Dead Reckoning - Meccanismi per compensare la mancanza di risorse nei Multiplayer Mass Online Games

Nicola Corti - 454413
Corso di Laurea Magistrale in Informatica - Università di pisa

10 Dicembre 2013

Sommario

Questa relazione ha lo scopo di presentare il meccanismo del *Dead Reckoning*, meccanismo largamente utilizzato nel campo dei MMOG (Multiplayer Mass Online Games) per ridurre i ritardi del gioco generati dalla latenza di comunicazione della rete. Verranno dapprima presentate le metodologie base, con i vari benefici ed le varie difficoltà che si possono incontrare utilizzando il *Dead Reckoning*, e successivamente verranno analizzate alcune alternative ed evoluzioni proposte al sistema del *Dead Reckoning*.

Indice

Introduzione

Fra tutti software che sono stati mai realizzati dall'industria informatica, forse i più complessi che si possano pensare sono i videogame.

Per poter realizzare un videogames è infatti necessario avere competenze che toccano tutti gli aspetti dell'informatica, dalla grafica tridimensionale in primis, ai database, senza ovviamente escludere l'intelligenza artificiale.

Il mercato videoludico, nato negli anni '70 con il primo *Pong*[1] o altri giochi dello stesso calibro, ha subito fino ad oggi notevoli impennate fino ad invadere la vita di tutte le

[1] `http://en.wikipedia.org/wiki/Pong`

persone, con console presenti nella casa di molte persone e videogiochi che possono essere fruiti facilmente e gratuitamente tramite ogni tipo di piattaforma.

A fare da padrone a questa crescita è stato probabilmente l'introduzione di un'altra tecnologia all'interno dello sviluppo dei videogiochi: il networking. A partire dagli anni '90 sono nati i primi MMORPGs (Multiplayer Mass Online Role Play Games [1]), giochi di ruolo multigiocatore quali *Ultima Online*[2], *Lineage*[3] o *EverQuest*[4] che sono riusciti ad unire alla componente videoludica una componente multiplayer, permettendo ai videogiocatori di incontrarsi e di scontrarsi attraverso la rete, offrendo un livello di sfida ovviamente non raggiungibile dai personaggi gestiti dalla CPU.

Attualmente il mercato MMOG è considerato uno fra i mercati più importante in campo videoludico, con giochi quali *World of Warcraft*[5](sviluppato da Blizzard - figura 1) o *League of Legends*[6](sviluppato da Riot) che toccano picchi di 30 milioni di videogiocatori connessi al mese (di cui una media di 10 milioni di videogiocatori connessi contemporaneamente)[7].

Figura 1: Screenshot tratto dal videogioco World of Warcraft - Si possono notare i personaggi di tutti i giocatori collegati

Ovviamente l'introduzione del networking all'interno dei videogiochi ha portato con se notevoli criticità che devono essere prese in considerazione durante lo sviluppo del software e durante la realizzazione dell'infrastruttura che possa supportare un tale carico. Si consideri inoltre che questi software richiedono un numero molto elevato di risorse: potenza di calcolo e memoria per effettuare il rendering 3D del mondo di gioco, spazio disco per memorizzare mappe, mondi, video ed altro materiale multimediale, banda di rete per effettuare la comunicazione con il server e con gli altri giocatori.

In particolare la banda e la latenza della rete possono molto spesso diventare dei *bottleneck* e rallentare il gioco, si consideri infatti che possono connettersi a giocare persone geograficamente dislocate in luoghi molto distanti del globo. Il software deve occuparsi di gestire questi problemi, al fine di offrire un'esperienza di gioco che non sia affetta da ritardi, che potrebbero altrimenti rendere l'esperienza talmente pessima, tanto da rendere

[2] `http://www.uo.com/`
[3] `http://lineage.plaync.com/`
[4] `https://www.everquest.com/`
[5] `http://battle.net/wow/`
[6] `http://leagueoflegends.com/`
[7] Game Industry - League of Legends: 32 million monthly active players

il gioco ingiocabile, con conseguente perdita della clientela da parte della software house produttrice.

Fra i vari meccanismi sviluppati per gestire questi ritardi, uno fra i più utilizzati è il *Dead Reckoning* che si occupa sostanzialmente di prevedere gli spostamenti dei giocatori in base ai loro precedenti spostamenti, prendendo in considerazione informazioni quale la posizione, la velocità e l'accelerazione.

1 Il fenomeno del *Lag*

Il compito del server che gestisce un MMOG è quello di:

- Mantenere lo stato del gioco consistente,
- Ricevere e gestire gli eventi provenienti dagli utenti e modificare lo stato di conseguenza.

Lo stato del gioco e gli eventi da gestire variano ovviamente in base al tipo di gioco come descritto nella tabella 1.

Tipo di gioco	**Esempi di stato**	**Esempi di Eventi**
FPS (*First Person Shooter*)	Posizione degli altri giocatori e dei power-up sulla mappa	Spostamento dei giocatori e sparo di un colpo
RTS (*Real Time Strategy*)	Posizione delle truppe, degli edifici e delle risorse	Spostamento delle truppe ed attacchi fra truppe
RPG (*Role Play Game*)	Posizione del proprio avatar, dei mostri e degli avatar degli altri giocatori	Spostamento dell'avatar, uso o raccolta di un oggetto, attacco di un mostro o di un altro avatar, interazione con gli altri avatar (chat, etc...)
Simulazione	Posizione del proprio veicolo (aereo, macchina, etc...) e posizione dei veicoli avversari	Cambiamento della velocità e della direzione del veicolo
Sportivo	Posizione del proprio personaggio e dei personaggi avversari	Cambiamento di posizione, passaggi di palla, tiri, altre azioni dipendenti dallo sport.

Tabella 1: Tipi di gioco con esempi di stato e di eventi da gestire

Le comunicazioni fra i client dei giocatori e il server di gioco avvengono attraverso la rete, sia essa una rete LAN oppure Internet, che introduce dei ritardi, in parte generati dal fatto che la larghezza di banda dei client è limitata, ed in parte generati dal fatto che è presente una latenza di comunicazione fra il server e il client (dovuto ai vari router che ogni pacchetto deve attraversare per raggiungere il server). La latenza di comunicazione fra il server e il client viene chiamata anche *Ping*.

Se consideriamo lo scenario delle connessioni attuale, la larghezza di banda è diventata sufficientemente ampia (molti utenti riescono infatti a connettersi tramite connessioni ADSL ad elevate velocità) da non creare rallentamenti significativi, mentre invece esistono latenze di comunicazione che possono generare ritardi notevoli (si pensi ad esempio alla latenza di comunicazione fra Europa ed America oppure fra Europa ed Asia).

Inoltre i ritardi possono essere amplificati dal fenomento della pardita di pacchetti (*packet loss*) oppure dal fatto di utilizzare macchine con potenza di calcolo ridotta, che tardano nell'inviare e nel ricevere i pacchetti.

Nel caso in cui il ritardo di comunicazione complessivo fra il client e il server risulti troppo alto è possibile che il server non riesca più a mantenere lo stato consistente, gli eventi generati dai client possono essere ricevuti in ritardo e non nell'ordine temporale in cui erano stati effettivamente generati dai client. Il server deve quindi affrontare questa situazione e riportare lo stato ad una situazione consistente.

Si verifica quello che viene appellato dai videogiocatori come *Lag* ([3] - dall'inglese ritardo oppure *Latency GAp*), i videogiocatori possono notare personaggi che si muovono a scatti, personaggi che non si muovono e possono addirittura non riuscire più a controllare l'intero gioco. L'esperienza del *Lag* può essere talmente frustrante da portare il giocatore a cambiare videogioco e si deve quindi cercare di evitarla per quanto possibile.

2 Il *Dead Reckoning*

Il *Dead Reckoning* ([2]) è un meccanismo per mitigare i ritardi introdotti dalla rete al fine di evitare il manifestarsi dei *Lag* e rendere l'esperienza utente più fluida.

Un sistema che non utilizza il *Dead Reckoning* obbliga i client ad inviare tutti i cambiamenti di posizione dell'avatar del giocatore con una frequenza molto elevata. Utilizzando invece il *Dead Reckoning* è possibile diminuire la frequenza con cui vengono inviati i messaggi dal client al server, i messaggi inviati contengono però informazioni ulteriori oltre alla semplice posizione quali ad esempio la velocità (modulo, direzione e verso) e l'accelerazione (modulo, direzione e verso). Utilizzando queste informazioni e le leggi fisiche della cinematica è possibile prevedere gli spostamenti degli altri giocatori.

Utilizzando questo sistema il client invia un nuovo messaggio solamente quando effettua degli spostamenti significativi, oppure quando varia la velocità e l'accelerazione, riducendo quindi notevolmente il carico complessivo del traffico in entrata al server e comportando un significativo miglioramento delle performance ([4]).

Questa fase del meccanismo di *Dead Reckoning* viene detta fase di *Prediction*, in base al fatto che si cerca di prevedere dove si troverà il giocatore partendo dai dati ricevuti.

Nel caso in cui la posizione predetta si discosti dalla posizione effettiva ricevuta successivamente è necessario effettuare una correzione della posizione del personaggio: questa fase prende il nome di *Convergence* e deve essere realizzata nel modo più gradevole possibile per l'utente.

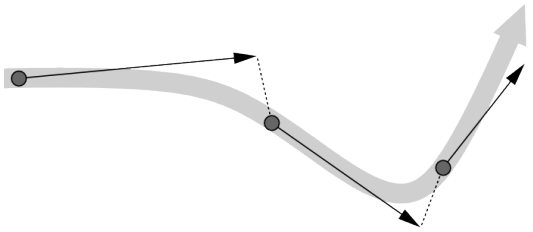

Figura 2: Esempio di utilizzo del meccanismo del *Dead Reckoning*, in grigio è rappresentato il movimento del giocatore, i cerchi rappresentano le posizioni con i vettori velocità (le frecce) inviate ad ogni messaggio. Le linee tratteggiate rappresentano gli spostamenti di posizione che devono essere gestiti nella fase di convergence.

2.1 Fase di *Prediction*

La fase di prediction consiste fondamentalmente nella predizione della posizione a partire dai dati ricevuti. Generalmente, oltre alla posizione del personaggio, si implementa la prediction utilizzando la velocità e l'accelerazione. In particolare consideriamo la velocità e l'accelerazione rispettivamente come la derivata prima e seconda della posizione rispetto al tempo; risulta cruciale la decisione dell'approssimazione da utilizzare, un'approssimazione al primo ordine che prende in considerazione solo la velocità, un'approssimazione del secondo ordine che prende in considerazione velocità ed accelerazione, oppure un'approssimazione di ordine superiore.

Utilizzando velocità ed accelerazione è possibile anche determinare quando il client deve inviare un nuovo messaggio: è sufficiente determinare un valore di soglia da confrontare con la variazione di velocità ed accelerazione: nel caso in cui la variazione sia superiore alla soglia il client invierà un nuovo messaggio, altrimenti la piccola variazione risulta tollerabile e non va ad impattare sul gioco.

Predizione utilizzando approssimazione al primo ordine La predizione con approssimazione al primo ordine prende in considerazione solamente la velocità e calcola la nuova posizione utilizzando la seguente legge della cinematica:

$$p(t) = p(0) + v(0)t$$

Dove $p(t)$ rappresenta la posizione al tempo t e $v(t)$ rappresenta la velocità al tempo t.

Si potrebbe essere portati a pensare che un'approssimazione di secondo ordine, malgrado sia più onerosa dal punto di vista computazionale, possa offrire dei risultati migliori in termini di precisione, ma non è sempre così.

Un discorso del genere può valere per videogiochi di simulazione dove l'accelerazione non varia molto, ma può portare a risultati fuorvianti nel caso di giochi RTS, FPS o RPG: in questi giochi i giocatori possono effettuare repentini cambi di direzione e l'accelerazione può portare a predizioni molto distanti dalla realtà.

Predizione utilizzando approssimazione la secondo ordine La predizione con approssimazione al primo ordine prende in considerazione anche l'accelerazione e calcola la nuova posizione utilizzando la seguente legge della cinematica:

$$p(t) = p(0) + v(0) \cdot t + \frac{1}{2}a(0)t^2$$

Dove $p(t)$ rappresenta la posizione al tempo t, $v(t)$ rappresenta la velocità al tempo t e $a(t)$ rappresenta l'accelerazione al tempo t. Si noti come il termine dell'accelerazione consideri il tempo al quadrato per cui piccoli errori nell'accelerazione possono portare a grosse differenze nella posizione.

Questo genere di approssimazione è dunque utilizzata per i videogiochi dove l'accelerazione non varia troppo: ad esempio nei giochi di simulazione di veicoli.

Predizione utilizzando approssimazione di ordini superiori Le predizioni di ordine superiore (quelle che prendono in considerazione gli *strappi*) vengono spesso scartate in quanto introducono errori maggiori a quelle del secondo ordine e risultano inoltre molto difficili da calcolare, in quanto molto spesso l'accelerazione varia in base alle decisioni dell'utente e risulta difficile da prevedere.

2.2 Fase di *Convergence*

Nella fase di prediction vengono quindi calcolate le nuove posizioni assunte dal personaggio, considerando i dati ricevuti dal client in ogni messaggio. Nel caso in cui la posizione ricevuta dal client sia troppo distante dalla posizione predetta è necessario correggere questa discrepanza: ha dunque luogo la fase di convergence.

La fase di convergence consiste fondamentalmente nel riposizionamento del giocatore, in modo da correggere l'errore generato dalla previsione. Questo riposizionamento può essere effettuata in svariati modi, più o meno gradevoli (immagine 3).

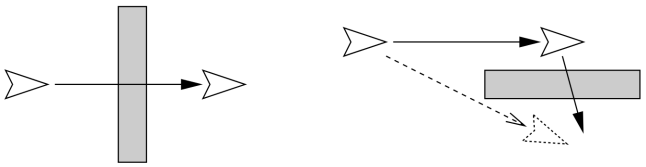

Figura 3: Esempio di utilizzo del meccanismo di *convergence*, nell'immagine di sinistra si usa la versione *snap* mentre nell'immagine di destra si usa la versione lineare

Convergenza tramite *Snap* La convergenza tramite *snap* consiste semplicemente nel riposizionare il giocatore alla nuova posizione, senza effettuare posizionamenti intermedi.

Questa tecnica risulta la più semplice da implementare, ma al contempo è anche quella più sgradevole per l'utente, dato che il giocatore non vede il personaggio che si "teletrasporta" fra un punto ed un altro, senza percorrere alcun percorso; si avrà quindi una forte percezione del lag che potrebbe inficiare l'esperienza di gioco.

Per questo motivo si cerca di implementare quantomeno una convergenza di tipo lineare, per rendere meno sgradevole l'evento.

Convergenza Lineare La convergenza lineare consiste in realizzare un'interpolazione lineare fra il punto di partenza e il nuovo punto. Così facendo il giocatore ha la percezione che il personaggio si stia muovendo lungo una linea retta.

Per la convergenza lineare è possibile ad esempio utilizzare la formula seguente: sia s_1 la posizione al tempo t_1 e sia s_2 la posizione al tempo t_2, si vuole calcolare la posizione s al tempo t con $t_1 < t < t_2$:

$$s = \frac{(t - t_1)}{(t_2 - t_1)} \cdot (s_2 - s_1) + s_1$$

La convergenza lineare è da preferirsi a quella di tipo *snap*, anche se può portare a percorsi che possono comunque risultare innaturali. L'utente non si aspetta infatti che tutti i personaggi si muovano lungo linee rette, ma si aspetta movimenti che siano lungo linee curve (quella che viene chiamo un percorso di tipo *smooth*), per questo si può ricorrere alle convergenze di ordine superiore.

Convergenza di ordine superiore La convergenza di ordine superiore permette di ottenere curve che siano più dolci di una semplice linea retta. Per utilizzare questo meccanismo si può ricorrere a curve di tipo differente, una curva di tipo famoso è la curva di Bézier cubica.

In particolare assumiamo che s_1 sia la posizione al tempo t_1, e sia s_2 la posizione al tempo t_2, consideriamo $\bar{s_1}$, $\bar{t_1}$, $\bar{s_2}$ e $\bar{t_2}$ le predizioni a partire da s_1 e da s_2 con i loro relativi

tempi, è possibile creare una curva di Bézier che unisce i punti s_1 e $\bar{s_2}$ usando la formula seguente:

$$s(t') = s_1(1-t')^3 + 3 \cdot \bar{s_1}t'(1-t')^2 + 3 \cdot s_2 t^2(1-t') + \bar{s_2}t'^3 \qquad t' = \frac{(t-t_1)}{(\bar{t_2}-t_1)}$$

Con $t_1 < t' < \bar{t_2}$.

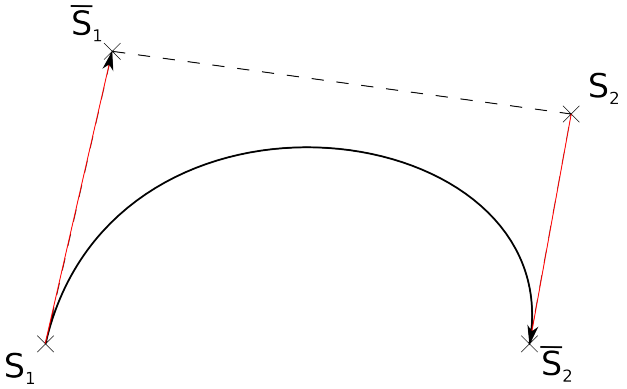

Figura 4: Esempio di percorso generato dalla *convergence* con curva di bezier quadratica fra i punti s_1 e s_2 e le loro relative predizioni $\bar{s_1}$ e $\bar{s_2}$

Questa curva offre una convergenza migliore di quella lineare (immagine 4), ma può portare a computazioni più complesse in quanto sono presenti termini al quadrato e al cubo che possono rallentare il calcolo.

3 Inconsistenza dello stato

Sebbene l'utilizzo del *Dead Reckoning* riduca notevolmente il carico di rete, non si è preso in considerazione ciò che accade se i pacchetti che contengono le informazioni di velocità ed accelerazione vengono persi.

Il sistema del *Dead Reckoning* gode fortunatamente della proprietà del *Self-healing*: nel caso in cui un pacchetto venga perso oppure che subisca un grosso delay, verrà effettuata la convergenza utilizzando il pacchetto successivo e il sistema del *Dead Reckoning* continuerà a funzionare.

Purtroppo questo meccanismo, malgrado sia robusto, non tiene in considerazione il fatto che può portare lo stato di gioco ad una situazione di inconsistenza, si consideri l'esempio seguente:

A Dead Man that Shoots Si supponga che il giocatore A spari un colpo di proiettile verso il giocatore B. Ovviamente il client memorizza velocità ed accelerazione del proiettile e li invia al server.

Purtroppo il pacchetto contente questi dati subisce un forte delay a causa dell'infrastruttura di rete; nel mentre il giocatore B spara un colpo verso il giocatore C. Il pacchetto contenente queste informazioni non viene rallentato a causa dell'infrastruttura di rete di B (che risulta essere meno affetta da ritardi rispetto a quella di A) e arriva al server provocando la morte di C.

Il pacchetto contente la pallottola sparata di A arriva in un secondo momento al server e provoca la morte di B.

3.1 Utilizzo dei *timewarp*

Per sopperire a questo problema è necessario definire un ordinamento fra gli eventi che sono stati registrati dai client, in modo da poterli eseguire in modo ordinato e a non portare ad un'inconsistenza dello stato ([5], [6]).

Ciò è possibile utilizzando dei *timestamps*, cioè corredando l'informazione sulla velocità e sull'accelerazione di una marca temporale che indichi il tempo a cui è stata generato l'evento. Ciò è possibile utilizzando un protocollo quale NTP[8] (Network Time Protocol), che permette di sincronizzare l'orologio dei client al fine di avere un ordinamento temporale degli eventi.

Utilizzando questa informazione ulteriore è quindi possibile utilizzare un algoritmo che si basa sui *timewarp*: ogni client invia i propri eventi a tutti gli altri client aggiungendo i dati sul tempo in cui è stato generato l'evento; i client memorizzano tutti gli eventi che hanno ricevuto in ordine cronologico ed effettuano dei *timewarp* alla ricezione di nuovi eventi, ovvero ricalcolano lo stato eseguendo in ordine gli eventi che hanno ricevuto.

In questo modo tutti i client si "accordano" sullo stato effettivo e non si genera inconsistenza, in particolare nel caso del *Dead Man that Shoots* A, B e C eseguono un *timewarp* subito dopo la ricezione dell'evento dello sparo di A e lo stato viene riportato ad uno stato di consistenza.

Ovviamente c'è da considerare che questo algoritmo presenta alcune criticità: in primis assumere che gli eventi generati da un client siano inviati a tutti gli altri client può risultare molto difficile in architetture con milioni di nodi; inoltre la complessità computazionale di questo algoritno può risultare molto elevata; infine l'operazione di riportare lo stato ad una situazione consistente deve essere effettuato entro un certo lasso di tempo (se infatti viene renderizzato sullo schermo la morte di C, è possibile che C si ritrovi in vita successivamente e non riesca a comprenderne il motivo).

4 L'*AntDeadReckoning*

Il *Dead Reckoning* non prende in considerazione un informazione importante che si può evincere dal gioco: la semantica.

Si pensi ad un gioco di sparatutto (ad esempio Quake 3[9]), ovviamente un giocatore sarà maggiormente attratto da eventuali armi, medikit o altri power-up presenti sulla mappa e cercherà di evitare luoghi dove subisce danni.

Per prendere in considerazione anche queste informazioni è stato proposto il modello dell'*AntDeadReckoning* ([7]). Questo modello parte dal principio del modello comportamentale delle formiche (da questo il nome *Ant*) che seguono il percorso dei loro simili attraverso i feromoni che rilasciano.

Quindi ogni entità del gioco (sia gli oggetti del gioco che i personaggi stessi) rilascerà dei "feromoni" che eserciteranno una forza attrattiva verso tutti i personaggi del gioco.

Applicazioni di questo modello hanno mostrato che si possono generare predizioni che sono più precise di quelle generate dal semplice *Dead Reckoning* fino a circa il 40%.

4.1 Misurazioni

I fondamenti di questo modello possono essere ritrovati da alcune misurazioni effettuate nella mappa `q3dm01` del gioco Quake 3 (immagine 5).

[8] RFC 958 - `http://tools.ietf.org/html/rfc958`

[9] `http://www.quake3arena.com/`

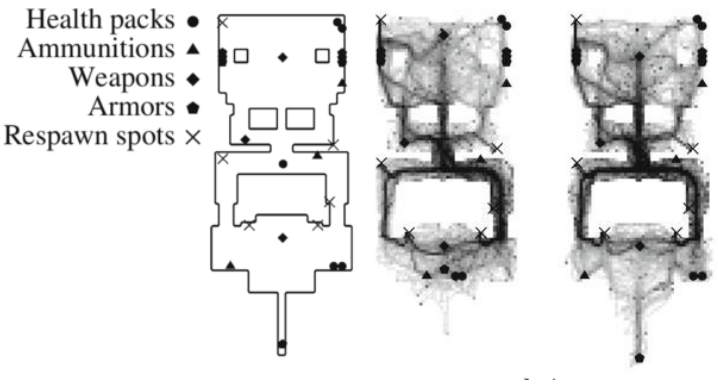

Figura 5: Mappa del livello `q3dm01` del gioco Quake 3, le ultime due immagini sulla destra mostrano di un colore più scuro i punti che sono stati percorsi più spesso

Nella mappa sono rappresentate di un colore differente le zone che sono state maggiormente percorse: si può notare come alcune zone, in particolare quelle che non contengono oggetti di interesse, siano state maggiormente trascurate rispetto alle zone in prossimità di armi e power-up.

Le misurazioni cambiano drasticamente se si spostano gli oggetti all'interno della mappa (si veda il corridoio nella parte bassa della mappa).

Inoltre si può notare come siano presenti dei percorsi fissi, che i giocatori percorrono per arrivare prima al power-up desiderato, oppure per raggiungere punti importanti della mappa (colline, o zone in cui si ha una maggiore visibilità e quindi una maggiore probabilità di vittoria).

I risultati delle misurazioni possono quindi essere riassunti nei punti seguenti:

- I personaggi sono attratti dagli oggetti del gioco, in modo proporzionale alla loro potenzialità ed in particolare se ne hanno urgente bisogno (si pensi ad un medikit).
- I personaggi sono attratti da altri personaggi che sono amichevoli nei loro confronti (compagni di squadra, altri personaggi con cui commerciare, etc...)
- I personaggi sono attratti da altri personaggi che risultano indifesi (si nota infatti che un personaggio che inizia a sparare ad un altro personaggio che risulta indifeso tende ad avvicinasi e ad inseguirlo)
- I personaggi sono attratti da luoghi interessanti (anche detti punti di interesse - POI o *hot-spot)* che possono aiutarli nell'esito della partita: quali colline, torri, avamposti, etc...
- I personaggi tendono ad allontanarsi da altri personaggi che abbiano un comportamento offensivo nei propri confronti, tendono infatti ad arretrare per evitare di ricevere ulteriori colpi e possono infine scappare
- I personaggi tendono ad allontanarsi dai luoghi che gli possono arrecare un danno e tendono ad evitare anche le zone in loro prossimita, quali ad esempio pozze di acido, di lava, oppure torri o altre armi che fanno parte della mappa e danneggiano i giocatori.

Si noti che molte informazioni sulle relazioni fra i personaggi possono essere raccolte dalla storia stessa dei personaggi: se il personaggio A ha ucciso molte volte il personaggio B

e non ha mai ucciso il personaggio C ne risulterà che A sarà più attratto da B piuttosto che da C (permettendogli di effettuare più probabilmente un uccisione) e B tenderà a scappare da A con una probabilità maggiore rispetto a quella con cui C tenderà a scappare da A.

Partendo da questi risultati ogni entità genera una quantità di *feromoni* proporzionale alla sua appetibilità.

4.2 Modello Analitico

Nel modello *AntDeadReckoning* il mondo è diviso in celle di eguale grandezza, ogni cella contiene un certo numero di feromoni che esercitano una certa attrattiva verso i propri vicini.

Per ogni personaggio si definisce una *attraction region* ovvero una regione prossima al personaggio in cui le caselle contenente feromoni vengono considerate per calcolare la forza risultante sul personaggio.

Utilizzando questo modello si può calcolare la posizione come:

$$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \boldsymbol{v}(0) \cdot t + \frac{1}{2}\left(\alpha \cdot \frac{1}{m}\boldsymbol{F} + (1-\alpha) \cdot \boldsymbol{a}(0)\right) t^2$$

Dove $\boldsymbol{F}$ rappresenta la somma vettoriale delle forze attrattive esercitate dai feromoni presenti nelle celle dell'*attraction region*. Il parametro α varia da 0 a 1 e permette di far pesare la predizione fra l'accelerazione e l'attrazione generata dai feromoni vicini. Il termine $\boldsymbol{F}$ risulta essere diviso per la massa del personaggio m in ottemperanza alla formula fisica di Newton $\boldsymbol{F} = m \cdot \boldsymbol{a}$.

La forza viene calcolata come un vettore centrato nel personaggi, avente come direzione la retta passante fra il personaggio e la cella contenente i feromoni, e ha modulo direttamente proporzionale alla quantità di feromoni e inversamente proporzionale alla distanza (immagine 6).

$$||\boldsymbol{f}_t(cell, x_t)|| = \frac{\mathrm{ph}_t(cell)}{d(cell, x_t)^k}$$

Dove $\mathrm{ph}_t(cell)$ rappresenta la quantità di feromoni al tempo t e $d(cell, x_t)$ rappresenta la distanza fra la cella e la posizione di x_t e k è un parametro del sistema.

Nel caso di entità che si respingono ovviamente il modulo è negativo.

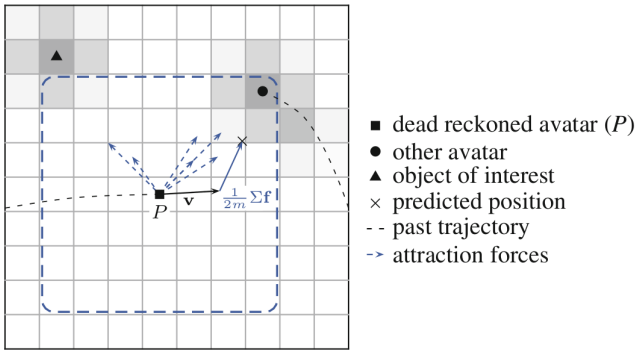

Figura 6: Esempio di *AntDeadReckoning*, con *attraction region* e vettori che rappresentano le forze esercitate dalle caselle contenenti feromoni.

4.3 Dinamica dei Feromoni

Il numero di feromoni che sono presenti in una cella non è statico, ma varia nel tempo in base ad una serie di fenomeni:

Generation Ogni entità presente in una cella genera un numero di feromoni proporzionale alla sua appetitibilità nei confronti del giocatore P. È possibile definire un valore (ph_{max}) come valore massimo dei feromoni presente in una casella in modo da limitarne l'aumento;

Evaporation I feromoni presenti in un cella diminuiscono ("evaporano") rispetto a quelli presenti nel passato in base ad un parametro ε compreso fra 0 ed 1: $\varepsilon = 0$ sta ad indicare che i feromoni vengono dispersi immediatamente, $\varepsilon = 1$ sta ad indicare che i feromoni non si disperdono mai;

Dissemination I feromoni si spostano fra caselle adiacenti in base ad una velocità e vengono rallentati da muri ed altri ostacoli, in modo da modellare il fatto che i personaggi non passeranno attraverso muri ed ostacoli.

Questi fenomeni vengono sintetizzati nella formula seguente che rappresenta la quantità di feromoni in un casella *cell* al tempo t per il giocatore P.

$$\begin{aligned} \mathrm{ph}_t(\mathrm{cell}) = \overbrace{\varepsilon \cdot \mathrm{ph}_{t-1}(\mathrm{cell})}^{\text{Evaporation}} + \overbrace{\sum_{\mathrm{ent} \in \mathrm{cell}} \mathrm{attract}(\mathrm{ent}, P)}^{\text{Generation}} + \\ + \underbrace{\sum_{c \in N(\mathrm{cell})} \frac{\varepsilon \cdot \gamma}{N(c)} \cdot \mathrm{ph}_{t-1}(\mathrm{c})}_{\text{In Dissemination}} - \underbrace{\varepsilon \cdot \gamma \mathrm{ph}_{t-1}(\mathrm{cell})}_{\text{Out Dissemination}} \end{aligned} \tag{1}$$

Dove attract è una funzione che rappresenta l'attrattività di ogni entità presente nella cella nei confronti del personaggio P, $N(\mathrm{cell})$ rappresenta i vicini della cella (cell) dove avviene la dissemination, e γ è un parametro tra 0 ed 1 indicante la percentuale di feromoni che vengono disseminati fra i vicini (dopo che parte dei feromoni sono evaporati, per questo è moltiplicato per ε).

Riferimenti bibliografici

[1] `http://en.wikipedia.org/wiki/History_of_massively_multiplayer_online_games` - Wikipedia

[2] Smed, J., Hakonen H., *University of Tarku, Finland* (2006). Algorithms and Networking for Computer Games. *Wiley* (191-195 pp.)

[3] `http://en.wikipedia.org/wiki/Lag_(online_gaming)` - Wikipedia

[4] Pantel, L. & Wolf, L. C., (2002, April) On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st workshop on Network and system support for games* (79-84 pp.). ACM.

[5] Aggarwal, S., Banavar, H., Khandelwal, A., Mukherjee, S., & Rangarajan, S. (2004, August). Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (161-165 pp.). ACM.

[6] Mauve, M. (2000, January). How to keep a dead man from shooting. In *Interactive Distributed Multimedia Systems and Telecommunication Services* (199-204 pp.). Springer Berlin Heidelberg.

[7] Yahyavi, A., Huguenin, K., & Kemme, B. (June 2013). Interest modeling in games: the case of dead reckoning. *Multimedia Systems* (Volume 19, Issue 3, 255-270 pp.).