UNIVERSITA' DEGLI STUDI DI BOLOGNA
II FACOLTA' DI INGEGNERIA

Corso di laurea magistrale in Ingegneria Informatica

"FANTASY WAR GAME"

Relazione progetto
Laboratorio di Sistemi e Applicazioni LM
A.A. 2010-2011

Studenti: Enrico Gualandi, Luca Spazzoli
Matricole: 0000417795, 0000523939

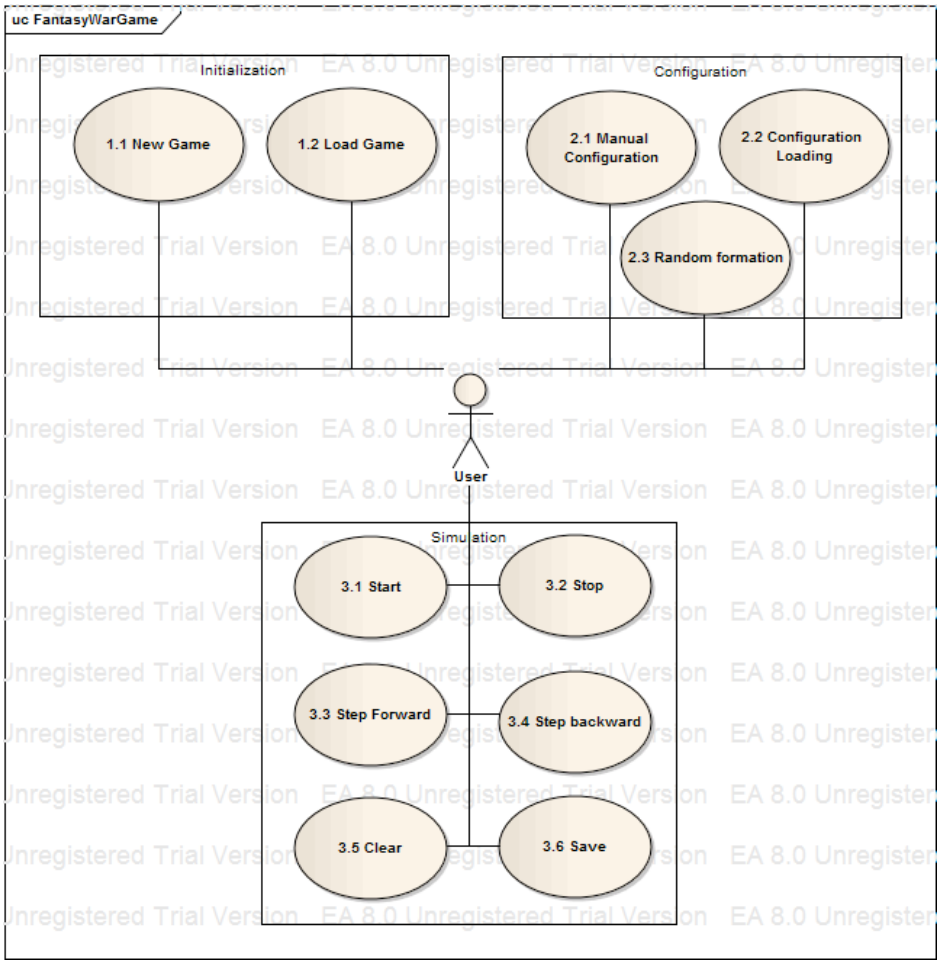
1. Analisi dei requisiti

Il progetto prevede la realizzazione della simulazione di una battaglia fantasy-medievale. L'applicazione offre la possibilità di vedere contrapposti su un **campo di battaglia** un numero n di **giocatori**, ad ognuno dei quali è associato un **esercito**. Sono previste diverse tipologie di **obiettivo**, quali, ad esempio, l'annientamento reciproco, la distruzione di un determinato bersaglio o la difesa di una posizione o di un edificio.
Ogni esercito è composto di **unità** dotate di obiettivi specifici, impartiti dal giocatore o valutati dall'unità stessa presa visione della contingente situazione di gioco. Un'unità è un insieme di **soldati**, ognuno dei quali appartiene ad una determinata categoria (fanteria, cavalleria, ecc.). Aspetto fondamentale è che l'unità abbia una propria visione del mondo per quanto concerne l'area in prossimità della quale essa compie le proprie azioni.
Il sistema, ai fini della gestione della simulazione, prevede la figura di un **orchestratore** onniscente che fornisce alle altre entità in gioco le informazioni sullo stato del mondo e sull'evolversi della battaglia.
Il vincitore sarà valutato al termine della simulazione sulla base della tipologia di gioco selezionata.
All'utente dell'applicazione viene offerta la possibilità di creare una propria configurazione di partenza per inizializzare la simulazione. Egli può scegliere se crearla manualmente, inserendo ad esempio il numero di eserciti presenti e di unità per ogni esercito, se caricarla da file o se farla generare in maniera casuale dal sistema.
Una volta inizializzata, l'utente può avviare la simulazione, fermarla, farla proseguire di un turno, farla tornare indietro di un turno e salvare lo stato corrente.

1.1 Glossario dei termini

Entità	Descrizione
Campo di battaglia	Mondo della simulazione in cui si svolge la battaglia
Obiettivo	Scopo finale della simulazione
Giocatore	Gestore del proprio esercito composto da unità alle quali impartisce comandi più o meno specifici
Esercito	Insieme di unità
Unità	Elemento di cui è composto un esercito, dotata di obiettivi specifici impartiti direttamente o dedotti dallo svilupparsi della simulazione. Al suo interno è formata da soldati
Soldato	Entità dotata di un profilo e caratteristiche proprie dipendenti dalla tipologia a cui appartiene
Orchestratore	Gestore della simulazione, conosce in ogni istante lo stato del mondo della simulazione

1.2 Casi d'uso



Scenari

Nome	1.1 New Game
Descrizione	Permette di inizializzare un nuovo gioco
Attori	Utente
Precondizioni	...
Scenario principale	L'utente può creare una nuova configurazione
Scenari alternativi	...
Postcondizioni	L'utente sceglie una nuova configuarazione

Nome	1.2 Load Game
Descrizione	Permette di caricare un gioco salvato precedentemente
Attori	Utente
Precondizioni	Un gioco deve essere stato salvato
Scenario principale	L'utente carica un gioco salvato
Scenari alternativi	Errore di caricamento
Postcondizioni	Il gioco salvato è stato caricato

Nome	2.1 Manual Configuration
Descrizione	Permette di configurare manualmente i dati della battaglia. Occorre inserire dati quali: • Nome degli eserciti • Tipo e nome delle unità da inserire • Coordinate iniziali delle unità • Numero di soldati di cui le unità sono composte • Numero di soldati sul fronte
Attori	Utente
Precondizioni	...
Scenario principale	Viene creata la simulazione sulla base dei dati forniti dall'utente
Scenari alternativi	I dati inseriti non sono validi
Postcondizioni	La simulazione viene inizializzata

Nome	2.2 Configuration Loading
Descrizione	Permette di caricare da file una configurazione precedentemente salvata.
Attori	Utente
Precondizione	Almeno una configurazione deve essere precedentemente salvata
Scenario principale	La configurazione viene caricata da file
Scenari alternativi	Errore nella lettura da file Il file cercato non esiste
Postcondizioni	La simulazione viene inizializzata

Nome	2.3 Random Formation
Descrizione	Permette di creare una formazione random. Occorre inserire dati quali: • Numero di eserciti • Numero di unità per esercito
Attori	Utente
Precondizione	...
Scenario principale	La formazione viene creata
Scenari alternativi	...
Postcondizioni	La simulazione viene inizializzata

Nome	3.1 Start
Descrizione	Permette di avviare la simulazione
Attori	Utente
Precondizione	Una configurazione del sistema deve essere stata creata. La simulazione deve essere in stop
Scenario principale	La simulazione viene avviata
Scenari alternativi	...
Postcondizioni	La simulazione è in esecuzione

Nome	3.2 Stop
Descrizione	Permette di fermare la simulazione
Attori	Utente
Precondizione	La simulazione deve essere in esecuzione
Scenario principale	La simulazione viene fermata
Scenari alternativi	...
Postcondizioni	La simulazione non è in esecuzione

Nome	3.3 Step forward
Descrizione	Permette di far avanzare la simulazione di un turno
Attori	Utente
Precondizione	La simulazione deve essere in stop
Scenario principale	La simulazione avanza di un turno
Scenari alternativi	...
Postcondizioni	La simulazione è avanzata di un turno

Nome	3.4 Step backward
Descrizione	Permette di far tornare indietro la simulazione di un turno
Attori	Utente
Precondizione	La simulazione deve essere in stop
Scenario principale	La simulazione torna indietro di un turno
Scenari alternativi	...
Postcondizioni	La simulazione è tornata indietro di un turno

Nome	3.5 Save
Descrizione	Permette salvare su file lo stato corrente della simulazione
Attori	Utente
Precondizione	La simulazione deve essere in stop
Scenario principale	La simulazione viene salvata su file
Scenari alternativi	Errore di I/O
Postcondizioni	E’ stato creato un file corrispondente al salvataggio della simulazione

Nome	3.6 Clear
Descrizione	Permette di fare il reset della simulazione
Attori	Utente
Precondizione	La simulazione deve essere in stop
Scenario principale	La simulazione viene resettata
Scenari alternativi	
Postcondizioni	La simulazione è stata resettata

2. Analisi del problema

L'applicazione descritta dal testo dei requisiti richiede la costruzione di un sistema multiagente. A tale fine occorre definire le caratteristiche principali di un agente software che verranno evidenziate nello sviluppo del sistema.

2.1 Agente software

Un agente software è un'**entità attiva** che incapsula un criterio per le proprie attività (*task*) al fine di autogestire il proprio flusso di controllo. Lo scopo della sua esecuzione è quello di perseguire un *goal*, uno stato da raggiungere. Questo determina un comportamento non deterministico e garantisce all'agente un certo grado di **autonomia**.
Quest'ultima è definita in funzione del fatto che un agente opera all'interno di un **ambiente** assieme ad altri suoi simili, tutti dotati di una rappresentazione esplicita del mondo, seppure limitata.
Un agente è **reattivo** nella misura in cui percepisce cambiamenti dell'ambiente in cui "vive", ed opera di conseguenza variazioni nel piano delle proprie azioni. Al tempo stesso l'agente è **proattivo** in quanto delibera il proprio corso di azioni basato sulla rappresentazione che ha del mondo.
Un agente è inoltre un "essere sociale" che comunica con altri agenti al fine di ottenere informazioni o collaborazione. Per fare questo utilizza un linguaggio di comunicazione, Agent Communication Language (ACL).

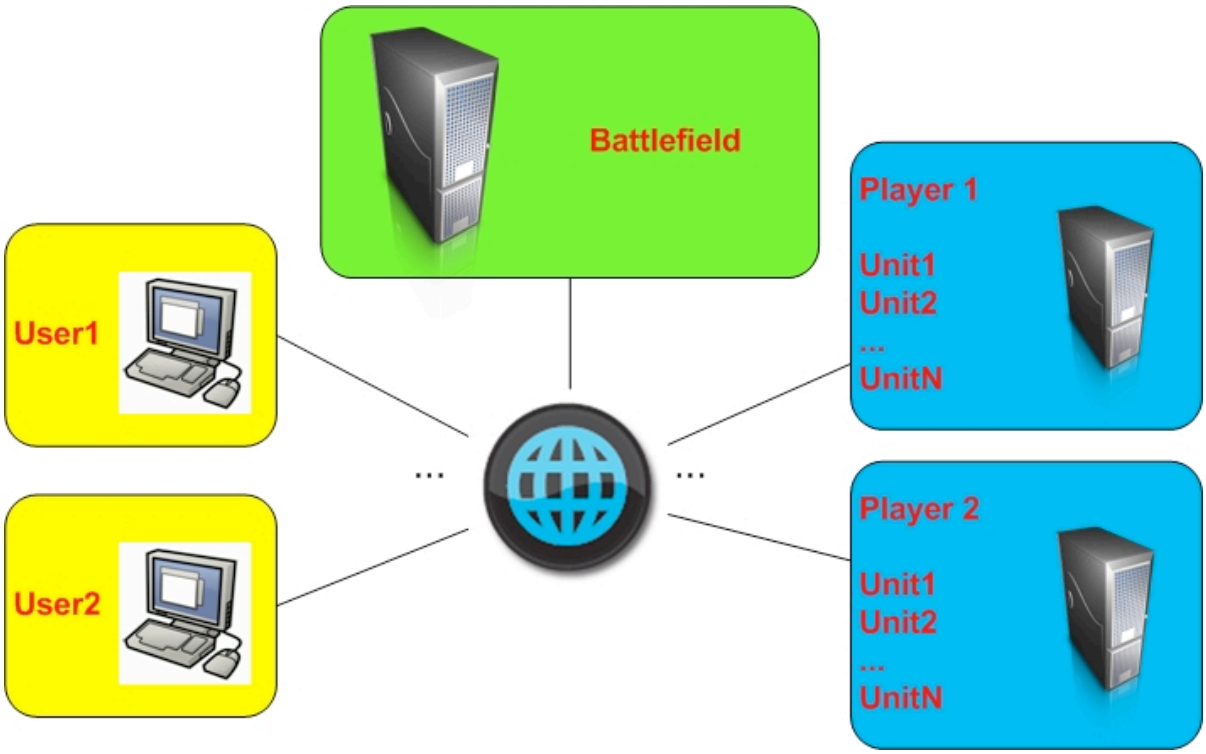
2.2 Punti critici

Dalla definizione di agente software presentata nel paragrafo precedente emergono alcuni punti critici che caratterizzano l'applicazione.

1. Le unità che compongono un esercito devono ricevere ordini impartiti dal giocatore che comanda l'esercito cui appartengono. Considerando un'unità come agente software, tuttavia, essa deve possedere un certo grado di discrezionalità al fine di perseguire l'obiettivo corrente. In alcune contingenti circostanze di gioco, dunque, un'unità può scegliere di non eseguire l'ordine direttamente impartito dal giocatore, ma compierne un altro valutato in base alla propria percezione del mondo e alle proprie priorità.
2. L'identificazione tra unità e agente consente inoltre lo scambio di informazioni tra unità, siano esse appartenenti allo stesso esercito o ad eserciti contrapposti.
3. Il sistema prevede la figura di un orchestratore che è a conoscenza dello stato dell'ambiente della simulazione e che può diffondere le informazioni locali riguardanti le diverse unità. Per fare questo l'orchestratore stesso può essere un agente in grado di comunicare con gli altri attraverso il linguaggio ACL.
4. Le unità devono avere una propria rappresentazione della parte del mondo all'interno della quale operano. E' necessario che richiedano queste informazioni all'orchestratore al fine di valutare il proprio stato all'interno del mondo della simulazione confrontandolo con gli obiettivi ricevuti dal giocatore.
5. Il giocatore, infine, viene identificato come agente in grado di impartire gli ordini alle unità appartenenti al proprio esercito sulla base della tipologia di gioco scelto.

2.3 Architettura logica

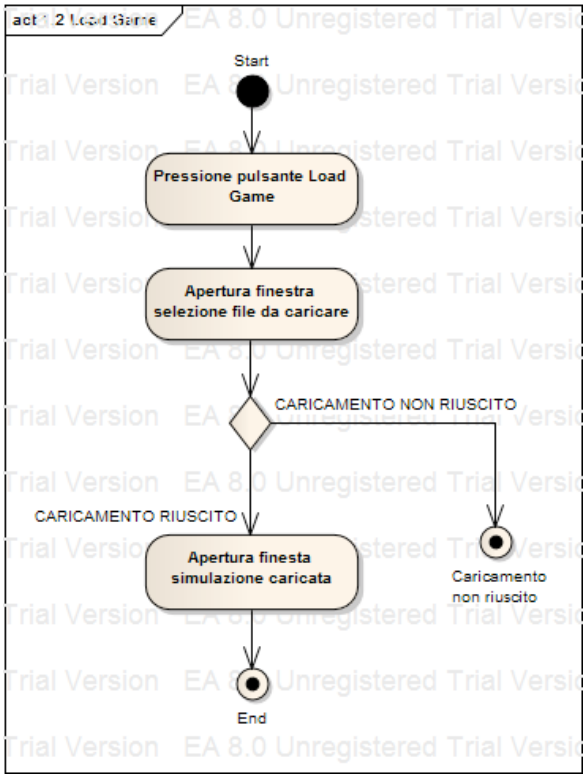
2.3.1 Struttura



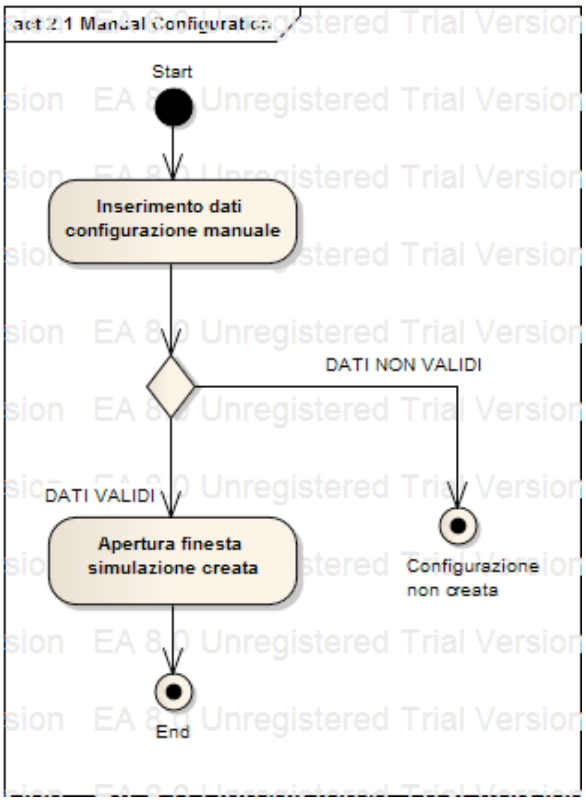
2.3.2 Comportamento

Di seguito vengono riportati i diagrammi delle attività relativi ai casi d'uso.

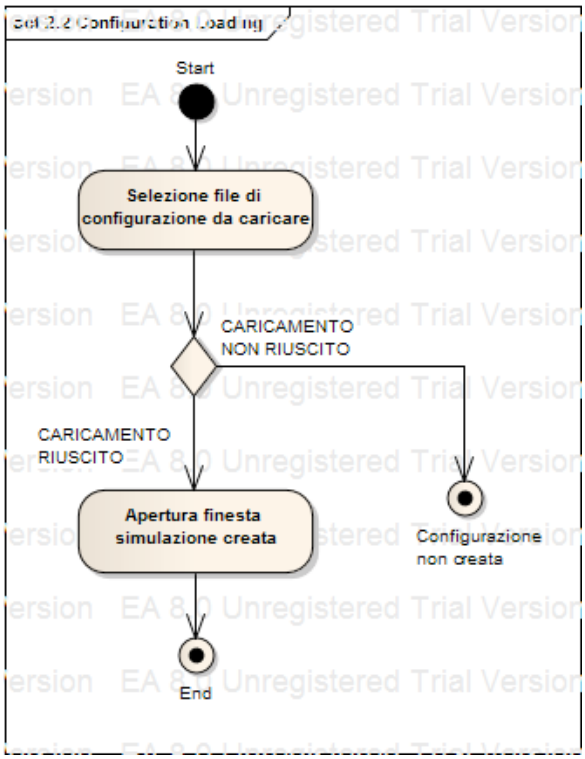
1.1 New Game



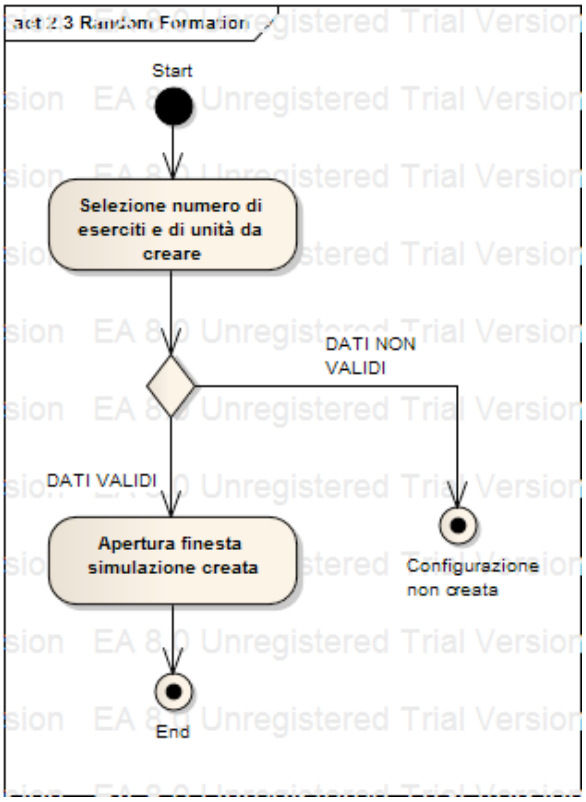
1.2 Load Game



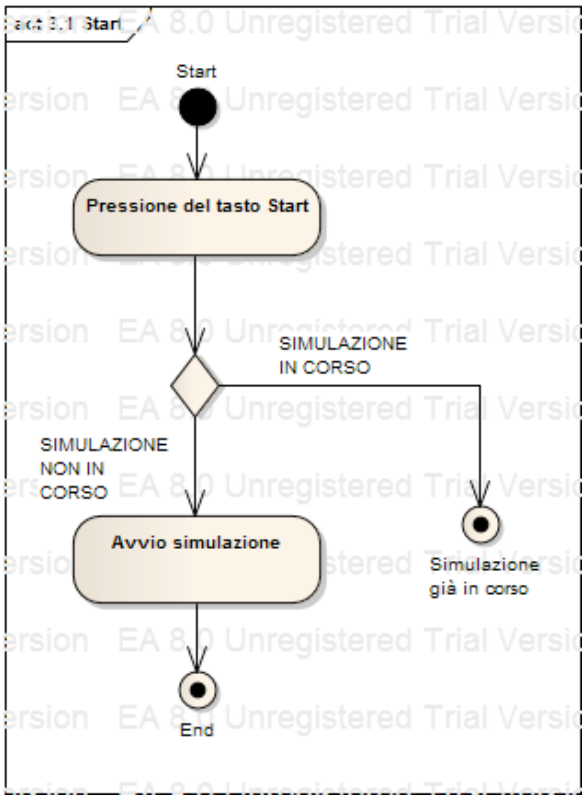
2.1 Manual Configuration



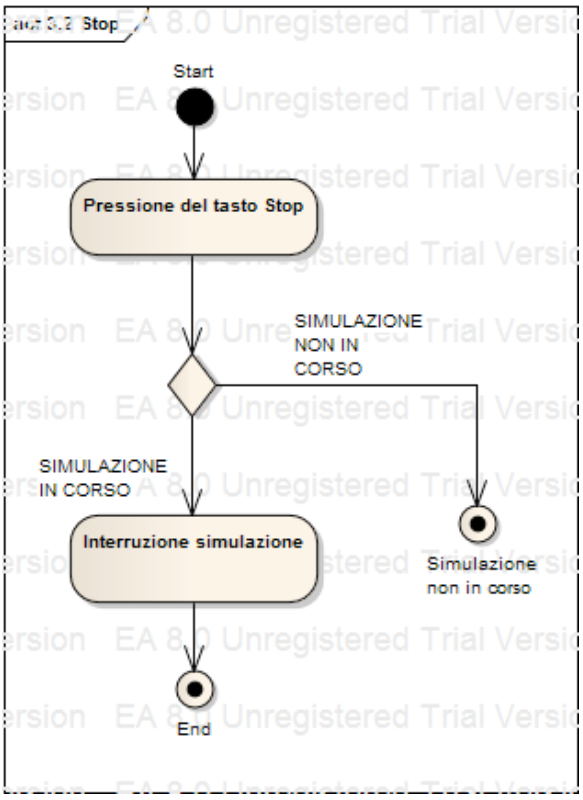
2.2 Configuration Loading



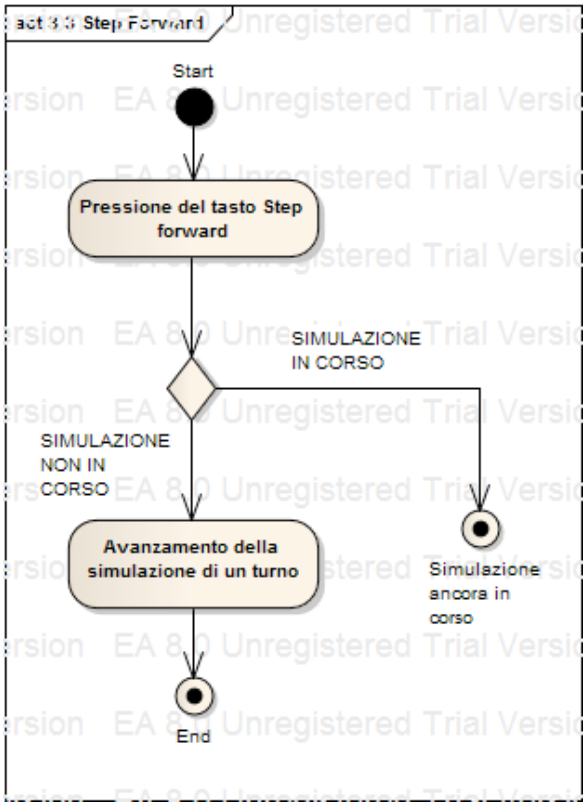
2.3 Random Formation



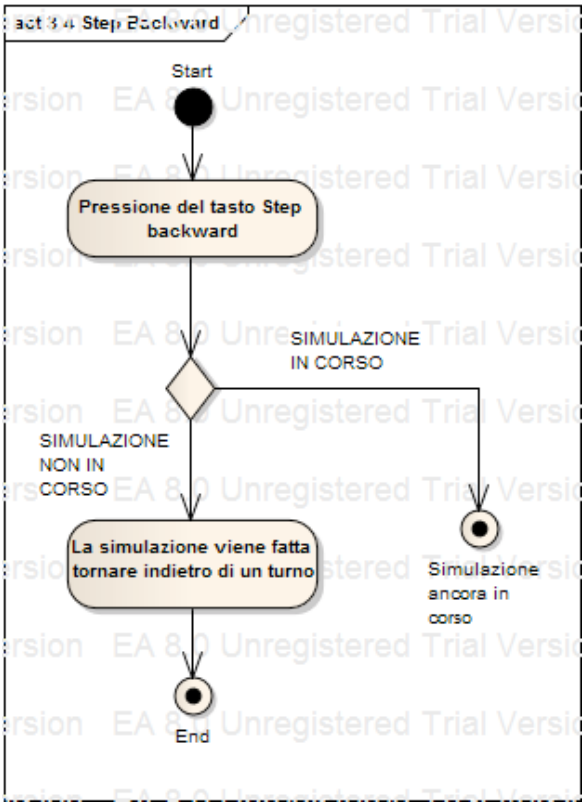
3.1 Start



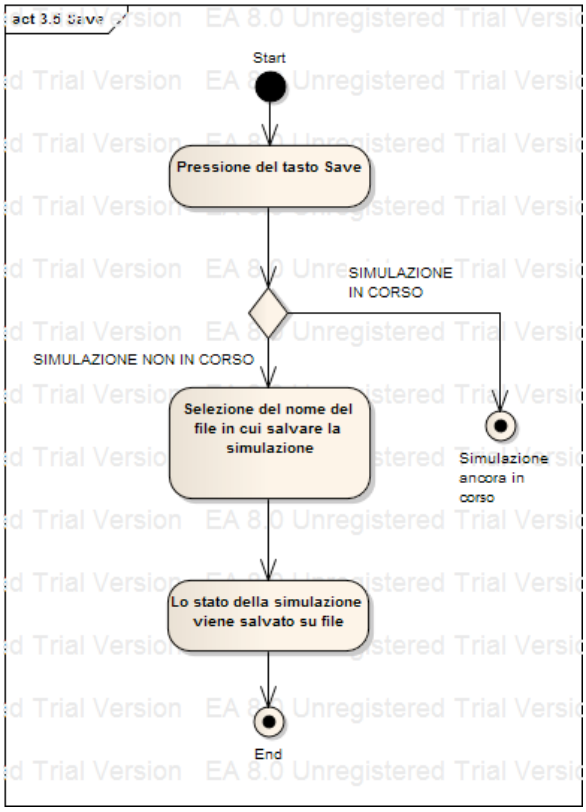
3.2 Stop



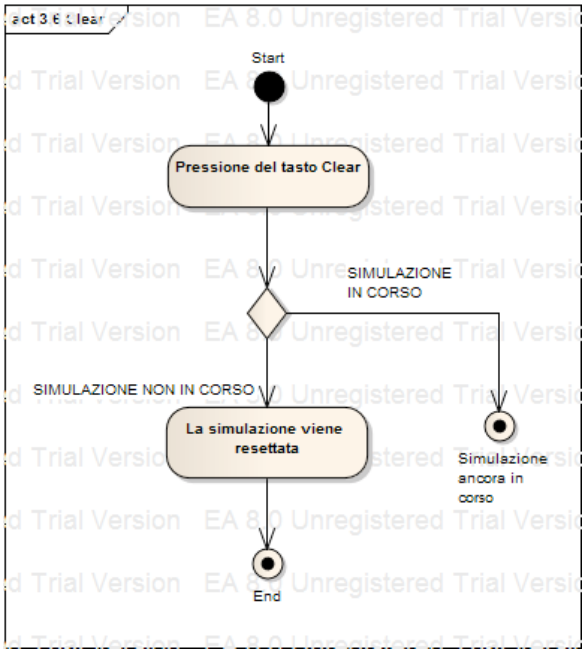
3.3 Step Forward



3.4 Step Backward



3.5 Save


Start
Pressione del tasto Clear
SIMULAZIONE
IN CORSO
SIMULAZIONE NON IN CORSO
La simulazione viene
resettata
Simulazione
ancora in
corso
End

3.6 Clear

3. Progetto della soluzione

La tecnologia utilizzata per sviluppare l'applicazione multiagente è il framework JADE.

3.1 JADE (Java Agent DEvelopment Framework)

Jade è un framework software implementato interamente in linguaggio Java. Semplifica l'implementazione di sistemi multiagente attraverso un middleware compatibile con le specifiche FIPA e una serie di tool che supportano le fasi di debugging e deployment.
Le piattaforme degli agenti possono essere distribuiti tra varie macchine e la configurazione può essere controllata attraverso una GUI remota. JADE permette inoltre di modificare la configurazione run-time creando e spostando agenti da una macchina all'altra. L'architettura di comunicazione offre uno scambio di messaggi flessibile ed efficiente. JADE crea e gestisce una coda di messaggi ACL (Agent Comunication Language) privata per ogni agente. Il meccanismo di trasporto si adatta ad ogni situazione, utilizzando il migliore protocollo disponibile.
Il middleware JADE include:

- Un **runtime environment** in cui gli agenti "vivono" e che deve essere attivo all'interno di un determinato host prima che uno o più agenti possano essere eseguiti in quell'host.
- Una **libreria di classi** che il programmatore può usare per sviluppare gli agenti.
- Una serie di **tool grafici** che consentono l'amministrazione e il monitoraggio dell'attività degli agenti.

3.1.1 Containers and Platforms

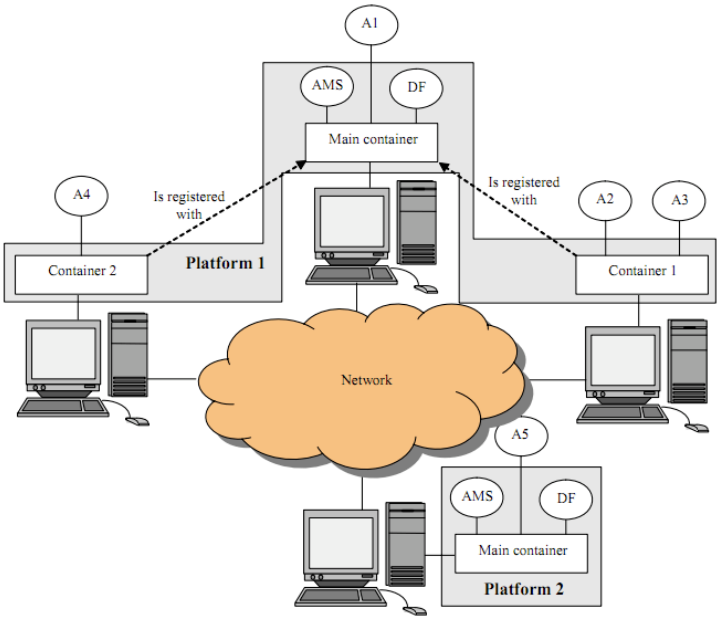


Figure 1Containers and Platforms

Ogni istanza di un runtime environment è chiamata *Container* e contiene uno o più agenti. L'insieme dei container attivi è chiamata *Platform*. Deve essere sempre attivo in ogni piattaforma un *Main Container* e deve essere il primo ad essere inizializzato.
Creare platform diverse consente quindi al sistema di essere distribuito in rete.

3.1.2 Behaviour

Il compito (job) di un agente è rappresentato dalla classe JADE *Behaviour* che specifica la modalità con cui un agente esegue il proprio task. A tal fine è sufficiente aggiungere il behaviour desiderato all'agente che deve eseguirlo. I behaviour possono essere aggiunti in ogni momento, all'avvio di un agente o all'interno di un altro behaviour.
I behaviour possono essere di tre tipi:
- One-shot: la cui esecuzione avviene una sola volta
- Cyclic: la cui esecuzione è ciclica
- General: che mantiene uno stato e esegue diverse operazioni a seconda dello stato

3.1.3 ACLMessage

JADE prevede la comunicazione attraverso un paradigma a scambio di messaggi asincrono. Ogni agente è infatti dotato di una coda di messaggi all'interno della quale il runtime environment di JADE inserisce i messaggi inviati dagli altri agenti. Quando un messaggio viene aggiunto alla coda, l'agente viene notificato.
Il formato dei messaggi è specificato dal linguaggio ACL definito da FIPA, standard internazionale per l'interoperabilità tra agenti.

3.1.4 The Yellow Pages service

Il servizio di "pagine gialle" permette agli agenti di pubblicare uno o più servizi da loro offerti in modo che gli altri agenti possano trovarli ed utilizzarli. In JADE questo è fornito da un agente chiamato DF (Directory Facilitator) che si trova in ogni platform.

3.1.5 JADE Tools

JADE comprende una serie di Tool per fornire supporto alle fasi di debugging e di deployment degli agenti. Tra i più importanti si trovano:
- JADE RMA: offre un'interfaccia grafica per l'amministrazione delle platform attraverso un RMA Agent. Questo agente mostra lo stato della platform cui appartiene e offre vari tool per testare e amministrare applicazioni basate su JADE.
- JADE Sniffer: framework di sviluppo software finalizzato alla realizzazione di sistemi multi-agente conformi allo standard FIPA. Consente il tracking dei messaggi scambiati dagli agenti.
- JADE DF GUI: offre un'interfaccia grafica per l'agente DF, consentendo di osservare e compiere operazioni sui servizi registrati presso le "pagine gialle".

3.2 Scelte progettuali

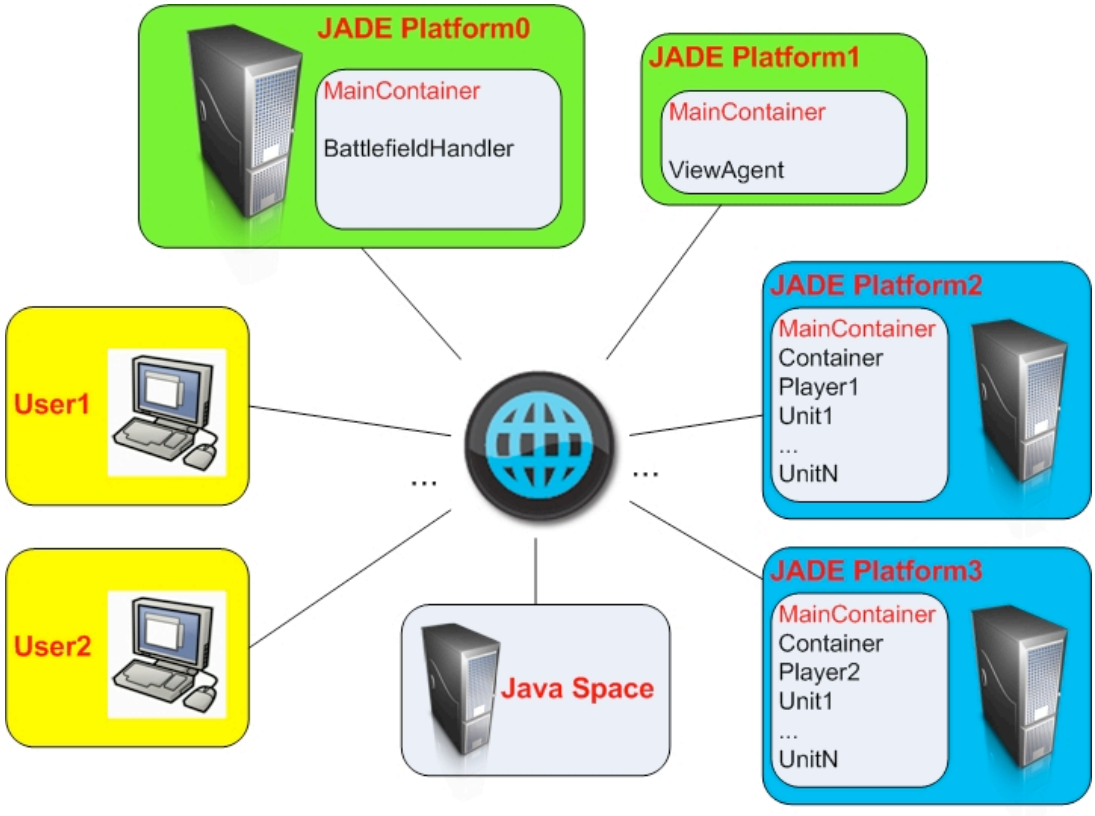
- La simulazione è organizzata a turni, uno per ogni giocatore presente.
- Ogni turno è suddiviso in quattro fasi: combattimento, magia, tiro e corpo a corpo. Durante la prima si dichiarano le cariche e si effettuano i movimenti, nella seconda i maghi lanciano gli incantesimi, la terza è riservata per le unità con armi da tiro e alle macchine d'assedio e infine nella quarta avviene il combattimento corpo a corpo.
- I dati relativi all'ambiente della simulazione sono amministrati da un agente, *BattlefieldHandler,* che è in grado di fornire ad ogni turno lo stato del mondo corrente sia ai giocatori che alle unità e di ricevere i loro aggiornamenti.
- Il giocatore è rappresentato da un agente, *Player,* che diffonde i goal generali alle unità del proprio esercito. Questo avviene attraverso un **Java Space** unitamente alla

tecnologia Jini. I goal vengono quindi inseriti in uno spazio condivisio in rete sotto forma di tuple. Le informazioni relative agli obiettivi sono ottenute dal BattlefieldHandler ad ogni turno.

- L'introduzione dello spazio di tuple è finalizzato alla diffusione "broadcast" degli obiettivi eliminando così la necessità di comunicarli direttamente ad ogni unità attraverso un paradigma di comunicazione basato su scambio di messaggi.
- Ogni unità presente sul campo di battaglia è rappresentata da un agente, *UnitAgent*. Esse ad ogni turno richiedono l'aggiornamento della propria specifica percezione del mondo. Successivamente leggono dallo spazio le tuple con i goal inseriti dal Player che controlla il corrispondente esercito. In funzione della contingente situazione della simulazione esse valutano quale task eseguire.
- La gestione del conflitto tra due UnitAgent è demandata ad un agente detto *ConflictResolver*, il quale viene creato dall'unità attaccante trasmettendogli le proprie informazioni e il proprio profilo. L'unità attaccata viene informata della creazione del ConflictResolver ed invia ad esso i propri dati. Una volta terminato il calcolo del risultato del conflitto, il ConflictResolver informa le due unità coinvolte dell'esito del conflitto e termina la propria esecuzione.
- Un agente denominato *ViewAgent*, infine, si occupa della gestione dell'interfaccia grafica rimanendo reattivo agli eventi da essa generati tramite l'interazione con l'utente. Il ViewAgent si occupa anche dell'aggiornamento della rappresentazione su GUI dello stato della simulazione.

3.3 Architettura fisica

3.3.1 Struttura



L'applicazione è distribuita in rete. In particolare si prevede che i seguenti componenti si trovino su nodi distinti:

- *BattlefieldHandler*

- *ViewAgent*
- Ogni *Player* facente parte della simulazione assieme ai relativi *UnitAgent*
- Il *Java Space* utilizzato dai Player per distribuire gli obiettivi

3.3.2 Comportamento

Di seguito viene riportato il funzionamento dettagliato di ogni agente presente nel sistema

ViewAgent

Il *ViewAgent* rappresenta l'agente attraverso il quale il sistema viene inizializzato.
Alla sua creazione esso effettua la registrazione al sistema di "pagine gialle", inizializza il BattlefieldHandler e crea la GUI principale del gioco. Svolte le suddette operazioni esso rimane reattivo agli eventi che giungono dall'interfaccia grafica (la pressione dei pulsanti da parte dell'utente) ed ai messaggi che provengono dal BattlefieldHandler.
In particolare all'evento di creazione della simulazione, sia essa stata ottenuta attraverso configurazione manuale, casuale o tramite caricamento da file, il ViewAgent inizializza tutti gli UnitAgent e i Player in gioco e rappresenta graficamente la situazione iniziale della battaglia. Successivamente invia un messaggio al BattlefieldHandler contenente la configurazione iniziale della simulazione (lista di unità e di oggetti in gioco).
Contestualmente alla pressione del pulsante "Start" esso invia un messaggio di notifica di avvio della simulazione a tutti gli agenti del sistema.
Mentre la partita è in corso, al termine di ogni fase e di ogni turno di cui fa parte, il ViewAgent riceve dal BattlefieldHandler la lista aggiornata contenente lo stato corrente della simulazione e si occupa della sua rappresentazione all'interno della GUI.

BattlefieldHandler

Il *BattlefieldHandler* è l'agente che contiene lo stato dell'ambiente della simulazione.
Alla sua creazione esso effettua la registrazione presso il sistema di "pagine gialle" e attende la notifica di inizio della simulazione da parte del ViewAgent.
All'interno del proprio comportamento ciclico esso si mette in attesa di ricevere messaggi da parte degli altri agenti che agiscono nella simulazione. Agli agenti di tipo Player invia su richiesta il turno corrente e, nel caso esso sia il loro, lo stato del mondo relativo alle unità nemiche. Dagli agenti di tipo UnitAgent invia, in risposta ad una richiesta, il turno corrente e l'arco di vista relativo. Il BattlefieldHandler si occupa anche di aggiornare lo stato della simulazione in funzione delle notifiche che riceve da parte degli UnitAgent. Esse consistono nelle notifiche di un nuovo stato, di "morte", di fine fase, di risveglio e di fuga.
Il ViewAgent viene notificato nel momento in cui tutte le unità hanno concluso una fase ed il turno di gioco, al fine di aggiornare lo stato della simulazione nella GUI.
Al cambio di turno il BattlefieldHandler avverte tutti gli agenti in gioco della fine del turno. Se il turno raggiunto è quello corrispondente al numero di turni predisposto per la durata della simulazione, viene calcolato e mostrato in interfaccia grafica l'esito della battaglia. Nel caso in cui un esercito non avesse più unità "vive" senza aver raggiunto il numero di turni stabilito, il BattlefieldHandler decreterà vincitore il giocatore dell'altro esercito.

Player

Il *Player* rappresenta il giocatore e ha il compito di stabilire quali sono gli obiettivi da fornire alle unità che compongono il proprio esercito.

Alla sua creazione effettua la registrazione al sistema di “pagine gialle”, cerca lo spazio di tuple in cui inserire gli obiettivi e attende dal ViewAgent la notifica di avvio della simulazione. Il suo comportamento è ciclico: per prima cosa rimuove dallo spazio di tuple gli obiettivi relativi al precedente turno di gioco, successivamente richiede al BattlefieldHandler il turno corrente, al fine di capire se è il proprio. In caso positivo, richiede al BattleFieldHandler lo stato corrente delle unità nemiche. Ricevuta la risposta esso predispone gli obiettivi del turno inserendo nello spazio una tupla per ognuno di essi. Infine attende da parte la notifica di fine turno.

UnitAgent

Lo *UnitAgent* rappresenta un’unità appartenente ad uno degli eserciti che compongono la simulazione. Esso come prima azione esegue una richiesta di qual è l’esercito di turno nello stato corrente. Se il turno è il suo si porta in modalità attiva, altrimenti rimane in modalità passiva in attesa di stimoli dalle altre unità.
In modalità attiva, se in stato di “Idle”, compie la seguente serie di azioni che lo portano ad aggiornare la sua visione del mondo: richiede gli obiettivi primari al proprio Player e il proprio arco di vista (i 90° frontali fino a 20 unità logiche di distanza) al BattlefieldHandler. Con questi dati crea la propria scala di obiettivi con una priorità; quest’ultima viene assegna in base alla distanza dall’unità e se questi sono stati segnalati dal Player come primari.
Completata la lista esegue l’obiettivo a priorità maggiore, che tendenzialmente è un attacco ad un’unità. Per prima cosa comunica all’unità che è sotto attacco e contemporaneamente comunica al BattlefieldHandler di aver svegliato un’altra unità così da sapere quante notifiche quest’ultimo deve ricevere. Rimane quindi in attesa di un messaggio contenente la reazione dell’attaccato, in base alla quale esegue il turno di gioco. Le variazioni sono, però, solo nel calcolo delle distanze da percorrere. Il turno e la sequenza di comportamenti resta la stessa: prima effettua il movimento (dipendentemente da un tiro di dadi e dalla reazione dell’attaccato), esegue le fasi di magia e di tiro, poi passa al combattimento (se non ha già eliminato il proprio avversario in una delle fasi precedenti).
Durante la fase di combattimento lo UnitAgent crea un ConflictResolver in cui sono registrati i suoi dati e ne comunica l’Id all’attaccato. Quest’ultimo invierà i suoi dati al ConflictResolver via messaggio. Dopo aver ricevuto l’esito del combattimento si coordina con l’attaccato per verificare chi ha vinto ed il comportamento dello sconfitto.
L’attaccato, tuttavia, può trovarsi in altre tre situazioni: in corpo a corpo, in fuga o morto. Nel caso sia morto si sospende. Nel caso sia in fuga prova di richiamarsi a raccolta, ovvero tornare lucido e quindi a combattere, altrimenti prosegue la sua “folle” corsa verso uno dei bordi del campo.
Nel caso sia in corpo a corpo salta tutte le fasi ed esegue solo quella del combattimento.
In modalità passiva, lo UnitAgent rimane reattivo solo ai messaggi che gli vengono inviati, reagendo diversamente in base al messaggio ricevuto. Se esso è un EndPhaseNotify, richiede di chi è il turno, aggiorna nella sua memoria la fase e si rimette in attesa fino al suo turno. Diversamente può riceve altri due messaggi: di attacco e di creazione del ConflictResolver.
In caso di attacco, analizzando il profilo proprio e dell’attaccante valuta la possibilità di sopravvivere ad un attacco. Nel caso questa sia bassa, si gira e comunica all’altro che fugge, altrimenti rimane sul posto a ricevere la carica.
Nell’altro caso comunica al ConflictResolver i suoi dati e attende l’esito. Poi si coordina con l’attaccante per vedere il vincitore e avere la reazione dello sconfitto.

ConflictResolver

Il ConflictResolver si occupa della gestione del combattimento tra UnitAgent. Esso riceve il messaggio da parte dell'attaccato, calcola il risultato del combattimento e comunica alle unità le proprie ferite e lo scarto di vittoria. Terminate queste operazioni si sospende.

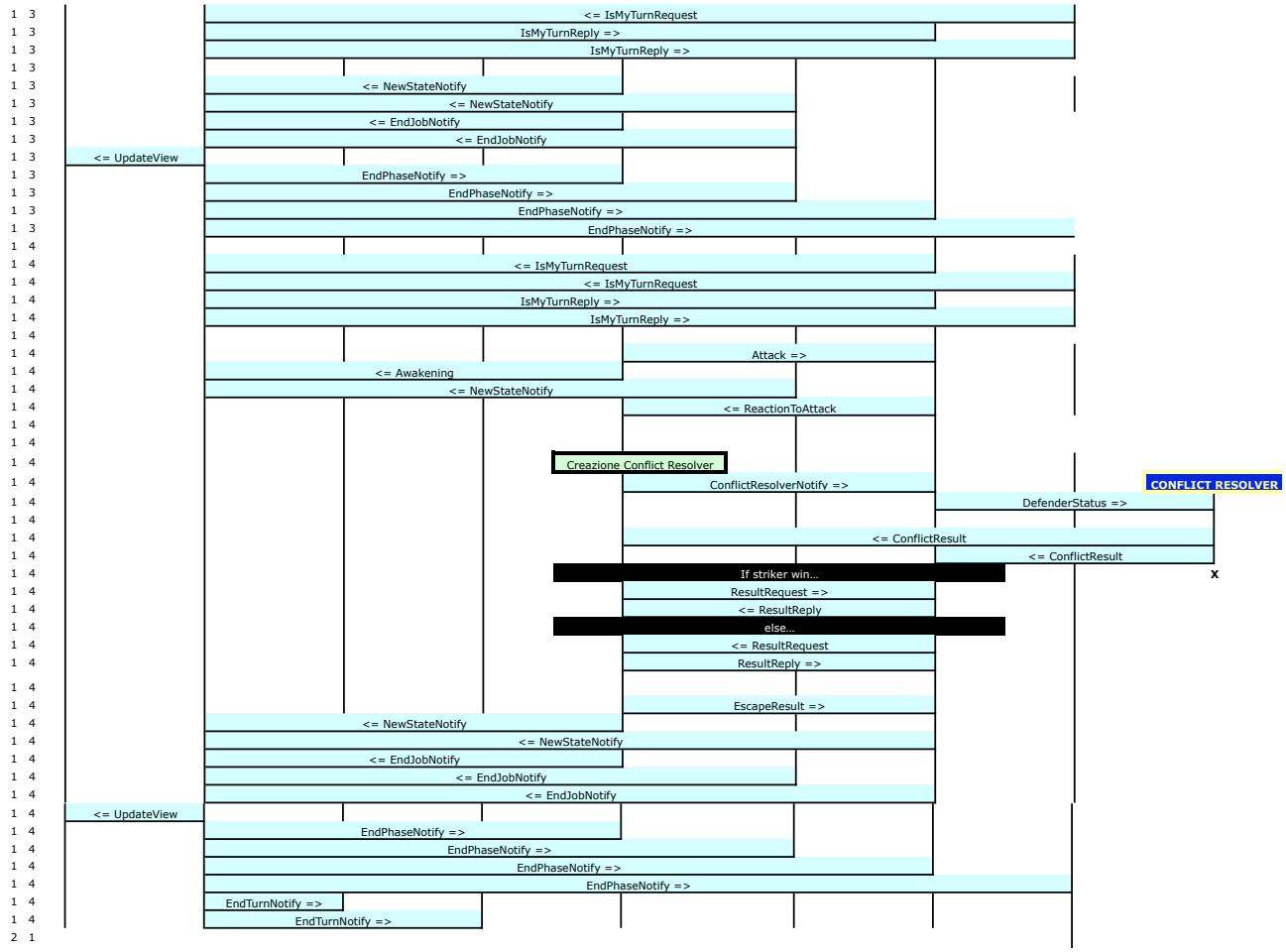
3.3.3 Interazione

La distribuzione degli obiettivi da parte dei Player nei confronti degli UnitAgent avviene attraverso uno spazio di tuple. Tutte le altre interazioni sono basate su scambio di messaggi. Di seguito vengono elencati i messaggi che compongono il sistema.

Messaggio	Mittente	Destinatario	Descrizione
ListCreated(unitList)	ViewAgent	BattlefieldHandler	Configurazione iniziale sistema
StartNotify()	ViewAgent	BattlefieldHandler Player UnitAgent	Notifica avvio simulazione
StopNotify()	ViewAgent	BattlefieldHandler	Notifica stop simulazione
IsMyTurnReply(turn)	BattlefieldHandler	Player UnitAgent	Risposta richiesta turno
WorldStateReply (unitList)	BattlefieldHandler	Player	Risposta richiesta stato del mondo
ArcViewReply (unitList)	BattlefieldHandler	UnitAgent	Risposta richiesta arco di vista
ClockNotify()	BattlefieldHandler	UnitAgent	Notifica clock
MovementNotify (unitList)	BattlefieldHandler	ViewAgent	Notifica di movimento
EndTurnNotify()	BattlefieldHandler	Player	Notifica fine turno
EndPhaseNotify()	BattlefieldHandler	UnitAgent	Notifica fine fase
UpdateNotify (unitList)	BattlefieldHandler	ViewAgent	Notifica di aggiornamento
IsMyTurnRequest()	UnitAgent Player	BattlefieldHandler	Richiesta turno
WorldStateRequest()	Player	BattlefieldHandler	Richiesta stato mondo

Messaggio	Mittente	Destinatario	Descrizione
ArcViewRequest (myId)	UnitAgent	BattlefieldHandler	Richiesta arco di vista
NewStateNotify (UnitGenericInfo)	UnitAgent	BattlefieldHandler	Notifica aggiornamento dati unità
MovementNotify()	UnitAgent	BattlefieldHandler	Notifica di movimento avvenuto
EndJobNotify()	UnitAgent	BattlefieldHandler	Notifica di terminazione lavori (per fase)
Attack (Coordinate,Dimension,moviment, Profile, NLine, NColumn)	UnitAgent	UnitAgent	Messaggio di attacco
AwakeningNotify()	UnitAgent	BattlefieldHandler	Notifica di risveglio di un'altra unità
ReactionNotify (reaction, distance)	UnitAgent	UnitAgent	Messaggio di reazione all'attacco
EscapeResult(result)	UnitAgent	UnitAgent	Risultato della fuga
ChargeResult(result)	UnitAgent	UnitAgent	Risultato della carica
DeathNotify (UnitGenericInfo)	UnitAgent	BattlefieldHandler	Notifica di distruzione della propria unità
ConflictResolverNotify(resId)	UnitAgent	UnitAgent	Comunicazione dell'ID del ConflictResolver
ResultRequest()	UnitAgent	UnitAgent	Richiesta esito combattimento
ResultReply(esito, percentualeMorti)	UnitAgent	UnitAgent	Notifica esito finale combattimento
AttackedUnitData (mySoldier, myBonus)	UnitAgent	ConflictResolver	Comunicazione dati al ConflictResolver
ConflictResult(hit, ris)	ConflictResolver	UnitAgent	Notifica risultato combattimento.

Esempio di scambio di messaggi durante un turno di gioco ipotizzando che la simulazione sia composta da due eserciti ciascuno dei due formati da due unità:


1 3
<= IsMyTurnRequest
1 3
IsMyTurnReply =>
1 3
IsMyTurnReply =>
1 3
1 3
<= NewStateNotify
1 3
<= NewStateNotify
1 3
<= EndJobNotify
1 3
<= EndJobNotify
1 3
<= UpdateView
1 3
EndPhaseNotify =>
1 3
EndPhaseNotify =>
1 3
EndPhaseNotify =>
1 3
EndPhaseNotify =>
1 4
1 4
<= IsMyTurnRequest
1 4
<= IsMyTurnRequest
1 4
IsMyTurnReply =>
1 4
IsMyTurnReply =>
1 4
1 4
Attack =>
1 4
<= Awakening
1 4
<= NewStateNotify
1 4
<= ReactionToAttack
1 4
1 4
1 4
Creazione Conflict Resolver
1 4
ConflictResolverNotify =>
CONFLICT RESOLVER
1 4
DefenderStatus =>
1 4
1 4
<= ConflictResult
1 4
<= ConflictResult
1 4
If striker win...
x
1 4
ResultRequest =>
1 4
<= ResultReply
1 4
else...
1 4
<= ResultRequest
1 4
ResultReply =>
1 4
1 4
EscapeResult =>
1 4
<= NewStateNotify
1 4
<= NewStateNotify
1 4
<= EndJobNotify
1 4
<= EndJobNotify
1 4
<= EndJobNotify
1 4
<= UpdateView
1 4
EndPhaseNotify =>
1 4
EndPhaseNotify =>
1 4
EndPhaseNotify =>
1 4
EndPhaseNotify =>
1 4
EndTurnNotify =>
1 4
EndTurnNotify =>
2 1

4. Estensioni del sistema

Le estensioni implementate riguardano principalmente tre aspetti:

4.1 Inserimento di ostacoli ed edifici all'interno della simulazione

La simulazione è composta, oltre che dalla unità facenti parte degli eserciti contrapposti, anche da elementi statici che compongono l'ambiente.
Al momento il sistema presenta la possibilità di inserire edifici ed altri elementi quali, ad esempio, muri. Essi influenzano l'arco di vista delle unità, alcuni presentano la proprietà di intransitabilità, altri invece rallentano il movimento. Gli edifici consentono inoltre l'implementazione di altre due modalità di gioco all'interno della simulazione: presidio e conquista. Secondo queste modalità aggiuntive, uno degli eserciti si dovrà occupare della difesa di un edificio prestabilito (ad esempio una torre) mentre l'altro (o gli altri) dovranno conquistarlo.

4.2 Combattimento multiplo tra unità

Diversamente dalla versione precedente del sistema, che prevedeva un combattimento uno contro uno tra le unità, la nuova implementazione gestisce combattimenti multipli.
E' quindi possibile che più unità facenti parte di un esercito sostengano un combattimento con una o più unità dell'esercito o degli eserciti avversari.

4.3 Aspetti di coordinazione tra le unità

Al fine di garantire la funzionalità di cui al punto 4.2, occorre che le unità siano in grado di comunicare tra loro e di implementare protocolli di coordinazione. Per questo motivo il comportamento dello UnitAgent, unitamente a quello del ConflictResolver descritti precedentemente hanno subito variazioni.

4.3.1 UnitAgent

Per garantire il combattimento multiplo, al momento dell'invio di un messaggio di attacco, lo UnitAgent riceve in risposta anche l'AID del ConflictResolver già utilizzato dall'attaccato, quindi si aggiunge alla lista di attaccanti presente in esso.
La nuova versione dello UnitAgent possiede inoltre un algoritmo che consente all'unità di evitare gli ostacoli presenti all'interno dell'ambiente della simulazione. L'arco di vista che l'unità riceve dal BattlefieldHandler infatti comprende sia la lista delle unità nemiche che quella degli ostacoli che l'unità è in grado di vedere. Per prima cosa essa verifica se sulla propria traiettoria (calcolata sulla base della posizione del bersaglio) sono presenti ostacoli. In caso affermativo l'unità individua il vertice dell'ostacolo più vicino alla traiettoria originale ed esegue ricorsivamente la stessa verifica su ogni nuova frazione di traiettoria trovata. Dopo aver effettuato le suddette valutazioni, lo UnitAgent possiede al suo interno una lista di punti, detti mete intermedie, e calcola i passi da compiere sulla base di essa.

4.3.2 ConflictResolver

Il compito del ConflictResolver è quello di comunicare con gli UnitAgent coinvolti in un combattimento e fornire loro il risultato. Nel contesto di un combattimento multiplo, esso deve essere dunque dotato di due liste di unità: gli Strikers e i Defenders. La prima

rappresenta la lista delle unità di turno che compiono l'attacco, la seconda è composta dalle unità che si devono difendere. Il ConflictResolver inoltre deve rimanere reattivo a nuovi messaggi che provengono dalle unità coinvolte nel combattimento, quali l'aggiunta e la cancellazione delle unità dalle due liste.
Contestualmente alla ricezione del messaggio di risoluzione del combattimento, il ConflictResolver esegue il vecchio algoritmo applicandolo alle liste degli attaccanti e dei difensori.
Volendo infine mantere il minore possibile il numero di agenti attivi all'interno del sistema al fine di garantire la migliore prestazione possibile, alla ricezione del messaggio di fine turno, il ConfictResolver inverte le liste degli attaccanti e difensori, e continua a calcolare il risultato del combattimento. In questo modo non è necessario inizializzare un agente per ogni turno per ogni combattimento.

Sono inoltre stati implementate altre tipologie di messaggio che vengono riportate in seguito:

Messaggio	Mittente	Destinatario	Descrizione
SimulationCreated (unitList,landscapeList, simulationType, target)	ViewAgent	BattlefieldHandler	Configurazione iniziale simulazione
EnemyDataReply (unitList)	BattlefieldHandler	UnitAgent	Risposta dati avversario fornito dal ConflictResolver
GiveMeEnemy()	UnitAgent	ConflictResolver	Richiesta al ConflicResolver di uno degli avversari già coinvolti nel combattimento
EnemyDataRequest (enemyAID)	UnitAgent	ConflictResolver	Richiesta al BattlefieldHandler dei dati dell'avversario
DeleteUnit()	UnitAgent	ConflictResolver	Richiesta rimozione di se stessa
ReactionNotify (reaction, distance, resolverAID)	UnitAgent	UnitAgent	Reazione messaggio di attacco
ResolverCombat()	UnitAgent	ConflictResolver	Richiesta risoluzione combattimento
InsertUnit(lineUp, bonus, tipo)	UnitAgent	ConflictResolver	Richiesta insertimento unità in una delle liste del ConflictResolver

Messaggio	Mittente	Destinatario	Descrizione
ResolverNotify (resolverAID)	UnitAgent	BattlefieldHandler	Notifica di creazione di un ConflictResolver
GiveMeEnemyReply (enemyAID)	ConflictResolver	UnitAgent	Risposta alla GiveMeEnemy
IsMyTurnRequest()	ConflictResolver	BattlefieldHandler	Richiesta del turno

5. Note

5.1 Note sulla versione demo

- La modalità di creazione della configurazione dell'ambiente di gioco implementata è il caricamento da file. A tal fine è stato creato un ArmyGenerator (package fmw.generator) per mezzo del quale è possibile generare e salvare su file una configurazione ordinata specificando il numero di eserciti, di unità, di oggetti e di edifici.
- Tramite l'ArmyGenerator sono state create alcune configurazioni d'esempio della simulazione.
- La versione dell'applicazione implementata è predisposta per avere n giocatori ed n unità, tuttavia la parte relativa all'interfaccia grafica prevede una battaglia tra due eserciti composti da n unità.
- Il controllo della simulazione prevede al momento soltanto la possibilità di avviarla e di fermarla.
- Il sistema, nonostante sia eseguito in locale e dunque in una sola Platform JADE, è predisposto per essere eseguito su nodi distribuiti, avendo suddiviso gli agenti in gioco in Container JADE differenti. Occorre dunque creare una Platform per ogni nodo sul quale si vuole che gli agenti vengano eseguiti.

5.2 Istruzioni di avvio dell'applicazione

Per avviare l'applicazione è necessario installare il plugin Eclipse **EJADE** (http://disi.unitn.it/~dnguyen/ejade/) e **Jini** (http://www.jini.org/wiki/Main_Page).
Una volta installati occorre avviare JADE RMA da Eclipse e l'eseguibile "launch-all" di Jini.
A questo punto si deve cliccare con il tasto destro del mouse sulla classe ViewAgent all'interno del package fmw.controller, selezionare EJADE --> Deploy Agent(s) e cliccare su "Finish" nella schermata successiva.
Nel caso in cui le immagini relative al gioco non vengano caricate, occorre specificare l'indirizzo assoluto del file system in cui si trovano.