ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

INTELLIGENZA ARTIFICIALE PER GIOCHI STRATEGICI REAL-TIME

Elaborato in
Fondamenti di Informatica B

Relatore:
ANDREA ROLI

Presentata da:
FATTORI MATTEO

III Sessione

Anno Accademico 2012/2013

Ai miei genitori che hanno sempre creduto in me

Indice

Introduzione

I giochi commerciali per computer rappresentano una parte crescente dell'industria dell'intrattenimento; con la comparsa di personal computer sempre più veloci quelli basati su simulazione in particolare sono diventati sempre più popolari. Gli elementi che accomunano i giochi strategici in real-time sono i vincoli temporali basati sulle azioni compiute dal giocatore ed una robusta reattività, i quali caratterizzano lo sviluppo di un'intelligenza artificiale che deve essere in grado di risolvere problemi decisionali in modo rapido e soddisfacente.

I giochi di simulazione rappresentano quindi un ambiente ideale su cui effettuare test in fase di ricerca, infatti i giochi strategici in tempo reale possono essere visti come giochi di simulazione semplificati. Più in generale l'ambiente di simulazione può variare ed è questo il motivo per cui i ricercatori risultano interessati a questo campo, che trova sviluppo non solo nel settore industriale dei giochi ma anche in svariate applicazioni reali fra cui la gestione del traffico automobilistico, la finanza oppure le previsioni del tempo. Si vuole far notare che tutti questi settori di applicazione prevedono ambienti in cui le informazioni percepite dal sistema non sono complete, pertanto è previsto l'utilizzo di apposite tecniche di intelligenza artificiale che permettono al sistema di effettuare ragionamenti decisionali solamente sulle informazioni di cui si dispone.

Si vuole introdurre quindi uno studio preliminare sulle particolarità riscontrate in tipici giochi strategici in tempo reale facendo riferimento alle caratteristiche di giochi da tavolo strategici tradizionali. Valutando inoltre una stima sulla complessità computazionale, viene esaltata ulteriormente la differenza di ognuna di queste due categorie.

Viene introdotto quindi uno studio accurato sui problemi che un sistema di intelligenza artificiale deve affrontare durante il corso di una partita, in cui vengono individuati principalmente sei problemi che incapsulano aspetti spe-

cifici di gioco e le caratteristiche precentemente studiate. Dopo aver studiato singolarmente ogni problema, sono state individuate le attività primarie che identificano il comportamento da seguire per il raggiungimento dello scopo del gioco. Vengono comparate quindi tali attività con quelle di un giocatore umano.

Sulla base delle attività riscontrate viene fatta successivamente una suddivisione concettuale sulle entità che compongolo il sistema, le quali vengono poi composte in un'architettura astratta che identifica principalmente tre livelli di astrazione e ne determina i ruoli. A questo punto, utilizzando un approccio di progettazione top down, è stato definito il comportamento delle singole entità nel dettaglio.

Gli studi fatti finora portano allo sviluppo di una serie di architetture ad **agenti** di intelligenza artificiale in grado di affrontare e risolvere i sei problemi riscontrati. All'interno di questo elaborato verranno prese in considerazione solo due di queste chiamate SORTS ed EISBot, le quali vengono progettate con lo scopo di riuscire a giocare ai giochi strategici in tempo reale nelle stesse condizioni in cui si trova un giocatore umano, ovvero utilizzando solamente le informazioni accessibili.

In particolare SORTS è un'architettura basata su un middleware che si interfaccia con gli agenti chiamati *Soar*, i quali sono grado di effettuare ragionamenti tattici ed impartire comandi di alto livello, sarà quindi il middleware ad occuparsi della maggior parte dei problemi. L'architettura di EISBot invece si basa sulla suddivisione delle attività in moduli ed stata progettata per affrontare un gioco strategico in tempo reale in particolare chiamato *Starcraft*.

Una volta studiate in dettaglio le due architetture e giunti in fase di implementazione, verranno riportati i risultati dei test specifici sui due sistemi. In particolare i test di SORTS hanno previsto partite individuali contro altri agenti ad intelligenza artificiale, mentre quelli effettuati su EISBot hanno previsto solamente partite contro avversari umani. L'elaborato si conclude con un resoconto sui risultati ottenuti dai test descritti.

Capitolo 1

Introduzione ai Giochi Strategici Real-Time

Questo tipo di giochi, anche chiamati RTS[1], rappresenta un sottoinsieme dei giochi di strategia, nei quali il giocatore deve portare a termine un insieme di obbiettivi preliminari per riuscire a sconfiggere il proprio avversario fra cui l'acquisizione e la gestione di una solida economia che permette al giocatore di guadagnare risorse, costruire edifici, e la formazione di una potenza militare.

1.1 Differenze fra RTS e tipici giochi strategici

Inoltre questo sottoinsieme di giochi presenta alcune particolarità che ci permettono di distinguerli dai comuni giochi da tavolo come ad esempio gli scacchi, per visualizzare al meglio questa distinzione elenchiamo le principali differenze:

- Negli scacchi ad esempio ogni giocatore deve aspettare il proprio turno e, quando questo arriva, può al massimo muovere una delle proprie pedine che non sono ancora state mangiate, mentre negli RTS non esiste il concetto di turno ma ogni giocatore può muovere le proprie unità a piacimento anche nello stesso istante di tempo così come è possibile eseguire più azioni contemporaneamente, le quali sono continuative nel tempo ovvero non sono istantanee e richiedono un determinato tempo per il loro completamento.

- Sempre soffermandoci sul concetto di turno presente nel gioco degli scacchi, nel quale esiste un tempo limitato per eseguire la propria mossa, di solito qualche minuto,notiamo che gli RTS si sviluppano in tempo reale, il che significa che ogni giocatore ha un tempo molto piccolo nel quale decidere la prossima azione(oppure un insieme di azioni), se ad esempio il gioco viene eseguito a 24 frame/secondo il giocatore avrà la possibilità di eseguire una azione ogni 42 millisecondi prima che lo stato del gioco cambi.

- Nel gioco degli scacchi ogni giocatore conosce la posizione di tutte le pedine, sia le proprie che quelle dell'avversario,mentre negli RTS il gioco è parzialmente osservabile, ciò implica che non si conosce a priori ne la posizione, ne lo stato di evoluzione della base dell'avversario. Ogni area della mappa di gioco viene mascherata da una sorta di nebbia temporanea che svanirà una volta che il giocatore avrà esplorato il suddetto territorio, per questo motivo tale concetto prende il nome di "nebbia di guerra".

- Quando nel gioco degli scacchi un giocatore effettua una mossa non vi è alcuna possibilità che questa fallisca mentre la maggior parte dei giochi RTS sono non deterministici, ovvero alcune azioni hanno una qualche possibilità di fallimento, prendiamo come esempio la costruzione di un edificio ed ipotizziamo un attacco da parte delle unità nemiche, se l'edificio viene distrutto prima che la costruzione sia terminata si può considerare l'azione "costruzione dell'edificio" fallita.

- Per confrontare la complessità di queste due categorie di giochi dobbiamo introdurre un concetto che ci permette di quantificare gli stati di un sistema dinamico, ovvero lo *spazio degli stati* che rappresenta il numero di stati possibili che il sistema può assumere. Grazie alla dimensione dello spazio degli stati, ed al numero di azioni che è possibile eseguire in un unico ciclo decisionale, riusciamo in un primo momento a comprendere tale differenza, per esempio lo spazio degli stati del gioco degli scacchi è stimato all'incirca a 10^{50} mentre il Poker Texas Holdem all'incirca a 10^{80}. Nei capitoli successivi verrà studiato a fondo il problema della complessità ma già in questa fase possiamo dire che lo spazio degli stati di un RTS è molti ordini di grandezza superiore ai due esempi sopra citati.

È grazie a tali differenze che riusciamo a comprendere l'impossibilità di utilizzare gli stessi algoritmi per risolvere gli RTS usati in giochi strategici tradizionali, come ad esempio il game-tree search, senza aver prima introdotto una serie di livelli di astrazione in modo da semplificare i problemi legati alla dinamica ed evoluzione del gioco.

1.2 Obbiettivo finale e condizioni iniziali

L'obbiettivo primario di un tipico RTS è quello di distruggere tutti gli edifici dell'avversario in modo tale che questo non possa più reagire agli attacchi subiti, per fare in modo che ciò accada i giocatori posso controllare, sia in gruppo che singolarmente, le proprie unità ed edifici sul campo di battaglia.

In alcuni RTS tipicamente la varietà di tipi di unità è molto vasta, è per tale motivo che uno dei requisiti fondamentali per raggiungere la vittoria è quello di scoprire, esplorando il territorio, sia la posizione che il tipo di unità ed edifici costruiti dal nemico in modo tale da assemblare il più velocemente possibile il tipo di unità adeguato per fronteggiarlo in battaglia. Tipicamente viene offerta la possibilità di effettuare partite con più di due giocatori, in questo contesto il singolo giocatore deve fornire supporto ai giocatori alleati in caso di necessità, in tale situazione il gioco di squadra determina un ruolo fondamentale nel corso di una partita in quanto rappresenta uno dei fondamentali fattori critici per il raggiungimento dell'obbiettivo finale.

Si vuole sottolineare inoltre che le regole fondamentali con cui i giocatori agiscono nella maggior parte di questi giochi è la stessa, ad esempio tutti i giocatori iniziano la partita nelle stesse condizioni iniziali, solitamente l'unica cosa che cambia è la scelta, fatta prima che la partita cominci, della civiltà e/o fazione che avrà alcune caratteristiche uniche riguardo alle unità e gli edifici.

1.3 Ambiente di Gioco

Questo sottoinsieme di giochi sintetizza un ambiente dinamico complesso in cui non sono accessibili tutte le informazioni riguardo lo stato corrente e la dinamica di evoluzione dell'ambiente,un esempio immediatamente comprensibile lo troviamo nella vita di tutti i giorni come il traffico automobilistico,la finanza oppure le previsioni del tempo, esempi che ci forniscono un'idea della complessità che i nostri sistemi devono affrontare.

Gli RTS possono essere visti come una semplificazione di uno di questi ambienti riscontrati nella vita reale, introducendo un "mondo" ben delimitato avente una complessità finita, si vuole far notare che essendo l'ambiente dinamico il giocatore dovrà rispondere molto velocemente alle variazioni di quest'ultimo.

L'ambiente in cui questi giochi si sviluppano è descritto tipicamente da una mappa che cambia a seconda dell'ambientazione del gioco, può essere una regione geografica ad esempio, in cui vengono distribuite un certo numero di risorse, in zone ben precise, che solo alcune unità sono in grado di raccogliere e danno la possibilità ai giocatori di migliorare la propria economia.
Grazie a queste informazioni preliminari il giocatore esperto ad esempio cercherà di portarsi in una situazione di vantaggio appropriandosi di più risorse possibile in modo da espandere e migliorare la propria potenza militare in un tempo minore rispetto a quello che impiegherebbe il suo avversario.

Questo però potrebbe non bastare per assicurare al giocatore una vittoria assoluta in quanto tipicamente la mappa su cui viene sviluppato il gioco è oscurata agli occhi di ogni giocatore, per questo motivo non si conoscono ne la posizione ne lo stato, istante per istante, della base nemica.

1.4 Risorse ed Evoluzione Tecnologica

Le risorse inoltre possono essere spese dal giocatore in modo da creare nuove unità, costruire edifici oppure sviluppare nuove tecnologie, la sequenza scelta per effettuare tali acquisti va a discrezione del giocatore in quanto essendo le risorse limitate egli dovrà decidere una strategia per conseguire la vittoria.

In aggiunta alle particolarità già indicate, tipicamente negli RTS è presente una qualche forma di evoluzione tecnologica, ovvero i giocatori possono acquisire, spendendo risorse, la creazione di nuovi tipi di unità che in fase iniziale del gioco non possedevano, questo obbiettivo secondario è raggiungibile solo quando sono stati soddisfatti i requisiti che ogni singolo tipo di unità richiede. Oltre a questo l'evoluzione tecnologica può portare un potenziamento in termini di attributi legati alla singola unità come ad esempio un aumento dei punti danno recati alle unità avversarie.

Abbastanza interessante è l'approccio umano a questi videogiochi che risulta essere di gran lunga migliore a quello del computer. Nei capitoli successivi si approfondirà questo aspetto introducendo opportune tecniche per sviluppare un'adeguata intelligenza artificiale negli RTS.

Capitolo 2

Problemi nei Giochi Strategici Real-Time

Una ricerca preliminare sviluppata sugli RTS da M. Buro [2] ha portato all'identificazione di sei problemi principali, riguardanti la gestione di questi giochi da parte di un'intelligenza artificiale, che vado ad elencare:

- Gesione delle Risorse
- Processo decisionale in condizioni di incertezza
- Ragionamento spaziale e temporale
- Collaborazione (fra più intelligenze artificiali)
- Modellazione dell'avversario ed apprendimento
- Pianificazioni in tempo reale contraddittorie

Gran parte del lavoro svolto è centrato alla risoluzione solo di alcuni di questi punti fondamentali, mentre altri non sono stati presi in considerazione, come ad esempio la collaborazione, inoltre per contribuire allo sviluppo di recenti ricerche in questo settore è stata utilizzata una grande quantità di materiale già esistente come ad esempio replay di partite e strategie conosciute.

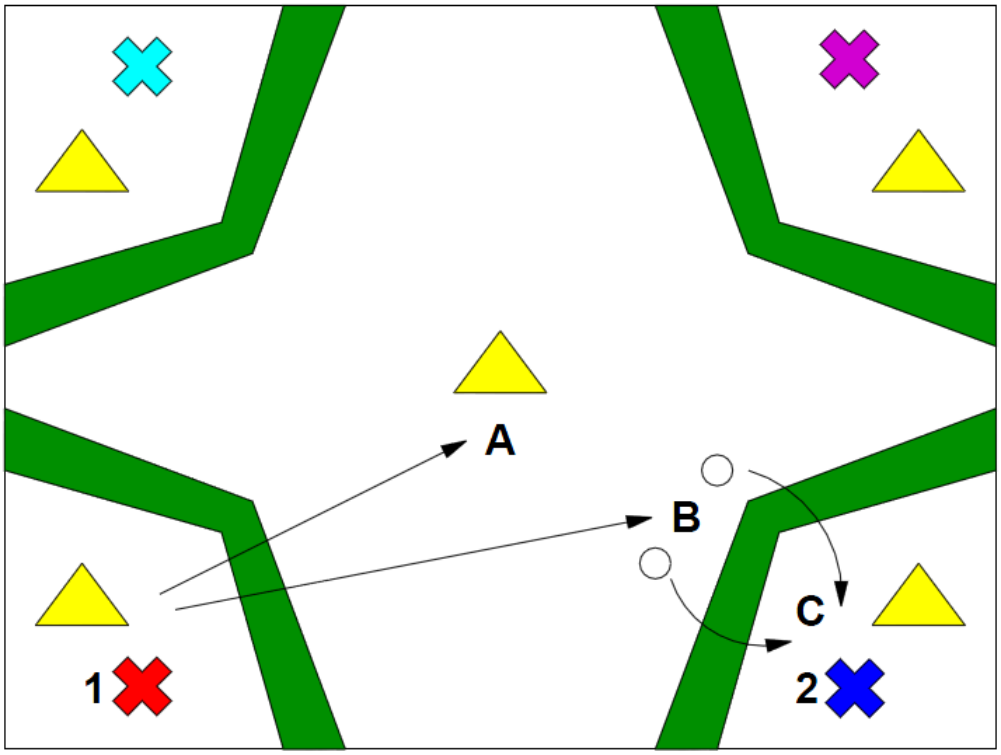
2.1 Pianificazione

Come descritto nel paragrafo di introduzione lo spazio degli stati di un gioco RTS è nettamente più ampio di quello riscontrato nei giochi da tavolo tradizionali come ad esempio gli scacchi, così come lo è anche il numero di azioni

che possono essere eseguite in un qualsiasi istante di tempo durante il gioco. Per questi motivi gli approcci come il game-tree search, che risolvono questo problema nei giochi tradizionali, non possono essere utilizzati.

A questo punto occorre introdurre un adeguato livello di astrazione in cui si è decisa l'istituzione di più livelli, in particolare al livello più alto i giocatori hanno bisogno di una capacità di pianificazione a lungo termine con lo scopo di sviluppare un'economia robusta durante il gioco, mentre al livello più basso le singole unità hanno bisogno di una particolare coordinazione che permette loro di muoversi durante le battaglie tenendo conto della geometria del terreno e delle unità dell'avversario.

Un tipico problema strategico che mostra un esempio del problema appena citato è descritto dalla seguente figura:



In questa mappa le zone verdi rappresentano le risorse di legname e le zone gialle rappresentano le risorse minerarie, inoltre si vuole far notare che non vi è la possibilità di utilizzare unità aeree.

Per un giocatore umano (1) non ci vuole molto tempo a capire che deve iniziare il prima possibile a tagliare il legname che lo imprigiona per impadronirsi e difendere la risorsa mineraria centrale A, invece il giocatore del computer (2) ,essendo incapace di effettuare tale ragionamento, inizia a tagliare il legname solamente dopo aver esaurito la risorsa mineraria più vicina, pertanto perderà la partita dopo essere stato assediato dal giocatore umano (1) nella zona B-C.

2.2 Apprendimento

Date le difficoltà di gioco riscontrate nel punto precedente introduciamo ora un ulteriore problema legato agli RTS che sostanzialmente mette in evidenza una delle carenze maggiori dei sistemi di intelligenza artificiale sviluppati finora per questo tipo di giochi, ovvero la loro incapacità di imparare in fretta. Suddividiamo ora il problema generale dell'apprendimento in tre principali sottoproblemi:

- Apprendimento precedente:
  Questo problema non persiste durante il corso della partita ma è legato al periodo antecedente, ed è caratterizzato da un'unica domanda, come possiamo sfruttare i dati accessibili, ovvero i replay esistenti di partite già giocate, le informazioni riguardanti il terreno di gioco oppure le strategie già scoperte finora?

- Apprendimento in gioco:
  Anch'esso è caratterizzato da una domanda principale, come può l'intelligenza artificiale implementare alcune tecniche di apprendimento in modo tale da permettere al sistema di migliorare il proprio gioco durante la partita? Queste tecniche includono una sorta di rinforzo all'apprendimento ma anche una modellazione dei dati riguardanti il comportamento dell'avversario, in realtà il problema principale legato a questo tipo di apprendimento è dato dal fatto che lo spazio degli stati degli RTS è troppo grande in aggiunta alla condizione iniziale: il gioco è parzialmente osservabile.

- Apprendimento intergame:
  Questo problema persiste ogni volta che una partita è terminata, cosa può essere appreso da una partita in modo da poter essere utilizzato nella partita successiva per aumentare le possibilità di vittoria? Alcuni approcci in questo senso hanno previsto l'utilizzo di una serie di strategie predefinite, ma il problema principale è rimasto.

2.3 Incertezza

Inizialmente i giocatori non sono a conoscenza ne del luogo in cui si trova la base nemica, ne delle intenzioni che l'avversario ha adottato. Per questi motivi tipicamente i giochi RTS riscontrano due tipi principali di incertezza, il primo è legato al fatto che il gioco è parzialmente osservabile, pertanto ogni giocatore non è in grado di osservare in ogni istante la composizione del territorio di gioco per intero ma dovrà inviare le proprie unità per esplorarlo al fine raccogliere informazioni riguardo alla posizione ed alle azioni compiute dall'avversario fino a quel momento. Utilizzando al meglio questa tecnica di esplorazione il giocatore potrà abbassare questo primo tipo di incertezza.

Il secondo tipo di incertezza deriva dal fatto che gli RTS presentano un aspetto conflittuale, ovvero non si può prevedere a priori ne il numero ne il tipo di azioni che l'avversario ha compiuto, oppure sta compiendo, in un determinato istante di tempo. Per quanto riguarda questo secondo tipo di incertezza l'intelligenza artificiale, come giocatore umano dovrà assemblare un modello ideale che rispecchia la serie di azioni che, più probabilmente, l'avversario a compiuto fino a quell'istante.

2.4 Ragionamento Spaziale e Temporale

Un ulteriore strategia fondamentale per l'aumento delle possibilità di vittoria in un RTS è rappresentata dall'analisi statica e dinamica del terreno di gioco, in particolare il ragionamento spaziale ha come obbiettivo quello di sfruttare, nel migliore modo possibile, la mappa di gioco.

Questo problema riguarda sostanzialmente due aspetti fondamentali, il posizionamento degli edifici e l'espansione della base, nel primo i giocatori dovranno valutare attentamente, nel corso della partita, la posizione degli edifici in modo tale da garantire una protezione efficace contro le invasioni nemiche, come ad esempio creare coni di bottiglia, ed allo stesso tempo costruire una sorta di barricata per proteggere le strutture più importanti. L'analisi territoriale va sostenuta anche per evitare che l'intelligenza artificiale costituisca una formazione di edifici tale da bloccare il passaggio delle unità più ingombranti.

Per valutare l'importanza del secondo aspetto si consideri la situazione in cui le risorse vicine alla base del giocatore stiano per esaurire,in questo contesto si deve introdurre il problema correlato all'espansione della base per

favorire l'acquisizione di nuove risorse più lontane in modo tale che ciò avvenga nel minor tempo possibile, a questo punto l'analisi fatta in precedenza fornirà al giocatore, tenendo conto anche della posizione del nemico, la zona migliore in cui insediarsi.

Inoltre l'analisi del territorio non trova utilizzo solamente nei due aspetti sopra elencati, bensì quando il giocatore deve affrontare una battaglia, il posizionamento delle unità è la chiave per la vittoria di uno scontro individuale, si può pensare di iniziare una battaglia quando le unità avversarie sono racchiuse in un cono di bottiglia ad esempio.

In modo analogo il ragionamento temporale è fondamentale per quanto riguarda le scelte tattiche e strategiche, per esempio ogni singola unità ha un determinato tempo di attacco ed una distanza massima, se il giocatore è abbastanza abile da far indietreggiare le unità al momento giusto egli acquisirà un vantaggio che in alcuni casi determina la vittoria della battaglia.

Oltre a questo i giocatori, al livello strategico più alto, devono effettuare una serie di azioni economiche molto costose che possono impiegare molto tempo per il loro completamento, un'evoluzione tecnologica oppure un cambiamento di strategia ad esempio, ad occuparsi di questi aspetti è sempre il ragionamento temporale.

2.5 Dominio di sfruttamento della conoscenza

In giochi strategici tradizionali come gli scacchi è stato sfruttato gran parte del dominio di conoscenza già esistente per fare una stima delle funzioni principali che l'intelligenza artificiale dovrà sfruttare per portare a termine l'obbiettivo principale del gioco, in questo caso tale dominio è rappresentato da una serie di libri e da configurazioni, a partita terminata, riscontrate nelle partite già giocate. Per quanto riguarda gli RTS, data la complessità dello spazio degli stati, non è ancora del tutto chiaro quale siano i limiti del dominio di conoscenza,sotto forma di replay e guide strategiche per esempio, pertanto risulta difficile implementare a priori tali funzionalità.

Il lavoro svolto finora per colmare questo problema ha portato i ricercatori ad intraprendere due strade principali, la prima si concentra sull'implementazione, a livello di codice, di strategie già esistenti e documentate, in modo tale che il sistema non debba far altro che decidere,sulla base delle informazioni ottenute fino a quell'istante, quale strategia adottare senza dover

effettuare ogni minima scelta strategica per quanto riguarda le singole azioni di movimento delle unità per esempio. La seconda strada invece prevede un massiccio utilizzo dei dati ricavati dai replay esistenti, da cui sono state estrapolate strategie e tendenze di gioco.

Tuttavia le situazioni, che si sviluppano durante una partita negli RTS, rappresentano una complessità tale che il concetto di imparare in maniera automatica, dato un insieme di dati, risulta un problema aperto ancora oggi.

2.6 Decomposizione delle attività

Nel mondo dell'informatica vige la regola *Divide et impera* il cui significato sostanziale riguarda la decomposizione di un problema più grande in tanti piccoli sottoproblemi la cui soluzione risulta più semplice. Anche per questo tipo di giochi la decomposizione dei problemi elencati finora, in una raccolta di problemi più piccoli, è fondamentale in fase di progettazione e più in dettaglio la suddivisione comune è la seguente:

- Strategia:
  Corrisponde al processo decisionale che troviamo al livello di astrazione più alto, in particolare questo è essenziale per la comprensione del gioco. Come accennato in precedenza, trovare una strategia efficace ed essere in grado di modificarla durante il gioco, a seconda delle informazioni ottenute, è la chiave per raggiungere l'obbiettivo primario negli RTS, questo problema riguarda anche la valutazione del complesso di unità che un giocatore possiede.

- Tattiche:
  Questo sottoproblema riguarda l'utilizzo di tattiche già esistenti, le quali implicano i movimenti di singole, o gruppi di unità, il posizionamento e la tempistica in battaglia, il posizionamento degli edifici, la tempistica con cui eseguire le determinate azioni e molto altro.

- Controllo reattivo:
  Questa parte riguarda l'implementazione delle tattiche citate nel punto precedente, sostanzialmente consiste in funzioni come ad esempio sparare, mirare, muoversi, fuggire durante una battaglia relative alla singola unità, e non al gruppo.

- Analisi del terreno:
  Consiste principalmente allo studio accurato della mappa di gioco e delle regioni che la compongono le quali possono essere isole basse o alte(se un'unità si trova su un'isola bassa, il giocatore non avrà visibilità su quella alta ad esempio) , strozzature, terreni percorribili dalle unità e così via.

- Raccolta di informazioni:
  Riguarda le informazioni raccolte fino ad un dato istante di tempo sull'avversario, come citato in precedenza gli RTS sono caratterizzati dalla nebbia di guerra ovvero il gioco è parzialmente osservabile, pertanto i giocatori dovranno inviare periodicamente unità esploratrici per localizzare e spiare le basi nemiche, questo aspetto facilita la scelta strategica da effettuare, il giocatore esperto solitamente effettua attacchi periodici non solo per danneggiare la base nemica ma anche per raccogliere informazioni sui tipi di unità il nemico ha prodotto.

Se mettiamo a confronto un tipico approccio umano a questi giochi strategici ci accorgiamo che il processo decisionale è totalmente differente, in particolare questo viene suddivido un due voci principali:

- Micro:
  Rispecchia la capacità di controllare singolarmente le unità, per questo in parte corrisponde alle voci Controllo reattivo e Tattiche sopra citate, solitamente un giocatore che presenta un approccio di questo tipo molto sviluppato tenderà a mantenere in vita il più possibile le proprie unità.

- Macro:
  E' l'abilità di produrre costantemente unità ed espandersi con il giusto tempismo per evitare di rallentare tale produzione, un giocatore più sviluppato in questo ambito solitamente è in grado di ottenere l'esercito più grande.

Una delle sfide significative riscontrate negli RTS è rappresentata dalla progettazione di architetture che mirano, come scopo principale, a rendere possibile la comunicazione e l'interazione in modo efficace fra le singole tecniche di intelligenza artificiale che si rivolgono alla risoluzione delle attività sopra citate in modo efficace, in grado di prevedere e risolvere inoltre conflitti interni.

Si vuole far notare infine che la decomposizione delle attività non è l'unico approccio possibile ma esistitono altre tecniche come ad esempio una valutazione dell'assegnamento delle risorse sulle singole attività che tiene conto dei vantaggi che questa può portare.

Capitolo 3

Possibili soluzioni ai problemi

Per progettare un sistema di intelligenza artificiale che riesca a giocare una partita ad un RTS è fondamentale che questo affronti la maggior parte, se non tutti, dei problemi citati nel capitolo precedente. In particolare per raggiungere in maniera efficace questo obbiettivo si è deciso di suddividere il lavoro[1] in tre principali livelli di astrazione (si veda la Fig. 1):

- Strategia(che corrisponde vagamente alla voce Macro)
- Tattica
- Controllo reattivo(che corrisponde vagamente alla voce Micro)

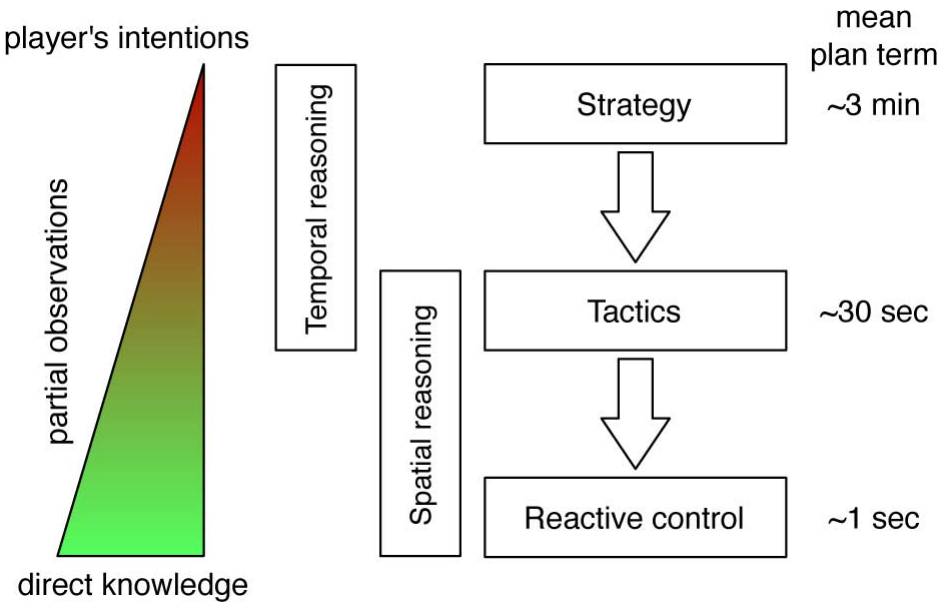


Figura 1.

L'immagine mostra come i tre livelli sono collegati e qual è la scala gerarchica che il sistema deve seguire. In particolare al livello più alto troviamo la strategia, che corrisponde ai processi decisionali a lungo termine, mentre il livello costituito dal controllo reattivo corrisponde alle decisioni, secondo per secondo, di più basso livello.

Inoltre il livello strategico si sofferma sul ragionamento che considera le informazioni dell'intero gioco in una sola volta, mentre le decisioni degli altri due livelli sono localizzate ed interessano solamente specifiche unità, o gruppo. Tipicamente le decisioni strategiche pongono dei vincoli sulle decisioni tattiche future, che a loro volta pongono delle condizioni al controllo reattivo; inoltre le informazioni raccolte dall'esecuzione dei comandi imparatiti dall'ultimo livello possono provocare una riconsiderazione delle tattiche utilizzate, che a loro volta possono causare una modifica nel ragionamento strategico.

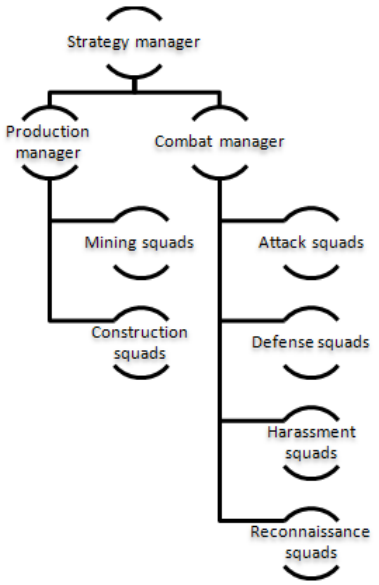
Seguendo questo principio, al livello strategico viene attribuito tutto ciò che riguarda lo sviluppo tecnologico, l'ordinamento con cui costruire gli edifici e la composizione dell'esercito. Questo rappresenterà il livello più risolutivo, si consideri come esempio un giocatore che mette in atto una strategia sulla futura presa di posizione aggressiva avendo bene in mente la tattica da seguire.

Al livello di tattica invece verrà retribuito tutto ciò che riguarda lo sviluppo di battaglie fra unità, in particolare questo livello si occuperà di gestire il ragionamento spaziale,ovvero lo sfruttamento territoriale, ed il ragionamento temporale, ovvero i movimenti delle unità, e sarà vincolato da tutti i possibili tipi di attacchi subiti e dalla composizione dell'esercito nemico.

Il livello di controllo reattivo descrive come il giocatore controlla individualmente le proprie unità per massimizzare l'efficienza in tempo reale, ciò che distingue questo livello da quello tattico è che quest'ultimo effettua ragionamenti che in genere comportano una pianificazione delle azioni future in tempi brevi, mentre il controllo reattivo non prevede alcuna pianificazione.

Un esempio del lavoro svolto dai tre livelli è il seguente, si consideri una generica partita appena iniziata; il giocatore decide di scegliere una strategia che implica la rapida costruzione di un esercito per poi inviarlo ad attaccare il nemico nel minor tempo possibile, poi in fase di attacco egli deciderà di utilizzare una tattica che mira a circondare il nemico così da non lasciargli possibili vie di fuga. Inoltre mentre l'attacco è in corso il giocatore potrebbe utilizzare tecniche di controllo reattivo che comandano le unità, in modo

tale da infliggere un colpo e scappare ripetutamente, al fine di massimizzare l'efficienza dell'attacco.



L'immagine [6] mostra la suddivisione gerarchica delle entità presenti in un agente atto alla gestione dei problemi in un RTS, andrò ora ad effettuare lo zooming sui tre livelli sulla base di questa architettura.

3.1 Strategy Manager

Nei giochi RTS questo rappresenta sicuramente l'aspetto più importante, indipendentemente da quanto un giocatore sia abile nel comandare le singole unità sul campo di battaglia, se la strategia scelta si rivela inferiore a quella adottata dall'avversario, le possibilità di raggiungere l'obbiettivo del gioco diminuiscono drasticamente.

In fase di progettazione, questo elemento dovrebbe risultare uno dei più ardui da pensare, infatti la parte di sviluppo della strategia e l'implementazione su agenti si è rivelata, a causa della complessità, una delle più lente. Gli esseri umani utilizzano spesso un approccio strategico intuitivo basato sulla valutazione della situazione attuale di gioco. Un giocatore esperto ad esempio riesce, solamente guardando il tempo di gioco, a determinare, con elevata precisione, una stima sulla composizione delle unità di qualche mi-

nuto più tardi, e se la strategia utilizzata risulta completamente inefficace, riuscirà inoltre a determinare l'esito della partita.

Per ottenere tale risultato, un agente strategico richiederebbe una banca dati di tutte le strategie esistenti assieme ad un algoritmo atto a selezionare quella giusta a seconda dei casi, senza tener conto poi del fatto che le strategie continuano ad evolvere, se ne aggiungono altre e così via. Si noti anche che, al momento del rilascio del gioco, la banca dati non esisteva e per tale motivo si è deciso di sviluppare una strategia di base.

Prima di proseguire, si vuole fare inoltre la distinzione tra due principali tipi di agenti, quelli che barano e quelli che non barano. Il primo tipo riguarda gli agenti che non giocano una partita equilibrata in quanto prevedono l'utilizzo di tutte le informazioni del gioco, ovvero per tutta la durata della partita conoscono la posizione, la strategia adottata, e la quantità di risorse ed unità accumulate dell'avversario. Questo corrisponde ad un enorme vantaggio in quanto non vi è il bisogno di effettuare continue esplorazioni e previsioni sulla strategia adottata fino ad un dato istante di tempo, rende inoltre impossibile l'utilizzo di una strategia falsa in modo da nascondere gli elementi strategici principali. In questo articolo ci si concentrerà principalmente sulla seconda categoria di agenti, ovvero quelli che simulano l'approccio umano a questo tipo di giochi.

Il primo passo che possiamo fare verso la progettazione dello *Strategy Manager* è quello di considerare le differenze fra le varie razze/fazioni scelte dai giocatori in fase iniziale, in quanto vengono scelte dalla macchina casualmente. Questa separazione permette un design più pulito in modo tale che il sistema prenda in considerazione solamente decisioni strategiche specifiche alle varie tipologie di razze/fazioni in gioco nel corso della partita.

La scelta di una strategia specifica viene fatta all'inizio di una partita e va immediamente a tradursi in un numero di ordini di produzione, questo tipo di approccio provoca come conseguenza la necessità di avere, all'interno di questo livello, un meccanismo che permette al sistema di rappresentare lo stato del gioco, il quale deve essere aggiornato continuamente. Per ottenere tale risultato si hanno sostanzialmente due modi per manipolare le informazioni a disposizione dello *Strategy Manager*, ovvero attraverso l'accesso diretto allo stato del gioco oppure attraverso l'interpretazione delle notifiche di eventi.

Per esempio ogni volta che un'unità diventa visibile viene scatenato un evento ed il livello strategico riceverà una notifica contenente l'informazione sul tipo di unità, altri eventi comportano la distruzione, la scomparsa e così via. Altrimenti un modo per rappresentare lo stato del gioco può comprendere l'utilizzo di un vettore contenente componenti fra cui il tempo trascorso, le risorse occupate, quelle acquisite, il numero di unità presenti ed alcune variabili sullo stato di gioco dell'avversario come ad esempio una stima delle dimensioni del suo esercito, è possibile memorizzare anche variabili statiche come ad esempio le dimensioni della mappa di gioco ed, al fine di evitare la prevedibilità, si possono introdurre variabili casuali.

Sfruttando tali informazioni questo livello può quindi eseguire azioni strategiche una volta che sono state soddisfatte le condizioni basate sullo stato del gioco. Ovviamente tentare una strategia che consiste nell'assemblare velocemente un esercito per poi inviarlo il prima possibile alla base nemica in una mappa molto grande è meno ragionevole e comporta una diminuzione della probabilità di successo in quanto risulta meno probabile che le unità raggiungano la base nemica prima che le difese siano state innalzate.

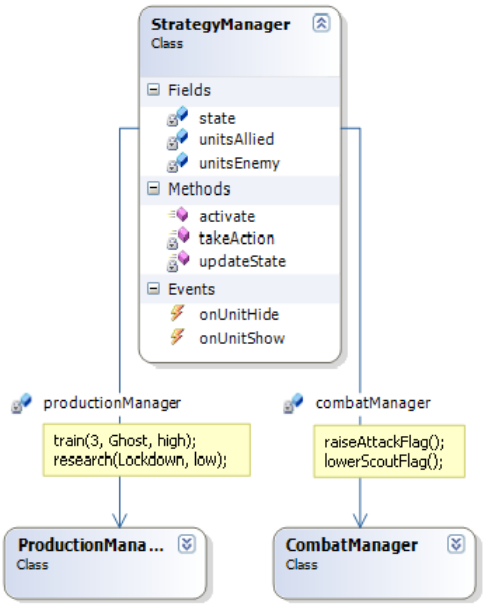
Un tipo di azioni strategiche eseguite da questo livello sono quelle legate alla produzione, più nello specifico al *Production Manager*, le quali modificano lo stato di produzione che a sua volta sarà composto da una serie ordinata di azioni di produzione formate da parametri come ad esempio la priorità, il tipo di unità e la quantità da produrre. Inoltre il caso di revoca dell'ordine di modifica può essere gestito utilizzando simili direttive.

Lo *Strategy Manager* può pianificare in anticipo ed inviare ordini di produzione con differenti priorità al livello sottostante, e sarà in grado anche di bloccare gli ordini in esecuzione per favorire il rapido completamento di quelli più prioritari. Quando poi gli ordini più prioritari saranno terminati verra ripristinata la coda precedente. Il meccanismo appena spiegato consente all'agente di realizzare modifiche sulle unità da produrre in base a quelle dell'avversario senza dover modificare completamente la strategia adottata, in tal modo il sistema risulta più adattabile a fronte di comportamenti imprevisti.

Un ulteriore tipo di azioni compiute da questo livello, oltre a quelle di produzione, sono le azioni di combattimento che vanno a modificare lo stato interno del *Combat Manager* il quale interpreta il tipo di comando impartito utilizzando principalmente variabili di tipo booleano. In particolare se il livello strategico decide di sferrare un assalto, l'azione verrà trasmessa al

Combat Manager tramite la modifica dell'opportuna variabile, che si traduce nell'acquisizione da parte delle unità di un nuovo obbiettivo, sia nel caso in cui queste ne abbiano già uno oppure no. In modo analogo può avvenire una richiesta di difesa oppure una richiesta di esplorazione, tramite le opportune variabili, quando l'agente è sotto attacco oppure quando è necessario l'aggiornamento delle informazioni raccolte sull'avversario.

L'immagine sottostante mostra una possibile struttura dello *Strategy Manager*:



3.2 Production Manager

Il processo di produzione non risulta un'attività semplice in questo tipo di giochi, come già accennato in precedenza questo livello si sofferma sulla gestione di risorse, l'evoluzione tecnologica ed i vincoli territoriali, in particolare si consideri che per acquisire la possibilità di produrre una specifico tipo di unità devono essere soddisfatte tutte le condizioni necessarie, sotto forma di edifici e/o avanzamenti tecnologici.

Inoltre, solitamente gli edifici atti alla produzione di unità sono in grado di eseguire una richiesta per volta, pertanto si consideri la produzione consecutiva di cinque unità utilizzando un'unico edificio per esempio, nello stato interno di questo livello si verrà a creare una coda di produzione che blocca la quantità di risorse pari al costo di tutte le unità ma non i rifornimenti. Si noti inoltre che secondo questo tipo di architettura, spetta allo *Strategy Manager* decidere se più edifici di questo tipo sono necessari.

Un altro esempio di funzionalità sostenute da questo livello è costituito dai vincoli di produzione legati al terreno di gioco; quando viene impartito un ordine di costruzione ad un'unità, specializzata in questo tipo di compiti, può venire inoltre indicata una locazione specifica, tenento conto del fatto che non è possibile costruire su tutto il terreno di gioco. Pertanto escludendo a priori la possibilità di costruire strutture sovrapposte, si devono prendere in considerazione altre limitazioni di costruzione correlate al terreno che spesso determinano intere regioni di spazio.

Nel caso in cui la locazione di costruzione dell'edificio non venga indicata sarà compito di questo livello trovare l'adeguata soluzione, un modo semplice per raggiungere tale scopo consiste nell'effettuare una scansione, tipicamente a spirale, del terreno di gioco fino a quando non è stata trovata una posizione valida. Questo metodo risulta utile ad esempio quando si posizionano strutture difensive in quanto il livello stategico dovrà solamente indicare la posizione in cui il *Production Manager* dovrà costruire il primo, di conseguenza per fare in modo di creare una zona altamente difensiva, per quanto possibile, basterà impostare la locazione di partenza della scansione uguale a quella del primo edificio.

Anche se le attività di produzione vengono decise dal livello più alto, il *Production Manager* deve risolvere alcuni problemi di fondo come l'ottimizzazione economica, la costruzione ed il collocamento delle strutture, pertanto il suo ruolo principale consiste nel coordinare tali attività in modo tale che la produzione risultante sia coerente con le esigenze dello *Strategy Manager*. Tali attività risultano prevalentemente di tipo meccanico, anche se alcune come il posizionamento di una struttura richiede un minimo di intelligenza, quindi lo sviluppo implementativo di questo livello risulterà più semplice.

L'obbiettivo primario di questo livello è quello di produrre unità ed edifici nell'ordine stabilito dal livello strategico attraverso l'utilizzo di un'apposita coda di attività in cui ogni singolo elemento comprende un meccanismo, attraverso il quale il sistema comprende la priorità di esecuzione di tale attività.

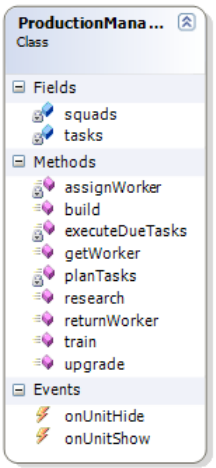
In questo modo risulta impossibile che un'attività di bassa priorità rallenti il complemento di una di alta priorità, la motivazione primaria è che i tempi di produzione non sono istantanei.

Una volta stimato il tempo di produzione per il completamento di un'attività e l'ordine con cui queste devono essere eseguite, egli deve quindi distribuire le risorse acquisite e future, un modo efficace per raggiungere tale scopo è quello di distribuirle in base alla priorità in ordine decrescente, per effettuare una stima delle risorse future basta considerare quelle acquisite per frame.

Nel caso in cui una specifica attività non possa essere eseguita a causa di una limitazione tecnologica il *Production Manager* dovrà provvedere all'acquisizione di quest'ultima per proseguire con il completamento dell'attività, ad esempio interrompendo la produzione e rimanendo in attesa di aver portato a termine diverse attività secondarie con la stessa priorità.

Al fine di ottimizzare l'acquisizione di risorse a seconda dell'attività da svolgere, questo livello si occupa anche di distribuire le unità, in sostanza l'obbiettivo di quest'ultime viene costantemente riassegnato a seconda delle necessità. Notiamo inoltre che anche il *Production Manager* riceverà le notifiche, da parte del livello strategico, riguardo la comparsa e scomparsa delle unità in modo tale da ottimizzare l'aspetto appena introdotto.

L'immagine sottostante mostra una possibile struttura del *Product Manager*:



3.3 Combat Manager

Nei giochi RTS uno degli aspetti più difficili, da controllare a livello umano, è senza dubbio la gestione delle battaglie fra unità in quanto richiede una solida organizzazione ed alta reattività. Risulta impossibile gestire interi gruppi di unità senza un'organizzazione, man mano che queste verranno prodotte il giocatore si troverà a controllare diverse unità di tipi differenti e dovrà effettuare alcune scelte per quanto riguarda la formazione da esibire sul campo di battaglia, avverrà quindi una suddivisione in gruppi.

In aggiunta a queste considerazioni il giocatore dovrà tener conto delle variazioni del terreno e reagire opportunamente, in quanto è possibile che queste siano sia a suo vantaggio che svantaggio, come esempio si consideri il combattimento fra unità aeree ed unità terrene, in questo caso se il terreno presenta variazioni d'elevazione, le unità terrene avranno scarse possibilità di successo.

Il ruolo del *Combat Manager* quindi è quello di trasmettere decisioni strategiche, relative alla battaglia, alle unità militari assieme alla gestione e formazione dei gruppi a seconda dei tipi di unità a disposizione. L'assegnazione corretta di unità agli appositi gruppi è un'operazione delicata che può aumentare notevolmente le performance di un agente quando viene effettuata in modo corretto, in molti casi l'utilizzo delle unità in un'unico gruppo può portare a risultati catastrofici e di coseguenza la qualità di questo livello è una caratteristica essenziale in fase di progettazione.

Pertanto il comportamento del *Combat Manager* dovrà essere in gran parte influenzato dal livello strategico, se così non fosse il comportamento complessivo risultante sarebbe irregolare. Inoltre dato che questo livello è a conoscenza delle informazioni legate alla composizione dei gruppi dovrebbe essere in grado di determinare, prima e durante una battaglia, se questa avrà successo o meno. Eppure, lasciando tali decisioni al *Combat Manager* non sono stati riscontrati buoni risultati, in quanto tale esito non dipende solo dai fattori descritti.

Questo tipo di decisioni sono strettamente legate alla strategia adottata, di cui questo livello non è a conoscenza, ad esempio se il primo livello decide di adottare una strategia difensiva senza effettuare attacchi, al fine di portarsi in vantaggio sviluppando una consistente macrogestione, il lancio di un attacco da parte del *Combat Manager* a causa di stima basata sulla potenza militare attuale risulterebbe un'azione incoerente.

Si può notare quindi che tali decisioni non sono basate unicamente sulla composizione dei gruppi ed i dettagli risultano irrilevanti, pertanto sarà lo *Strategy Manager* ad occuparsi della gestione e la temporizzazione degli attacchi. Questo livello, tuttavia, può organizzare attacchi specificando la zona e controllando la composizione della squadra, inoltre sarà lui a controllare l'acquisizione del bersaglio di ogni singola unità e dividerle, per ottenere un vantaggio tattico, in zone.

Questo può essere utilizzato per attaccare una base nemina su due fronti ad esempio, oppure per attaccare due basi allo stesso tempo quando le forze militari del nemico sono concentrate unicamente nella base principale, tale metodo viene usualmente utilizzato dai giocatori esperti per limitare l'acquisizione di risorse dell'avversario distruggendo un punto di raccolta, l'attacco congiunto serve per ritardare l'invio di rinforzi nemici.

Ovviamente le informazioni in possesso al *Combat Manager* devono essere costantemente aggiornate in quanto, un gruppo di unità che sta raggiungendo la base nemica per sferrare un attacco, potrebbe incontrare, durante il tragitto, unità nemichei che non possono essere ignorate, pertanto l'acquisizione del bersaglio deve essere rinnovata o almeno adattata per includere tali unità. In tal modo questo livello è in grado di decidere se un gruppo di unità può di perseguire con successo l'obbiettivo che gli è stato assegnato o meno.

L'assegnazione di un un'unità ad un gruppo può essere divisa in due livelli, il primo riguarda l'attribuzione di un compito a quest'ultima, i gruppi vengono composti a seconda delle attività, ciò comporta che unità dello stesso gruppo svolgano la stessa attività. Quindi una volta settata l'attività, nel secondo livello, avviene l'assegnazione dell'unità al gruppo al quale questa deve appartenere. Per effettuare questo compito il *Combat Manager* utilizza delle variabili di tipo booleano che possono essere sovrascritte anche dal livello strategico. Può capitare inoltre il caso in cui ad un'unità vengano assegnate due attività nello stesso istante, in questo caso l'agente può utilizzare una regola per evitare tale fenomeno considerando ad esempio l'affinità di un'unità nello svolgere un compito.

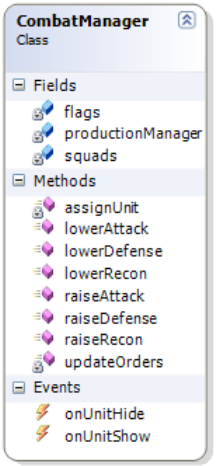
Le configurazioni dei gruppi non sono statiche e devono essere continuamente aggiornate in base alle variabili di stato e dall'arrivo di nuove unità, se consideriamo inoltre l'affinità, si può pensare di raggruppare unità affini e di conseguenza questo livello non dovrà far altro che gestire l'assegnazione dei compiti sul gruppo anzichè sulle singole unità.

Il *Combat Manager* deve, data un'attività, gestire solamente i gruppi che effettivamente riescono a portare a termine quest'ultima, se consideriamo ad esempio due squadre, una sta attaccando la base nemica nella parte opposta della mappa e l'altra si trova in base, nel caso in cui l'agente riceva, nello stesso istante un attacco nemico, per difendere la base dovrà essere impiegato solamente il gruppo che si trova più vicino. Pertanto se questo livello rivaluta le configurazioni di gruppi che si trovano in un dato perimetro vicino alla base, i gruppi attaccanti non verranno richiamati, tale comportamento si avvicina a quello umano ed in alcuni casi risulta più efficiente. Se un gruppo ha subito troppi danni, i sopravvissuti vengono riassegnati ad altri gruppi.

Un ulteriore compito svolto da questo livello è la gestione delle unità lavoratrici sul campo di battaglia, in particolare può sollevarsi la necessità di ripare edifici oppure costruirne di tipo difensivo durante una battaglia. Dato che questo tipo di unità sono sotto il controllo del *Production Manager* in caso di necessità, non appena vengono prodotti, questo livello dovrà comunicare una richiesta per prenderne possesso prima che l'unità possa essere utilizzata per altri scopi.

Quest'ultimo caso particolare non risulta essenziale per la progettazione dell'agente, è utile però al fine di comprendere fino a che punto devono poter comunicare fra loro entità dello stesso livello.

L'immagine sottostante mostra una possibile struttura del *Combat Manager*:



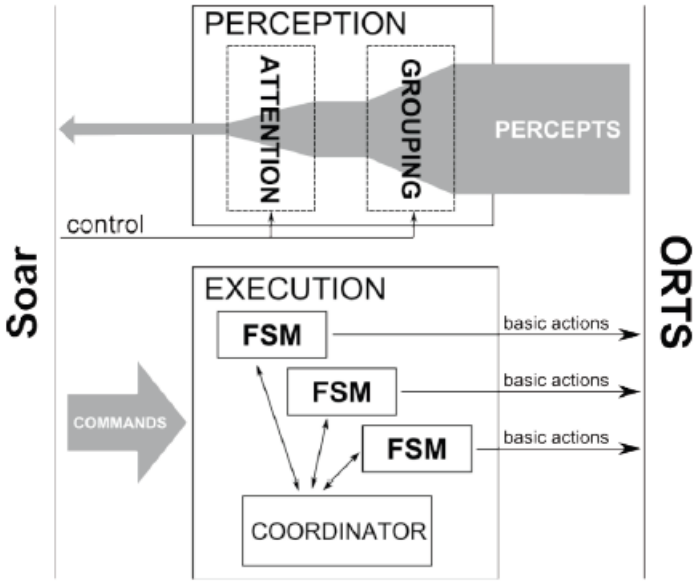
Capitolo 4

Intelligenza Artificiale a livello umano

Questo capitolo introduce due architetture basate su intelligenza artificiale per giochi RTS la cui progettazione mira alla composizione di un sistema con caratteristiche simili al ragionamento umano e che sfrutta le informazioni a cui può accedere un giocatore umano. SORTS è un'architettura cognitiva che si basa sull'astrazione dello stato del gioco, la pianificazione di attività e la suddivisione dei compiti principali. EISBot a differenza di quest'ultima suddivide le sfide proposte da questo tipo di giochi in moduli distinti che si focalizzano maggiormente sul coordinamento delle attività.

4.1 SORTS

Questo tipo di architettura[4] prevede l'utilizzo di ORTS come motore di gioco, ed un middleware che si pone fra questo componente e l'intelligenza artificiale chiamata *Soar*, il cui scopo è quello di rendere possibile la comunicazione fra le due entità. In particolare *Soar* è un'entità contenente un insieme di agenti che, inizialmente, sono stati progettati per affrontare i problemi legati ai giochi sparatutto in prima persona, è stata quindi necessaria l'implementazione di un nuovo agente in grado di gestire giochi RTS.



Questa immagine mostra come viene organizzato SORTS, come già detto in precedenza ORTS è il motore di gioco e *Soar* è il motore di intelligenza artificiale, tutto ciò che si pone fra queste due entità è il middleware che viene suddiviso in *Perception* ed *Execution*, la prima parte contiene le entità di *Attention* e *Grouping* che vengono direttamente controllate da *Soar* mentre la seconda parte si occupa dell'esecuzione di comandi, impartiti sempre da questa entità, che vengono interpretati attraverso l'utilizzo di macchine a stati finiti atte allo svolgimento di azioni primitive in ORTS.

4.1.1 ORTS

Rappresenta un motore di gioco per RTS open source che venne sviluppato dall'Università di Alberta, il quale è stato progettato da zero al fine di essere utilizzato nella ricerca in questo settore ed offre, a basso livello, una serie di

API che consentono un approccio client-server sicuro per ottenere una solida competizione su internet. Questo componente favorisce inoltre la modifica delle meccaniche di gioco, pertanto è in grado di simulare la maggior parte dei giochi RTS sul mercato.

L'architettura di ORTS presenta diverse sfide nello sviluppo di un'intelligenza artificiale in quanto è stato adottato un approccio minimalista che porta il motore grafico a simulare solamente l'aspetto fisico del gioco, lasciando così funzionalità complesse, come la gestione delle unità, al programma utente. Le API discusse precedentemente sono inoltre in grado di fornire informazioni di tipo percettivo all'intelligenza, ad ogni battito dell'orologio di gioco lo stato di ogni oggetto presente sulla mappa che è cambiato viene inviato.

4.1.2 Soar

Questo componente incapsula l'agente di intelligenza artificiale utilizzando un'architettura che permette la codifica della conoscenza procedurale a lungo termine in regole di produzione ed è in grado di rappresentare lo stato attuale in una memoria di lavoro dichiarativa. Grazie a questa suddivisione *Soar* possiede inoltre la capacità di mandare in esecuzione gli ordini di produzione in parallelo quando le regole coincidono.

Il comportamento viene organizzato in cicli decisionali in cui prima di tutto viene elaborata la situazione attuale dove, sempre attraverso l'utilizzo di regole, vengono sfruttati pattern per la gestione delle notifiche e strutture dati in grado di incapsulare le attività rilevanti come ad esempio "il lavoratore inviato ad esplorare ha terminato". Vengono inoltre utilizzate regole supplementari chiamate **operatori** per verificare lo stato attuale e proporre lo svolgimento di attività alternative, alcune delle quali prevedono azioni motorie nell'ambiente, come ad esempio la costruzione di una struttura, mentre altre modificano le strutture dati interne, se un'unità riceve il comando di costruire un'edificio questa informazione va memorizzata.

Se un **operatore** non può essere eseguito in maniera diretta, questo diventa un obbiettivo e viene decomposto in operatori più semplici introducendo così una pila di obbiettivi. *Soar* è in grado di selezionare ed applicare un solo **operatore** alla volta, anche se questo può comportare più attività, pertanto nel sistema sarà presente solamente una pila di obbiettivi, limitandolo così a seguire un'unica linea di pensiero.

Alcuni svantaggi nell'utilizzo di un'architettura che vuole mantenere separato questo componente sono:

- Lo spostamento di una grande quantità di dati dal componente di **percezione** del middleware a *Soar* risulta molto costoso.
- Questo componente non ha la capacità di astrazione visiva o filtraggio che hanno gli esseri umani.
- Essendo un sistema basato sul ragionamento non è in grado di effettuare un'elaborazione numerica complessa, diversamente dagli esseri umani.

4.1.3 Difficoltà riscontrate nell'interfacciamento

Esperienze preliminari nella modellazione di *Soar* per i giochi sparatutto in prima persona hanno messo in evidenza alcune difficoltà riguardo l'interfacciamento di questa entità con l'ambiente descritto da ORTS, in particolare il middleware deve fornire all'agente lo stesso tipo di informazioni che un giocatore umano è in grado di percepire, non sotto forma di pixel ad esempio, ma in termini di astrazione che hanno gli umani subito dopo aver percepito tali informazioni.

Per esempio un giocatore umano è in grado di percepire gruppi di unità ma, in un istante di tempo, è capace di focalizzare la propria attenzione solamente su un sottoinsieme di queste, più in generale si focalizza un sottoinsieme di informazioni percepite e non è in grado di controllare il flusso completo. Pertanto l'agente deve poter comandare le unità rispettando gli stessi vincoli, potrà di conseguenza impartire comandi ad un'unico gruppo di unità alla volta.

Seguendo questi principi il middleware deve disporre di comandi di basso livello in grado di impartire ordini alle unità, come la pianificazione del percorso e combattimento, mentre *Soar* dove disporre di comandi di alto livello, come "vai a questa locazione" oppure "combatti questo nemico".

4.1.4 Perceptual System

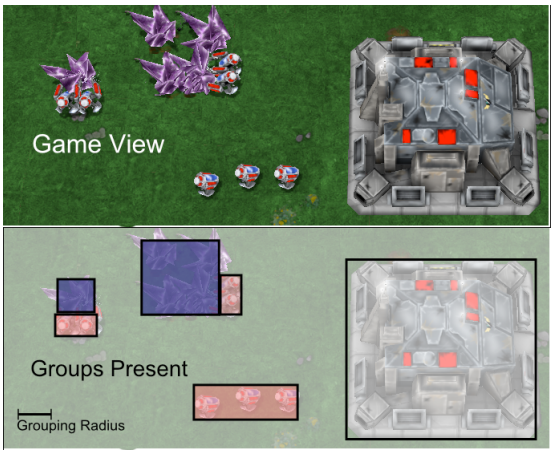
Questo componente ha lo scopo di ricevere i dati riguardanti lo stato del gioco, ricevuti dal server ORTS, per poi creare apposite strutture dati che andranno ad occupare la memoria di lavoro di *Soar*. In particolare le informazioni che vengono ricevute riguardano lo stato ogni singolo oggetto presente

nel gioco, per ogni frame, e se consideriamo che in un tipico gioco RTS possono esserci centinaia di oggetti il cui stato potrebbe essere cambiato risulta chiaro che la mole di dati gestita da questo sistema è enorme.

Il compito principale di questo componente consiste in un'operazione di filtraggio delle informazioni percepite, in modo tale da fornire a *Soar* solamente quelle percepibili a livello umano, pertanto il middleware è in grado di supportare due operazioni chiamate *Grouping*, che riassume le informazioni riguardanti i singoli oggetti, ed *Attention*, che esclude le informazioni non necessarie. Utilizzando entrambe queste operazioni il sistema è in grado di fornire un livello di astrazione tale da permettere al ragionamento tattico di interpretare le informazioni risultanti.

Grouping

Lo sviluppo di questa funzionalità è stata ispirata dalla capacità umana di vedere un insieme di oggetti simili come un insieme unitario, che gli studiosi psicologi hanno chiamato *Raggruppamento Gestalt*. I principi di questa teoria specificano che, se gli oggetti sono vicini in termini di spazio e presentano caratteristiche comuni come la forma e il colore, allora questi possono essere percepiti come un unico gruppo. Modellando questo concetto in fase di progettazione è stato possibile introdurre la capacità di percepire oggetti ed unità raggruppati per tipo, proprietario e vicinanza e si è deciso di utilizzare, come impostazione predefinita, tutte e tre queste informazioni per comporre la regola di raggruppamento. In sostanza se unità dello stesso tipo e proprietario si trovano in una zona di terreno che rientra nel raggio di raggruppamento, definito dall'agente, allora queste fanno parte dello stesso gruppo.



L'immagine sopra indicata mostra un esempio di raggruppamento in cui sette unità lavoratrici, cinque risorse minerarie ed un edificio vengono tradotte in tre gruppi di unità lavoratrici, due gruppi di risorse ed un gruppo di edifici. Si noti inoltre che utilizzando tale metodo se si vogliono organizzare gruppi contenenti le unità singolarmente, basterà impostare il raggio di raggruppamento a zero.

Una volta inserita un'unità nel rispettivo gruppo, attributi come la salute ed i danni vengono riassunti ed attribuiti all'intero gruppo, non più alla singola unità. Questa, rappresenta l'informazione che viene inviata direttamente a *Soar*, in particolare le informazioni sulle singole unità vengono inviate soltanto se queste formano gruppi indipendenti, un agente può quindi diminuire la quantità di informazioni percepite aumentando il raggio di raggruppamento, questo però influirà sul livello di dettaglio di quest'ultime.

La manipolazione del raggio di raggruppamento fornisce un ulteriore meccanismo al sistema che lo rende capace di regolare il livello di ragionamento, se un agente vuole gestire le unità a livello *Micro* non dovrà far altro che impostare il raggio a zero e ragionare sulle singole unità, mentre se desidera avere accesso alla distribuzione generale sul terreno di gioco imposterà un elevato raggio di raggruppamento.

Un ulteriore vantaggio introdotto da queste considerazioni è che l'agente può utilizzare le stesse regole per ragionare su situazioni specifiche le cui informazioni differiscono solamente a livello di dettaglio, ad esempio una regola che permette di riconoscere una manovra di avvicinamento di un avversario può essere applicata alle singole unità oppure a gruppi a seconda del raggio di raggruppamento applicato.

Attention

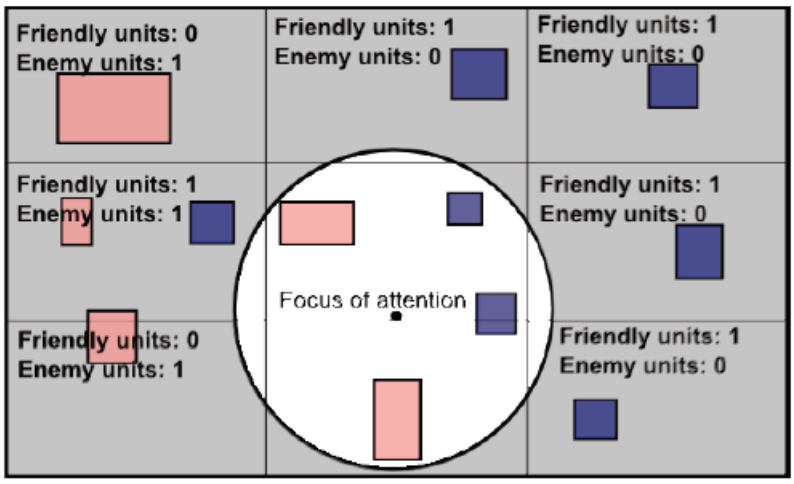
Anche dopo aver applicato la tecnica di raggruppamento la quantità di informazioni risulta essere ancora eccessiva per un'efficiente elaborazione da parte dell'intelligenza artificiale, pertanto si è deciso di introdurre un'ulteriore funzionalità che evita il passaggio di informazioni di scarsa importanza. Dato che gli eventi tattici significativi nei giochi RTS tendono a verificarsi in un'area di terreno ristretta, vale la pena concentrare la percezione di informazioni soltanto in quest'ultima limitando così quelle riguardanti il resto della mappa.

Anche in questo caso si è deciso di prendere spunto dal comportamento umano, più in particolare dal sistema visivo che ci permette di focalizzare l'attenzione effettuando uno "zoom" su un'area più piccola rispetto al nostro campo visivo totale consentendoci di raccogliere informazioni più dettagliate sull'area in questione e meno dettagliate sul resto. L'area ristretta su cui ci focalizziamo può essere spostata in modo guidato, per esempio ad un essere umano risulta semplice focalizzare la propria attenzione su di un oggetto rosso se tutto intorno vi è una distesa di colore nero.

Per spiegare il fenomeno appena introdotto, viene utilizzato un modello chiamato *FIT*[1] il cui concetto di base è che gli oggetti incustoditi sono disponibili solo attraverso caratteristiche come il colore o la forma, ma queste non vengono integrate insieme nei singoli oggetti. Se consideriamo l'esempio precedente dell'oggetto rosso situato su una distesa nera si nota che l'unica informazione percepita dall'occhio umano è l'esistenza di qualcosa di rosso, ma le informazioni più dettagliate, riguardo la forma dell'oggetto ad esempio, possono essere integrate solamente quando l'attenzione si focalizza su di esso. Inoltre non vi è il bisogno di effettuare una ricerca se nel range visivo su cui è focalizzata l'attenzione è presente solamente un oggetto rosso, ma se vi sono più oggetti di questo colore ed è necessario trovarne uno in particolare allora bisognerà focalizzare l'attenzione su ciascun oggetto individualmente.

Per implementare questi concetti negli RTS è stato utilizzato un campo visivo rettangolare ridimensionabile sulla mappa di gioco in modo che tutte le informazioni al di fuori di questo campo non vengano considerate, pertanto la "lente di zoom" risulta un punto mobile che focalizza l'attenzione solamente sul campo visivo in cui tutti i gruppi di unità che rientrano nel range della lente verranno percepiti nel dettaglio. Il rettangolo descritto viene suddiviso ulteriormente in nove settori formando così un layout a griglia e le informazioni di ogni gruppo, situato al di fuori della lente di zoom, vengono poi riunite in una mappa caratteristica di ciascun settore della griglia in modo tale da effettuare un conteggio dei gruppi, specifico di ogni settore. Questa suddivisione permette a *Soar* di effettuare, in base ai gruppi presenti nel campo visivo, ricerche efficienti e di limitare la dimensione delle informazioni di percezione ricevute da quest'ultimo, tutto basandosi sui conteggi effettuati.

[1] Acronimo di Featured Integration Theory un modello sviluppato da Anne Treisman e Garry Gelade nel 1980 sulla teoria dell'attenzione.



L'immagine mostra il campo visivo percepito dal sistema di attenzione, in particolare in questo istante l'attenzione viene focalizzata sui quattro gruppi che rientrano nella lente di zoom, pertanto *Soar* riceverà informazioni dettagliate su di loro, mentre quelle sui gruppi rimanenti vengono riassunte. L'agente inoltre può spostare l'attenzione selezionando uno dei gruppi che non rientrano nella lente di zoom grazie alle caratteristiche di settore.

Se il campo di visione risulta più piccolo dell'intera mappa allora vi è la possibilità che ci siano gruppi al di fuori di esso, ma data la suddivisione fissa effettuata sul campo, una riduzione di visibilità comporta un aumento della risoluzione sui settori. Questa situazione particolare può essere utile nell'identificazione di un'unità nemica situata in un luogo insolito, mentre se il campo visivo comprendesse sempre l'intera mappa si otterrebbe uno svantaggio, se ad esempio il nemico ha posizionato le unità in molti luoghi è probabile che ci sia almeno un'unità nemica da qualche parte in ogni settore.

Inoltre se uno di questi settori contiene una zona che l'agente sta cercando di controllare, non vi è un rapido modo per sapere se al suo interno sono presenti unità nemiche, nonostante ciò se si limita il campo di visione alla zona in questione verranno opportunamente aggiornate e comunicate le informazioni a *Soar*.

4.1.5 Execution System

Questa entità è in grado di ricevere comandi da *Soar*, interpretarli adeguatamente attraverso macchine a stati finiti e poi inviarli opportunamente al motore di gioco. In particolare le unità in ORTS sono in grado di riceve comandi di base come muoversi in linea retta oppure attaccare, se per esempio ad un'unità viene impartito il comando di attaccare un bersaglio che è fuori dalla sua portata di tiro, questa non si sposterà autonomamente verso il bersaglio prima di eseguire l'azione di attaccare.

Nei giochi RTS commerciali non viene richiesto al giocatore umano di comandare le unità ad un così basso livello in quanto risulterebbe opprimente, viene fornita quindi la possibilità di impartire comandi di alto livello come quello descritto precedentemente, le unità inoltre sono in grado di eseguire comportamenti predefiniti come attaccare bersagli vicini o scappare quando vengono disarmati e sono sotto attacco. Per riuscire a sviluppare un'agente il cui comportamento risulti più vicino a quello umano questo componente, situato nel middleware, è in grado di supportare i comportamenti basilari delle unità a fronte della ricezione di un comando di alto livello attraverso l'utilizzo di macchine a stati finiti ed entità coordinatrici.

Finite State Machines

Notiamo ora che *Soar*, come un giocatore umano, può impartire all'*Execution System* comandi atomici, ovvero non divisibili in comandi più semplici, e comandi non atomici. In particolare quelli atomici possono essere inviati direttamente ad ORTS in quanto tali, mentre quelli non atomici devono subire un processo di decomposizione e pertanto vengono assegnati a FSM[2]. Se consideriamo un comando che comprende un gruppo, questo componente è in grado di fornire il controllo per l'esecuzione, passo per passo, dell'attività non atomica sulle singole unità che lo compongono ed insiste su questa decomposizione fino a quando il comando è terminato oppure viene annullato.

In particolare ogni FSM viene aggiornata ad ogni frame di gioco ed ha accesso ad ogni tipo di informazione percepita avente complessità arbitraria, tuttavia aderendo alla politica di simulazione di giochi RTS commerciali, sono stati implementati solamente comandi comuni di alto livello e comportamenti predefiniti.

[2]Acronimo di Finite State Machines

Inoltre il comportamento delle FSM non viene in alcun modo influenzato dal ciclo decisionale di *Soar* e viceversa, una volta che una FSM viene assegnata ad un'unità l'agente non dovrà fermarsi ad aspettare alcuna uscita, questo consente alle unità di eseguire le proprie istruzioni anche quando *Soar* sta partecipando ad altre attività.

Global Coordinators

Tipicamente nei giochi RTS le unità non devono solamente dimostrare un certo livello di autonomia nello svolgimento delle attività individuali, ma spesso possono cooperare nell'esecuzione di un comando assegnato ad un gruppo, questo comporta in alcuni casi una suddivisione dell'obbiettivo dell'attività, si consideri ad esempio la battaglia fra gruppi di unità, in questo caso se non arrivano comandi specifici da parte dell'agente, il computer assegnerà la potenza di fuoco in modo tale da suddividerla su tutte le unità nemiche, cosa che risulta sconveniente a livello strategico.

Riuscire a gestire questo tipo di comportamento risulta impossibile a meno che ogni FSM conosca la presenza e lo stato delle altre entità dello stesso tipo con cui sta cooperando, pertanto sono state introdotte le entità coordinatrici globali che si occupano di dirigere il comportamento che ogni FSM in gruppo dovrebbe adottare, questa soluzione non è l'unica ma risulta più efficiente rispetto all'utilizzo di un sistema multiagente senza *Execution System*.

A tale scopo sono stati implementati due coordinatori, uno per la l'acquisizione di risorse ed uno per la gestione degli attacchi. Il primo si occupa di ottimizzare l'assegnazione delle unità, apposite per questo scopo, al fine di massimizzare l'aspetto economico utilizzando un semplice tipo di apprendimento che tiene traccia delle prestazioni effettive di ciascuna unità e riassegna la rotta a quelle che acquisiscono meno risorse in modo tale da rendere il tragitto più corto possibile. Il secondo invece tenta di attuare la strategia di focalizzare la potenza di fuoco dell'intero gruppo su una singola unità nemica alla volta e per ottenere questo obbiettivo è in grado inoltre di comandare i movimenti ed il posizionamento di quest'ultime durante l'attacco.

Si vuole far notare che la gestione delle unità, effettuata dal middleware, richiede molto più tempo di elaborazione rispetto a quello impiegato da *Soar* in quanto operazioni come il calcolo del percorso ottimo risulta computazionalmente costoso a fronte del fatto che devono essere considerate le unità singolarmente.

4.1.6 Multi-Tasking

Come si può intuire dalle cosiderazioni fatte finora, i giochi RTS sono caratterizzati da un insieme di attività che interagiscono fra loro, le quali sono di natura gerarchica, se vogliamo effettuare un attacco ad esempio dobbiamo prima raccogliere abbastanza risorse e poi produrre unità sufficienti, attività secondarie come queste inoltre possono essere decomposte ulteriormente.

Quando si deve portare a termine un compito ed una delle attività secondarie, necessarie per portare a termine quella principale, è in attesa di un qualche evento per il suo completamento, risulta utile passare ad altri compiti. Inoltre vi sono situazioni in cui bisogna interrompere l'adempimento di un compito per portare a termine un'attività di priorità più alta, come ad esempio difendere la base da un attacco.

In SORTS i compiti sono le azioni che possono essere unificate sotto un unico obbiettivo ed in *Soar* le attività vengono mappate in subgoal[3], più in particolare:

- Il meccanismo dei subgoal è di tipo gerarchico, definendo così una gerarchia di compiti.
- *Soar* può selezionare solo **operatori** di subgoal più bassi e mentre un subgoal è in atto egli potrà solamente inviare comandi a ORTS atti al raggiungimento dello scopo di tale compito.
- Un subgoal viene considerato compiuto se le condizioni di uno più prioritario vengono soddisfatte, questo non avviene per subgoal aventi la stessi priorità.
- Dato che *Soar* segnala quando non vi sono in corso attività riguardanti un compito specifico basterà cambiare il compito in esecuzione con un altro avente pronte le attività.

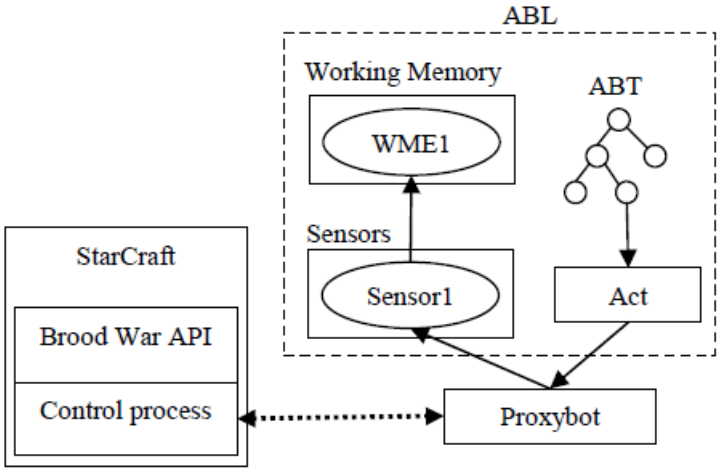
Consideriamo come esempio la gestione di due compiti da parte dell'agente, la costruzione della propria base e un attacco alla base nemica. Poichè prima che effettivamente inizi l'attacco, le unità devono arrivare alla base nemica, l'agente passa il controllo al compito di costruzione fino a quando la battaglia non ha inizio dato che il compito di attacco avrà una priorità più elevata, pertanto il controllo non tornerà al compito di costruzione finchè questa non sarà terminata.

[3]obbiettivo secondario

Si noti che questo comportamento porta un vantaggio all'agente rispetto ad un giocatore umano in quanto egli tenderà a completare un obbiettivo una volta avviate le attività senza effettuare scambi. L'operazione che comporta il cambiamento di obbiettivo risulta più semplice all'agente e questo avvantaggia maggiormente l'intelligenza artificiale rispetto all'essere umano. Questo sistema riduce tale vantaggio grazie alle limitazioni introdotte in *Soar*.

4.2 EISBot

L'architettura di questo agente[3] si focalizza sull'ottimizzazione di un RTS in particolare chiamato *Starcraft* che comprende tutte le considerazioni riguardo ai problemi fatte finora:



Il nucleo di questo agente è basato su un pianificatore reattivo, chiamato ABL[4], che si interfaccia con l'ambiente di gioco ed è in grado di gestire obbiettivi ed attività. In particolare questo componente si sviluppa su di un'architettura ibrida il cui scopo è quello di gestire la pianificazione ed il processo decisionale. Il principale punto di forza di questo tipo di approccio è il supporto ad agire in base a piani strategici parziali ed allo stesso tempo perseguire attività mirate al completamento di obbiettivi e rispondere ai cambiamenti dell'ambiente.

[4]Acronimo di A Behavior Language

EISBbot si interfaccia con *Starcraft* attraverso l'utilizzo di sensori, in grado di percepire lo stato del gioco, ed attuatori, che consentono l'invio di comandi alle unità. Inoltre in questa architettura sono compresi una memoria di lavoro ed un ABT[5] il cui scopo è quello di perseguire gli obbiettivi attivi selezionando, a seconda della situazione, il comportamento adeguato da un'apposita libreria contenente quelli esistenti. I componenti *Proxybot*, *The Brood War API* e *Control Process* servono ad interfacciare l'agente con l'ambiente di gioco.

Questo agente si basa principalmente sulla suddivisione concettuale del gioco in apposite aree di competenza, le quali vengono gestite da entità manager che si distinguono in base all'aspetto del gioco di loro competenza. Ogni manager implementa un insieme di comportamenti atti alla scomposizione del singolo aspetto di gioco in sottoproblemi mantenendo la possibilità di gestire problemi trasversali come la condivisione di risorse fra manager. I manager che compongono l'agente sono:

- **Strategy Manager**
  Si occupa di selezionare la strategia e di gestire la temporizzazione degli attacchi.
- **Income Manager**
  Gestisce la raccolta di risorse e le unità annesse, assieme all'espansione della base.
- **Construction Manager**
  E' responsabile delle richieste di costruzione di edifici.
- **Tactics Manager**
  Si occupa dei compiti relativi alle battaglie fra unità assieme ai comportamenti di microgestione.
- **Recon Manager**
  Gestisce l'esplorazione del territorio.

Ciascuna di queste entità implementa un'interfaccia che rende possibile la comunicazione fra manager, in cui viene specificato il metodo di scrittura dei dati nella memoria di lavoro. Questo approccio consente una semplice sostituzione per quanto riguarda implementazioni future sui singoli manager in quanto basterà rispettare le intefacce proposte.

[5]Acronimo di Active Behavior Tree

Inoltre in EISBot il processo decisionale, che porta alla costruzione di un certo edificio, viene disaccoppiato dal processo di costruzione, che prevede l'allocazione in un determinato luogo ed il monitoraggio dell'attività mentre questa è in corso. In questo modo è possibile specializzare il comportamento di ogni manager, come ad esempio la gestione del comportamento delle singole unità nel *Tactics Manager*.

4.2.1 Approccio all'Integrazione

L'entità di pianificazione reattiva è stata integrata con componenti esterni attraverso l'utilizzo di interfacce che offrono la possibità di effettuare le seguenti operazioni:

- Modificare la memoria di lavoro.
- Formulare un obbiettivo esterno.
- Generare pianificazioni di attività esterne.
- Attivare comportamenti specifici.

Tali operazioni non sono specifiche della pianificazione reattiva e possono essere utilizzare da architetture esterne. In particolare la memoria di lavoro permette la comunicazione fra manager, come detto in precedenza, ma consente anche ai componenti esterni, attraverso l'intefaccia, di memorizzare i dati riguardanti l'ambiente e previsioni sullo sviluppo del gioco. Dati che verranno elaborati dall'agente per ottenere previsioni sul comportamento specifico da adottare, come ad esempio la scelta del luogo per attaccare.

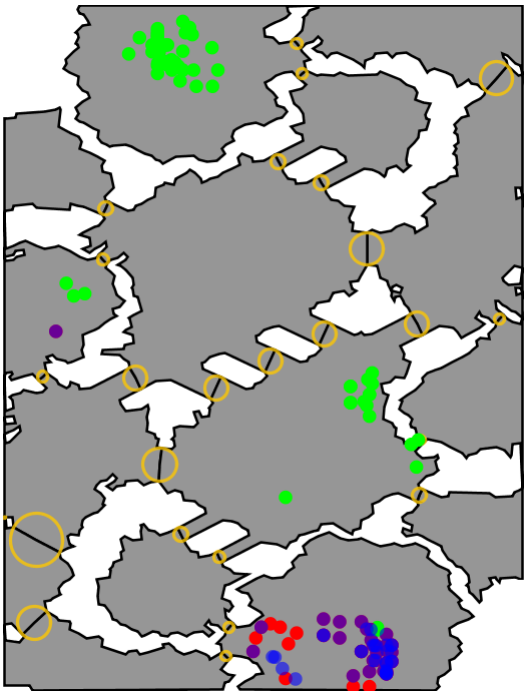
Inoltre viene fornita la possibilità di perseguire obbiettivi che vengono formulati esternamente a questa entità, i quali vengono poi aggiunti a quelli già presenti nel pianificatore reattivo attraverso un'opportuna modifica dell'albero degli obbiettivi attivi(ABT). I componenti esterni sono in grado di modificare la pianificazione delle attività, interna a questo componente, aggiungendo nella memoria di lavoro un piano in grado di attivare una sequenza di azioni che l'agente deve eseguire. L'operazione di attivazione di comportamenti specifici consente la modifica delle condizioni di attivazione, le quali possono contenere precodizioni che obbligano l'agente ad interrogare componenti esterni al momento dell'attivazione.

4.2.2 Microgestione

Come descritto nei capitoli precedenti, questo problema risulta complesso in quanto richiede una reattività accurata e diverse tecniche di gestione riguardo unità specifiche, pertanto la microgestione in questo agente prevede l'utilizzo di una libreria di comportamenti specifici per ogni tipo di unità. E' stato scelto questo tipo di approccio perchè la creazione di tecniche emergenti in grado di essere competitive a livello umano è una sfida ancora aperta.

4.2.3 Analisi del Terreno

EISBot utilizza inoltre una libreria contenente le caratteristiche spaziali qualitative delle mappe di gioco di *Starcraft* che principalmente contengono regioni, strozzature e posizioni di espansione per le basi. Queste informazioni vengono aggiunte alla memoria di lavoro e vengono utilizzate da molti manager per effettuare un'analisi sul terreno specifica in base all'aspetto di gioco riscontrato, il *Recon Manager* utilizza un elenco delle zone di espansione per decidere i luoghi di destinazione per le unità esploratrici, il *Tactics Manager* utilizza un elenco dei punti dove il terreno forma una strozzatura per posizionare le unità difensive, il *Construction Manager* utilizza un elenco delle regioni su cui andare a posizionare gli edifici. Queste informazioni migliorano inoltre la percezione dell'agente sullo stato di gioco.



La figura mostra un esempio della percezione dello stato di gioco dell'agente in base alle informazioni della libreria dove le zone verdi rappresentano la locazione delle unità, quelle blu una stima sulla posizione delle unità nemiche e quelle grigie sono le regioni. Le zone rosse indicano la presunta locazione delle unità nemiche che l'agente non è in grado di osservare.

4.2.4 Selezione della Strategia

In questo aspetto del gioco si è cercato di suddividere la scelta della strategia e la sua esecuzione introducendo così un modello specifico per l'adempimento degli obbiettivi. Nella fase iniziale di gioco l'agente seleziona ed esegue un ordine di costruzione, formato da una sequenza di azioni che segue la scelta strategica fatta fino a quell'istante, lo *Strategy Manager* utilizza una libreria per selezionare ed eseguire il comportamento atto all'esecuzione di tale ordine. In particolare ogni ordine di costruzione conterrà un elenco di aspettative che devono essere rispettate durante l'esecuzione, se questo non avviene allora si crea una discrepanza e l'agente sceglie una nuova strategia da seguire, questo approccio fornisce all'agente la possibilità di reagire a fronte di situazioni inaspettate.

Dopo aver portato a termine la strategia iniziale, l'agente si focalizza su una nuova strategia scelta in base alle informazioni sull'avversario. Per effettuare questa transizione viene introdotto un componente di pianificazione in grado di scegliere la strategia più adatta a seconda dei casi e di selezionare le attività da mandare in esecuzione in base ad una libreria contenente un elenco di situazioni, ricercando quella più simile. Le attività scelte infine vengono aggiunte alla memoria di lavoro.

4.2.5 Temporizzazione di Attacco

Gli aspetti dell'architettura introdotti finora offrono diverse possibilità per la gestione degli attacchi, ad esempio quando un ordine di produzione viene selezionato dallo *Strategy Manager* è possibile che in esso vi sia incapsulata una nota che specifica l'istante in cui effettuare l'attacco all'avversario. Questo può essere realizzato semplicemente aggiungendo un elemento alla memoria di lavoro che viene utilizzato a fronte dei controlli sulle precondizioni dell'attacco. Il sistema può inoltre iniziare una fase di attacco quando è stata raggiunta una dimensione in termini di potenza militare, inoltre per riuscire a mantenere il giusto livello di pressione sull'avversario vengono utilizzati comportamenti specifici, ricavati sempre da un'apposita libreria.

Capitolo 5

Confronto a Livello Implementativo

In fase di implementazione di un applicativo software solitamente il tempo impiegato per l'effettivo sviluppo di un prototipo iniziale risulta minimo a confronto con la fase subito successiva chiamata testing, in cui è prevista una serie di test progettati ad hoc per verificare che effettivamente il sistema segua il comportamento atteso.

5.1 Modalità Utilizzate in Fase di Test

Gli sviluppatori hanno implementato gli agenti descritti nei capitoli precedenti in base alle sfide proposte nei concorsi a cui hanno partecipato, in particolare SORTS partecipò al concorso chiamato AIIDE[1] nel 2006 che consisteva nella suddivisione del gioco in tre categorie distinte, fra cui una competizione basata sull'acquisizione di risorse, una basata sulle battaglie ed una che comprende una partita intera di gioco, ma limitata. Per ognuna di queste categorie è stato creato appositamente un agente che si interfaccia con il middleware, uguale per tutti.

Mentre per verificare le effettive prestazioni di EISBot si è deciso di effettuare una serie di partite uno contro uno fra questo agente e giocatori umani all'ICCup[2]. Questo rappresenta un torneo di competizione a cui partecipano più di diecimila giocatori attivi di *Starcraft* dove l'abilità di ogniuno viene descritta da una scaletta che varia dal titolo di principiante a quello di professionista.

[1] Acronimo di Association for the Advancement of Artificial Intelligence
[2] Acronimo di International Cyber Cup

5.2 Risultati ottenuti

5.2.1 SORTS

I test su SORTS prevedono quindi una serie di partite, giocate contro altri agenti, il cui scopo principale è quello di ricavare le prestazioni effettive e verificare il funzionamento del middleware.

Partita 1

La prima prova consiste nel raccogliere il maggior numero di risorse minerarie, utilizzando le apposite unità, in una determinata quantità di tempo. Questo compito risulta difficile in quanto prevede il coinvolgimento di molte unità che si muovo in un'area di terreno ristretta, pertanto l'agente dovrà alternare l'utilizzo di apposite tecniche di controllo in modo tale da ridurre il numero di collisioni fra i percorsi delle unità.

La gestione di questi aspetti in SORTS è prevista, per la maggior parte, nel middleware in quanto l'entità *Soar* non farà altro che impartire ordini alle unità e sarà l'*Execution System* a coordinarle ed assegnargli i percorsi adatti. Per evitare gli ostacoli presenti sulla mappa di gioco e la collisione frà unità, sono state introdotte una serie di regole nelle FSM che hanno portato questo agente a raccogliere il 78% delle risorse presenti nella mappa durante il tempo previsto, vincendo così la competizione.

Partita 2

In questa competizione ogni giocatore inizia la partita con 5 basi e 50 carri armati e l'obbiettivo primario è quello di distruggere le basi nemiche preservando quelle alleate. Nell'implementazione di questo agente sono state introdotte una serie di strategie euristiche che includono il raggruppamento delle unità. In questo caso si cercherà di attaccare il nemico con un gruppo di grandi dimensioni dando priorità alla distruzione dei carri armati nemici che attaccano le basi alleate, in caso di separazione durante le battaglie, le unità tenderanno a ritirarsi per formare nuovamente un gruppo.

Anche se *Soar* potrebbe gestire l'attenzione su tutte le singole unità contemporaneamente, l'utilizzo di queste strategie ha facilitato il processo di ragionamento dell'agente, ma tuttavia alcuni errori riscontrati durante il processo di percezione hanno portato il sistema in una situazione di stallo, con conseguente perdita della competizione.

Partita 3

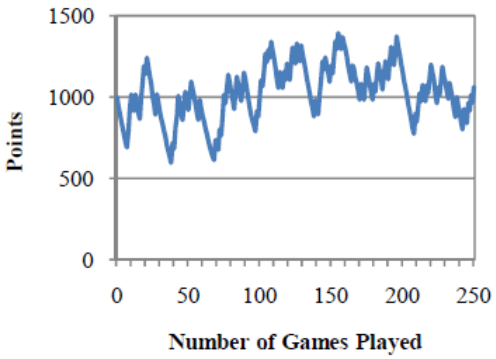
Questa prova prevede lo sviluppo di tutti gli aspetti, compreso l'obbiettivo, di una tipica partita ad un RTS con la possibilità di controllare poche unità in grado di raccogliere risorse.

L'agente implementato per affrontare questa competizione non riflette le capacità complessive che l'architettura, per come è stata progettata, è in grado di raggiungere in quanto il gioco risulta semplificato. Il sistema quindi si limiterà a seguire una strategia semplice che verrà riconsiderata a seconda delle situazioni. Grazie alla capacità di suddivisione e riassegnamento delle attività in base ai compiti, questo agente ha vinto il 60% delle partite 400 partite giocate contro quello sviluppato dall'Università di Alberta, nel restante 40% il sistema ha riscontrato lo stesso errore della seconda competizione nel *Perceptual System*.

5.2.2 EISBot

La competizione ICCup prevede l'utilizzo di un metodo di classificazione dei giocatori in base all'Elo[3]. Inizialmente vengono assegnati 1000 punti a tutti i giocatori che aumentano o diminuiscono in base all'esito delle partite giocate, in particolare i dilettanti rimangono sotto i 1000 punti Elo mentre i giocatori esperti arrivano a raggiungere i 2000 punti.

Per classificare le prestazioni di questo agente sono state giocate 250 partite contro avversari umani che avevano in media 1205 punti Elo con una deviazione standard di 1600. EISBot ha avuto un tasso medio di vittoria del 32% ottenendo così un punteggio finale di 1063 punti, pertanto è stato classificato come giocatore amatoriale di *Starcraft*.



[3]Metodo utilizzato per calcolare la forza relativa ad un giocatore di scacchi.

Conclusione

Come si può notare la complessità dello spazio degli stati riscontrata negli RTS risulta elevata; per riuscire a risolvere in maniera efficace il problema, è stata fatta una suddivisione in sei problemi più semplici in cui sono emersi aspetti specifici di gioco. Dopo aver studiato in dettaglio tali problemi, è stata effettuata una decomposizione delle attività che risulta essenziale in fase di progettazione, in quanto è in questa parte preliminare che viene delineato il dominio di competenza del sistema e vengono descritti gli obbiettivi primari.

E' stata quindi identificata un'architettura formata da tre livelli di astrazione in cui ogni livello si occupa della gestione di alcuni dei problemi precedentemente studiati. Durante questa analisi è emerso un aspetto che risulta essere, a livello teorico, più importante rispetto agli altri ovvero quello strategico, ma si è giunti alla conclusione che i livelli sottostanti, produzione e controllo reattivo delle unità, sono la chiave per aumentarne considerevolmente le prestazioni.

Sulla base di questa suddivisione teorica sono state sviluppate due architetture ad agenti di intelligenza artificiale il cui scopo è quello di riuscire ad ottenere un comportamento simile a quello umano sfruttando solamente le informazioni sullo stato del gioco che anche un giocatore umano riesce a percepire, senza barare.

In particolare l'interfacciamento tra *Soar* ed ORTS è stato guidato da due obbiettivi principali: l'impegno ad un approccio al ragionamento a livello umano e la necessità di risolvere i conflitti pratici fra queste due entità. Un essere umano impiega le stessa difficolta ad interfacciarsi con il gioco ed è in grado di sviluppare appositi meccanismi nella risoluzione di questi problemi; è proprio l'ispirazione a questi meccanismi che ha portato alla progettazione di un sistema che utilizza approcci differenti rispetto a quelli convenzionali nell'intelligenza artificiale negli RTS.

Durante la progettazione di questa architettura cognitiva sono state introdotte possibili soluzioni a problemi come il raggruppamento delle unità, l'attenzione ed il controllo gerarchico dei compiti. Un possibile sviluppo futuro può includere l'integrazione dell'apprendimento assieme al ragionamento spaziale in modo tale da sviluppare agenti in grado di ricavare sempre più informazioni autonomamente.

SORTS è stato testato durante l'AIIDE del 2006 che ha previsto la suddivisione del torneo in tre sfide principali, il sistema pur riscontrando qualche errore durante l'elaborazione dei dati percepiti, è riuscito ad ottenere un ottimo risultato vincendo due sfide su tre.

La seconda architettura proposta invece mira alla costruzione di un agente per *Starcraft*, un gioco RTS in particolare, la quale è stata motivata dalle proprietà riscontrate durante l'analisi del dominio e delle competenze di gioco. Il nucleo di questo agente è rappresentato da ABL, un pianificatore reattivo, che gestisce l'interfacciamento con l'ambiente di gioco e monitora l'esecuzione di attività, all'interno del quale, a differenza di SORTS, sono state utilizzate una serie di librerie contenenti i comportamenti da seguire a fronte di determinate situazioni, introducendo così l'utilizzo del materiale esistente. Inoltre questo agente utilizza alcuni approcci all'integrazione di componenti esterni che prevedono la manipolazione dei dati nella memoria di lavoro sotto forma di obbiettivi da perseguire, piani di esecuzione e condizioni per l'attivazione di un determinato comportamento.

Data l'impostazione di questa architettura esistono diversi possibili sviluppi futuri, fra cui l'estensione dell'agente nel caso in cui vengano individuate ulteriori competenze in questo campo, le quali possono essere implementate utilizzando comportamenti reattivi di pianificazione oppure un componente esterno. Oltre a questo un ulteriore obbiettivo futuro può comportare la modifica dell'interfacciamento con il gioco, in quanto attualmente il sistema è in grado di percepire le posizioni e le traiettorie delle unità interrogando lo stato del gioco, questo aspetto può essere migliorato con l'introduzione di tecniche di visione artificiale per l'astrazione dello stato di gioco utilizzando immagini catturate dello schermo nel corso della partita. Inoltre non vi è alcuna restizione sul numero di azioni che l'agente può eseguire, per imitare ulteriormente il comportamento umano, si potrebbe valutare come limitare questo numero al minuto, riducendo così le prestazioni dell'agente.

Una prima valutazione dell'agente EISBot è stata fatta al torneo ICCup contro giocatori umani ed ha portato risultati interessanti, in quanto il sistema ha ottenuto un tasso di vittoria del 32% sorpassando, nella classifica del torneo, il 33% dei giocatori. Vi è ancora un ampio divario fra le prestazioni di questo sistema ed i giocatori umani esperti, ma nel frattempo EISBot è riuscito a superare il livello di giocatore principiante ed acquisire il titolo di giocatore amatoriale.

Un ulteriore innovazione in questo settore porterà benefici non solo nell'industria dei giochi commerciali, ma anche in applicazioni reali in cui l'approccio ai problemi richiede un'intelligenza artificiale specifica.

Bibliografia

[1] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A Survey of Real Time Strategy Games AI Research and Competition in Starcraft, *IEEE transactions on computational intelligence and AI games,vol. 5,no. 4,December 2013 pages 293-311*

[2] M.Buro. Real time strategy games: A new AI research challenges, *In Proc. Int. Joint Conf. Artif. Intell. , 2003 , pages 1534-1535*

[3] Ben G. Weber, M. Mateas and A. Jhala. Building Human-Level AI for Real-Time Strategy Games, *Association for the Advancement of Artificial Intelligence 2011*

[4] S. Wintermute, J. Xu and John E. Laird. SORTS: A Humal-Level Approach to Real-Time Strategy AI, *University of Michigan 2260 Hayward St. Ann Arbor pages 55-60*

[5] M. Buro and Timothy M. Furtak. RTS Games and Real-Time AI Research, *Department of Computing Science, University of Alberta*

[6] F. Safadi and D. Ernst. Organization in AI design for real-time strategy games, *University of Liège pages 1-26*