle scelte fatte sia in fase di progettazione che di implementazione. Vengono illustrate le linee guida per la comprensione delle cinque strategie proposte, ponendo l'accento sulle differenze tra ognuna di esse. Infine è stata dedicata un'intera sezione al confronto tra le varie tecniche adottate.

Indice

Capitolo 1

Introduzione

Tutte le strategie illustrate in questo documento hanno in comune la struttura. Il codice è strutturato in maniera gerarchica, in modo tale che ogni modulo si occupi di un compito particolare da svolgere, il tutto coordinato da una serie di controllori inseriti nel file `5_agent.clp`, fornito nello scheletro iniziale del progetto. Partiamo illustrando il modulo agent: questo modulo, concettualmente, si occupa solo di gestire e coordinare i vari moduli a cui è assegnato un compito specifico. Agent controlla che la condizione di entrata per un modulo sia soddisfatta, quindi dà il focus al modulo in questione attendendo che il lavoro sia svolto. Quando questo modulo rilascerà il focus attraverso una pop, agent avrà a questo punto la possibilità di dare il focus a un altro modulo che svolgerà un altro sottocompito. Ogni modulo, come detto, ha un compito particolare (verranno illustrati in seguito i vari moduli con i rispettivi compiti). La struttura di ogni modulo è abbastanza semplice ed è così composta con le azioni elencate in ordine cronologico:

1. riceve il focus dal modulo agent, il quale controlla che la condizione affinché sia svolto il compito assegnato al modulo specifico sia soddisfatta;

2. svolge le operazioni cui è deputato;

3. asserisce una condizione per cui sia comprensibile, al modulo agent, che il compito richiesto è stato eseguito con successo;

4. rilascia il focus mediante una pop.

Ci sono poi, a seconda delle varie strategie, delle varianti a questo comportamento, ma generalmente il flusso di esecuzione è quello illustrato. Alcune eccezioni possono essere la necessità di spezzare la routine di ogni turno, ad esempio quando il modulo del tempo inferisce che il tempo rimanente per

dirigersi verso un gate scarseggia, ma entreremo nel dettaglio durante l'illustrazione specifica delle varie strategie. Verranno approfonditi poco più avanti i moduli più nel dettaglio anche se, per una spiegazione esaustiva, il rimando è alla sezione della strategia considerata.

1.1 Controllore

Entriamo ora più nello specifico nel comportamento del controllore. Il controllore si occupa, come detto, di dare il focus al modulo corretto. Quest'ultimo viene fatto identificando una determinata routine da svolgere per ogni passo (identificato dallo slot `step` del template status fornito nello scheletro iniziale del progetto):

1. controllo che ci siano celle ancora da visitare (sezione 1.2)
2. controllo del tempo rimanente (sezione 1.3)
3. individuazione della cella verso cui spostarsi (sezione 1.4)
4. calcolo del percorso per raggiungere la cella individuata mediante l'algoritmo A* (sezione 1.5)
5. controllo che la cella individuata permetta di raggiungere un'uscita, ai fini di non finire in vicoli ciechi (sezione 1.6)
6. computazione effettiva dei passi calcolati e memorizzati dall'algoritmo A* (sezione 1.7)

1.2 Modulo: Controllo delle celle da visitare

Questo modulo controlla che vi siano, sulla mappa, ancora delle celle che dovrebbero essere visitate. Se ne esistono allora si continua nel flusso dei controllori, altrimenti viene notificato che, al fine di non sprecare tempo, ci si deve dirigere verso la più vicina cella di tipo gate, il cui percorso viene calcolato in questo modulo.

1.3 Modulo: Controllo del tempo rimanente

Questo modulo si occupa di controllare che il tempo rimasto a disposizione sia sufficiente. Generalmente questo significa controllare il tempo necessario a raggiungere l'uscita più vicina e confrontarlo con il tempo rimanente. Se

il tempo rimanente è sufficiente si continua nel flusso normale delle azioni asserendo la condizione di successo per il modulo agent. Se, invece, il tempo scarseggia si intraprende un'azione che può variare a seconda delle varie strategie.

1.4 Modulo: Individuazione della cella target

In questo modulo viene scelta la cella verso cui dirigersi (target). I criteri con cui viene stabilita la cella target variano molto a seconda della strategia adottata.

1.5 Modulo: Esecuzione dell'algoritmo A*

Questo modulo, pressoché invariato in ogni strategia, computa semplicemente l'algoritmo A* partendo dalla posizione attuale per arrivare al target. All'interno del modulo stesso vengono prodotti fatti che serviranno poi al modulo 1.7.

1.6 Modulo: Controllo di uscita

Questo modulo si occupa di verificare che, una volta che verrà raggiunto il target trovato, ci sia la possibilità di raggiungere una cella di tipo `gate`. Questo è un'*escamotage* studiato al fine di non ritrovarsi in situazioni che portino l'UAV a non poter concludere il suo lavoro (ad esempio finendo in un vicolo cieco dal quale non si possa uscire), con relativa penalità altissima. Nel caso in cui sia possibile raggiungere una cella di tipo `gate` dal target stabilito si prosegue normalmente nel flusso dei moduli, asserendo la condizione di successo. Viceversa, se non fosse possibile raggiungere alcun `gate`, si aggiunge un fatto che indica la condizione di impossibilità di uscire da quella cella, in modo che non venga più presa in considerazione nella ricerca di nuovi target candidati; quindi si procede ritraendo il fatto di successo dei moduli 1.5 e 1.4 così che il controllore (1.1), una volta effettuata la pop, dia nuovamente il focus al modulo 1.4 che si occuperà di trovare un nuovo target papabile (scartando la cella trovata precedentemente perché contrassegnata dal fatto che indica l'impossibilità di uscire da quella cella). Vale la pena spendere due parole sul meccanismo con cui questo viene fatto: una volta computato A* sappiamo la direzione verso cui l'UAV sarà rivolto una volta arrivato nella cella target. Se si scopre che l'UAV non ha la possibilità di uscire, una volta arrivato in quella cella, quella cella nella determinata direzione in cui l'UAV

verrebbe a trovarsi viene contrassegnata come illegale con un fatto di tipo invalid-target. Il modulo 1.4 si occupa di escludere le celle così contrassegnate dalla ricerca della cella più appetibile verso cui dirigersi.

1.7 Modulo: Movimento

Anche quest'ultimo modulo, che in realtà è stato scritto, per questioni di economia di codice, direttamente nel modulo agent, è invariato per ogni strategia scelta, infatti si occupa solamente di computare un passo dell'UAV mediante la regola, già fornita nello scheletro iniziale, exec. Per eseguire l'azione di movimento ci si basa sui fatti di tipo PATH prodotti nel modulo 1.5, già ordinati, che indicano i singoli movimenti che l'UAV deve compiere a ogni passo.

1.8 Inform delle celle visibili

A ogni esecuzione di un movimento dell'UAV viene dato il focus a un modulo di inform. Questo modulo ha la responsabilità di effettuare azioni di tipo `inform` di celle sulle quali non è ancora stata effettuata tale azione secondo i criteri della strategia. Generalmente si è scelto di effettuare una `inform` base per ogni nuova cella avvistata, in quanto il costo dell'operazione di `inform` è basso: tuttavia questo può dipendere dalle varie strategie. Si è scelto di non dare il focus a questo modulo da un proprio controllore, questo perché a ogni movimento dell'UAV si possono avere, potenzialmente, fino a tre nuove celle da informare: è quindi un'azione da fare ogni qualvolta venga effettuata una `exec-move-path`. È da notare che in questo modulo non vengono effettuate direttamente le inform, bensì vengono asseriti dei fatti di tipo `inform-act` con tutti i parametri necessari della inform. È stata implementata una regola chiamata `exec-inform` nel modulo `AGENT` che si occupa di fare le inform previste dai fatti `inform-act` e di cancellare questi ultimi. Questo si è reso necessario al fine di una corretta segnalazione delle inform, che comportano un cambio dello slot `step` del template `status`. Questo comporta che, al fine di una corretta inform, è necessario delegare quest'operazione al modulo agent in cui vengono effettuate le exec.

Capitolo 2

Strategia Safe

La prima strategia illustrata è quella più semplice, che useremo come metro di paragone. Ha la proprietà desiderabile di essere Safe, ovvero intraprende sempre un procedimento che porta a un risultato per cui l'UAV riesce a uscire entro il tempo stabilito anche se non è detto che, necessariamente, riesca a massimizzare le inform effettuate nel tempo a disposizione. La strategia è molto semplice: ci si basa sempre sui controllori implementati nel modulo AGENT, tuttavia non a tutti viene dato il focus a ogni passo dell'UAV, a differenza di quanto detto nella descrizione generica delle strategie nel capitolo 1. In effetti, la principale differenza dalla maggior parte delle altre strategie implementate (e da quanto descritto precedentemente), è che il modulo che si occupa della computazione dei passi non rilascia il focus fino a quando non vengono computati tutti i passi calcolati da A*: questo comporta che i vari controllori vengano azionati solo una volta eseguiti tutti i passi previsti, rendendo la strategia meno adattativa alla variazione di punteggi conseguente alle inform effettuate nel percorso. Un'altra peculiarità di questa strategia è che viene completamente ignorata la possibilità di effettuare una loiter monitoring per poi compiere una inform precisa sullo stato delle celle. Questo è dovuto al fatto che si volesse implementare una strategia il più possibile semplice da poter poi usare come metro di paragone e confronto con le altre strategia e che avesse, come unica condizione realmente necessaria, la proprietà di essere Safe. Per poter rendere sufficientemente interessante la strategia si è quindi dovuto ristrutturare la routine di AGENT che, in effetti, si discosta abbastanza da quella basilare illustrata nel capitolo 1. Entriamo più nei dettagli illustrando, modulo per modulo, il funzionamento della strategia.

2.1 Controllore

Come detto, vi sono diverse differenze nella routine di controllo nel modulo AGENT rispetto alla strategia generica tratteggiata nel capitolo 1. Il modulo AGENT è composto dai seguenti controllori:

- controllo che ci siano celle ancora da visitare (sezione 2.2)
- individuazione della cella verso cui spostarsi (sezione 2.3)
- calcolo del percorso per raggiungere la cella individuata mediante l'algoritmo A* (sezione 2.4)
- controllo del tempo rimanente a partire dal raggiungimento della cella identificata come obiettivo (sezione 2.5)
- controllo che la cella individuata permetta di raggiungere un'uscita, ai fini di non finire in vicoli ciechi (sezione 2.6)
- computazione effettiva dei passi calcolati e memorizzati dall'algoritmo A* (sezione 2.7)

Si noti che il modulo 2.7, come già accennato nell'introduzione di questo capitolo, computa (e consuma) tutti i fatti prodotti dal modulo 2.4. Entriamo ora più nel dettaglio dei vari moduli.

2.2 Modulo: Controllo delle celle da visitare

Il controllo della presenza di celle da visitare viene fatto considerando i punteggi (vedere modulo 2.3) delle celle sulla mappa. Se esistono celle con un punteggio superiore a zero[1] si asserisce un fatto di tipo `finish_checked` e si prosegue con il prossimo controllore. Altrimenti, ovvero se non ci sono più celle da informare presenti sulla mappa, si calcola il percorso verso l'uscita più vicina e si asserisce un fatto di tipo `finished`: questo comporta che i successivi controllori vengano inibiti[2] per far sì che l'UAV si diriga verso una cella di tipo gate senza sprecare ulteriore tempo.

[1] le celle con punteggi negativi sono celle di tipo hill, border, gate o lake, oppure celle di tipo Urban o Rural su cui è già stata effettuata una inform

[2] a eccezione di 2.7

2.3 Modulo: individuazione della cella target

L'individuazione della cella più appetibile nella strategia di base è molto semplice. Durante la fase di inizializzazione dell'ambiente viene assegnato un punteggio a ogni cella, in base alla propria tipologia, e salvato nello slot `val` del template `score_cell`. Questo template serve a memorizzare i punteggi delle celle per individuare la più appetibile presente sulla mappa. I valori, che ricordo non vengono calcolati in questo modulo, bensì nella fase di inizializzazione dell'ambiente, vengono assegnati nel seguente modo: DA COMPLETARE UNA VOLTA STABILITI I PUNTEGGI ESATTI

- Rural: 900[3]
- Urban: 1000[4]
- Hill: -100
- Lake: -5
- Border: -100
- Gate: -100

Ciò che viene fatto in questo modulo è semplicemente trovare la cella con il punteggio più alto presente in `val` e impostarla come target per l'algoritmo A* che verrà computato nel modulo 2.4. È importante notare che, in questo procedimento, vengono escluse automaticamente le celle di tipo Hill, Border e Gate: le prime due per impossibilità di muoversi su celle di quel tipo, l'ultima perché non è desiderabile dirigersi verso una cella di questo tipo fino a quando non si è concluso il compito. Vengono altresì escluse le celle che sono già state contrassegnate come invalid-target nei moduli 2.6 e 2.5. Nel modulo viene quindi scelta la cella con `val` maggiore e impostata come target, quindi il focus viene rilasciato asserendo la condizione di controllo `punteggi_checked`.

2.4 Modulo: Esecuzione dell'algoritmo A*

Il modulo A* calcola il percorso dalla posizione dell'UAV alla posizione desiderata contrassegnata dal fatto di tipo target, ricalcando l'algoritmo illustrato a lezione. L'esecuzione dell'algoritmo parte dal nodo 0, corrispondente

[3] il valore viene abbassato a -5 una volta informata
[4] il valore viene abbassato a -5 una volta informata

alla posizione dell'UAV, nonché il nodo corrente all'inizio dell'esecuzione. Da questo nodo saranno individuati tutti gli spostamenti applicabili[5], tramite le regole aventi un template del tipo `nome_azione`[6]`-apply-direzione`[7]. I fatti prodotti da queste regole sono di tipo `(id ?curr) (op ?op) (direction ?dir) (pos-x ?r) (pos-y ?c)))`. Questi ultimi servono alle regole del tipo `nome_azione-exec-direzione` che, dato un fatto di tipo `apply`, ne asseriscono un fatto di tipo:

```
(newnode
(ident (+ ?n 11))
(pos-r ?r)
(pos-c (- ?c 1))
(direction ?dir)
(gcost (+ ?g 15))[8]
(fcost (+ (+(*(+(abs (- ?x ?r)) (abs (- ?y (- ?c 1)))) 10) 5) ?g
15))[9]
(father ?curr)) )
```

Ogni `newnode` viene controllato da alcune regole:

- `check-closed`: se un newnode ha già un nodo "chiuso" corrispondente, ma il suo costo è minore, esso viene aggiornato;
- `check-open-better`: se un newnode ha già un nodo aperto corrispondente, ma il suo costo è minore, il nodo viene aggiornato;
- `check-open-worse`: se un newnode ha già un nodo aperto corrispondente, ma il suo costo è maggiore, il nwnode viene retratto;
- `add-open`: se non esiste un nodo corrispondente, il newnode diventa un node aperto.

Infine, vi è una regola chiamata `changecurrent` che si occupa di impostare come nodo corrente il nodo aperto con costo minore.

2.5 Modulo: Controllo del tempo rimanente

La principale differenza in questo modulo, rispetto al normale funzionamento in altre strategie, è che il tempo viene calcolato a partire dalla cella che si sta

[5] ovvero quelli che non portano a collisioni da parte dell'UAV

[6] può essere go-forward, left o right

[7] north, east, south, west

[8] rappresenta il costo effettivo per arrivare al nodo

[9] rappresenta il costo complessivo, calcolato con l'euristica distanza di Manhattan, per arrivare al target

cercando di raggiungere. È noto il tempo rimanente disponibile (dallo slot time del template perc-vision), mediante il modulo 2.4 è noto anche il tempo che verrà consumato una volta raggiunta la cella desiderata: ad esso viene aggiunto un delta di tempo che tiene in considerazione la possibilità di fare una inform a ogni passo (in questa strategia, quindi, il delta sarà pari a 3 per ogni passo che computerà A*, visto che è necessario fare al più 3 inform a ogni passo dell'UAV). Si detrae, dal tempo rimanente, questo ammontare di tempo necessario a spostarsi verso il target, ottenendo il tempo che rimarrà una volta raggiunto il target. A questo punto viene calcolato il tempo, a partire da quel target, per raggiungere ogni uscita. L'uscita più vicina avrà il tempo inferiore per essere raggiunta quindi si confronta questo tempo con il tempo rimanente una volta raggiunto il target. Se il tempo sarà sufficiente per uscire verrà asserita la condizione di successo del modulo, altrimenti la cella target verrà contrassegnata come `invalid-target` e, ritraendo i fatti opportuni (`astar_checked` e `punteggi_checked`, il focus, una volta tornato a `AGENT`, verrà assegnato nuovamente al modulo 2.3 che questa volta escluderà la cella appena contrassegnata come `invalid-target` dalle sue computazioni.

2.6 Modulo: Controllo di uscita

Questo modulo si occupa di verificare che, una volta che verrà raggiunto il target trovato, ci sia la possibilità di raggiungere un gate. Questa è un'escamotage studiata al fine di non ritrovarsi in situazioni che portino l'UAV a non poter concludere il suo lavoro (ad esempio finendo in un vicolo cieco dal quale non si possa uscire), con relativa penalità altissima. Nel caso in cui sia possibile raggiungere una cella di tipo gate dal target stabilito si prosegue normalmente nel flusso dei moduli, asserendo la condizione di successo (`exit_checked`). Viceversa, se non fosse possibile raggiungere alcun gate, si aggiunge un fatto che indichi la condizione di impossibilità di uscire da quella cella, in modo che non venga più presa in considerazione nella ricerca di nuovi target papabili; quindi si procede ritraendo il fatto di successo dei moduli 2.5, 2.4 e 2.3 così che il controllore (2.1), una volta effettuata la pop, dia nuovamente il focus al modulo 2.3 che si occuperà di trovare un nuovo target papabile (scartando la cella trovata precedentemente perché contrassegnata dal fatto che indica l'impossibilità di uscire da quella cella). Come già illustrato nel capitolo 1 una volta computato A* sappiamo la direzione verso cui l'UAV sarà rivolto una volta arrivato nella cella target: se si scopre che l'UAV non ha la possibilità di uscire, una volta arrivato in quella cella, quella cella nella determinata direzione in cui l'UAV verrebbe a trovarsi viene contrassegnata come illegale con un fatto di tipo `invalid-target`. Il modulo

2.3 si occupa di escludere le celle così contrassegnate dalla ricerca della cella più appetibile verso cui dirigersi.

2.7 Modulo: Movimento

In questo contesto vengono semplicemente computati i passi dell'UAV prodotti dall'algoritmo A* del modulo 2.4 o dal modulo `sec:safe-finish`. L'algoritmo, come detto, produce fatti di tipo path. In questo modulo essi vengono computati in passi effettivi per l'UAV e ritratti, in modo da mantenere il più pulita possibile la Knowledge Base. In questa sede vengono analizzati fatti di tipo `path` (prodotti dal modulo 2.2) e fatti di tipo `path-star` (prodotti dal modulo 2.4) e vengono asseriti fatti di tipo `move-path`. Questi ultimi vengono poi computati effettivamente dalla regola `exec-move-path` presente nel modulo `AGENT`, che si occupa di far muovere effettivamente l'UAV e di consumare i fatti precedentemente creati, al fine di mantenere la Knowledge Base il più pulita possibile.

2.8 Inform delle celle visibili

A ogni esecuzione di un movimento dell'UAV viene dato il focus a un modulo di inform. Questo modulo ha la responsabilità di effettuare azioni di tipo `inform` di celle sulle quali non è ancora stata effettuata tale azione secondo i criteri della strategia. Viene effettuata una `inform` base per ogni nuova cella avvistata, in quanto il costo dell'operazione di `inform` è basso. Una volta effettuata una inform viene abbassato il punteggio della cella informata.

Capitolo 3

Stategia di base

Questa strategia risulta essere una leggera variante di quella illustrata nel capitolo 2. Ciò che varia, rispetto alla precedente, è il sistema di calcolo dei punteggi al fine di individuare la cella più promettente da esplorare verso cui dirigersi. Verranno enfatizzate solo le differenze rispetto alla strategia Safe.

3.1 Modulo: individuazione della cella target

L'individuazione della cella più appetibile nella strategia di base è molto semplice. Durante la fase di inizializzazione dell'ambiente viene assegnato un punteggio a ogni cella, in base alla propria tipologia, e salvato nello slot `val` del template `score_cell`. Questo template serve a memorizzare i punteggi delle celle per individuare la più appetibile presente sulla mappa. I valori, che ricordo non vengono calcolati in questo modulo, bensì nella fase di inizializzazione dell'ambiente, vengono assegnati nel seguente modo:

- Rural: 900[1]
- Urban: 1000[2]
- Hill: -100
- Lake: -5
- Border: -100
- Gate: -100

[1] il valore viene abbassato a -5 una volta informata
[2] il valore viene abbassato a -5 una volta informata

Il modulo in questione, come prima cosa, si occupa di assegnare, nello slot `abs_score` del template `score_cell`, un punteggio, per ogni cella, che tenga conto delle tipologie della cella in questione e di quelle limitrofe. Per fare ciò, semplicemente, vengono sommati i valori del campo val delle celle disposte a nord-ovest, nord, nord-est, ovest, est, sud-ovest, sud, sud-est e il campo val della cella in questione, ottenendo un valore che verrà appunto salvato nel campo `abs_score`. Viene chiamato `abs_score` che sta per "punteggio assoluto", vedremo nell'illustrazione delle altre strategie il perché di questa nomenclatura: in effetti è superfluo l'aggettivo "assoluto" in questa strategia ma, tuttavia, per consistenza nella nomenclatura, si è scelto di mantenere questo nome anche in questo contesto. Ciò che viene fatto in questo modulo è semplicemente trovare la cella con il punteggio più alto presente in `abs_score` e impostarla come target per l'algoritmo A* che verrà computato nel modulo corrispettivo di quello illustrato nella sezione 2.4. È importante notare che, in questo procedimento, vengono escluse automaticamente le celle di tipo Hill, Border e Gate (a cui, in effetti, non è nemmeno stato assegnato un valore al campo `abs_score`): le prime due per impossibilità di muoversi su celle di quel tipo, l'ultima perché non è desiderabile dirigersi verso una cella di questo tipo fino a quando non si è concluso il compito. Vengono altresì escluse le celle che sono già state contrassegnate come invalid-target nei moduli corrispettivi di 2.6 e 2.5. Al termine dei calcoli viene scelta la cella con `abs_score` maggiore e impostata come target, quindi il focus viene rilasciato asserendo la condizione di controllo `punteggi_checked`.

Capitolo 4

Stategia con punteggi relativi

Questa seconda strategia è molto simile alla precedente, perciò non ci inoltreremo in una disamina dettagliata di ogni caso, mentre ci limiteremo a enfatizzare le sole differenze, allo scopo di una lettura più fluida e comprensibile. L'unica differenza, anche se piuttosto corposa, è relativa al modulo che si occupa del calcolo della cella target (2.3).

4.1 Modulo: individuazione della cella target

L'individuazione della cella più appetibile nella strategia safe a punteggi relativi è un raffinamento di quella illustrata nel capitolo 3, sezione 2.3. Durante la fase di inizializzazione dell'ambiente viene assegnato un punteggio a ogni cella, in base alla propria tipologia, e salvato nello slot val del template `score_cell`. Questo template serve a memorizzare i punteggi delle celle per individuare la più appetibile presente sulla mappa. I valori, che ricordiamo non essere calcolati in questo modulo, bensì nella fase di inizializzazione dell'ambiente, vengono assegnati nel seguente modo:

- Rural: 900[1]
- Urban: 1000[2]
- Hill: -100
- Lake: -5
- Border: -100

[1] il valore viene abbassato a -5 una volta informata
[2] il valore viene abbassato a -5 una volta informata

- Gate: -100

Il modulo in questione, come prima cosa, si occupa di assegnare, nello slot `abs_score` del template `score_cell`, un punteggio, per ogni cella, che tenga conto delle tipologie della cella in questione e di quelle limitrofe. Per fare ciò, semplicemente, vengono sommati i valori del campo val delle celle disposte a nord-ovest, nord, nord-est, ovest, est, sud-ovest, sud, sud-est e il campo val della cella in questione, ottenendo un valore che verrà appunto salvato nel campo `abs_score`. Viene chiamato `abs_score` che sta per "punteggio assoluto", risulterà chiaro in questo contesto, a differenza della precedente strategia, il perché. In questo modulo ci si occupa di raffinare i punteggi presenti nel campo `abs_score` di ogni cella inserendolo nel campo `rel_score` della cella stessa. Per ogni cella, infatti, viene calcalolata la distanza di Manhattan (i due cateti del triangolo rettangolo la cui ipotenusa è la distanza "aerea" tra la cella su cui è l'UAV e la cella considerata). Con questa distanza viene calcolato un nuovo punteggio, `rel_score` appunto, che consiste nel rapporto tra il valore del campo `abs_score` e la distanza di Manhattan. Questo procedimento serve a dare un grado di interesse alle celle non solo in base alla "densità" di tipologie di celle interessanti nelle celle limitrofe a quella interessata, come avveniva nella strategia illustrata nel capitolo 3, bensì, il grado di interesse, sarà inversamente proporzionale anche alla distanza della cella dall'UAV. L'idea è di favorire un'esplorazione più intelligente della mappa, cercando di procedere in maniera omogenea limitando il più possibile gli spostamenti da un estremo all'altro della mappa. Come nella strategia precedente vengono escluse automaticamente le celle di tipo Hill, Border e Gate (a cui, in effetti, non è nemmeno stato assegnato un valore al campo `abs_score`): le prime due per impossibilità di muoversi su celle di quel tipo, l'ultima perché non è desiderabile dirigersi verso una cella di questo tipo fino a quando non si è concluso il compito. Vengono altresì escluse le celle che sono già state contrassegnate come `invalid-target` nei corrispettivi moduli 2.6 e 2.5 per questa strategia. Al termine dei calcoli viene scelta la cella con `rel_score` maggiore e impostata come target, quindi il focus viene rilasciato asserendo la condizione di controllo `punteggi_checked`.

Capitolo 5

Loiter Monitoring di base

La strategia chiamata Loiter Monitoring con punteggi relativi è una variante della strategia illustrata nel capitolo 4. La principale variante consiste nel modulo inform, in cui viene prevista un'azione di Loiter Monitoring se l'UAV si trova su una cella di tipo urban (e ha sufficiente tempo rimanente), come spiegato nel modulo 5.2.

5.1 Modulo: Controllo del tempo rimanente

Questo modulo è molto simile a quelli corrispondenti nelle altre strategia, vi è tuttavia un'aggiunta significativa giustificata dalla variazione della strategia di inform. Siccome l'operatione di Loiter Monitoring è piuttosto costosa, nel momento in cui il tempo comincia a scarseggiare, ovvero quando scende sotto la soglia di X (STABILIRE LA SOGLIA), si asserisce un fatto di tipo `no-loiter` che impedisca di effettuare quest'onerosa operazione.

5.2 Inform delle celle visibili

A ogni esecuzione di un movimento dell'UAV viene dato il focus a un modulo di inform. Questo modulo ha la responsabilità di effettuare azioni di tipo `inform` di celle sulle quali non è ancora stata effettuata tale azione secondo i criteri della strategia. Viene effettuata una `inform` base per ogni nuova cella avvistata, in quanto il costo dell'operazione di `inform` è basso. Quando, però, la nuova cella avvistata è di tipo `urban`, e si ha una percezione di tipo water su di essa, allora, se non è stato asserito il fatto `no-loiter`, si procede con una azione di loiter monitoring sulla cella in questione, quindi si predispone, attraverso un fatto di tipo `inform-act`, una inform precisa (`severe-flood`

o `initial-flood`) che verrà poi effettuata dalla regola `exec-inform` nel modulo `AGENT`.

Capitolo 6

Loiter Monitoring con punteggi relativi

La strategia chiamata Loiter Monitoring di Base è una variante della strategia illustrata nel capitolo 3. La principale variante consiste nel modulo inform, in cui viene prevista un'azione di Loiter Monitoring se l'UAV si trova su una cella di tipo urban (e ha sufficiente tempo rimanente), come spiegato nel modulo 5.2.

6.1 Modulo: Controllo del tempo rimanente

Questo modulo è molto simile a quelli corrispondenti nelle altre strategia, vi è tuttavia un'aggiunta significativa giustificata dalla variazione della strategia di inform. Siccome l'operatione di Loiter Monitoring è piuttosto costosa, nel momento in cui il tempo comincia a scarseggiare, ovvero quando scende sotto la soglia di X (STABILIRE LA SOGLIA), si asserisce un fatto di tipo `no-loiter` che impedisca di effettuare quest'onerosa operazione.

6.2 Inform delle celle visibili

A ogni esecuzione di un movimento dell'UAV viene dato il focus a un modulo di inform. Questo modulo ha la responsabilità di effettuare azioni di tipo `inform` di celle sulle quali non è ancora stata effettuata tale azione secondo i criteri della strategia. Viene effettuata una `inform` base per ogni nuova cella avvistata, in quanto il costo dell'operazione di `inform` è basso. Quando, però, la nuova cella avvistata è di tipo `urban`, e si ha una percezione di tipo water su di essa, allora, se non è stato asserito il fatto `no-loiter`, si procede con una azione di loiter monitoring sulla cella in questione, quindi si predispone,

attraverso un fatto di tipo `inform-act`, una inform precisa (`severe-flood` o `initial-flood`) che verrà poi effettuata dalla regola `exec-inform` nel modulo `AGENT`.

Capitolo 7

Confronti tra le strategie

provola

Capitolo 8

Bonus

In quest'ultima sezione illustreremo alcune features interessanti, usate per questo progetto, che esulano dal programma d'esame.

8.1 Repository Git

Ai fini di un adeguato meccanismo di versioning per la collaborazione tra gli studenti autori del progetto, si è scelto di utilizzare git come meccanismo di versioning. Questo ha comportato un sistema affidabile per il tracciamento di versioni e modifiche, con la possibilità di tenere traccia, ed eventualmente facendo delle operazioni di revert, degli sviluppi incrementali dei vari progetti. È stato reso disponibile il repository su github[1].

8.2 Sistema di collaborazione

Un altro strumento particolarmente utile alla collaborazione e alla coordinazione del lavoro è stato l'utilizzo del tool gratuito Zoho Projects[2], che ci ha consentito di dividerci i compiti, tener traccia dei compiti da svolgere, dei problemi da risolvere e di avere una message board ordinata che consentisse di tener traccia del lavoro svolto e da svolgere. È stato inoltre possibile fissare delle date di scadenza sui vari compiti in maniera da organizzare in maniera chiara, ordinata e funzionale il flusso di lavoro.

[1] il repository è disponibile su github all'indirizzo https://github.com/axedre/ialab
[2] https://www.zoho.com/projects/

8.3 Migliorie nell'interfaccia grafica

Importanti migliorie nell'interfaccia grafica java sono apprezzabili. La principale è stata l'integrazione dei punteggi sulla mappa, in maniera da rendere più intuitivo il comportamento dell'UAV. A essa è stato aggiunto anche il disegno di un bersaglio per rappresentare la cella verso cui l'UAV è diretto. Sempre ai fini di una più chiara situazione sulla mappa vengono colorate in verde le celle su cui è già stata fatta una inform mentre sono illustrate in rosso quelle su cui deve essere ancora fatta.

8.4 Generatore di mappe

Infine, mediante l'utilizzo delle tecnologie AngularJS e Node.js, è stato costruito un generatore di mappe[3] che consente di generare, in maniera grafica, i file di configurazione .clp delle mappe pre e post allagamento, interamente personalizzabili in ogni aspetto. Questo ha reso più snelli e veloci i test delle varie strategie con ogni tipologia di mappa.

[3]disponibile all'indirizzo http://mmg-axedre.rhcloud.com/