

CORSO DI
INTELLIGENZA ARTIFICIALE E LABORATORIO

ANNO ACCADEMICO 2012/2013

Relazione del progetto Monitor

Marco Varesano

Luca Violanti

Indice

I	Il progetto	2
1	Proprietà dell'ambiente e dell'agente	3
2	Moduli del progetto	4
2.1	Moduli MAIN e INITIAL_MAP	4
2.1.1	MAIN	4
2.1.2	INITIAL_MAP	4
2.2	Moduli ENV e ACTUAL_MAP	5
2.2.1	ENV	5
2.2.2	ACTUAL_MAP	5
2.3	Modulo AGENT	6
2.4	Modulo CHECKER	7
2.5	Moduli PLANNER e NEW	7
3	Scelte progettuali	8
3.1	Comportamento di base dell'agente	8
3.1.1	La prima mossa	9
3.1.2	Fase di esplorazione	10
3.1.3	Fase di fuga	11
3.1.4	Ritorno all'esplorazione (<i>switch</i>)	11
3.2	Gestione dell'utilità	12
3.3	Schema di funzionamento dell'agente	14
4	Strategie realizzate	16
4.1	"Random walk"	17
4.2	"Random walk" con <i>switch</i>	18
4.3	Prima implementazione di un agente <i>greedy</i>	19
4.4	Agente <i>greedy</i> : 1 <i>chance</i>	20
4.5	Agente <i>greedy</i> : 1 <i>chance</i> con <i>switch</i>	21
4.6	Agente <i>greedy</i> : 2 <i>chances</i>	22
II	La sperimentazione	23
5	Misura delle prestazioni	24
6	Test effettuati e risultati ottenuti	25
6.1	Mappa default	27
6.1.1	Agente "random walk"	28
6.1.2	Agente "random walk" con <i>switch</i>	29

6.1.3	Primo agente greedy	30
6.1.4	Agente greedy 1 chance	31
6.1.5	Agente greedy 1 chance con switch	32
6.1.6	Agente greedy 2 chances	33
6.1.7	Confronto prestazionale tra le strategie implementate	34
6.2	Mappa generata randomicamente	37
6.2.1	Agente “random walk”	38
6.2.2	Agente “random walk” con switch	39
6.2.3	Primo agente greedy	40
6.2.4	Agente greedy 1 chance	41
6.2.5	Agente greedy 1 chance con switch	42
6.2.6	Agente greedy 2 chances	43
6.2.7	Confronto prestazionale tra le strategie implementate	44
6.3	Mappa “realistica”: Torino	46
6.3.1	Agente “random walk”	47
6.3.2	Agente “random walk” con switch	49
6.3.3	Primo agente greedy	51
6.3.4	Agente greedy 1 chance	53
6.3.5	Agente greedy 1 chance con switch	55
6.3.6	Agente greedy 2 chances	57
6.3.7	Confronto prestazionale tra le strategie implementate	59
7	Conclusioni	61
7.1	Riflessioni sui risultati dei test	63
7.2	Considerazioni personali	63

Parte I

Il progetto

Capitolo 1

Proprietà dell'ambiente e dell'agente

Secondo la classificazione di Russell e Norvig [1], l'ambiente del progetto Monitor ha le seguenti caratteristiche:

- è **parzialmente osservabile** poiché i sensori forniscono all'agente percezioni relative alle sole 8 celle attorno ad esso, oltre alla cella su cui l'agente si trova;
- ha **un singolo agente**, ovvero non è prevista la presenza di altre entità che possano influenzare l'agente (né cooperative, né competitive);
- è **deterministico**, ovvero lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseguita dall'agente;
- è **sequenziale**, perché ogni azione dipende dalle scelte effettuate in precedenza e ogni decisione presa può influenzare tutte le successive;
- è **statico** in quanto non può cambiare mentre l'agente sta “pensando” alla successiva azione da compiere;
- è **discreto** in quanto il problema è stato modellato tramite l'uso di una griglia limitata di celle, che portano ad avere un numero finito di stati distinti;
- è **noto** in quanto i risultati di tutte le azioni sono conosciuti dall'agente.

L'agente:

- è **dotato di sensori** tramite i quali interpreta gli input percettivi provenienti dall'ambiente;
- è **basato sull'utilità**, anche se la misura e l'utilizzo variano tra le differenti strategie implementate;
- è in grado di **apprendere** tramite l'esplorazione: il suo comportamento dipende da una conoscenza a priori del mondo (precedente all'allagamento) costituita dalla mappa iniziale, ma anche dalla sequenza percettiva osservata fino a quel momento.

Capitolo 2

Moduli del progetto

2.1 Moduli MAIN e INITIAL_MAP

2.1.1 MAIN

Il modulo `MAIN` è usato per la comunicazione tra i moduli `ENV` ed `AGENT`. In esso sono presenti varie definizioni di fatti non ordinati (*template*) che rappresentano:

- la codifica delle azioni che l'agente può compiere (`exec`);
- lo stato globale del mondo (`status`);
- la codifica delle percezioni visive fornite all'agente dopo ogni azione (`perc-vision`);
- la codifica delle percezioni più precise, fornite all'agente solo dopo un'azione di `loiter-monitoring` (`perc-monitor`);
- lo stato iniziale dell'agente (`initial_agentstatus`);
- la codifica della conoscenza a priori che l'agente ha della mappa dell'ambiente **prima** che avvenga l'inondazione (`prior_cell`).

I fatti definiti in questo modulo sono importati sia dal modulo `ENV` che dal modulo `AGENT`.

2.1.2 INITIAL_MAP

In questo modulo sono specificati vari fatti che rappresentano:

- lo stato iniziale dell'agente in termini di riga, colonna e direzione;
- il tempo totale a disposizione dell'agente per esplorare la mappa;
- la descrizione delle singole celle che compongono l'intera mappa **prima** dell'inondazione.

2.2 Moduli ENV e ACTUAL_MAP

2.2.1 ENV

ENV è il modulo che simula l'effetto delle azioni sull'ambiente e sull'agente fornendo le percezioni relative alla porzione di mondo attualmente interessata.

Si occupa inoltre di simulare lo scorrere del tempo e ha anche il compito di gestire le penalità.

Al suo interno sono specificate le definizioni dei *template* che rappresentano:

- lo stato dell'ambiente **dopo** l'inondazione (**actual_cell**);
- lo stato dell'agente (**agentstatus**);
- per ogni cella della mappa, uno stato interno che rappresenta la conoscenza che il simulatore possiede circa le interazioni che l'agente ha avuto con ogni singola cella della mappa (**discovered**); questi fatti verranno usati in fase di assegnazione delle penalità.

2.2.2 ACTUAL_MAP

In questo modulo è specificata la regola di asserzione dei vari fatti che rappresentano la descrizione delle singole celle che compongono l'intera mappa **dopo** l'inondazione.

2.3 Modulo AGENT

Questo modulo rappresenta il *core* del progetto, in quanto modella il comportamento dell'agente:

- riceve e interpreta le percezioni che gli giungono dall'ambiente;
- in combinazione con i moduli **CHECKER** e **PLANNER**, effettua la pianificazione delle successive azioni da compiere;
- in seguito, attua le strategie di movimento esplorativo e segnalazione di zone inondate alla base operativa.

Lo scopo principale di questo modulo è quello di asserire le azioni che l'agente dovrà compiere con l'obiettivo di massimizzare il numero di segnalazioni effettuate e quindi minimizzare la penalità ricevuta.

Al suo interno sono contenute le definizioni dei *template* che rappresentano:

- la conoscenza dell'agente della sua posizione e del tempo trascorso (**kagent**);
- la conoscenza parziale che l'agente ha del mondo durante la fase di esplorazione, che ha luogo **dopo** l'inondazione (**kagent_cell**).

Gli altri *template* che trovano posto all'interno di questo modulo sono necessari per il movimento e la pianificazione.

I fatti definiti in questo modulo sono importati sia dal modulo **CHECKER** che dal modulo **PLANNER**.

2.4 Modulo CHECKER

Il modulo **CHECKER** è deputato alla scelta della successiva azione da compiere (sia che si tratti di un'azione di movimento, sia di segnalazione di zone inondate).

L'azione che verrà scelta, tra quelle possibili, dipende da quale variante della strategia è adottata dall'agente.

2.5 Moduli PLANNER e NEW

Questi moduli implementano la strategia di ricerca informata A^* ¹ che fornisce all'agente un piano: esso è costituito da una sequenza di *way-point* che rappresentano la via più breve in termini di tempo per raggiungere una cella obiettivo (*goal*) a partire da una determinata cella di partenza.

Il piano risultato viene restituito al modulo **CHECKER**, che lo sfrutterà per scegliere la successiva azione da compiere.

¹Il calcolo della funzione di costo ($f(x) = g(x) + h(x)$) è indispensabile per la strategia A^* . Questa funzione, nel caso di mondi "a griglia", viene solitamente implementata usando un'euristica $h(x)$ realizzata tramite la funzione di distanza di Minkowski nota come *distanza Manhattan* [2], la quale valuta la distanza tra due punti come la somma del valore assoluto delle differenze delle loro coordinate.

$$Dist_Manhattan(p1, p2) = |x_1 - x_2| + |y_1 - y_2|$$

Questa funzione è un'euristica ammissibile ed è consistente (monotona), ovvero fa sì che il costo stimato per raggiungere l'obiettivo, diminuisca sempre avvicinandosi ad esso.

Per meglio adattarla al fatto che le azioni di movimento dell'agente hanno costi differenti (avanzare costa 10, mentre virare a destra o sinistra costa 15 unità di tempo), abbiamo pesato il calcolo dell'h-costo moltiplicandolo per 10, mantenendola così un'euristica ancora ammissibile e consistente, ma più vicina al "costo reale" dei futuri movimenti.

$$f(x) = g(x) + 10 \times h(x)$$

Per il calcolo dei movimenti già effettuati, abbiamo invece implementato la funzione $g(x)$ tramite un incremento di 10 o 15 unità a seconda dell'azione compiuta.

Capitolo 3

Scelte progettuali

In questo capitolo descriviamo le scelte che abbiamo preso durante la realizzazione dell'agente.

Caso per caso presentiamo le soluzioni che abbiamo adottato, le eventuali alternative che abbiamo scartato, motivando le scelte fatte.

Evidenziamo inoltre analogie e differenze di ogni variante della strategia che abbiamo implementato, mostrandone pregi e difetti.

3.1 Comportamento di base dell'agente

La misura di performance dell'agente è inversamente proporzionale alla quantità di penalità raccolta al termine dell'esplorazione.

Per questo motivo abbiamo sviluppato il progetto con l'obiettivo di ridurre al minimo i comportamenti che causano penalità.

Tra essi, i più dannosi corrispondono a:

- **far schiantare l'agente:** sebbene non comporti un aumento di penalità, questo coincide con la “morte” dell'agente e quindi, in un caso reale, corrisponderebbe al fallimento dell'intera missione;
- **non terminare la missione entro il tempo massimo:** questa azione causerebbe il massimo della penalità;
- **segnalare che l'esplorazione è finita al di fuori di un *gate*.**

Il verificarsi di questi comportamenti è totalmente scongiurato dal nostro agente tramite ragionamenti di tipo geometrico-temporale effettuati in fase di pianificazione.

La nostra strategia di pianificazione ci consente inoltre di evitare altre due fonti di penalità legate alle segnalazioni: il nostro agente non effettua infatti segnalazioni errate o ripetute.

3.1.1 La prima mossa

Uscita dal gate di partenza

Poiché la conoscenza dello stato corrente del mondo è completa solo per il simulatore mentre è inizialmente ignota all'agente, quest'ultimo potrà basare la sua esplorazione solo sulle informazioni contenute nella mappa iniziale o nella propria mappa, costituita dalle `kagent_cell`.

La prima attività svolta dall'agente è quindi la creazione di una sua "idea del mondo" tramite l'asserzione di fatti `kagent_cell`.

Sfruttando le informazioni note a priori circa il tipo di territorio che è identificato da una cella, l'agente vi attribuisce un grado di utilità.

Questi dati saranno alla base della scelta dei successivi movimenti da compiere.

A questo punto l'agente localizza tutti i `gate` presenti sulla mappa ed effettua un tentativo di pianificazione verso ognuno di essi.

Per ciascuno di questi tentativi che sia andato a buon fine, l'agente registra la cella `gate` obiettivo come *punto d'uscita garantito* (ovvero raggiungibile sia fisicamente che entro il tempo a disposizione). I `gate` che non sono risultati raggiungibili con questa prima pianificazione verranno automaticamente esclusi nei tentativi successivi.

Quindi, se è stato trovato almeno un *punto d'uscita garantito* (potrebbe essere anche il solo `gate` di partenza), l'agente può compiere la sua prima azione per uscire dal `gate`, dando il via alla missione.

Resa a priori

Se durante la prima fase di pianificazione dal punto di partenza verso i `gate` non è stato trovato alcun *punto d'uscita garantito*, oppure la cella di fronte a quella di partenza è occupata da una collina, la missione dell'agente termina con una **resa** anticipata, ovvero l'unica azione che verrà eseguita sarà una **done**.

3.1.2 Fase di esplorazione

Ogni attività svolta durante l'esplorazione è preceduta da fasi di controllo e pianificazione, che permettono all'agente di mantenersi sempre in uno stato *safe*. Con questa espressione ci riferiamo alle condizioni per cui l'agente si muove solo su celle da cui ha sempre la possibilità di raggiungere, entro il tempo rimanente, almeno un *punto d'uscita garantito*. Questa condizione è garantita inoltre da continui controlli ed aggiornamenti di fatti che rappresentano la conoscenza che l'agente ha del mondo che lo circonda.

Controllo della *safety* dell'azione successiva

L'agente considera eseguibile in maniera *safe* un'azione di movimento se questa lo porta su una cella che rispetta le seguenti condizioni:

- la cella non è un bordo o una collina (*reasoning geometrico*);
- a partire da quella cella, l'agente ha la possibilità di raggiungere almeno uno dei *punti d'uscita garantiti*.

La scelta di quale tra le azioni candidate compiere è caratteristica della variante della strategia implementata dall'agente.

Per quanto riguarda le azioni che non comportano movimenti (*inform* o *loiter-monitoring*), l'agente può eseguirle restando in stato *safe* se:

- un'azione identica non è già stata svolta sulla stessa cella;
- vi è il tempo sufficiente a compierle¹.

Il comportamento di base dell'agente prevede inoltre che le azioni di *inform* e *loiter-monitoring*, se attuabili, vengano eseguite prima di quelle di movimento.

Gestione delle percezioni e delle segnalazioni

Ad ogni passo, l'ambiente fornisce percezioni relative allo stato delle 9 celle situate in prossimità dell'agente.

L'UAV sa che dovrà fare rapporto alla stazione circa lo stato di ogni cella percepita, che questa risulti allagata o meno. In particolare, poiché una cella inondata causa più penalità di una "asciutta" ad ogni istante di tempo, la nostra implementazione fa sì che vengano effettuate segnalazioni dando la precedenza alle celle che sappiamo essere allagate rispetto alle altre.

Dopo questo primo turno di segnalazioni (che nel caso di celle non inondate ha bloccato la penalità relativa, mentre nel caso di celle inondate l'ha dimezzata), se l'agente si trova esattamente sopra ad una cella segnalata come genericamente allagata (e sussistono le condizioni per farlo) effettua un'azione di *loiter-monitoring*.

Questa azione permette di scoprire la gravità dell'allagamento e sarà quindi seguita, se il tempo a disposizione lo permette, da una seconda segnalazione, questa volta più accurata.

¹La verifica del tempo rimanente è un punto chiave della strategia, allo scopo di mantenere l'agente in stato *safe*: le azioni che non comportano movimenti perciò sono considerate eseguibili (in tempo) solo se, al termine di esse, l'agente ha ancora il tempo necessario a percorrere *almeno* la strada che lo separa dal *punto d'uscita garantito* più vicino.

Proprio il fatto di evitare di eseguire queste azioni permette all'agente di raggiungere sempre un *punto d'uscita*.

Esecuzione dell'azione

Una volta scelta quale sarà la prossima azione, in base alla strategia adottata, l'agente la porta a termine. A questo punto l'ambiente gli restituisce le nuove percezioni locali e sarà quindi compito dell'UAV interpretarle per aggiornare la propria conoscenza del mondo.

Dopo l'esecuzione di un'azione alcuni *punti d'uscita garantiti* potrebbero essere diventati non più raggiungibili, sia perché il tempo rimanente potrebbe non essere più sufficiente a raggiungerli, sia perché l'agente potrebbe aver imboccato un cammino che gli preclude la possibilità di uscire da quel *gate*.

Per tener traccia di queste possibilità, oltre al fatto che muovendosi potrebbe essersi avvicinato o allontanato da questi *gate*, l'agente deve quindi aggiornare i fatti in cui sono salvate le informazioni sui *punti d'uscita garantiti* e i cammini che gli consentirebbero di raggiungerli dalla posizione attuale.

Sempre a questo livello, l'agente seleziona quale tra questi cammini sarebbe il più veloce da percorrere: in questo modo si assicura di avere sempre almeno il tempo per giungere dal *punto d'uscita garantito* percorrendo quel cammino (*fase di fuga*).

3.1.3 Fase di fuga

Questa fase particolare si attiva nel momento in cui il tempo residuo a disposizione dell'agente non gli consente di compiere altre azioni *safe*, di conseguenza vengono sospese le segnalazioni e all'UAV non resta altro da fare che seguire il percorso più rapido verso un *gate* d'uscita.

Poiché il piano di fuga più rapido era già stato calcolato nei passi precedenti, l'agente si limiterà solo ad eseguirlo senza effettuare ulteriori pianificazioni.

La nostra implementazione prevede che non venga invece interrotta neppure in questa fase l'attività di ricezione e memorizzazione delle percezioni ambientali.

Tutte le strategie che abbiamo realizzato permettono all'agente di entrare in questa fase anche se sono presenti altre condizioni.

Queste verranno dettagliate in seguito, ma hanno in comune l'idea che l'agente decida di fuggire verso un *gate* quando ritiene di non poter più ricavare nuove informazioni sull'ambiente dall'esplorazione (fase di *stallo*), e perciò valuti come più conveniente concludere la missione per non rischiare di accumulare nuove penalità ad ogni passo, a causa di celle eventualmente non segnalate (e magari irraggiungibili).

3.1.4 Ritorno all'esplorazione (*switch*)

Come detto al punto 3.1.3, l'agente può decidere di fuggire anche avendo del tempo residuo. Questo tempo potrebbe però essere sfruttato per compiere altre segnalazioni se l'agente si trovasse in territori non ancora esplorati. Proprio per questo motivo, se durante l'esecuzione del piano di fuga l'agente si dovesse trovare a passare per zone non ancora visitate o segnalate, in alcune varianti abbiamo previsto la possibilità per l'agente di effettuare uno *switch* nel comportamento, ovvero di interrompere la fuga per tornare in modalità esplorativa.

3.2 Gestione dell'utilità

Nel dominio di questo progetto, il concetto di utilità è strettamente legato a quello della penalità. Per l'agente, visitare una zona è tanto più utile quanto più questa è portatrice di penalità: una cella da cui riceve tanta penalità gli risulterà quindi più utile poiché, segnalandola, riuscirà a ridurne i "danni" ricevuti.

Come già accennato in precedenza (paragrafo 3.1.1), l'agente attribuisce un valore di utilità ad ogni cella a seconda del tipo:

- poiché le zone di mappa relative a territori urbani o rurali, se allagati, causano una quantità di penalità maggiore rispetto a tutte le altre zone, ad esse viene assegnato un valore di utilità di partenza più alto rispetto alle altre: 80 (se cella **urban**) e 60 (se **rural**);
- non è previsto che l'agente effettui segnalazioni relative alle zone **border** e **hill**, inoltre esse non sono esplorabili: l'agente le ignora e la loro utilità è pari a 0;
- anche zone corrispondenti agli specchi d'acqua **lake** non necessitano di segnalazioni, però all'agente può essere utile attraversarle nel corso dell'esplorazione; per questo motivo gli è assegnata un'utilità di 20;
- per premiare la scelta esplorativa, l'accesso a celle di tipo **gate** dev'essere posposto il più possibile. Il valore assegnato a queste celle è 9: in questo modo l'agente vi potrà accedere, ma al contempo non risulteranno mai preferibili rispetto alle altre celle.

Il comportamento dell'agente in fase di scelta dell'azione successiva è dipendente dall'utilità della cella di destinazione e costituisce un punto di differenza tra le strategie implementate. Vi sono però degli aspetti comuni a tutte le varianti:

- una cella **urban** o **rural**, se percepita come genericamente allagata, subirà un incremento di utilità pari a 40 unità (**urban** diventa pari a 120 e **rural** a 100) e in questo modo alcune versioni dell'agente potranno essere "attratte" verso di esse;
- una cella per cui la fase di pianificazione non ha trovato alcun piano che consenta all'agente di raggiungere in sicurezza almeno un punto d'uscita, è considerata "vietata" e la sua utilità verrà annullata;
- una cella già visitata e/o segnalata verrà considerata in generale meno utile dall'agente:
 - il valore delle celle su cui l'agente ha già effettuato una segnalazione di "stato OK" (**inform-ok**²) viene ridotto a 20, per rendere l'idea che visitarle sia poco utile;
 - segnalare che una cella è genericamente allagata fa dimezzare la penalità che essa comporta ad ogni passo, ma non la annulla completamente; abbiamo perciò fatto in modo che l'agente continui a considerarla come ancora utile ed interessante da visitare, lasciandone inalterato il valore di utilità anche in seguito ad una **inform-water**;

²Ricordiamo che dopo che è stata effettuata correttamente un'azione di **inform-ok** su una cella, questa non causerà più alcuna penalità.

- le celle che l'agente ha visitato vengono marcate come “non più utili”, portandone il valore di utilità a 10;
- poiché per effettuare segnalazioni più precise sullo stato di allagamento di una cella è necessario effettuare un'azione di **loiter-monitoring** che non può prescindere dal visitare la cella stessa, il decremento relativo a queste azioni è già compiuto durante la fase di visita.

Quantificare la diminuzione di utilità negli altri casi sarà una scelta specifica di ogni variante della strategia.

3.3 Schema di funzionamento dell'agente

Nelle figure 3.1 e 3.2 abbiamo riassunto ad alto livello il funzionamento dell'agente.

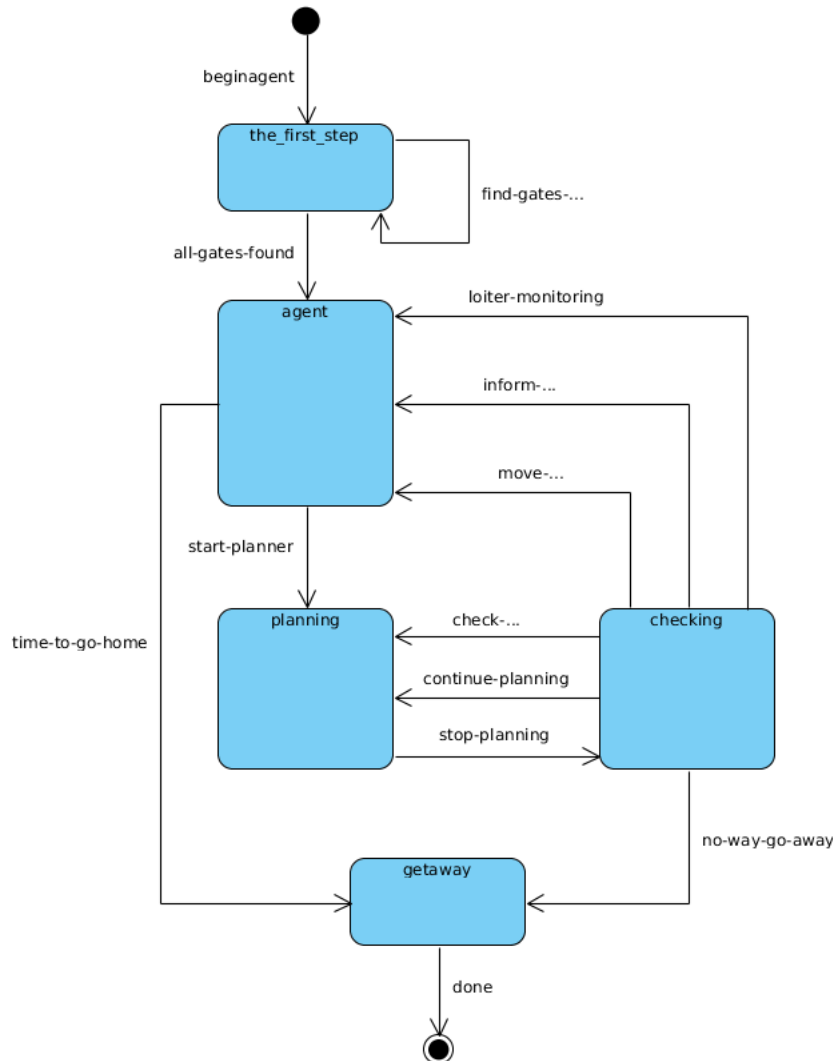


Figura 3.1: UML “State machine” dell’agente.

Lo stato **the_first_step** rappresenta i primi istanti della missione (descritti in dettaglio al punto 3.1.1) in cui l’agente si prepara ad uscire dal *gate* di partenza.

Localizzati tutti i *gate* ed almeno un *punto d’uscita garantito* (regola **all-gates-found**), l’agente inizia l’esplorazione.

Questa fase è costituita dall’alternarsi di stati di pianificazione (**planning**, implementato dal modulo **PLANNER**) e controllo (**checking** implementato nel modulo **CHECKER**).

Nello stato **checking** avvengono i controlli sulla *safety* delle future azioni, legati ai risultati ottenuti nello stato **planning**.

Se le condizioni sono verificate, si torna allo stato **agent** in cui l’azione selezionata verrà

asserita come eseguibile. L'esecuzione pratica dell'azione avverrà però nel modulo ENV.

Lo stato `getaway` rappresenta la situazione di fuga (vedere 3.1.3) e viene raggiunto se il tempo sta finendo (regola `time-to-go-home`) oppure se non ci sono più le condizioni per continuare l'esplorazione (regola `no-way-go-away`).

Il comportamento degli agenti che hanno la possibilità di effettuare *switch* è rappresentato nella figura 3.2: il ritorno dallo stato di `getaway` a quello di `checking` è causato dalla regola `switch-to-checking`.

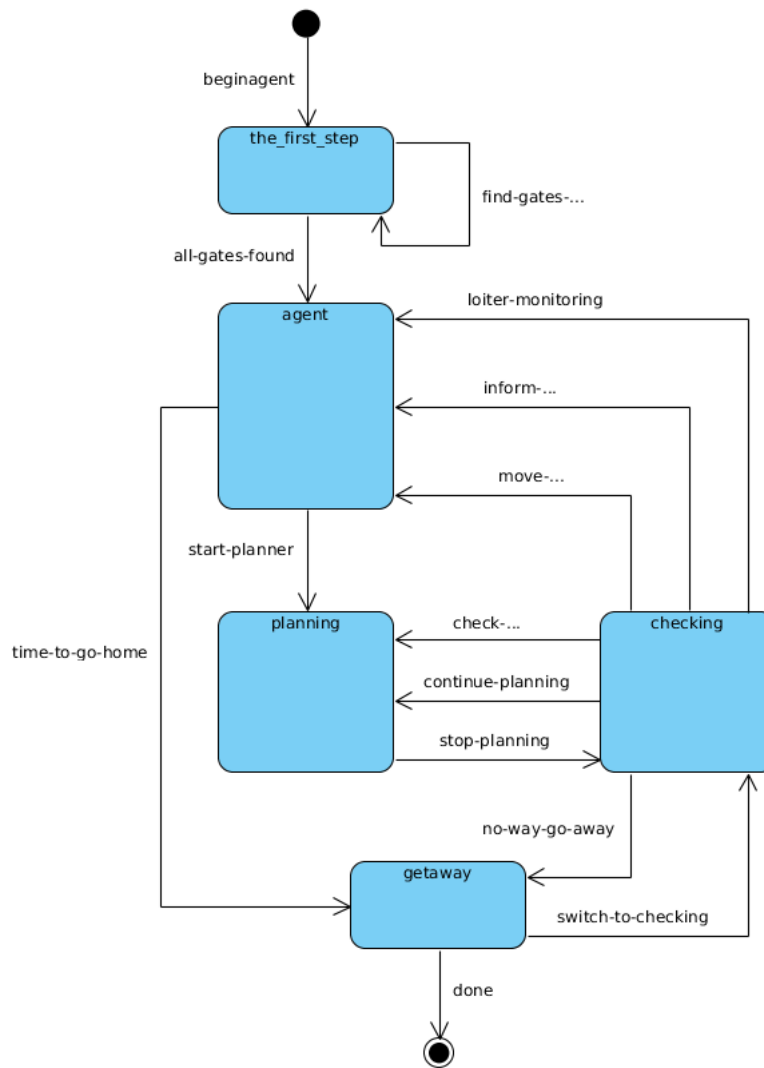


Figura 3.2: UML “State machine” dell’agente nel caso con *switch*.

I due schemi mostrati nelle figure precedenti sono alla base di tutte le strategie da noi implementate. Le differenze tra i comportamenti delle varie versioni degli agenti corrispondono agli stati `agent` (modulo AGENT) e `checking` (modulo CHECKER).

Capitolo 4

Strategie realizzate

No real designer attempts optimality in the sense of attaining perfect design.
Indeed, there is no such thing as perfect design. [3]

In questo capitolo sono riportate le descrizioni delle strategie che abbiamo realizzato. Per ognuna di esse descriviamo il comportamento che modellano e ne evidenziamo pregi e difetti.

Per quanto una strategia possa essere più o meno efficace ed efficiente, l'agente non può evitare alcune fonti di penalità.

Ad esempio esiste la possibilità che il mondo contenga celle non fisicamente raggiungibili: queste potrebbero essere comunque fonte di penalità e neppure un'esplorazione infinita permetterebbe di raggiungerle.

Preso atto di questa realtà, l'agente dovrebbe cercare di limitare la durata della propria missione (eventualmente terminandola in anticipo) qualora si accorgesse che le sue possibilità di movimento non andrebbero a migliorare ulteriormente la propria conoscenza del mondo, ma anzi aumenterebbero solo la penalità inflittagli.

4.1 “*Random walk*”

Questa prima strategia è stata nominata “*Random walk*” perché vuole simulare il comportamento di un agente che scelga a caso la prossima mossa da compiere¹.

Scelta della prossima azione e gestione dell'utilità

In questa strategia l'agente tenta di compiere come prima azione di movimento la **go-forward** poiché risulta meno dispendiosa in termini di tempo rispetto alle altre: se questa risulta non attuabile, tenta prima una **go-right** e quindi una **go-left**.

Se nessuna di queste azioni è utile a fini esplorativi, l'agente entra in fase di fuga: imbocca il cammino più rapido verso un **gate** e lo raggiunge, terminando la missione.

Questa strategia fa un uso molto rudimentale delle informazioni riguardanti l'utilità di una cella.

Impone infatti una soglia, al di sotto della quale una cella non viene più considerata percorribile in fase di esplorazione, ma solo in fase di fuga (utilità ≤ 10).

Di conseguenza tutte le celle risulteranno visitabili un'unica volta.

In ogni caso la scelta dell'azione successiva seguirà sempre, laddove possibile, lo schema *avanti-destra-sinistra*, senza essere influenzata dalla presenza o meno di allagamenti nelle celle circostanti.

Pro e contro

Questa strategia cerca di approfondire l'esplorazione sfruttando al massimo la possibilità di muoversi in linea retta, minimizzando così il tempo necessario per gli spostamenti.

Evitando di passare più volte sulle stesse celle, l'agente è in grado di scongiurare eventuali cicli. Il grosso svantaggio di questa soluzione è però che ad un certo punto dell'esplorazione l'agente potrebbe trovarsi circondato da celle già visitate e, non riuscendo più a compiere alcuna azione di movimento, entrerebbe direttamente in fase di fuga.

¹In realtà un vero agente *random* sceglierebbe tramite un'estrazione pseudocasuale la direzione in cui muoversi ma, poiché abbiamo considerato questa prima variante solo come base di partenza per lo sviluppo di tutte le altre strategie, la nostra implementazione non realizza un vero comportamento randomico.

4.2 “*Random walk*” con *switch*

Questa strategia è stata sviluppata con l’obiettivo di non terminare la missione troppo tempo prima dell’esaurirsi del tempo residuo.

Scelta della prossima azione e gestione dell’utilità

Quando l’agente si trova circondato da celle “non più utili”, entra in fase di fuga verso un gate ma, se durante il cammino verso il *punto d’uscita garantito* più conveniente incontra celle potenzialmente ancora utili (cioè con valore di utilità > 10), effettua uno *switch* del suo comportamento: torna cioè in fase di esplorazione, seguendo sempre lo schema “*random walk*” (vedere 3.1.4).

L’alternarsi di queste due fasi della vita dell’agente può continuare fino all’esaurirsi del tempo rimanente.

Pro e contro

La miglioria che abbiamo apportato al “*random walk*” è potenzialmente utile a ridurre la penalità totale al termine della missione.

Come qualsiasi scelta progettuale, i risultati reali sono fortemente influenzati dalla morfologia della mappa in cui l’agente si trova ad operare: non è garantito infatti che si verifichino le condizioni per cui lo *switch* possa scattare.

4.3 Prima implementazione di un agente *greedy*

Per sviluppare una strategia differente dal “*random walk*” abbiamo cercato di dotare l’agente di un meccanismo più sofisticato per la scelta del successivo passo di movimento. La prima implementazione che abbiamo prodotto introduce come sostanziale variazione la valutazione di tutti i movimenti potenzialmente effettuabili (al più 3, cioè avanti, destra o sinistra), prima di effettuare la scelta.

Questo consiste dunque nell’effettuare il passo in direzione della cella che risulta localmente più promettente in termini di utilità².

Scelta della prossima azione e gestione dell’utilità

Rispetto al caso del “*random walk*”, questo affinamento si traduce, a livello del modulo CHECKER, in un confronto tra i valori di utilità assegnati alle celle raggiungibili in un passo dall’agente (avanti, destra e sinistra).

Verrà quindi valutata la fattibilità³ di un movimento in direzione della cella di valore massimale; se questa non fosse garantita, la cella verrebbe marcata come “vietata” (utilità 0), si passerebbe ad analizzare la cella di valore immediatamente inferiore e così via.

In questa implementazione non c’è una vera soglia minima di utilità sotto la quale una cella non verrà visitata (è pari a 0): l’agente ha quindi la facoltà di visitare un numero indefinito di volte la stessa cella.

Pro e contro

Questa strategia corrisponde ad un comportamento maggiormente “*utility-based*” rispetto alle precedenti: l’agente è infatti attratto dalle celle allagate. Un tale approccio ha il vantaggio che, tra le celle locali, quelle che causano maggiore penalità saranno esplorate per prime, tentando così di ridurre il più possibile i “danni” da esse arrecati.

D’altro canto, la possibilità di esplorare un numero indefinito di volte la stessa cella potrebbe rivelarsi il tallone d’Achille di questa implementazione: a seconda della conformazione della mappa, l’agente potrebbe infatti compiere cammini ciclici.

In tal caso il numero di celle esplorate non progredirebbe e l’esplorazione sarebbe di fatto conclusa. Non avendo però modo di accorgersi di essere entrato in un ciclo infinito, l’UAV continuerebbe ad accumulare inutile penalità, laddove sarebbe preferibile scegliere di “tamponare i danni”, terminando in anticipo la missione raggiungendo un *gate*.

²Per disambiguare la scelta in caso di celle con pari valore di utilità, è stata mantenuta come base la priorità *avanti-destra-sinistra*

³Ricordiamo che una cella destinazione è considerata *safe* solo se da essa è possibile raggiungere, entro il tempo a disposizione, almeno un *punto d’uscita garantito*.

4.4 Agente *greedy*: 1 *chance*

Questa strategia è stata sviluppata con lo specifico obiettivo di impedire all'agente di compiere esplorazioni cicliche: ad ogni cella è infatti concessa una sola ed unica *chance* di essere visitata.

Scelta della prossima azione e gestione dell'utilità

Anche in questo caso la differenza dalla versione precedente risiede nelle regole definite nel modulo **CHECKER**: l'agente continua a preferire le celle con utilità maggiore, ma, dopo averle visitate una volta, le ritiene non più utili. La loro utilità viene infatti portata al pari della soglia minima (pari a 10) e non saranno più considerate visitabili.

Questa strategia torna quindi a fare uso della regola che individua una situazione di *stallo* per l'agente, in cui nessuna delle celle raggiungibili in un passo è in grado di risultare utile ai fini dell'esplorazione. In tal caso si entra in una fase (non interrompibile) di fuga verso il *gate* più vicino.

Pro e contro

Questa strategia è stata concepita come una variazione della precedente (4.3) e ne condivide i punti di forza.

La scelta di non concedere all'UAV la possibilità di visitare più di una volta la stessa cella scongiura sicuramente l'eventualità di ciclare indefinitamente, ma al contempo potrebbe condizionare le attività esplorative in maniera troppo limitante⁴.

⁴Similmente a quanto accadeva nel "caso base" della strategia *random walk* (4.1), l'agente potrebbe ritrovarsi circondato da celle non più utili, ed attivare di conseguenza la fase di fuga.

4.5 Agente *greedy*: 1 *chance* con *switch*

Il passo successivo è stato quello di dotare la strategia precedente della possibilità di tornare alla fase di esplorazione tramite lo *switch* (3.1.4).

Scelta della prossima azione e gestione dell'utilità

Come nei casi precedenti, la cella selezionata è quella a maggiore utilità.

Le celle già visitate non vengono più percorse in fase di esplorazione; una situazione di *stallo* attiva la fase di fuga, da cui però è possibile uscire tramite lo *switch*.

Pro e contro

Sebbene le condizioni di attivazione dello *switch* potrebbero non presentarsi, concedere questa possibilità all'agente non presenta svantaggi rispetto alla strategia 1 *chance* semplice (4.4).

4.6 Agente *greedy*: 2 chances

Una “variazione sul tema” che abbiamo previsto rispetto alla versione *greedy* 1 *chance* dell’agente (4.4) è costituita dal concedere all’UAV una seconda possibilità di visita di ogni cella.

Scelta della prossima azione e gestione dell’utilità

In questo caso la modifica a livello implementativo si è tradotta nell’introduzione di un ulteriore valore di utilità (15), intermedio tra quello iniziale (che indica una cella “nuova”) e quello finale (che indica una cella ormai “inutile”). Di conseguenza l’agente è in grado di distinguere celle visitate una o due volte.

Limitare in questo modo il passaggio su determinate celle, ripresenta, come nelle strategie precedenti, la possibilità di entrare in situazione di *stallo* e la conseguente attivazione della fase di fuga.

Pro e contro

Dare una seconda possibilità ad una cella può risultare utile in determinati scenari, in quanto può permettere di ampliare gli orizzonti esplorativi dell’UAV.

Il limite ai passaggi consentiti sulla stessa cella garantisce inoltre che gli eventuali percorsi ciclici saranno eseguiti al più due volte.

Parte II

La sperimentazione

Capitolo 5

Misura delle prestazioni

Come detto nel paragrafo 3.1, giudichiamo tanto migliore la prestazione dell'agente quanto più quest'ultimo è in grado di contenere la penalità raccolta durante la missione.

La scelta fondante di tutte le nostre strategie è stata quella di mettere in atto una serie di comportamenti volti a mantenere sempre l'agente in uno stato di *safety*.

Questa decisione ha reso necessario compiere, prima di ogni passo, molteplici attività di pianificazione che fanno uso dell'algoritmo A^* (come enunciato in 2.5), il quale, sotto opportune condizioni, ha il pregio di garantire completezza, ottimalità e di essere ottimamente efficiente.

Nonostante ciò, un'eccessiva dimensione della mappa (e di conseguenza dello spazio di ricerca) può rendere **intrattabile** il problema, poiché dilaterrebbe eccessivamente le richieste di memoria e tempo necessari per giungere ad una soluzione di una **singola esecuzione** della ricerca.

Questa possibilità ha reso necessario effettuare *tuning* sulla dimensione della mappa, per trovarne un limite superiore che garantisca tempi di esecuzione accettabili.

Inoltre, anche se il costo computazionale di una singola fase di pianificazione è dominato da quello dell'esecuzione di A^* , il numero di obiettivi verso cui effettuare pianificazione fa chiaramente aumentare in maniera lineare questi costi.

Questo risultato non può essere ignorato nella pratica e ha reso necessario imporre un limite alla quantità di **gate** che possono essere contemporaneamente presenti sulla mappa.

Possiamo quindi ragionevolmente considerare accettabili i tempi di esecuzione su mappe con un numero di celle non superiore a 400 e che non contengano più di 5 **gate**¹.

¹Si può notare che la nostra implementazione ha un interessante *side effect*: il numero di esecuzioni di A^* in ogni fase di pianificazione decresce monotonicamente nel tempo.

Questo risultato è dovuto a 2 fattori:

- gli eventuali **gate** non raggiungibili a priori saranno esclusi dalle successive pianificazioni;
- i movimenti dell'agente potrebbero portarlo ad allontanarsi così tanto da un **gate** da renderlo irraggiungibile geometricamente o entro il tempo a disposizione: tale **gate** pertanto non sarà più considerato un *goal* valido per le future ricerche.

Più formalmente, se un **gate** non è raggiungibile nello stato S_i , continuerà a non essere raggiungibile in qualsiasi stato S_j con $j \geq i$.

Capitolo 6

Test effettuati e risultati ottenuti

Nelle pagine seguenti presentiamo in forma grafica i risultati dei test effettuati.

Abbiamo previsto test su **3 mappe**, ciascuna con caratteristiche differenti:

- la mappa fornita con il testo del progetto, di dimensioni 10×11 (5 *gate*), a cui faremo riferimento nel seguito indicandola come “mappa default”;
- una mappa generata in maniera casuale, di dimensioni 15×15 (4 *gate*), a cui faremo riferimento nel seguito indicandola come “mappa random”¹;
- una mappa generata manualmente in cui gli allagamenti interessano solo le zone vicino agli specchi d’acqua², di dimensioni 18×18 (2 *gate*). Poiché ci siamo ispirati alla morfologia della città di Torino, faremo riferimento ad essa come “mappa Torino”.

Per velocizzare le simulazioni abbiamo scelto di mantenerci leggermente al di sotto dei **limiti dimensionali** discussi nel capitolo 5.

Per ogni mappa abbiamo verificato il comportamento di tutte le strategie implementate, analizzandone nel dettaglio tempi di esecuzione, penalità e numero di celle segnalate.

Per evidenziare quanto il **tempo** influisca sulle prestazioni dell’agente, ogni strategia è stata eseguita in 2 casi:

- con un limite di 1000 unità di tempo;
- con un limite di 6000 unità di tempo, per simulare un tempo “infinito”.

¹Le nostre strategie sono state sviluppate con l’ambizione di riuscire a gestire ambienti randomici, ovvero mappe nelle quali le celle allagate non seguono un particolare ordine o schema.

²Tra le scelte implementative che abbiamo scartato era contemplata la possibilità di realizzare un agente che fosse invogliato ad esplorare zone di mappa vicine agli specchi d’acqua, per simulare una situazione di allagamento reale. Tramite i test verificheremo se il nostro agente sarà in grado di esplorare efficacemente un tale ambiente.

Per interpretare le mappe delle pagine a seguire, sono fornite le figure 6.1 e 6.2.






	bordo
	cella non portatrice di penalità
	cella segnalata
	cella non segnalabile
	cella non segnalata ma segnalabile

Figura 6.1: Legenda per i risultati dei test











	cella di tipo border
	cella di tipo gate
	cella di tipo hill
	cella di tipo lake
	cella di tipo rural
	cella di tipo rural , allagamento iniziale
	cella di tipo rural , allagamento grave
	cella di tipo urban
	cella di tipo urban , allagamento iniziale
	cella di tipo urban , allagamento grave

Figura 6.2: Legenda della mappa

6.1 Mappa default

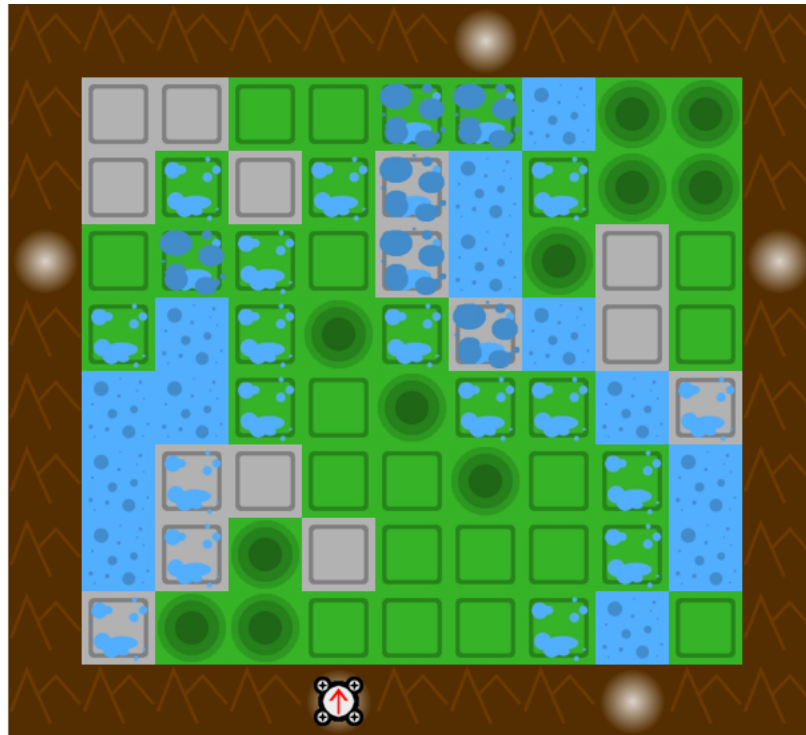


Figura 6.3: Mappa default, stato iniziale

In questa mappa sono presenti 72 celle, di cui 48 segnalabili e 24 non portatrici di penalità.

6.1.1 Agente “random walk”

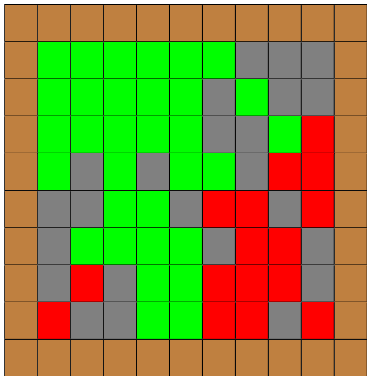


Tabella 6.1: Mappa default, random walk, tempo 1000

L’agente segnala 32/48 celle (67%) in tempo 973, raccogliendo una penalità di 8’872’852.

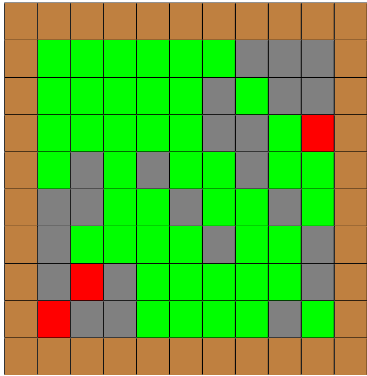


Tabella 6.2: Mappa default, random walk, tempo 6000

L’agente segnala 45/48 celle (94%) in tempo 1472, raccogliendo una penalità di 2’187’758.

6.1.2 Agente “random walk” con *switch*

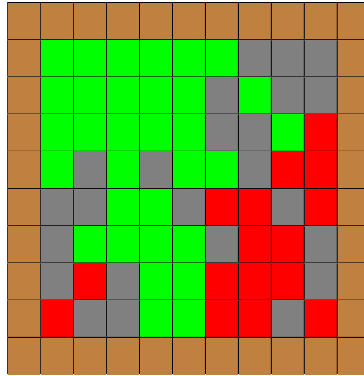


Tabella 6.3: Mappa default, random walk con switch, tempo 1000

L’agente segnala 32/48 celle (67%) in tempo 973, raccogliendo una penalità di 8’872’852.

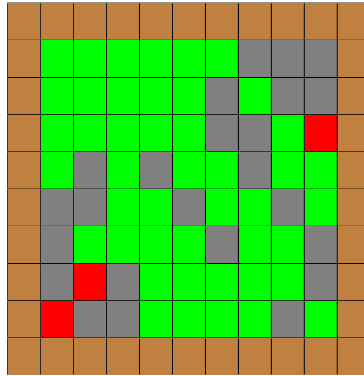


Tabella 6.4: Mappa default, random walk con switch, tempo 6000

L’agente segnala 45/48 celle (94%) in tempo 1472, raccogliendo una penalità di 2’187’758.

6.1.3 Primo agente greedy

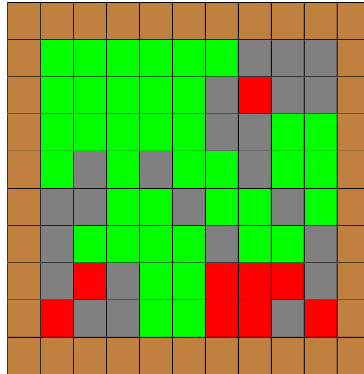


Tabella 6.5: Mappa default, primo agente greedy, tempo 1000

L'agente segnala 39/48 celle (81%) in tempo 997, raccogliendo una penalità di 5'469'554.

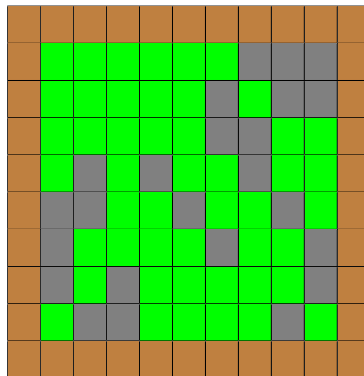


Tabella 6.6: Mappa default, primo agente greedy, tempo 6000

L'agente segnala 48/48 celle (100%) in tempo 5975, raccogliendo una penalità di 95'759.

6.1.4 Agente greedy 1 chance

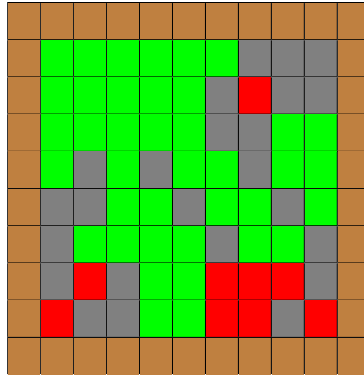


Tabella 6.7: Mappa default, agente greedy 1 chance, tempo 1000

L'agente segnala 39/48 celle (81%) in tempo 997, raccogliendo una penalità di 5'469'554.

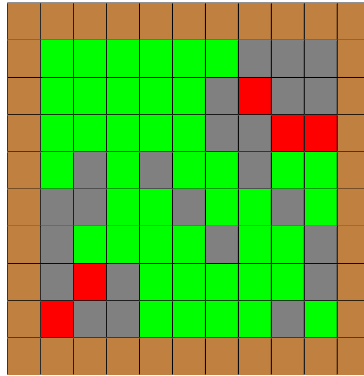


Tabella 6.8: Mappa default, agente greedy 1 chance, tempo 6000

L'agente segnala 43/48 celle (90%) in tempo 1279, raccogliendo una penalità di 3'276'769.

6.1.5 Agente greedy 1 chance con switch

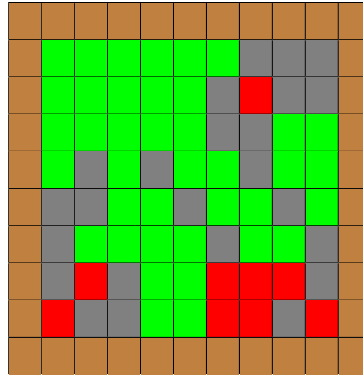


Tabella 6.9: Mappa default, agente greedy 1 chance con switch, tempo 1000

L'agente segnala 39/48 celle (81%) in tempo 997, raccogliendo una penalità di 5'469'554.

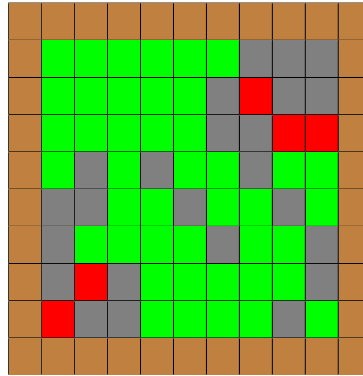


Tabella 6.10: Mappa default, agente greedy 1 chance con switch, tempo 6000

L'agente segnala 43/48 celle (90%) in tempo 1279, raccogliendo una penalità di 3'276'769.

6.1.6 Agente greedy 2 chances

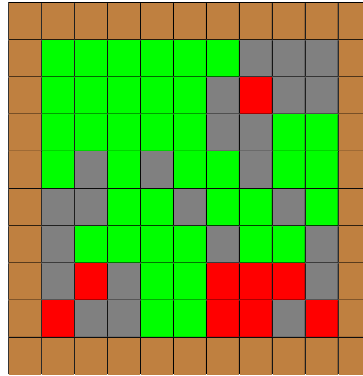


Tabella 6.11: Mappa default, agente greedy 2 chances, tempo 1000

L'agente segnala 39/48 celle (81%) in tempo 997, raccogliendo una penalità di 5'469'554.

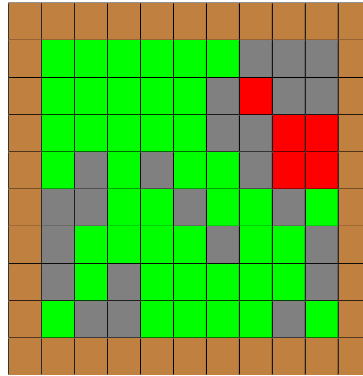


Tabella 6.12: Mappa default, agente greedy 2 chances, tempo 6000

L'agente segnala 43/48 celle (90%) in tempo 1850, raccogliendo una penalità di 1'282'944.

6.1.7 Confronto prestazionale tra le strategie implementate

Nei grafici seguenti usiamo:

- **RW** per indicare l'agente “*random walk*”;
- **RWS** per “*random walk*” con *switch*;
- **G** per il primo agente *greedy*;
- **G1C** per l'agente *greedy 1 chance*;
- **G1CS** per l'agente *greedy 1 chance* con *switch*;
- **G2C** per l'agente *greedy 2 chances*.

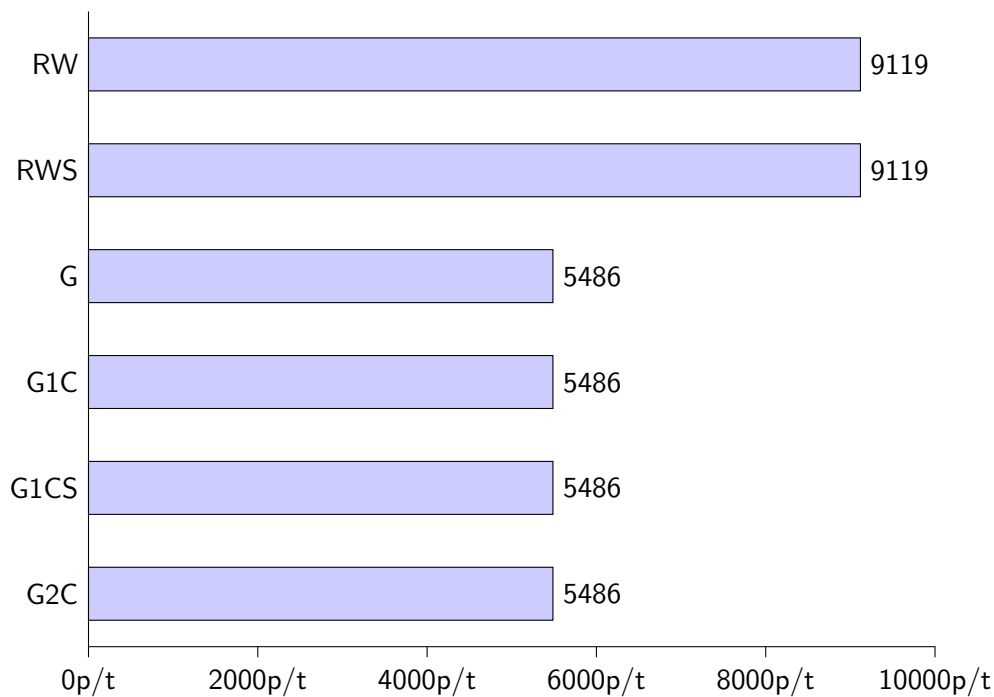


Figura 6.4: Confronto performance penalità/tempo usato su mappa di default, con tempo 1000.

In questo grafico un valore alto corrisponde a una bassa *performance* in quanto riflette il carico di penalità raccolto per ogni istante di tempo trascorso.

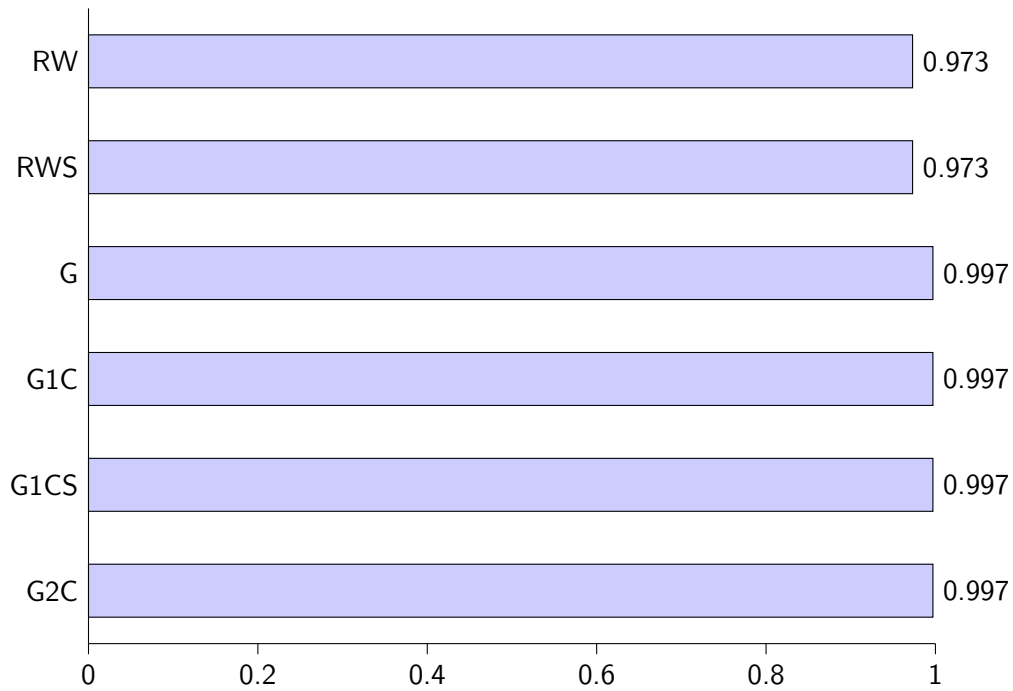


Figura 6.5: Confronto performance tempo usato/tempo totale su mappa di default, con tempo 1000.

Questo grafico mette in luce quanto l'agente ha sfruttato il tempo a sua disposizione. Sebbene in linea teorica un valore alto potrebbe rappresentare una buona esplorazione (e così è in molti casi), aver usato molto tempo potrebbe anche corrispondere alla situazione in cui l'agente ha imboccato un cammino ciclico.

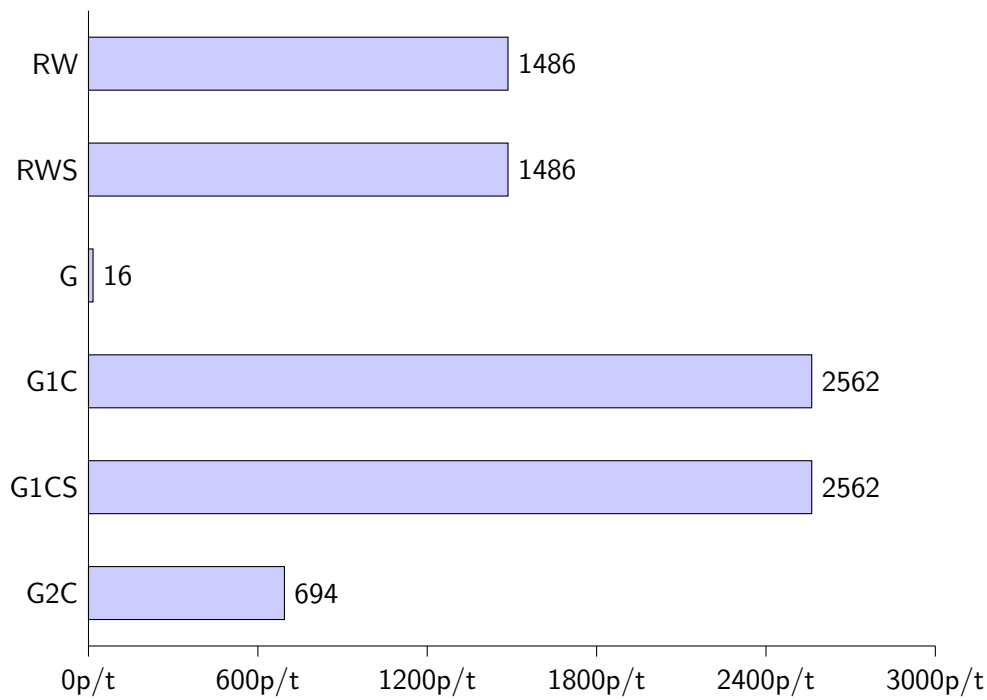


Figura 6.6: Confronto performance penalità/tempo usato su mappa di default, con tempo 6000.

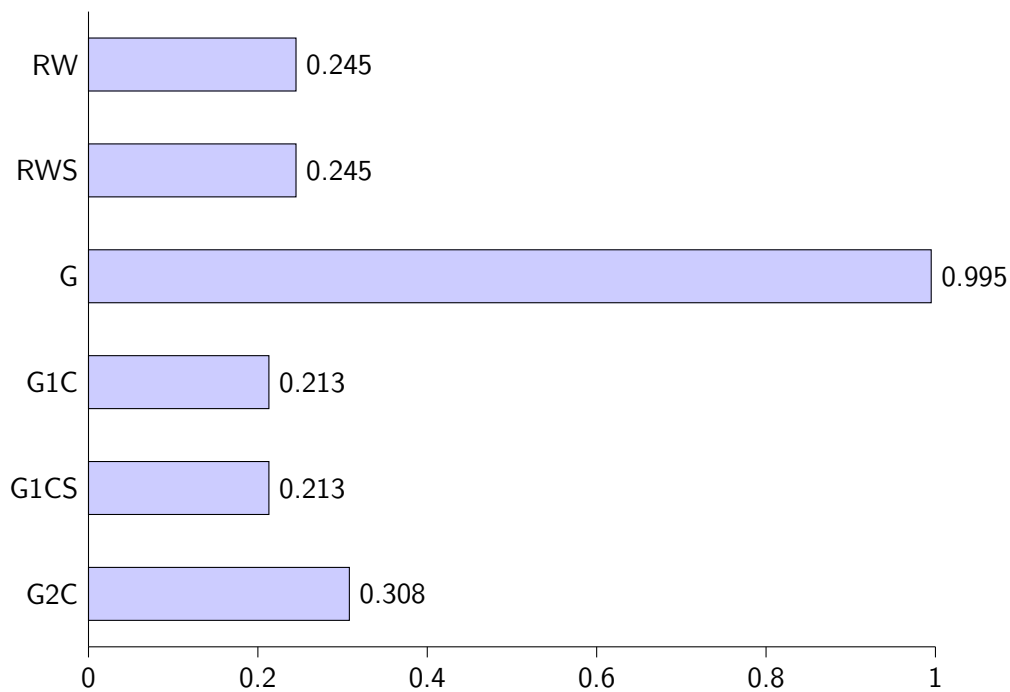


Figura 6.7: Confronto performance tempo usato/tempo totale su mappa di default, con tempo 6000.

6.2 Mappa generata randomicamente



Figura 6.8: Mappa random, stato iniziale

In questa mappa sono presenti 169 celle, di cui 81 segnalabili, 4 non segnalabili e 84 non portatrici di penalità.

6.2.1 Agente “random walk”

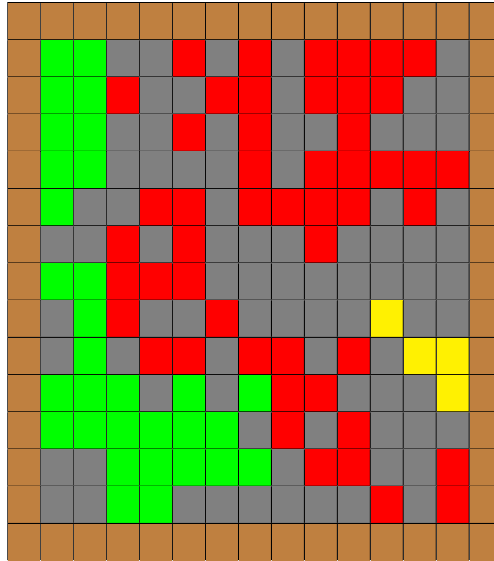


Tabella 6.13: Mappa random, random walk, tempo 1000

L'agente segnala 31/81 celle (38%) in tempo 791, raccogliendo una penalità di 45'204'421.

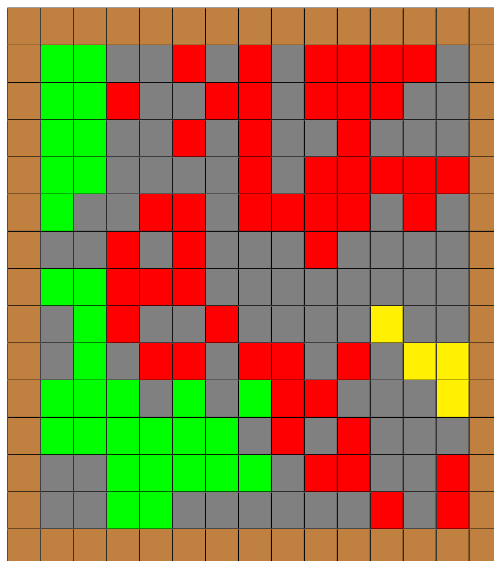


Tabella 6.14: Mappa random, random walk, tempo 6000

L'agente segnala 31/81 celle (38%) in tempo 791, raccogliendo una penalità di 45'204'421.

6.2.2 Agente “random walk” con switch

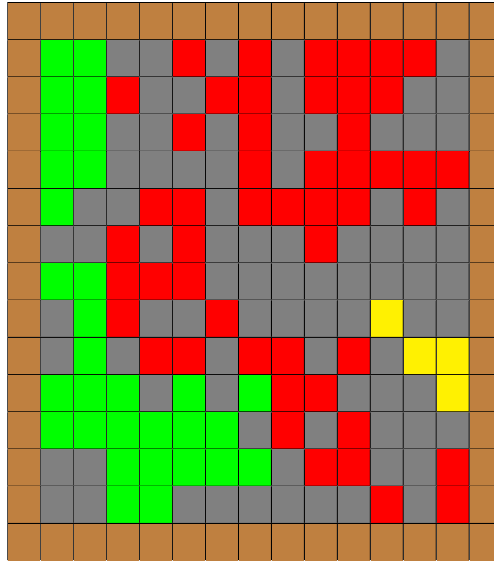


Tabella 6.15: Mappa random, random walk con switch, tempo 1000

L'agente segnala 31/81 celle (38%) in tempo 791, raccogliendo una penalità di 45'204'421.

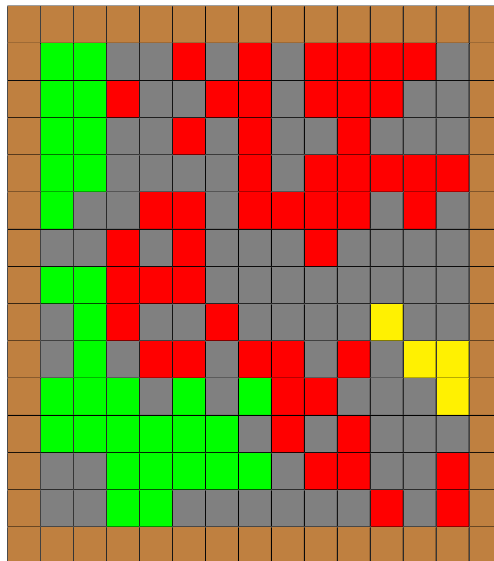


Tabella 6.16: Mappa random, random walk con switch, tempo 6000

L'agente segnala 31/81 celle (38%) in tempo 791, raccogliendo una penalità di 45'204'421.

6.2.3 Primo agente greedy

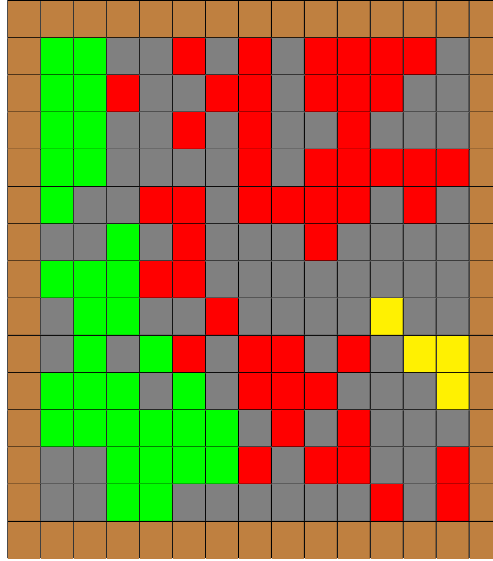


Tabella 6.17: Mappa random, primo agente greedy, tempo 1000

L'agente segnala 33/81 celle (41%) in tempo 996, raccogliendo una penalità di 45'047'303.

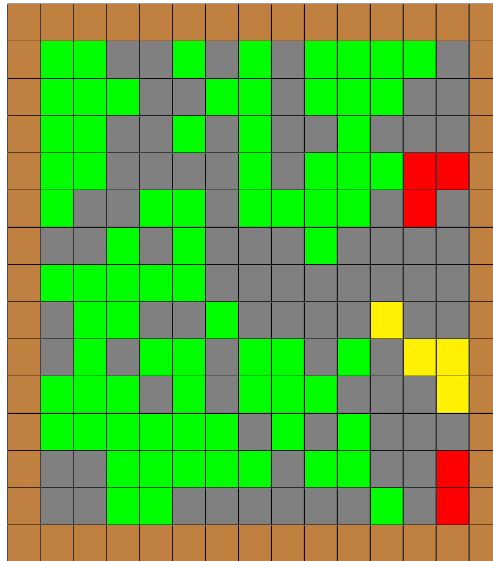


Tabella 6.18: Mappa random, primo agente greedy, tempo 6000

L'agente segnala 76/81 celle (94%) in tempo 5988, raccogliendo una penalità di 7'900'219.

6.2.4 Agente greedy 1 chance

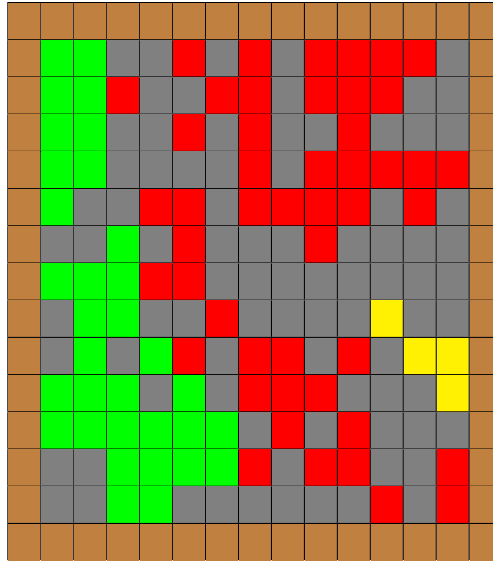


Tabella 6.19: Mappa random, agente greedy 1 chance, tempo 1000

L'agente segnala 33/81 celle (41%) in tempo 996, raccogliendo una penalità di 45'047'303.

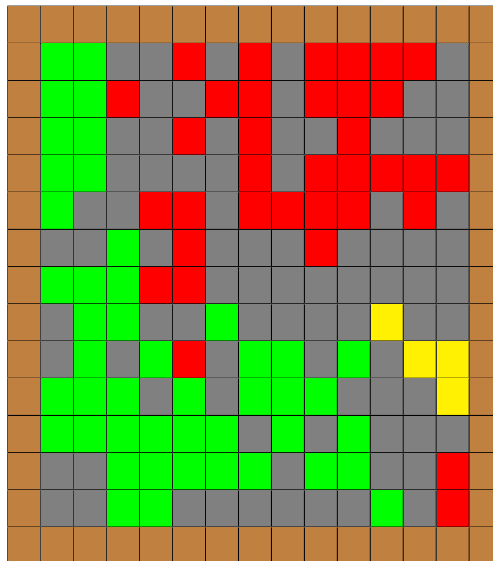


Tabella 6.20: Mappa random, agente greedy 1 chance, tempo 6000

L'agente segnala 46/81 celle (57%) in tempo 1780, raccogliendo una penalità di 33'086'468.

6.2.5 Agente greedy 1 chance con switch

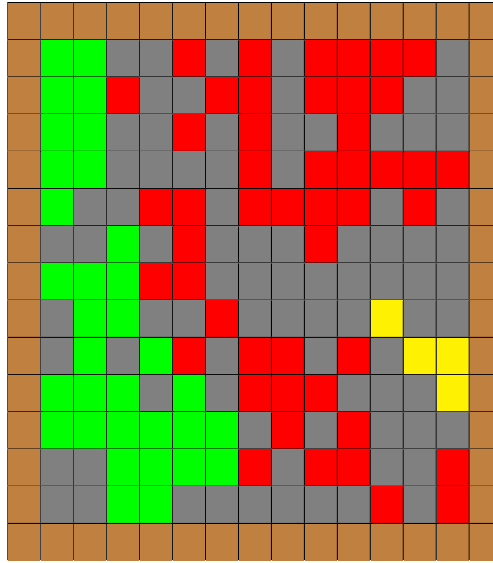


Tabella 6.21: Mappa random, agente greedy 1 chance con switch, tempo 1000

L'agente segnala 33/81 celle (41%) in tempo 996, raccogliendo una penalità di 45'047'303.

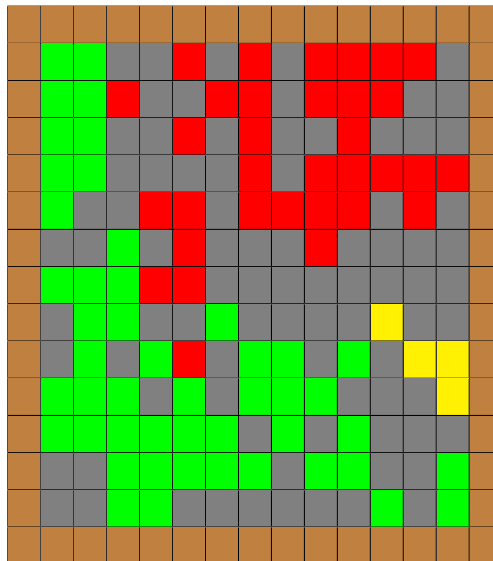


Tabella 6.22: Mappa random, agente greedy 1 chance con switch, tempo 6000

L'agente segnala 48/81 celle (59%) in tempo 1833, raccogliendo una penalità di 31'994'777.

6.2.6 Agente greedy 2 chances

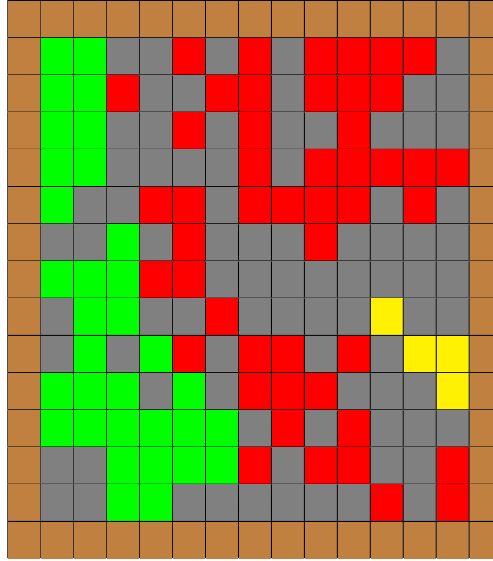


Tabella 6.23: Mappa random, agente greedy 2 chance, tempo 1000

L'agente segnala 33/81 celle (41%) in tempo 996, raccogliendo una penalità di 45'047'303.

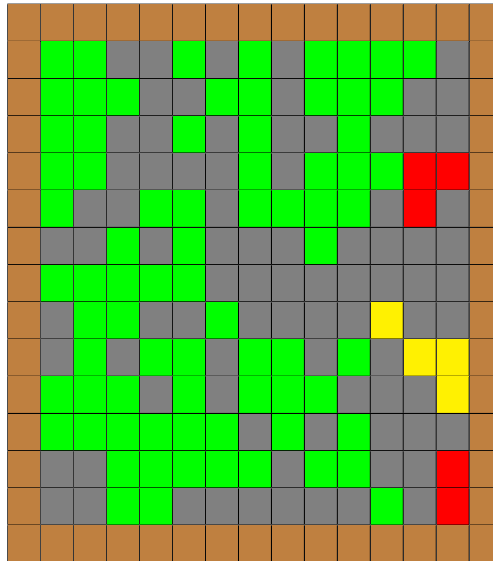


Tabella 6.24: Mappa random, agente greedy 2 chance, tempo 6000

L'agente segnala 76/81 celle (94%) in tempo 4188, raccogliendo una penalità di 7'824'619.

6.2.7 Confronto prestazionale tra le strategie implementate

Per il significato dei grafici facciamo riferimento a quanto già detto in 6.1.7.

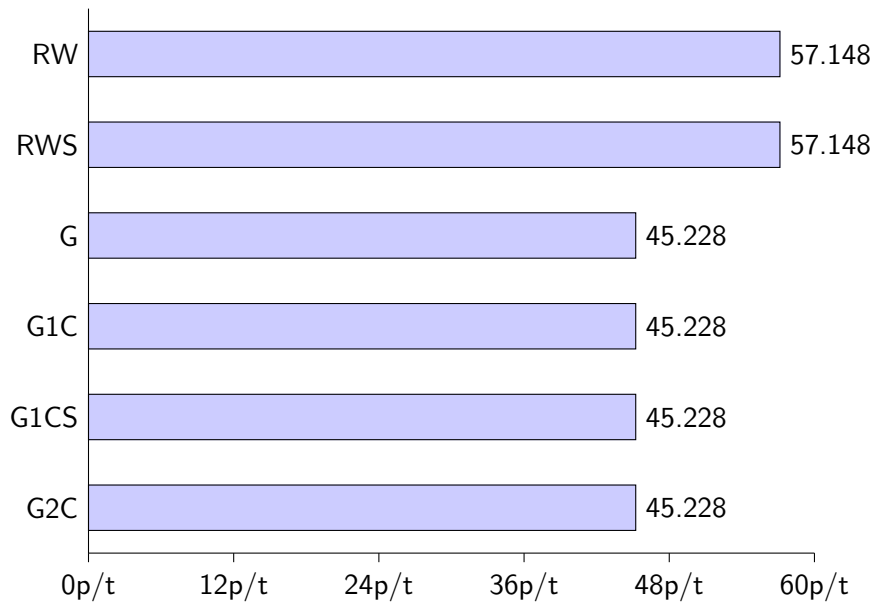


Figura 6.9: Confronto performance penalità/tempo usato su mappa random, con tempo 1000. I valori sono intesi $\times 1000$.

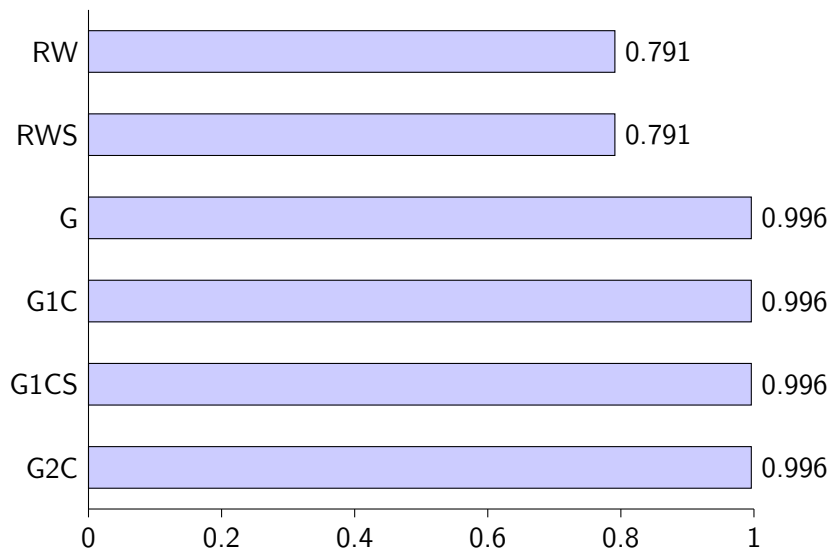


Figura 6.10: Confronto performance tempo usato/tempo totale su mappa random, con tempo 1000.

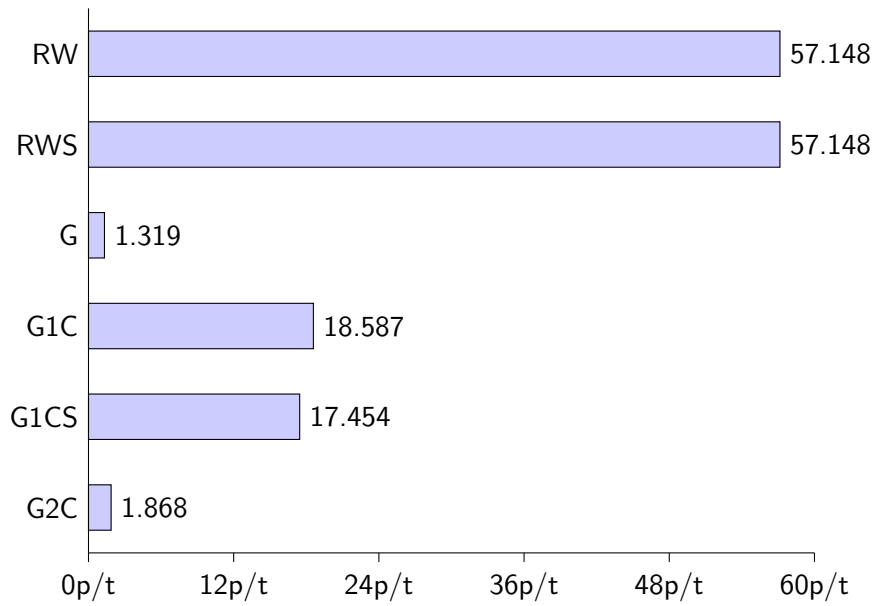


Figura 6.11: Confronto performance penalità/tempo usato su mappa random, con tempo 6000.
I valori sono intesi $\times 1000$.

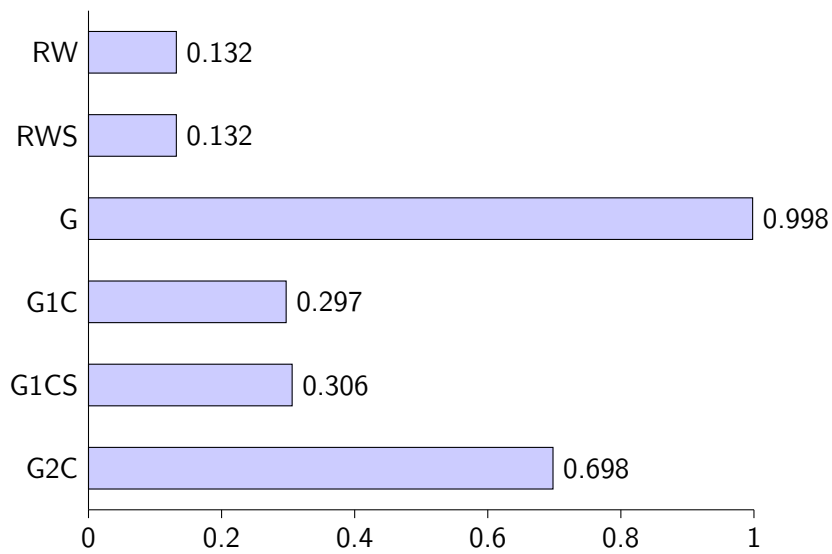


Figura 6.12: Confronto performance tempo usato/tempo totale su mappa random, con tempo 6000.

6.3 Mappa “realistica”: Torino



Figura 6.13: Mappa di Torino, stato iniziale

In questa mappa sono presenti 256 celle, di cui 149 segnalabili e 107 non portatrici di penalità.

6.3.1 Agente “random walk”

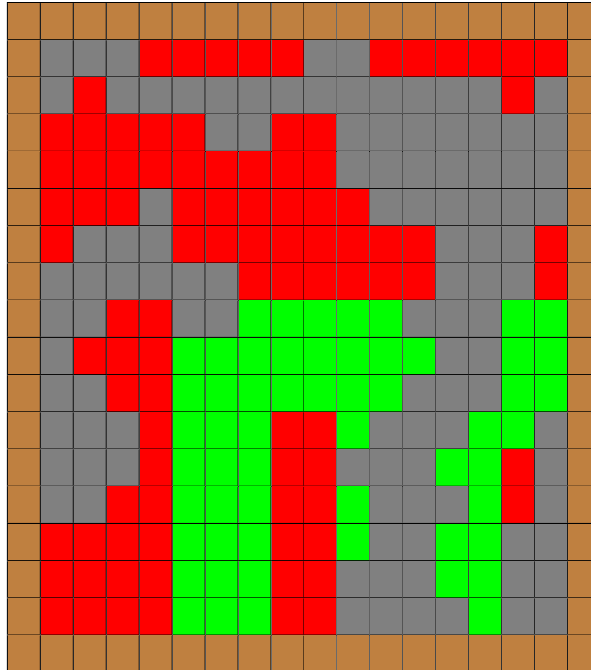


Tabella 6.25: Mappa Torino, random walk, tempo 1000

L'agente segnala 57/149 celle (38%) in tempo 975, raccogliendo una penalità di 38'297'445.

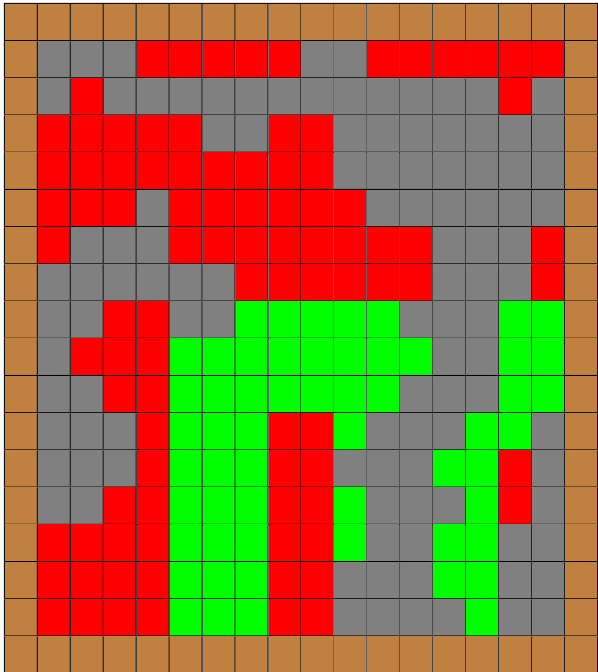


Tabella 6.26: Mappa Torino, random walk, tempo 6000

L'agente segnala 57/149 celle (38%) in tempo 1005, raccogliendo una penalità di 38'305'035.

6.3.2 Agente “random walk” con switch

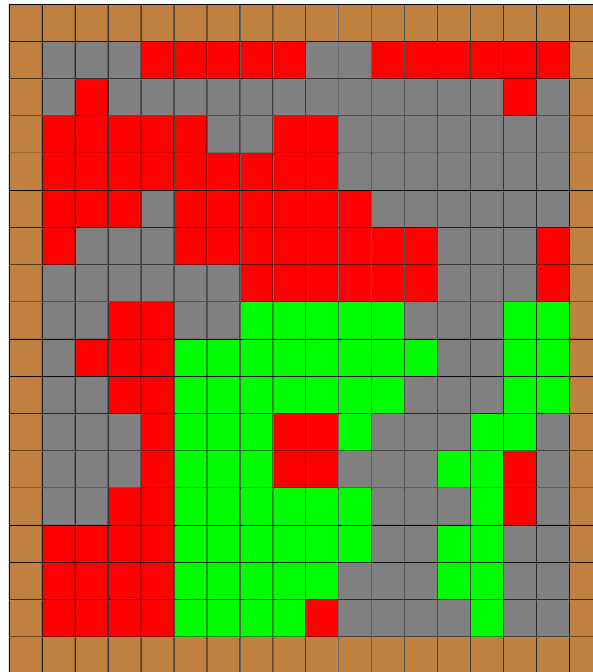


Tabella 6.27: Mappa Torino, random walk con switch, tempo 1000

L'agente segnala 64/149 celle (43%) in tempo 992, raccogliendo una penalità di 35'800'826.

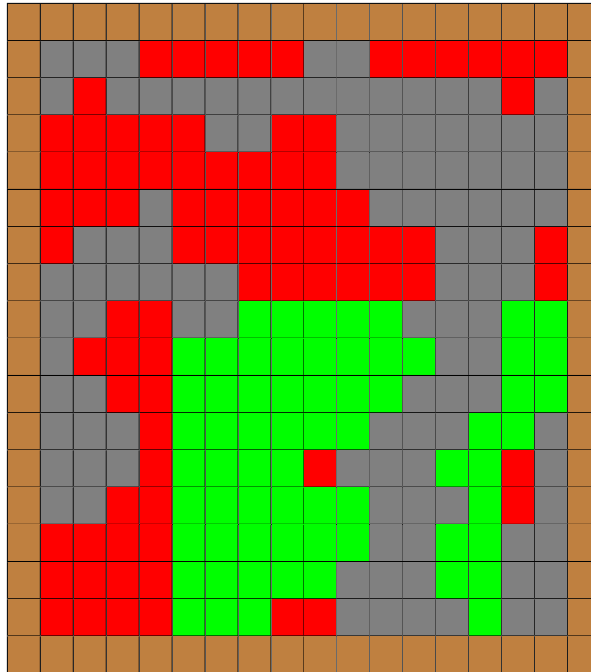


Tabella 6.28: Mappa Torino, random walk con switch, tempo 6000

L'agente segnala 66/149 celle (44%) in tempo 1367, raccogliendo una penalità di 35'689'455.

6.3.3 Primo agente greedy

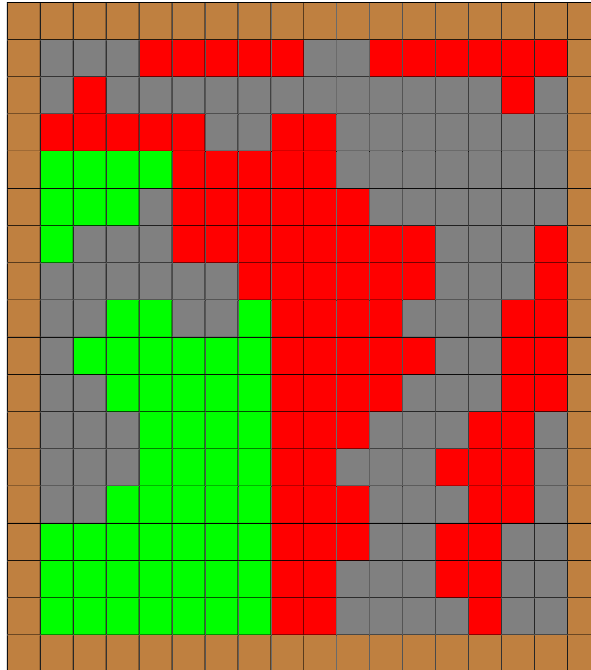


Tabella 6.29: Mappa Torino, primo agente greedy, tempo 1000

L'agente segnala 56/149 celle (38%) in tempo 976, raccogliendo una penalità di 47'427'475.

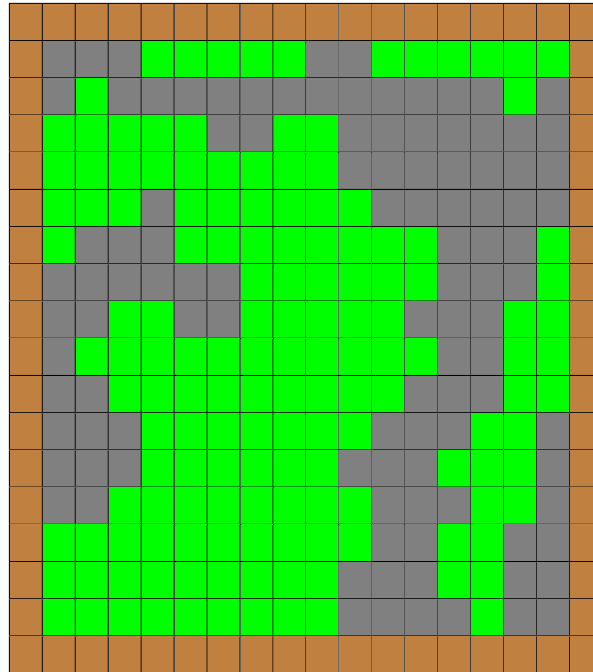


Tabella 6.30: Mappa Torino, primo agente greedy, tempo 6000

L'agente segnala 149/149 celle (100%) in tempo 5994, raccogliendo una penalità di 994'912.

6.3.4 Agente greedy 1 chance

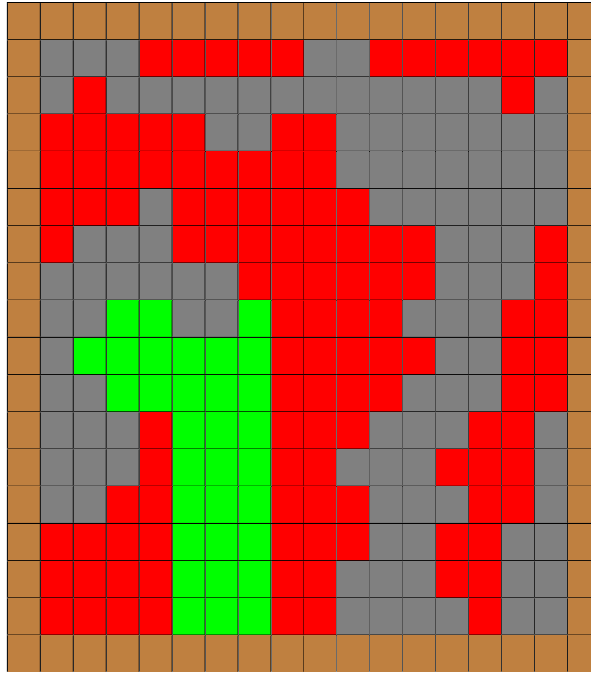


Tabella 6.31: Mappa Torino, agente greedy 1 chance, tempo 1000

L'agente segnala 32/149 celle (22%) in tempo 592, raccogliendo una penalità di 53'306'399.

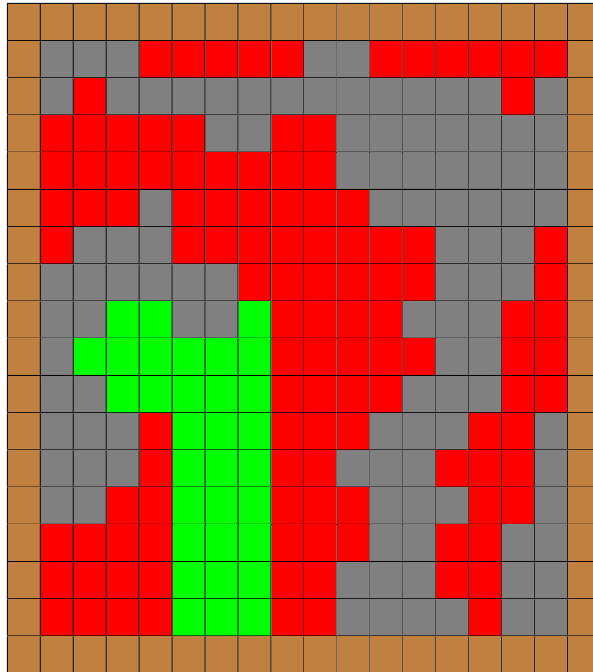


Tabella 6.32: Mappa Torino, agente greedy 1 chance, tempo 6000

L'agente segnala 32/149 celle (22%) in tempo 592, raccogliendo una penalità di 53'306'399.

6.3.5 Agente greedy 1 chance con switch

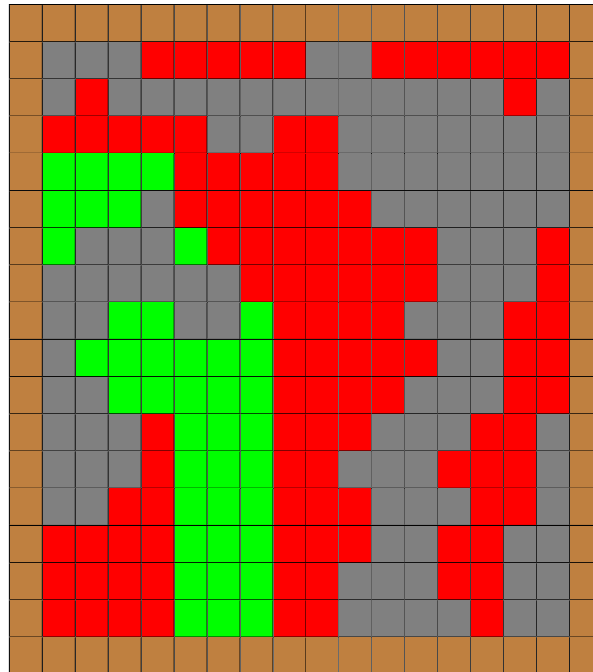


Tabella 6.33: Mappa Torino, agente greedy 1 chance con switch, tempo 1000

L'agente segnala 41/149 celle (28%) in tempo 995, raccogliendo una penalità di 48'934'698.

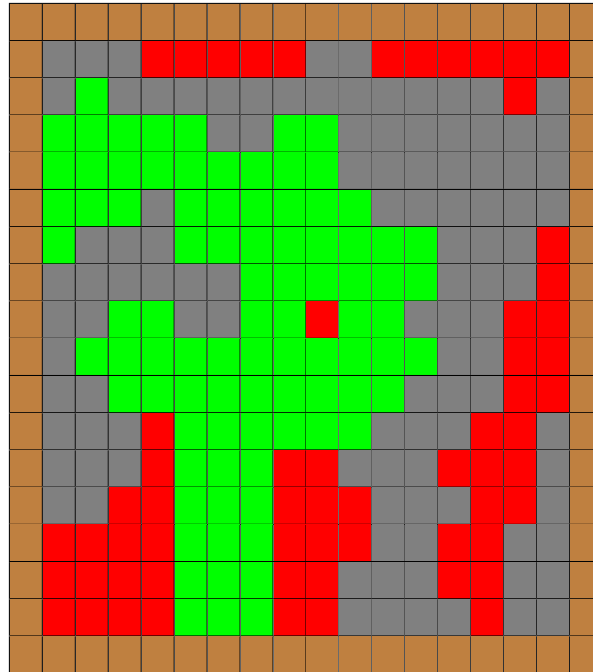


Tabella 6.34: Mappa Torino, agente greedy 1 chance con switch, tempo 6000

L'agente segnala 88/149 celle (59%) in tempo 2013, raccogliendo una penalità di 31'886'413.

6.3.6 Agente greedy 2 chances

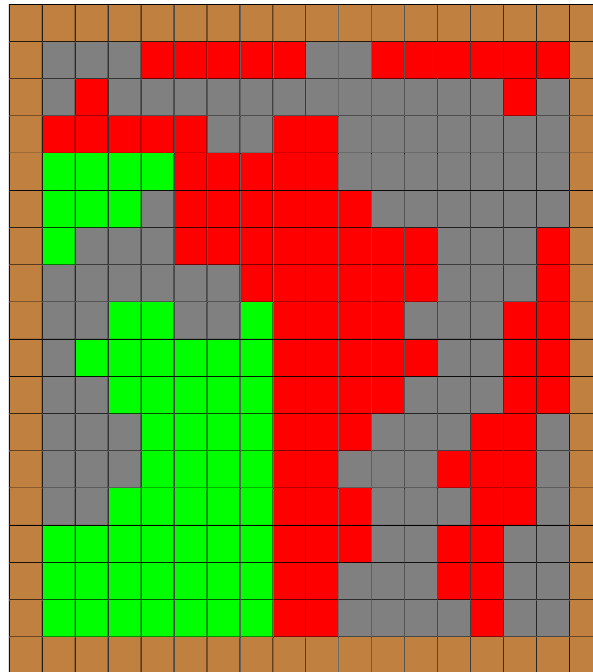


Tabella 6.35: Mappa Torino, agente greedy 2 chances, tempo 1000

L'agente segnala 56/149 celle (38%) in tempo 976, raccogliendo una penalità di 47'427'475.

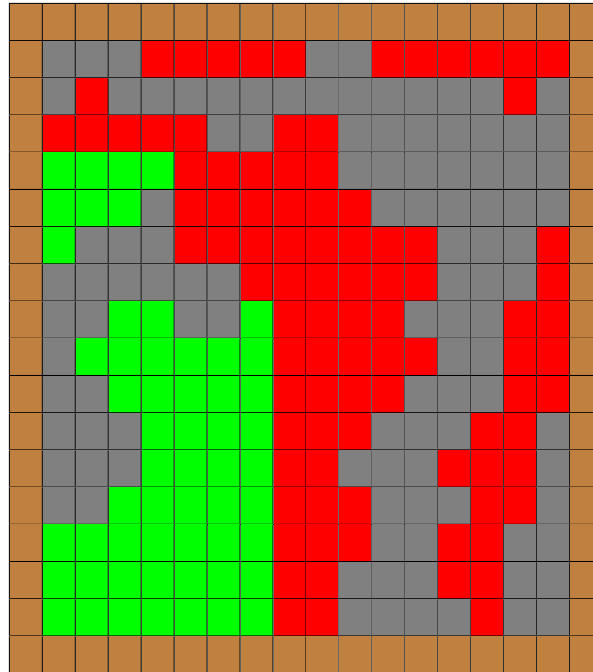


Tabella 6.36: Mappa Torino, agente greedy 2 chances, tempo 6000

L'agente segnala 56/149 celle (38%) in tempo 1310, raccogliendo una penalità di 47'528'578.

6.3.7 Confronto prestazionale tra le strategie implementate

Per il significato dei grafici facciamo riferimento a quanto già detto in 6.1.7.

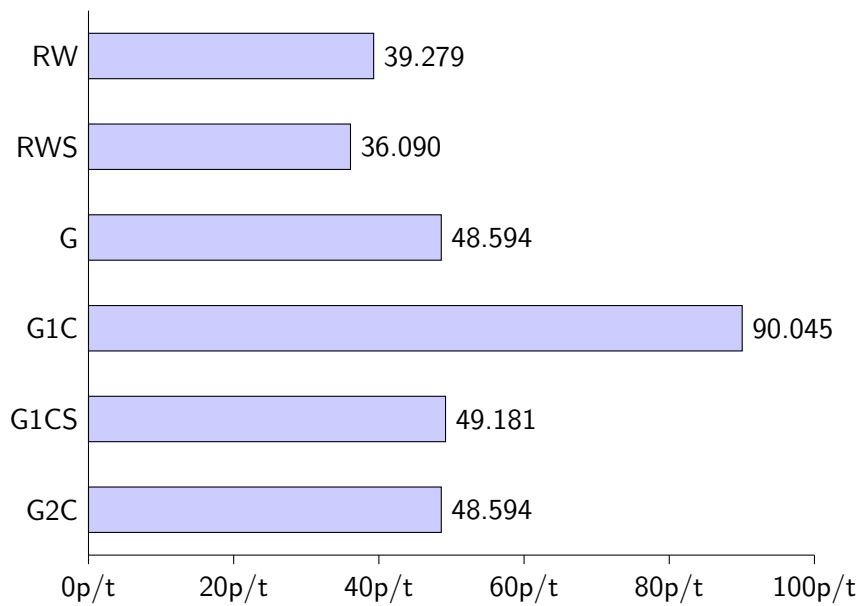


Figura 6.14: Confronto performance penalità/tempo usato su mappa di Torino, con tempo 1000. I valori sono intesi $\times 1000$.

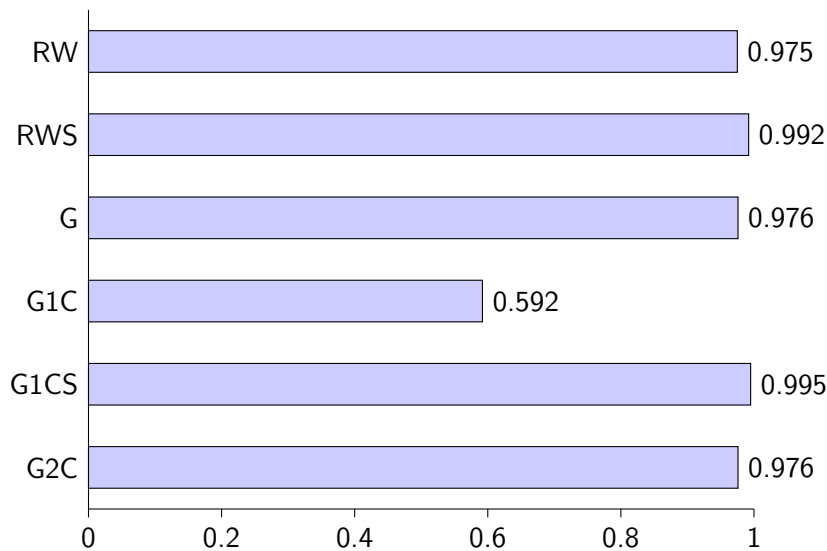


Figura 6.15: Confronto performance tempo usato/tempo totale su mappa di Torino, con tempo 1000.

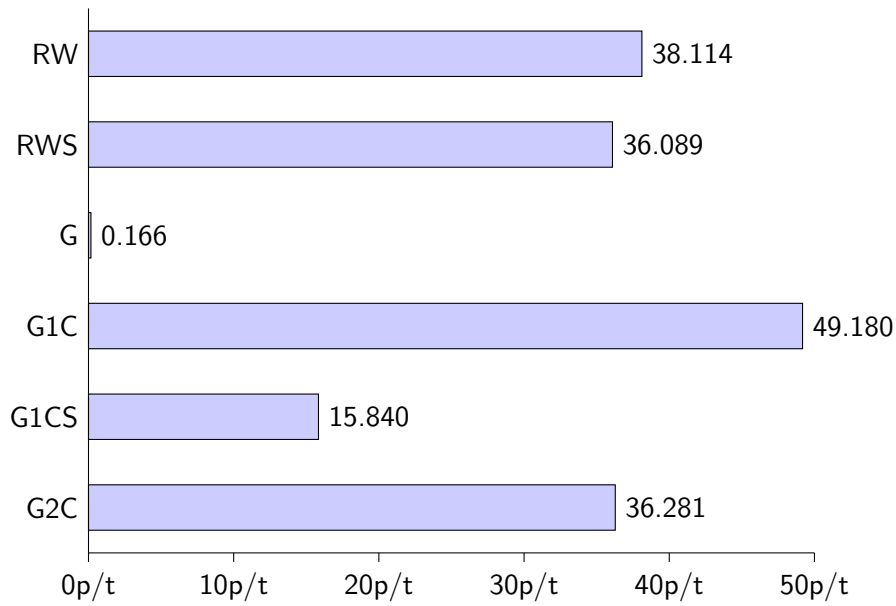


Figura 6.16: Confronto performance penalità/tempo usato su mappa di Torino, con tempo 6000.
I valori sono intesi $\times 1000$.

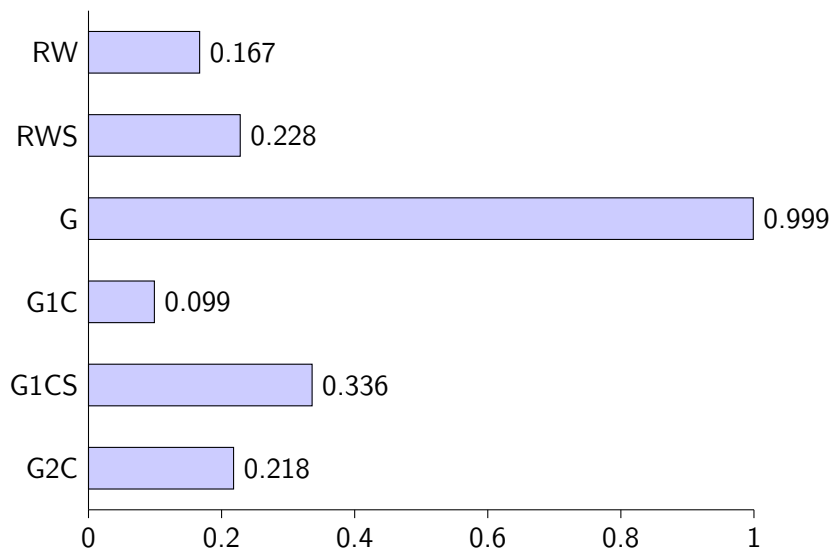


Figura 6.17: Confronto performance tempo usato/tempo totale su mappa di Torino, con tempo 6000.

Capitolo 7

Conclusioni

Dai risultati sperimentali emergono alcune proprietà interessanti riguardo le strategie che abbiamo sviluppato:

“Random walk”

Questa strategia esplora in maniera soddisfacente la mappa *default* sia in caso di tempo 1000 che 6000.

Nelle mappe *random* e Torino la sua performance è scarsa (38% di celle informate in entrambi i casi) e non migliora all’aumentare del tempo a disposizione.

Questi dati trovano spiegazione nel fatto che l’agente entra troppo presto in fase di fuga.

“Random walk” con *switch*

Questa strategia non presenta vantaggi rispetto alla versione senza *switch* né nella mappa *default* né in quella *random*. La mappa Torino fornisce invece all’agente l’opportunità di effettuare uno *switch*, consentendogli di migliorare anche se di poco il risultato (43% con 1000 e 44% con 6000 unità di tempo) e rendendolo il migliore con limite 1000.

Primo agente *greedy*

Questa strategia si è rivelata in molti casi la migliore:

nella mappa *default*, come tutti gli agenti di tipo *greedy* ottiene buoni risultati con tempo 1000 (81%); inoltre è l'unica implementazione in grado di segnalare tutte le celle della mappa a patto di avere 6000 unità di tempo a disposizione.

Queste performance si ripetono anche nella mappa *random*, in cui l'agente segnala rispettivamente il 41% ed il 94% delle celle.

Nella mappa Torino, a fronte di un risultato nella media nel caso di tempo 1000 (38%), l'agente ottiene la massima prestazione (100%) con limite a 6000.

I buoni risultati ottenuti da questo agente con un'elevata quantità di tempo a disposizione derivano dal fatto che non ha limiti al numero di visite effettuabili su ogni cella: questo lo porta ad assumere un comportamento "*brute force*" che, in assenza di cicli, lo lascia libero di esplorare tutta la mappa.

Agente *greedy 1 chance*

Porre un forte limite (1) al numero di possibilità di visita di ogni cella nel caso della mappa di *default* non ha influenzato troppo negativamente i risultati (81% e 90%).

Sulla mappa *random*, nel caso 6000, è invece possibile notare come questa scelta si sia rivelata penalizzante per l'esplorazione (segnalazioni scese al 57%).

La conformazione della mappa Torino mette in ginocchio questa strategia: l'agente con 1000 unità di tempo raccoglie un misero 22%, risultato che non riesce a migliorare neppure con 6000 unità di tempo.

Dai nostri test emerge quindi che questa strategia non presenta quindi vantaggi rispetto alle altre.

Agente *greedy 1 chance con switch*

La possibilità di effettuare uno *switch* è fortemente dipendente dalla conformazione della mappa.

Nel caso della mappa di *default* questo non si attiva e pertanto le prestazioni sono rimaste invariate rispetto alla soluzione senza *switch*.

Un lieve miglioramento si nota nella mappa *random* con tempo 6000, il cui grado di segnalazione passa da 57% a 59%. Un miglioramento più consistente si verifica nella mappa di Torino, in cui l'agente passa da 22% a 28% nel caso 1000, e da 22% a 59% nel caso 6000.

Agente *greedy 2 chances*

La seconda *chance* di visita di una cella offerta da questo agente ne lascia invariato il comportamento rispetto alle altre strategie *greedy* nella mappa di *default*.

Un sostanziale miglioramento si nota invece nella mappa *random* (da 57% a 94% in caso di tempo 6000), in cui l'agente ottiene un risultato analogo a quello del primo agente *greedy*. Sulla mappa di Torino le prestazioni migliorano leggermente rispetto alla 1 *chance* (da 22% a 38% per entrambi i tempi).

7.1 Riflessioni sui risultati dei test

I test effettuati hanno mostrato:

- la capacità del nostro agente di ottenere buone prestazioni sia in ambienti “strutturati” che totalmente randomici;
- le implementazioni di tipo *greedy*, come previsto, si comportano nella maggior parte dei casi meglio di quelle di tipo *random*;
- la scelta di concedere a ogni cella una sola *chance* di essere visitata risulta troppo limitante;
- in alcuni casi può invece risultare sufficiente ai fini esplorativi concederne 2;
- con le giuste condizioni morfologiche, avere la possibilità di effettuare uno *switch* può portare ad un miglioramento prestazionale.

7.2 Considerazioni personali

I risultati ottenuti ci hanno portato a riconsiderare alcune delle scelte effettuate in fase di implementazione e pertanto siamo giunti alle seguenti conclusioni:

- abbiamo impostato tutte le strategie in modo da concludere la missione appena le celle nelle immediate vicinanze dell’agente non fossero più “utili” (*stallo*), anche nel caso in cui ci fosse ulteriore tempo a disposizione, poiché temevamo che protrarre ulteriormente l’esplorazione avrebbe solamente causato ulteriore penalità.
I dati sperimentali hanno però smentito questa tesi;
- a questo proposito, siamo convinti che si potrebbero ottenere migliori risultati tenendo traccia delle celle utili ma non visitate incontrate durante il cammino ed andandole ad esplorare (se vi fossero le condizioni) al raggiungimento di una fase di *stallo*;
- per valutare meglio le differenze di comportamento degli agenti, avremmo potuto imporre un limite di tempo intermedio tra quelli analizzati (es. 2000 o 3000).
In questo modo l’agente avrebbe avuto un tempo più proporzionato alle dimensioni delle mappe più grandi.

Bibliografia

- [1] Stuart Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd Edition, 2009.
- [2] Eugene F. Krause, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover Publications, 1987.
- [3] William A. Dembski, *Intelligent Design is not Optimal Design*. Baylor University, 2000.