# POLITECNICO DI MILANO

## Facoltà di Ingegneria

### Corso di Laurea Specialistica in Ingegneria Informatica


# Sviluppo di un wargame per tablet con Unity3D

**Relatore: Prof. Pier Luca LANZI**

**Correlatore: Spartaco ALBERTARELLI**

**Tesi di Laurea di:**

**Valerio CERAUDO Matr. n. 767273**

**Anno Accademico 2012-2013**

# Prefazione

Lo sviluppo di un videogioco è un processo che offre sfide di notevole complessità dovendo coadiuvare diversi aspetti dell'informatica come l'ingegneria del software, la grafica, elementi di connettività, l'accesso alle strutture dati, la sicurezza informatica, l'intelligenza artificiale, tutti ugualmente importanti in questo particolare settore dello sviluppo software nel quale fra l'altro rientrano importanti fattori sociologici e psicologici. L'avvento delle tecnologie mobile ha innalzato ulteriormente il livello di sfida dovendo lavorare con risorse molto limitate se paragonate a quelle dei classici computer ed in contesti totalmente diversi da quelli del home entertainment.

Per questo lavoro di tesi è stato scelto di sviluppare un videogioco strategico per tablet partendo dal board game sviluppato da Guido Crepax negli anni '70 "La battaglia di Waterloo". Il genere strategico, chiamato anche wargame, è a sua volta un genere molto particolare che introduce elementi di gameplay complessi sia dal punto di vista del calcolo computazionale, sia dal punto di vista della giocabilità stessa, dovendo rappresentare su schermo la gestione di un esercito e di una battaglia con i quali il giocatore interagirà tramite il solo schermo del tablet e non con le classiche periferiche di mouse e tastiera. Fin dall'inizio dello sviluppo è stato previsto l'uso dell'intelligenza artificiale sia per fornire al giocatore la possibilità di giocare da solo, sia come strumento di testing. In questo documento sono presentate tutte le informazioni necessarie a comprendere appieno il contesto applicativo e tutte le scelte e le fasi di sviluppo utilizzate per la realizzazione del wargame in formato digitale.

# Indice

# Elenco delle Figure



# Elenco delle Tabelle

# Capitolo 1

## Introduzione

L'industria dei videogiochi sta subendo negli ultimi anni notevoli cambiamenti con l'avvento delle tecnologie mobile. Se con il termine mobile prima si intendeva il solo campo della telefonia, oggi le cose sono completamente diverse grazie alla disponibilità di strumenti hardware, primi fra tutti cellulari e tablet, in cui la telefonia riveste solo un ruolo marginale, o in alcuni casi secondario. La maggior interconnettività e la maggior potenza di calcolo di questi strumenti hanno portato a creare un nuovo mercato di applicazioni, le quali a loro volta hanno spinto i produttori hardware a creare sistemi mobile sempre più potenti e performanti. Le applicazioni che più di tutte hanno catalizzato questo fenomeno sono i videogiochi, che in alcuni casi sono diventati veri e propri fenomeni di massa.

A differenza dei classici giochi per console o PC, i videogiochi destinati al mercato mobile differiscono in molti aspetti, primi fra tutti la fruizione attraverso l'interazione con il solo schermo del tablet ( o movimento del tablet stesso) e le limitate risorse di calcolo a disposizione. Sviluppare un videogioco per ambienti mobile come se si stesse sviluppando un videogioco sulle piattaforme classiche risulta pertanto impossibile e necessità di una serie di scelte progettuali indispensabili per ottenere un prodotto funzionale e fruibile in questo particolare contesto.

Nel titolo che si è scelto di sviluppare per questo lavoro di tesi si è scelto di introdurre una Intelligenza Artificiale per permettere un gioco variegato in single player. Il campo dell'Intelligenza Artificiale è molto vasto e sono stati fatti moltissimi studi sulla sua applicabilità nel campo dei videogiochi, ma utilizzare un Intelligenza Artificiale richiede anche un livello di risorse computazionali che

aumenta proporzionalmente ai risultati che si desidera ottenere, risorse ancora non così disponibili nel settore mobile e che hanno dunque necessitato di uno studio per poter comprendere quale compromesso fosse possibile realizzare tra l'implementazione di un elemento fondamentale per alzare il livello di gioco e la corretta fruizione del titolo stesso.

Il documento di tesi si articola nelle seguenti parti:

*Capitolo 2*: in questo capitolo viene presentato il concetto di Intelligenza artificiale, le tappe che storicamente ne hanno segnato l'evoluzione e la sua applicazione in un contesto di entertainment come quello dei videogiochi, osservando, tramite esempi reali, i benefici che entrambi i settori hanno ottenuto da questa collaborazione. Nella parte finale del capitolo sono presentati alcuni esempi sullo stato dell'arte dello studio dell'Intelligenza Artificiale applicata ai wargame.

*Capitolo 3*: viene introdotto il contesto dei wargame, un particolare sottogenere dei giochi da tavolo e dei videogiochi, al quale appartiene il titolo sviluppato in questo documento. Successivamente sono illustrati nel dettaglio "La battaglia di Waterloo", il suo background, le regole, le fasi e le componenti del gioco da tavolo originale e mostrare il fine l'analisi fatta per passare alla versione digitale.

*Capitolo 4:* questo paragrafo è dedicato allo sviluppo vero e proprio della versione digitale, vengono illustrati gli strumenti utilizzati, le scelte architetturali fatte, i framework utilizzati e le problematiche legate allo sviluppo in ambiente mobile, nonché come queste siano state affrontate e risolte così da ottenere prestazioni performanti visto il contesto di sviluppo ed il particolare genere a cui appartiene il titolo.

*Capitolo 5:* in questa sezione sono riportate le conclusioni del progetto e i possibili miglioramenti da introdurre per eventuali sviluppi futuri.

# Capitolo 2

## Evoluzione dell’Intelligenza Artificiale nei Wargame

In questo capitolo vengono fornite le definizioni di intelligenza artificiale ed i campi ed i settori che contribuiscono alla sua identità, per presentare in seguito un sunto sull'evoluzione storica che l'intelligenza artificiale ha fatto dalle prime ricerche negli anni '50 fino ai giorni nostri.
Successivamente si passerà ad illustrare come l'intelligenza artificiale sia diventata un elemento determinante nello sviluppo dei videogame, di come questi siano cambiati e come alcuni titoli l'abbiano talmente sfruttata da farne il loro punto di forza e raggiungere il successo e di come a sua volta il settore dei videogame sia stato determinante nelle ricerche sull'intelligenza artificiale.
Nell'ultimo paragrafo verrà descritto lo stato dell'arte delle ricerche sull'intelligenza artificiale applicata principalmente ad il sottogenere dei wargame.

## 2.1 Definizione di Intelligenza Artificiale

Con intelligenza artificiale si intende la capacità di un computer di pensare ed agire secondo i meccanismi tipici del ragionamento umano.

L'intelligenza artificiale nasce intorno agli anni '50 come branca dell'ingegneria informatica con l'obbiettivo di studiare la possibilità di creare un calcolatore capace di lavorare e risolvere problemi indipendentemente dalla presenza di un operatore. Da un punto di vista puramente informatico si tratta dunque di fornire ad una macchina quelle basi teoriche e quelle tecniche necessarie alla formulazione di algoritmi in grado di mostrare un'attività intelligente.
In questa ricerca, l'intelligenza artificiale si è avvalsa di tecniche, punti di vista ed idee di altre discipline come:

- Matematica.
- Economia.
- Neuroscienza.
- Cibernetica.
- Linguistica.
- Filosofia.
- Psicologia.

Questi, e molti altri settori ancora, hanno a loro volta beneficiato dei risultati ottenuti nel campo dell'intelligenza artificiale.
In figura 2.1 sono mostrate otto definizioni di intelligenza artificiale ( da qui in poi abbreviata in IA), che spaziano lungo due dimensioni:

- Dall'alto verso il basso si va rispettivamente dal concetto di *pensare* a quello di *agire.*
- Dalla sinistra alla destra ci si riferisce rispettivamente ia concetti di *umanamente* e *razionalmente.*

Insieme le due dimensioni si combinano creando quattro differenti aree, ognuna con due famose definizioni: *Thinking Humanly, Thinking Rationally, Acting Humanly, Acting Rationally.*

Ognuna di queste quattro aree rappresenta un tipo di approccio con caratteristiche, motivazioni ed obbiettivi finali diversi dagli altri e verranno brevemente illustrati:

| **Thinking Humanly** | **Thinking Rationally** |
|---|---|
| *"The exciting new effort to make computers think ... machines with minds, in the full and literal sense."* (Haugeland, 1985)<br><br>"*[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning.. ."* (Hellman, 1978) | "*The study of mental faculties through the use of computational models.*" (Charniak and McDermott, 1985)<br><br>"*The study of computations that make it possible to perceive, reason, and act.*" (Winston, 1992) |
| **Acting Humanly** | **Acting Rationally** |
| "*The art of creating machines that perform functions that require intelligence when performed by people.*" (Kurzweil, 1990)<br><br>"*The study of how to make computers do things at which, at the moment, people are better.*" (Rich and Knight, 1991) | "*Computational Intelligence is the study of the design of intelligent agents.*" (Poole *et at*, 1998)<br><br>"*IA... is concerned with intelligent behavior in artifacts.*" (Nilsson, 1998) |

*Tabella 2.1: Definizioni di Intelligenza Artificiale.[1]*

### 2.1.1 Acting Humanly

Nel 1950 Alan Turing propose sulla rivista *Mind*[2] il famoso *Test di Turing*, costruito per dare una definizione operativa, di intelligenza artificiale:

Nel test sono coinvolte tre entità: una persona che pone delle domande, in forma scritta, ad un'altra persona ed ad un computer munito di intelligenza artificiale. L'interrogatore non può sapere a chi pone le domande.

Il test è passato dal computer se, dopo aver risposto, l'interrogatore non è in grado di distinguere se la risposta ricevuta gli è stata data dalla macchina o dalla persona.

Il test è stato in seguito esteso o modificato in più occasioni, ma nella sua pur apparente semplicità, pone tutte le sfide nel creare una macchina con funzioni cognitive tali da essere indistinguibili da quelle di un umano.

### 2.1.2 Thinking Humanly

Lo scopo principale di questo approccio è quello di creare una macchina in grado di pensare esattamente come se fosse un essere umano, implicando quindi tecniche e ragionamenti del modo di pensare degli esseri umani.

Per poter ricreare questo processo è necessario comprendere esattamente quali siano quei processi tali da permettere a l'uomo di formulare un ragionamento a partire dalla conoscenza che detiene.

E' in questo contesto che nascono le scienze cognitive, con lo scopo di affiancare le tecniche di IA agli esperimenti condotti nel campo della psicologia ( ma anche della neurofisiologia, della neuroscienza cognitiva e della linguistica cognitiva per citarne alcuni) al fine di indagare sui meccanismi del pensiero umano così da poterli successivamente esprimere in termini informatici.

### 2.1.3 Thinking Rationally

Il presupposto di questo approccio è quello che il "pensare" sia regolato da una serie di regole e leggi tali per cui partendo da delle premesse corrette si giunga sempre a delle conclusioni corrette. Uno dei primi studiosi a cercare di codificare queste norme fu il filosofo greco Aristotele che definì il ragionamento per sillogismo dimostrando come effettivamente fosse possibile descrivere un ragionamento secondo delle regole.

La disciplina che si occupa di questo studio è quello della *logica* che mira, attraverso le IA, a ricreare un sistema intelligente in grado di poter risolvere i problemi posti in forma logica. Vi sono però due grossi ostacoli: è necessario trasformare la conoscenza, spesso informale, nei termini formali richiesti dalle notazioni logiche, lavoro ancor più difficile quando in particolare la conoscenza non è al 100% corretta. Secondariamente, risolvere un problema nella pratica può risultare molto oneroso anche nel caso si abbiano poche centinaia di proposizioni, esaurendo in poco tempo le risorse computazionali.

### 2.1.4 Acting Rationally

Un agente razionale è un qualcosa (ad esempio un programma, o un robot, ecc ecc) in grado di operare autonomamente, percepire l'ambiente, adattarsi al cambiamento, darsi e perseguire degli obbiettivi cercando di farlo nel miglior modo possibile, o quando non esiste un modo migliore in assoluto, essere in grado di scegliere fra le possibilità a sua disposizione.

Mentre il pensare razionalmente si focalizza sul creare ragionamenti ed inferenze corrette, che sono comunque parte dell'agire razionalmente, vi sono casi in cui non vi sono ragionamenti corretti che portano a conclusioni corrette ma bisogna comunque prendere una decisione.

Il modello ad agenti presenta due grossi vantaggi rispetto gli altri approcci: è più generale rispetto al pensare razionalmente, dato che un inferenza corretta è solo una possibilità tra le molte inferenze, e secondariamente, gli agenti si prestano meglio allo sviluppo scientifico rispetto gli approcci che invece necessitano di studi sull'essere umano e sul suo modo di pensare.

## 2.2 Storia dell'IA

Il primo lavoro riconducibile al settore dell'IA fu presentato nel 1943 da Warren McCulloch e Walter Pins nel quale proposero il primo modello di neurone artificiale basato sullo studio di quelli umani: un neurone, concepito come una

proposizione logica e caratterizzato da due stati “on” ed “off”, passa allo stato “on” quando stimolato da un sufficiente numero di neuroni vicini (anch'essi strutturati come proposizioni logiche).

Nel 1949 Donald Hebb estese il lavoro sui neuroni artificiali introducendo una regola basata sull’aggiornamento ed il peso che un neurone ha nel processo di attivazione dei vicini. Questa regola di aggiornamento, chiamata *Hebbian Learning*, è un modello utilizzato ancora oggi in molte tecniche di apprendimento.
Alcuni anni dopo, nel 1951, Marvin Minsky e Dean Edmonds, costruirono lo SNARC, la prima macchina in grado di simulare una rete neurale ( nello specifico una rete neurale formata da 40 neuroni).

Il termine “Intelligenza Artificiale” fu usato per la prima volta soltanto nel 1956 durante la conferenza di Dartmouth organizzata da John McCarthy.

Nel 1958 McCarthy definì il linguaggio di alto livello Lisp, molto semplice da usare, orientato alla logica ed alla programmazione funzionale ricorsiva, capace di interpretare/compilare se stesso e con un polimorfismo in cui i dati diventano codice ed il codice può essere manipolato come un dato. Per queste sue caratteristiche, nei successivi 30 anni il Lisp fu il linguaggio predominante nel settore dell' intelligenza artificiale e tuttora rimane largamente impiegato.

Nel 1958 al MIT, e nel 1963 ad Harvard, furono fondati i primi laboratori universitari sulla ricerca nel settore dell' IA.

Frank Rosenblatt ampliò gli studi sull' *Hebbian Learning* giungendo, nel 1962, alla creazione del *Perceptron* ed al *Teorema di convergenza del Perceptron*: anziché usare un l’algoritmo di apprendimento statico, il teorema introduce un algoritmo più raffinato in grado ad ogni iterazione di regolare automaticamente i pesi delle connessioni fra i perceptron, così da poter convergere con qualsiasi dato di input in un tempo finito, ammesso che la convergenza esista.

Questo periodo d’oro della ricerca nel campo dell' IA, portò molti scienziati e ricercatori a sbilanciarsi nel fare dichiarazioni su come nel giro di 10 anni i

sistemi basati su IA avrebbero cambiato il mondo, conquiste che in realtà arrivarono all'incirca 40 anni dopo.

I domini in cui venivano applicate le tecniche di IA, come ad esempio i *microworlds* proposti da Marvin Minsky ai suoi studenti, problemi da affrontare e risolvere tramite tecniche di IA, erano problemi dai domini semplificati, e le stesse tecniche applicate in contesti più ampi o di diversa natura fallivano.

Si assumeva inoltre che bastasse semplicemente aumentare le risorse come capacità computazionale e spazio necessario a conservare i risultati parziali per poter affrontare tutti i problemi, ma, come poi dimostrato dall'informatica teorica, alcuni problemi erano, così posti, matematicamente irrisolvibili.

Altra forte limitazione era il fatto che molti programmi lavorassero con linguaggi di programmazione orientati alla manipolazione della sola sintassi e non della semantica e dunque con un potere espressivo limitato.

Un ulteriore freno alla ricerca sull'IA arrivo quando si comprese che, oltre alle già sopracitate problematiche, vi erano delle fondamentali limitazioni anche nelle strutture usate: il *Perceptron* infatti, per quanto potesse apprendere qualsiasi cosa fosse in grado di rappresentare, poteva in realtà rappresentare soltanto funzioni lineari, fattore che bloccò la ricerca sulle IA basate su reti neurali.

In questo periodo la ricerca si focalizzò sui sistemi basati sulla conoscenza (*Knowledge-based systems*): mentre i primi lavori di ricerca fatti nelle IA prevedevano di giungere ad una soluzione dei problemi tramite la concatenazione di successivi ed elementari ragionamenti, si utilizzò un nuovo approccio, basato sull'esperienza acquisita degli esperti ( relativi al settore del problema cui si applicava l'IA) che, forti della conoscenza di regole e comportamenti ben noti, potevano fornire ai ricercatori delle conoscenze di base più estese e precise, permettendo alle IA la formulazione di ragionamenti più estesi.

Da questo approccio deriva ad empio il programma *Dendrial*: sviluppato per inferire la struttura molecolare di un composto analizzato da uno spettrometro di massa, inizialmente venivano realizzate tutte le possibili strutture, ma dopo aver consultato degli esperti, fu possibile inserire nel sistema il riconoscimento di sottostrutture molecolari ricorrenti e già ben conosciute (a seconda dei picchi raggiunti dallo spettrometro di massa) così da giungere alla struttura dell'elemento da riconoscere in maniera più veloce e precisa.

Il successo del programma *Dendrial* richiamò l'attenzione delle aziende a richiedere i più svariati *Expert System* e ciò contribuì alla nascita di linguaggi basati su rappresentazioni logiche come il *Prolog*(1972).

L'invenzione dell'algoritmo di *Backpropagation*, avvenuta verso la fine degli anni '80 ad opera di Paul Werbos (in realtà era stato già definito nel '69 ma passò inosservato fino a quel momento) permise di superare l'iniziale limitatezza del *Perceptron* e di riprendere le ricerche sulle reti neurali e, maturati ormai i tempi, di adoperare un vero e proprio metodo di ricerca scientifico; le nuove ipotesi venivano sottomesse a rigorosi esperimenti ed i risultati analizzati statisticamente (esperimenti ora sempre riproducibili grazie alla conservazione dei dati di testing e del codice in repository condivisi e più facilmente accessibili).

Gli anni successivi videro la formalizzazione del *modello di Markov* (*hidden Markov model*) e delle *reti bayesiane* (*bayesian networks*) che introdussero importanti rivoluzioni, primo fra tutte, l'applicazione delle intelligenze artificiali in procedimenti di ragionamento in ambienti ad informazione incompleta.

Col tempo una miglior comprensione dei problemi e della loro complessità, procedimenti matematici ed algoritmi più sofisticati, nuove conoscenze e nuovi teoremi, hanno portato alla creazione di altri sotto settori dell'informatica come la robotica, la visione artificiale, la rappresentazione delle conoscenze, e non per ultimi, degli agenti intelligenti ( o autonomi), entità artificiali in grado di compiere delle azioni in autonomia e collaborare con altri agenti, chiamati comunemente *bot* e che hanno trovato le più svariate applicazioni in Internet.

Nel 1997 la macchina sviluppata da IBM e conosciuta con il nome di Deep Blue riuscì per la prima volta a vincere una partita a scacchi contro Garry Kasparov, all'epoca il campione mondiale di scacchi, realizzando così una di quelle previsioni che i primi ricercatori impegnati nel campo dell'IA avevano fatto 40 anni prima.

*Tabella 2.1 Foto dello storico incontro tra Kasparov e Deep Blue.*

## 2.3 IA nei Videogiochi

Intorno la fine degli anni '50, alcuni ricercatori delle università americane ebbero la geniale intuizione di utilizzare i computer a loro disposizione per eseguire programmi da loro scritti nel tempo libero per poter esplorare le potenzialità della programmazione e delle macchine stesse. William Higinbotham nel 1958 programmò *Tennis for Two*[3] su un computer analogico *Donner 30* ed utilizzando come display un oscilloscopio con lo scopo di creare un intrattenimento per i visitatori del *Brookhaven National Laboratory.* I primi

videogiochi prodotti con fini commerciali furono però programmati a partire dal 1970.

Fin da subito l'IA trovò grande applicazione all'interno dei videogiochi: questi infatti presentando complicate e variegate sfide, basti pensare al gioco degli scacchi ed al livello di sfida che il gioco stesso offre, dimostrarono di essere un ambiente ideale per implementare e testare le varie tecniche di IA. L'aspetto ludico e d'intrattenimento dei giochi diventò esso stesso un campo da esplorare suscitando nei programmatori l'interesse a creare videogiochi fini a se stessi, con il solo scopo di intrattenere e divertire.

Prendendo ancora come esempio il gioco degli scacchi, il gioco probabilmente più utilizzato nella ricerca sull'IA, il dominio del gioco è formato (ed quindi anche limitato) dalle stesse regole del gioco. Definendo come "stato" ogni possibile configurazione della scacchiera (32 pezzi su 64 caselle), il passaggio tra uno stato all'altro avviene di conseguenza con lo spostamento di un pezzo o la cattura di un pezzo avversario. Ogni partita sarà dunque una successione di stati composta da: uno stato iniziale ( la configurazione iniziale dei pezzi sulla scacchiera), una serie di stati intermedi, ed uno stato finale, equivalente ad uno scacco matto o ad un pareggio.

Un tipico esempio di IA applicata agli scacchi consiste nel far scegliere ad ogni turno giocato dal computer il successivo stato, con l'obbiettivo di giungere ad uno stato finale di scacco matto e dunque di vittoria.

Matematicamente è stato stimato che "*il numero di mosse ammesse dei 32 pezzi sulle 64 caselle della scacchiera sia compreso fra* $10^{43}$ *e* $10^{50}$ *e la dimensione dell'albero delle mosse è approssimativamente di* $10^{120}$*,mentre il numero di possibili partite a scacchi è di circa* $10^{10^{\wedge}50}$ "[4]. Usare un approccio di forza che elenchi ogni singolo stato ed esplori tutte le successive mosse per determinare quella che porterà il maggior vantaggio è irrealizzabile, pertanto il compito dell'IA è quello di determinare la successiva mossa qualsiasi sia lo stato in cui si trova con le risorse a disposizione.

I primi videogiochi come Pac-Man o Space Invaders più che delle vere e proprie IA avevano una sequenza di istruzioni scriptate in maniera tale da simulare un comportamento "intelligente", ma già nel 1986 la famosa serie *Chessmaster2000*[TheSoftwareToolwork 1986] poteva vantare un'IA di altissimo livello, anche se non in grado ancora di vincere contro un "maestro".

Un genere che vide lo sviluppo di IA particolarmente evolute e competitive fu quello degli strategici; mentre lo sviluppo della componente grafica veniva lasciata in secondo piano, i programmatori si focalizzarono sul come espandere la profondità della simulazione così da fornire sfide sempre impegnative e diversificate.

Uno dei migliori esempi del genere è la serie *Civilization*[Micropose 1991]: i titoli sviluppati per questa serie, che rientrano nella sottocategoria degli strategici a turni, si focalizzano tanto sulla gestione delle unità da combattimento ( e di quelle non da combattimento), quanto alla gestione di uno Stato formato da diverse città, con la necessità di affrontare problemi come l'approvvigionamento di risorse, la ricerca scientifica, la produzione di strutture ed unità.

Per simulare un avversario vengono utilizzate contemporaneamente più IA, ognuna delle quali specializzata nel gestire un particolare settore: un IA gestisce le scelte a breve termine delle unità per la realizzazione dei loro movimenti o attacchi, un'altra si occupa delle scelte a lungo termine che riguardano la gestione dell'intera nazione simulata, scelte come il fondare o no un'altra città, dichiarare guerra ad un avversario, o quali abilità migliorare.

Anche negli strategici real-time, sottocategoria dei videogiochi strategici in cui tutti i giocatori interagiscono contemporaneamente, ad esempio *Warcraft*[Blizzard Entertainment 1994] ed *Age of Empires*[Ensemble Studios 1997] per citare due famosissimi titoli, vengono utilizzate IA per simulare degli avversari.

Queste IA oltre alla gestione di strutture dedicate alla raccolta di risorse ed alla produzione di unità, devono anche muovere centinaia di unità

contemporaneamente, affrontando quindi problemi di ricerca del cammino migliore tra due punti e tutte le sfide aggiuntive di un ambiente real-time come l'adattamento della strategia in base all'evoluzione della partita.
Il genere degli strategici in generale ha tratto grandi benefici dalle IA; una sessione di gioco richiede generalmente un paio di ore per concludersi, e nel caso degli strategici a turni, anche giorni, implicando notevoli problemi nell'organizzare e portare avanti partite online in multiplayer con altre persone, motivo per cui è necessario inserire una simulazione che dia la possibilità di giocare contro il computer stesso e poter sospendere e riprendere la partita in ogni momento.

Un altro genere, gli *sparatutto in prima persona (FPS: First Person Shooter)* fanno largo uso di IA per gestire i bot ( ad esempio i nemici o i personaggi gestiti dal gioco e non dal giocatore) col fine di simulare realisticamente il loro comportamento così da sembrare "intelligenti" o gestiti da un essere umano.

Prendendo ad esempio *Thief: The dark project* [LookingGlassStudios 1998] in cui il giocatore riveste il ruolo di un ladro, deve raggiungere diversi obbiettivi evitando di essere catturato (ed ucciso) dai personaggi gestiti dal computer. A seconda delle azioni che il giocatore compie, si può generare un diverso livello d'allerta, e l'IA è necessaria per simulare le azioni che i bot devono intraprendere di conseguenza.

Molti altri giochi hanno dimostrato come fosse possibile applicare tecniche di IA più particolari, ad esempio le *fuzzy-state machines* e l'*artificial life* in *The Sims*[Maxis 2000], gli *autonomous agent system* nel reboot di *SimCity*[*Maxis* 2013] o l'uso di tecniche di *machine-learning* in *Black & White* [Lionhead 2001].

Questi sono soltanto alcuni esempi di titoli che fanno uso di IA e di come queste a loro volta influenzino e determinino il gameplay del gioco stesso.

Sono di seguito presentate le principali tecniche di IA[5] utilizzate nei videogiochi:

- **Expert Systems:** Una base di conoscenza è costruita partendo dall'esperienza messa a disposizione da esperti nel campo dell'applicazione, simulando così quei ragionamenti che guiderebbero lo stesso esperto messo al posto dell'IA.
- **Case-based reasoning:** Al sistema viene dato in input un set di dati da comparare con un database di casi ben conosciuti o già acquisiti così da scegliere il migliore output possibile per quel set.
- **Finite-state machines:** E' un sistema rule-based nel quale un numero finito di stati sono fra loro collegati in una struttura a "grafo" tramite transazioni. La *macchina a stati finiti* si trova sempre esattamente in un solo stato che cambia nel momento in cui viene eseguita una transizione in uno stato adiacente e collegato. Potendo ad esempio costruire l'intero spazio degli stati del gioco degli scacchi, sarebbe possibile creare una IA perfetta dato che saprebbe sempre quale sequenza di azioni può eseguire per andare in vantaggio.
- **Production systems:** Sono sistemi composti da un database di regole caratterizzate da uno o più stati condizionali e con un certo numero di azioni disponibili in seguito al soddisfacimento della regola stessa.

  Nel caso in cui più regole siano contemporaneamente soddisfatte esistono meccanismi di risoluzione dei conflitti.
- **Decision trees:** Molto simili agli stati condizionali dei *production systems*, le regole dei *decision trees*, sono strutturate ad albero in maniera tale che, partendo dalla radice dell'albero, siano attraversati i vari livelli a seconda dell'input fornito.

  Ogni possibile percorso dal nodo radice ad un nodo foglia porta a determinare quale output eseguire.
- **Search methods:** Sono tecniche focalizzate sulla ricerca di una serie di azioni ( o stati di un grafo) da compiere per raggiungere ad un obbiettivo ( o un particolare stato) secondo determinati criteri da ottimizzare.
- **Planning systems e scheduling systems:** Sono un estensione dei *search methods* che enfatizzano la ricerca della sequenza ottima, partendo da un qualsiasi stato, per raggiungere una precisa azione ( o

stato) in un determinato tempo, tenendo conto delle conseguenze derivanti dall'attraversamento di ogni stato o possibile azione all'interno del percorso.

- **First-order logic:** Estensione della logica proposizionale che permette di formalizzare ragionamenti sull'intelligenza artificiale contestualizzata all'ambiente che la circonda. L'ambiente è organizzato in "oggetti" con identità individuali, "proprietà" che contraddistinguono gli oggetti e "relazioni" che sussistono tra oggetti e proprietà.
- **Multi-agent system:** Approccio focalizzato sulle interazioni di agenti "intelligenti" in ambienti cooperativi o competitivi.
- **Artificial life:** Si riferisce a quei sistemi multi-agent nei quali si applicano alcune delle caratteristiche universali dei sistemi viventi agli agenti che compongono il sistema.
- **Genetic algorithms** e **genetic programming:** Queste tecniche permettono di imitare direttamente i processi evolutivi e di selezione ed ibridazione proprie della natura, tramite tecniche di crossover e mutazioni di popolazioni composte da programmi, algoritmi o set di parametri.
- **Neural networks:** Sono una particolare classe di *machine learning* formate da un'architettura basata sulle interconnessioni neurali del cervello e del sistema nervoso. Il loro modo di operare prevede un costante aggiustamento dei parametri numerici assegnati alle interconnessioni tra i neuroni componenti il network così da poter calibrare i processi di apprendimento.
- **Fuzzy logic:** Usando dei numeri reali per rappresentare i gradi di appartenenza di un input ad un determinato set, è possibile esprimere ragionamenti più ricchi e precisi rispetto la tradizionale logica booleana.
- **Belief networks** e **Bayesian inference:** Sono reti usate per modellizzare le relazioni causali tra differenti fenomeni utilizzando i principi matematici della teoria della probabilità per poter lavorare anche in caso di informazioni incomplete.

  Tramite queste reti è possibile anche inferire sulle probabilità dei possibili effetti che potrebbero verificarsi in seguito ad un azione.

Tutte queste tecniche sono state implementate, con più o meno successo, nei videogiochi, ma quelle più utilizzate, sono le *finite-state machines,*gli *expert systems,* i *decision tree* ed i *production rule system* per via della loro facile creazione, comprensione ed analisi, ma anche per il numero più ridotto di risorse richieste.

Questi sistemi basati su regole fisse, hanno una forte componente limitativa: dal momento che le regole sono fissate, scoperto come l'IA reagisce ad un determinato tipo di input, il giocatore può a sua volta trarre vantaggio da questa conoscenza ed adeguare la sua strategia a quella immutabile del computer, di fatto minando l'esperienza di gioco.

Gli approcci esplorativi dello spazio degli stati, come ad esempio viene fatto per i giochi da tavolo come gli scacchi, la dama o il go, hanno invece fortissime limitazioni non solo in termini di potenza di calcolo richiesta, ma anche di spazio e soprattutto tempo per poter esplorare gli stati elaborati, risultando inadeguati per i comuni computer e soprattutto per gli ambienti real-time.

Implementare una IA di un certo livello richiede risorse in termini di costi ed ore di lavoro, ed un buon risultato si ottiene solo se il design della IA inizia insieme a tutte le fasi di design, processo diventato ormai consuetudine nello sviluppo di un videogame, ma che precedentemente veniva rilegato alle ultime fasi di sviluppo, con risultati spesso imbarazzanti.

Alcuni giochi, ad esempio la serie *Civilization,* presentano una sorta di "trucco" nel momento in cui i giocatori decidono di giocare contro una IA più competitiva: mentre si ci aspetterebbe di competere contro una IA più intelligente e capace di ragionamenti in grado di creare un gameplay più variegato, ciò che in realtà avviene è la sola variazione dei valori utilizzati da parte dell'IA, ad esempio il tempo di costruzione di un edificio o dell'addestramento di un'unità è minore rispetto a quelli del giocatore. Il risultato è quindi una partita più difficile per il giocatore, ma dal punto di vista dell'IA, le scelte rimangono invariate rispetto a quelle che farebbe in una partita con un grado di difficoltà minore.

Mentre le IA sviluppate in ambito scientifico hanno come scopo quello di utilizzare l'enorme potenza di calcolo delle macchine su cui lavorano per ottenere risultati straordinari, come vincere contro i miglior giocatori di scacchi al mondo, risulterebbe invece inadeguato per un gioco di tipo commerciale; difficilmente i giocatori si impegnerebbero in una partita contro un'IA talmente performante da risultare invincibile. Un'IA deve quindi saper perdere, e saper perdere bene, senza cioè sconfinare nella banalità, poiché anche perdendo sempre otterrebbe lo stesso effetto suscitando ancora una volta poco interesse nei giocatori.

Un'IA dovrebbe dunque essere calibrata in maniera tale che un giocatore giocando una partita quanto meno buona, possa vincere ma soprattutto l'IA deve riuscire a sostenere sempre un adeguato livello di sfida tale da non annoiare o frustrare il giocatore umano.

Le altre tecniche precedentemente elencate hanno invece incontrato molta fortuna nel risolvere alcuni sotto-problemi o in determinati contesti, ma hanno trovato poco successo se applicati altrove o su problemi più grandi o soprattutto quando applicate in ambienti real-time o con risorse computazionali limitate.

Le *learning machine* sono delle tecniche più complesse da creare, modificare e testare, dunque con oneri più grandi per i programmatori, e basate su una funzione, detta di *fitness*, che giudica quanto imparato dalla macchina. Se da una parte l'obbiettivo è quello di migliorare le proprie capacità per vincere contro l'avversario, ciò che impara deve comunque portare alla soddisfazione dell'utente ed al suo divertimento. La funzione di fitness deve dunque tenere conto di questi fattori, misurare il grado di abilità può anche essere semplicemente definito in base al risultato della partita (vittoria +1, sconfitta -1, pareggio 0 ad esempio), misurare la soddisfazione del giocatore, e perché il giocatore è rimasto soddisfatto, risulta molto più difficile e soggettivo. Rimane inoltre il problema di una fase iniziale di training (fatta dagli sviluppatori) nella quale impari le basi.

Tutte queste tecniche più complesse devono inoltre affrontare un particolare problema: spesso, a seconda del genere del videogioco o del problema da affrontare, risultano troppo performanti per i brevi lassi di tempo in cui sono richieste e né avrebbero risultati molto differenti da quelle basate su conoscenze e regole pre-acquisite, con l'unica differenza di avere un costo d'implementazione maggiore.

Sviluppare una IA non è dunque una cosa semplice, e soprattutto non esiste una soluzione universale, perché ogni tipo di genere, ed all'interno di ogni genere, ogni tipo di gioco, presenta sfide tanto simili quanto diametralmente opposte, rendendo l'IA dipendente dal contesto applicativo.

Si evince come la ricerca nel campo dell'IA applicata ai videogiochi, sia ancora oggi un settore che offre moltissime sfide, oggi anche amplificate dalla richiesta di prestazioni performanti in ambienti mobile come i tablet di nuova generazione. Nel prossimo paragrafo verranno presentate diverse ricerche fatte in questo ambito, e soprattutto rivolte ai board-game strategici, genere a cui appartiene il gioco sviluppato in questo lavoro di tesi.

## 2.4 Stato dell'arte

Molti studi sono stati fatti in tutte le tecniche precedentemente elencate al fine di migliorare i livelli delle IA da applicare ai videogiochi. Gli studi quì presentati sono stati scelti per l'approfondimento che hanno apportato alla tecnica del Monte-Carlo Tree search, tecnica risultata molto promettente nel particolare settore dei giochi da tavolo classici, primi fra tutti scacchi e dama, e per un eventuale implementazione dell'IA nel titolo sviluppato in questo lavoro di tesi, gioco per molti aspetti vicino ai board game classici.

**Monte-Carlo Tree Search: A new Framework for Game AI.[6]**

Nell'implementazione dell'IA nei classici giochi da tavolo come gli scacchi, il go o il kriegspiel, il fattore più importante risulta essere la funzione di valutazione di uno stato intermedio di gioco. Usualmente la funzione di valutazione si basa su euristiche estratte dal dominio del gioco ad esempio, il numero ed il tipo di pezzi

mangiati o la posizione in scacchiera dei pezzi, informazioni comunque strettamente legate al dominio e che necessitano di un grosso lavoro d'analisi.

Il Monte-Carlo Tree Search (MCTS) è una tecnica "best-first" che usa simulazioni stocastiche. Questa tecnica può essere utilizzata su giochi di lunghezza finita e si basa sulla simulazione di una partita giocata fra IA che utilizzano mosse random, o meglio ancora, pseudo-random, con l'obbiettivo di inferire una strategia di gioco da una moltitudine di partite, ognuna delle quali in grado di fornire limitate informazioni che verranno utilizzate dall'algoritmo per costruire un albero degli stati da utilizzare per le future partite, secondo questo meccanismo:

- Selezione: Quando lo stato di gioco è già presente nell'albero, sceglie l'azione da compiere rispetto due differenti approcci, uno conservativo, cioè che sfrutta le informazioni già immagazzinate, tipicamente quindi orientato a fare la miglior scelta possibile, l'altro invece esplorativo, cioè capace di fare scelte non ottime affinché sia possibile esplorare stati che siano inizialmente poco promettenti ma che possono in seguito portare a situazioni migliori di quelle che si otterrebbero.
- Espansione: Quando il gioco raggiunge il primo stato non presente nell'albero, viene aggiunto come nuovo nodo. In questa maniera l'albero si espande di un nodo per ogni partita.
- Simulazione: Il resto della partita viene simulato giocando azioni possibilmente pseudo-random altrimenti se le azioni disponibile fossero scelte tutte con uguale probabilità si otterrebbe mediamente una strategia debole. Per questo alle azioni a disposizione si preferisce dare un peso basato sulle euristiche definite dal dominio del gioco.
- Backpropagation: Una volta finita la simulazione della partita, tutte le caratteristiche degli stati attraversati presenti nell'albero vengono aggiornati.

Le tecniche di "taglio dell'albero" (meglio conosciute come alpha-beta pruning) hanno ottenuto grandi successi in giochi come scacchi o dama, ma sotto due precise condizioni: una buona funzione di valutazione già esistente, è un fattore

di ramificazione dell'albero basso. MCTS ha ottenuto  ottimi risultati non solo in questa tipologia di gioco, ma anche in quelli dove non sono applicabili le tecniche di alpha-beta pruning, come il go.
Per quando riguarda invece la sua applicabilità nei videogiochi, costituiti solitamente da un dominio più realistico e complesso, la sua implementazione si è rivelata comunque efficiente nel costruire IA adattabili al contesto sin dalle prime fasi di gioco.

**Search Versus Knowledge in Game-Playing Programs Revisited[7]**

Nel momento in cui un'IA disponesse della conoscenza totale del dominio dell'applicazione per cui è sviluppata, non necessiterebbe di dover implementare fasi di ricerca dato che saprebbe in qualsiasi momento cosa fare. Dall'altro lato, disponendo di un sistema di ricerca esaustivo ed efficiente, l'IA non sarebbe dipendente dal contesto essendo in grado di determinare la miglior azione possibile partendo dalle informazioni raccolte con la ricerca stessa. Spesso, non disponendo ne della conoscenza totale, ne di un sistema di ricerca esaustivo ( e soprattutto efficiente) lo sviluppo dell'IA ricade nel mezzo di questi due estremi. Sono stati condotti diversi studi sulla possibilità di implementare, o per lo meno far tendere, un'IA verso uno degli estremi. Dei due, quello più facile da ottenere è senza dubbio il lato della ricerca, dal momento che il concetto di conoscenza è già per sua natura più complesso da definire.

Lo studio quì riportato è partito inizialmente da un'analisi teorica delle dimensioni coinvolte e di cosa implica trovarsi in uno degli estremi. Avendo una conoscenza perfetta del dominio, emerge comunque che è necessario avere almeno un minimo livello di ricerca per poter valutare la scelta migliore da fare, mentre è impensabile trovarsi effettivamente in condizione da non aver nessuna informazione, altrimenti non sarebbe neanche possibile determinare la vittoria, la sconfitta o il pareggio. Ne risultano delle isocurve concave verso l'alto, ognuna delle quali misura un livello di prestazione in cui il 100% corrisponderebbe ad una partita perfetta, mentre i livelli inferiori sono rappresentanti i livelli di gioco semi-random, ma tutte originarie da un livello di

ricerca uguale 1. Per ottenere la partita perfetta è necessario un oracolo in grado di predire perfettamente le azioni da compiere, naturalmente solo approssimabile. L'inaccuratezza di una partita non perfetta può essere vista invece come un disturbo nel livello di conoscenza. Dal momento che in molti giochi come gli scacchi il valore di uno stato dipende da quelli successivi, il disturbo è stato artificialmente introdotto alla valutazione della qualità di uno stato in modo da rispecchiare questo comportamento. Non potendo disporre di un oracolo perfetto, la valutazione di una partita giocata è stata fatta rispetto ad algoritmi ben noti ed altamente performanti già esistenti (*TheTurk* per gli scacchi, *Chinook* per la dama, *Keyano* per Othello) confermando sperimentalmente le ipotesi iniziali.

**Progressive Strategies for Monte-Carlo Tree Search. [8]**

L'algoritmo MCTS per quanto molto performante necessità di un elevato numero di partite prima che raggiunga livelli soddisfacenti. Il passaggio più importante di tutto il processo è quello della selezione, in cui l'algoritmo deve scegliere se seguire un approccio esplorativo oppure conservativo rispetto le informazioni possedute, scelta da cui si determina la velocità di convergenza dell'algoritmo stesso. Sono proposti in questo studio due differenti strategie per calibrare il passaggio tra i due approcci, *Progressive Bias* e *Progressive Unpruning*, applicate al gioco del Go.

Nella prima strategia, l'obbiettivo è quello di regolare la strategia di ricerca in base ai tempi necessari alla raccolta di informazioni. Nelle prime fasi infatti si prediligono gli aspetti esplorativi, per poi decrescere velocemente per tendere ad un approccio selettivo ( quindi basate sulle informazioni). Il passaggio viene fatto utilizzando la tecniche di selezione UCT (Upper Confidence bounds applied to Trees) pesata sulla configurazione dello stato di gioco.

La strategia *Progressive Unpruning* invece tende allo stesso obbiettivo ma seguendo un'altra via: ogni qual volta viene aggiunto un nuovo nodo, tutti i figli vengono rimossi dal padre, meno una quantità. I figli tagliati fuori vengono

aggiunti progressivamente, partendo dai figli più promettenti, in base al numero di volte che lo stato viene visitato ed alla valutazione dello stato di gioco.

La valutazione degli stati viene fatta su euristiche controllate temporalmente, fornendo lassi di tempo sempre più brevi ai nodi visitati al di sopra di un certo numero di volte. Entrambe le tecniche hanno dimostrato di riuscire a ridurre i tempi di convergenza del MCTS e di mantenere comunque un livello di competitività elevato.

# Capitolo 3

## Il wargame di Crepax

Il primo paragrafo di questo capitolo è incentrato sul genere dei *wargame* e sulla definizione di quelle caratteristiche comuni a tutti i titoli che ne fanno parte. Il secondo paragrafo si focalizza invece sull'evoluzione storica di questa categoria di giochi da tavolo dall'antichità fino ai giorni nostri. Infine, il terzo paragrafo introdurrà il wargame sviluppato da Guido Crepax nonché base del titolo sviluppato per tablet: verranno spiegate in dettaglio istruzioni, regole, il campo di battaglia e le unità.

## 3.1 Cos'è un wargame

Il termine wargame spazia dal settore delle simulazioni militari che gli eserciti usano per addestramento ed esercitazione, ai giochi da tavolo, ai videogiochi, in particolar modo a quelli appartenenti al genere degli strategici e degli FPS( First Person Shooter).

Il minimo comune denominatore di questi settori è espresso da tre grandi fattori che li contraddistinguono:

- **Una connessione al mondo militare:**

  Tutti i giochi da tavolo ed i videogiochi che ricadono nei wargame si inspirano in maniera più o meno realistica a quello militare, prendendone in prestito nomi, concetti ed idee. I pezzi degli scacchi ad esempio sono le storiche unità che troveremmo in un campo di battaglia medievale, così come il gameplay si sviluppa ed utilizza termini e concetti tipici della gestione, della tattica e della strategia del mondo militare.

- **Sono giochi:**

  Per definizione stessa di gioco, più partecipanti sono messi di fronte ad un obiettivo da raggiungere e per farlo devono fra loro competere/cooperare ed agire in modi nuovi e creativi rispetto un sistema di regole predefinite. Anche le esercitazioni militari, estratte dal contesto dell'addestramento bellico, ricadono perfettamente in questa definizione.

- **Simulano:**

  Tramite la simulazione i wargame rendono possibile la creazione di contesti facilmente ripetibili dai quali estrapolare ogni volta vecchie e nuove informazioni da elaborare ed analizzare con fini educativi.

## 3.2 Storia dei Wargame

Alcuni reperti ritrovati negli scavi archeologici delle antiche civiltà hanno dimostrato che i giochi da tavolo erano già all'epoca molto diffusi ed in uso, con molte variazioni da popolo a popolo. Uno dei primi reperti classificato come gioco da tavolo fu ritrovato presso le tombe degli antichi egizi: questo oggetto è una tavola con una superficie divisa in celle e colonne nella quali posizionare delle pedine che vi si spostano di un numero di spazi definito tramite un meccanismo di sorteggio casuale.

Il piano da gioco diviso in celle e l'uso del dado, sono due elementi presenti anche nei nostri moderni giochi da tavolo, per esempio il *"Gioco dell'oca"* e le variazioni come *Scale e serpenti, Scale e scivoli* o *Il gioco della vita*.

Altri reperti associati ai giochi da tavolo furono ritrovati anche presso le tombe degli antichi Greci e dei Romani. In particolar modo il gioco chiamato *latrunculi* fu ritrovato nella maggior parte dei territori conquistati dalle legioni romane), dimostrando come questo fosse considerato un'importante intrattenimento.

Sia il gioco dei *Latrunculi,* sia il gioco giapponese del *Go,* sono caratterizzati da dinamiche molto simili a quelle dei moderni scacchi.

Il gioco degli scacchi si credeva inizialmente derivasse dalla cultura europea e che nel corso dei secoli avesse subito diverse modifiche fino alla più recente versione.

Si è scoperto invece essere sì un'evoluzione, ma di un antico gioco indiano chiamato *Chaturanga* nel quale due avversari dovevano cercare di catturare il re nemico tramite l'utilizzo di altri pezzi.

Mentre il re poteva muoversi nelle quattro caselle adiacenti di un solo passo, gli altri pezzi, raffiguranti quelli che all'epoca erano i principali settori delle forze militari indiane, avevano possibilità di muoversi più liberamente, come ad esempio il carro che poteva percorrere un certo numero di caselle in avanti.

L'avvicinamento delle culture europee ed asiatiche ad opera degli scambi commerciali ha portato i giochi da tavolo a diffondersi nei vari paesi, subendo variazioni ed evoluzioni rispetto le regole di base.

Questi comunque contengono già tutte e tre le dimensioni (guerra,gioco,simulazione) comuni a tutti i wargame moderni.

Nei successivi secoli i giochi da tavolo continuarono ad influenzarsi vicendevolmente fino a quando, nel 1664 Christopher Weikhamann, nato ad Ulm, in Germania cominciò a porre le basi di uno studio scientifico dei giochi da tavolo.

I suoi studi si focalizzarono sul *Königspiel*: da lui stesso definito come un "sommario dei più utili principi militari e politici", era essenzialmente un'evoluzione degli scacchi formato da circa trenta pezzi ed una scacchiera più grande. Weikhamann voleva portare il gioco ad un livello simulativo più profondo ed introdusse nuovi pezzi e nuove regole afferenti agli ambienti delle accademie militari prussiane.

La versione del *Königspiel* di Weikhamann venne a sua volta modificata da *Christian Ludwig Hellwig* (1743-1831), creando un'enorme versione composta da 1666 caselle e oltre 200 pezzi rappresentati corpi specializzati dell'esercito e non più classi generiche.

Il gioco si diffuse moltissimo nelle corti europee, e soprattutto negli ambienti militari, per il suo maggiore livello di simulazione.

Tra il 1780 ed il 1824 vennero fatte ulteriori modifiche, basate sull'analisi delle vere situazioni di combattimento (velocità delle unità, statistiche dei pezzi, ecc, ecc) così da poter riprodurre, fedelmente ed in una scala più piccola, un vero campo di battaglia sul quale gli ufficiali potessero allenarsi e fare addestramento.

Nel 1824 il luogotenente prussiano *Von Reisswitz* creò una versione chiamata *Kriegspiel* (l'equivalente tedesco della parola wargame) che si distanziò sensibilmente da quelle precedenti: in questa versione venne abbandonato il campo di battaglia suddiviso in celle, e si adottò un campo senza spazi discreti nel quale le unità si potevano muovere secondo le sole limitazioni dovute alle caratteristiche delle unità che rappresentano.

*Figura 3.1 Moderna riproduzione del Kriegspiel.*

Delle tre dimensioni caratteristiche di un wargame, quella che più di tutte ne risulta rinforzata da tutte le modifiche introdotte, è quella simulativa, a scanso invece di quella legata al concetto di gioco fine all'intrattenimento, motivo per cui l'analisi dell'evoluzione del *Königspiel* risulta essere così importante.

Lo spostamento verso la dimensione simulativa comportò il delinearsi di due mentalità: da una parte quella degli ambienti militari che poneva maggior enfasi sulla profondità simulativa fine all'educazione militare (il *Kriegspiel rigido)*, mentre dall'altra parte vi sono coloro che continuano a vedere il *Kriegspiel* come un'evoluzione del gioco degli scacchi e come tale, un intrattenimento (il *Kriegspiel libero*), che, essendo però ancora troppo simulativo e con un elevato numero di regole, era difficile da giocare, sia per gli esperti che per i neofiti.

Nel 1913 *H.G. Wells*, ridisegnò le regole del *Kriegspiel Libero* per renderle meno rigide.

Il gioco che ne derivò, *Little War*, rimaneva comunque ambientato in un contesto militare e si basava sull'uso di miniature (militari e non) e di elementi

scenici da disporre per ricreare uno scenario ispirato ad un campo di battaglia realistico, o di pura fantasia.
Le regole meno rigide permisero al gioco una rapidissima diffusione, soprattutto al di fuori degli ambienti militari, creando una spaccatura tra i wargame simulativi usati come addestramento e quelli invece usati come intrattenimento e gioco.

## 3.3. Introduzione al gioco

Nel 1965, il fumettista italiano Guido Crepax rilasciò sulle pagine della rivista *linus* il gioco strategico a turni "*La battaglia di Waterloo"*, al quale successivamente seguirono *La battaglia di Pavia* e poi *Aleksandr Nevskij, la battaglia del lago ghiacciato, e* creando quello che fu il primo wargame italiano.
Il gioco, interamente da ritagliare dalle pagine della rivista, consisteva in quattro fogli ripiegati, i quali una volta affiancati, avrebbero rappresentato il campo di battaglia con le città, le fortezze, le alture e tutti i maggiori dettagli del reale campo di battaglia, un foglio contenente le istruzioni e le regole del wargame, ed altre otto tavole (queste disegnate fronte retro) dedicate alle unità coinvolte nello scontro: da una parte lo schieramento francese, dall'altra la coalizione formata da inglesi, prussiani ed olandesi.
Ogni singola unità, una volta ritagliata dal foglio, aveva una base con le sue caratteristiche di velocità e forza, che piegata su un lato, fungeva da piedistallo così che i pezzi potessero spiccare in tre dimensioni dal campo di battaglia.

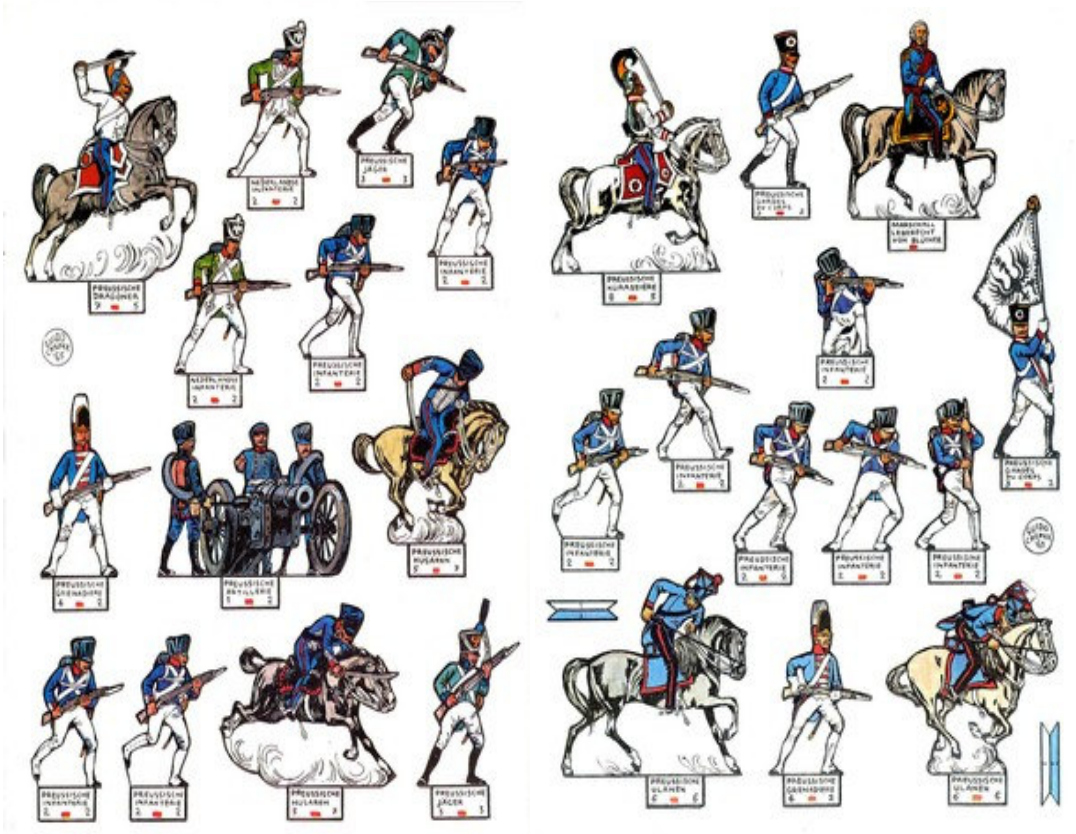

*Figura 3.2 Unità originali del board game di Crepax.*

E' doveroso ricordare che unità della stessa classe (ad esempio due fanti appartenenti alla *fanterie)* non condividevano lo stesso identico disegno, bensì ognuna aveva la propria raffigurazione, così che i due pezzi fossero fra loro unici.

Con un dado ed una forbice ogni abbonato praticamente poteva così usufruire di un gioco da tavolo completo di tutto.

Nel 1999 la rivista *Sandokan*[9] ripropose, con le stesse modalità della rivista *linus*, un inserto dedicato al gioco anche se leggermente riadattato, con un minor numero di unità ed una versione del campo di battaglia suddivisa in un minor numero di caselle.

Questa è l'edizione dalla quale si è partiti per modellare la versione digitale.

Nella battaglia di Waterloo si contrapposero le forze francesi guidate da Napoleone da un lato, e la coalizione formata da inglesi,olandesi e prussiani,

sotto il comando d Sir Arthur Wellesley, I duca di Wellington e del feldmaresciallo Gebhard von Blücher.

Sono mostrate in tabella (Tabella 3.1) le unità, divise per schieramento, che compongono il gioco.

Per ogni unità viene fornito il nome, il numero di pezzi da schierare, la tipologia di unità, la forza nel combattimento ravvicinato ed il numero di caselle di cui può spostarsi ogni turno.

| Armata | Coalizione | | | |
|---|---|---|---|---|
| Nome | Num | Tipo. | For | Vel |
| Cuirassiers | 1 | Cav. | 8 | 5 |
| Dragoons | 2 | Cav. | 7 | 5 |
| Horse Gurad | 1 | Cav. | 7 | 5 |
| Light Dragoons | 1 | Cav. | 6 | 5 |
| Hussars | 1 | Cav. | 5 | 6 |
| Scots Greys | 2 | Cav. | 5 | 7 |
| Grenadiers | 4 | Fant. | 4 | 2 |
| Infantry | 8 | Fant. | 2 | 2 |
| Black Watch | 2 | Fant. | 4 | 2 |
| Highlanders | 2 | Fant. | 3 | 3 |
| Artllery | 1 | Arti. | 1 | 2 |
| Infanterie | 4 | Fant. | 2 | 2 |
| Jaeger | 2 | Fant. | 3 | 3 |
| Infanterie Holand | 1 | Fant. | 2 | 2 |
| Husaren | 2 | Cav. | 5 | 7 |
| Kurassiere | 1 | Cav. | 8 | 5 |
| Dragoner | 1 | Cav. | 7 | 5 |
| Ulanen | 2 | Cav. | 6 | 6 |
| Garde du corps | 2 | Fant. | 3 | 2 |
| Jaeger | 2 | Fant. | 3 | 3 |
| Infanterie | 10 | Fant. | 2 | 2 |
| Grenadiere | 2 | Fant. | 4 | 2 |
| Artillerie | 1 | Arti. | 1 | 2 |

| Armata | Francia | | | |
|---|---|---|---|---|
| Nome | Num | Tipo | For | Vel |
| Cuirassiers | 2 | Cav. | 8 | 5 |
| Dragons | 2 | Cav. | 7 | 5 |
| Grenadier à cheval | 1 | Cav. | 6 | 5 |
| Chasseurs à cheval | 1 | Cav. | 7 | 6 |
| Lanciers | 2 | Cav. | 6 | 6 |
| Lanciers Polonais | 2 | Cav. | 6 | 6 |
| Hussards | 1 | Cav. | 5 | 7 |
| Vielle Guarde | 2 | Fant. | 5 | 2 |
| Grenadier | 4 | Fant. | 4 | 2 |
| Chasseurs | 4 | Fant. | 3 | 3 |
| Carabiniers | 2 | Fant. | 3 | 2 |
| Sapeurs | 2 | Fant. | 3 | 2 |
| Voltigeurs | 4 | Fant. | 2 | 3 |
| Infanterie | 12 | Fant. | 2 | 2 |
| Artillerie | 2 | Arti. | 1 | 2 |

*Tabella 3.1: Unità del board game.*

Lo sviluppo di questa tesi, nonché dello sviluppo del gameplay della versione digitalizzata e dell'aspetto grafico che utilizza i materiali originali di Guido Crepax, si è avvalso della preziosa collaborazione di Spartaco Albertarelli[10] : collaboratore di Editrice Giochi , è uno dei massimi editori italiani di giochi da tavolo.

**Schieramento**

Il gioco prevede un minimo di due giocatori, uno per lo schieramento francese e l'altro per la coalizione inglese, ma più giocatori possono partecipare suddividendo il comando degli schieramenti su sotto gruppi di unità, mantenendo naturalmente le fazioni originali.

Ogni giocatore schiera i propri pezzi sul tabellone in segreto ( è necessario utilizzare un divisorio, ad esempio un grosso foglio di carta) rispettando i limiti iniziali:

- L'esercito francese si colloca nel rettangolo formato dalle prime 8 righe e le prime 30 colonne contando dall'angolo in basso a sinistra.
- L'esercito inglese si colloca nel rettangolo formato dalle ultime 8 righe e le prime 30 colonne contando dall'angolo in basso a sinistra.
- Le truppe prussiane entreranno in un secondo momento suddivise in due scaglioni:
    - Il primo, formato da 8 pezzi ( 4 fanti, 2 jäger, 2 ussari), entrerà dal bordo della mappa all'altezza della strada per Placenoit ( la prima strada sul bordo destro contando dal basso).
    - Il secondo, formato dalle restanti truppe, entrerà dal bordo della mappa all'altezza dell'altra strada per Placenoit ( la seconda strada sul bordo destro contando dal basso).

Il momento in cui i prussiani faranno il loro ingresso in gioco è determinato dal dado.

Il giocatore che muove l'esercito inglese getterà a fine di ogni turno il dado per due volte, assegnando il primo valore all'ingresso del primo scaglione, ed il secondo all'ingresso del grosso delle truppe. Sommando di volta in volta i punteggi ottenuti ad ogni fine turno, le truppe verranno posizionate

effettivamente in campo al turno successivo in cui si è ottenuto il punteggio di 15 per il primo scaglione e di 20 per il secondo.

**Movimenti**

Lanciando il dado si sceglierà quale giocatore muoverà per primo le sue truppe.
Il tabellone è suddiviso in 1280 quadretti ed ogni unità reca sul piedistallo il numero massimo di quadretti di cui può muoversi. Un giocatore può muovere solo le proprie truppe e verso qualsiasi direzione libera da ostacoli: avanti, dietro, destra, sinistra, diagonalmente e percorsi mistilinei, sono validi.
E' assolutamente vietato per un pezzo occupare, anche solo momentaneamente o di passaggio, un quadretto su cui vi è già un pezzo (amico o nemico).
Un pezzo può anche non essere mosso e rimanere fermo per tutto il turno.

**Fuoco - Combattimento**

Pezzi di artiglieria ed unità di fanteria possono far fuoco dalla distanza. I pezzi che sparano non possono muoversi e, viceversa, un pezzo che muove non può sparare.
L'artiglieria spara nel seguente modo: dalla bocca del cannone nella direzione determinata dallo stesso cannone, si contano 6 quadretti in avanti ai quali si sommeranno il risultato del lancio di 1 o 2 dadi. Se si vuole colpire un bersaglio a destra ( o sinistra) si effettuerà un ulteriore lancio di 1 o 2 dadi sommando, alla posizione precedentemente ottenuta, l'equivalente numero di caselle in quella direzione. Se nella casella ottenuta alla fine di tutte le somme vi è un unità, amica o nemica, questa viene distrutta. Tutti i lanci devono essere dichiarati prima dell'esecuzione e successivamente eseguiti obbligatoriamente.
Dato che la direzione del tiro dell'artiglieria è determinata dalla direzione della bocca da fuoco, il cannone è l'unico pezzo che può sparare e cambiare direzione nello stesso turno.

Tutti i pezzi appiedati possono far fuoco secondo le seguenti regole:

- Contro la cavalleria: Ottenendo un numero di dado che non abbia uno scarto superiore al 2 fra il numero di quadretti che separa il fuciliere dal pezzo preso di mira.
- Contro la fanteria: Alla distanza minima di 2 quadretti e massima di 3.
  Un pezzo che dista 2 quadretti verrà colpito se il dado segnerà 2 o 3, mentre se il pezzo dista 3, il pezzo risulterà colpito soltanto se il dado segnerà 3.

Si tira sempre con un dado solo e la traiettoria del tiro di fanteria è determinata dal tratto mistilineo più breve che separa il fuciliere dal proprio bersaglio. Si può sparare soltanto su bersagli che non siano coperti da altre unità (amiche o nemiche) o ostacoli.

**Combattimento ravvicinato**

Quando due pezzi avversari si trovano su due quadretti adiacenti ( sia che si tratti del lato o di uno spigolo) avviene forzatamente uno scontro che si risolve nel seguente modo:

Getta per primo il dado chi ha attaccato e sommerà i punti indicati dal dado a quelli del valore del suo pezzo, indicato sul piedistallo.

Colui che si difende farà lo stesso ed ovviamente vincerà chi ha totalizzato il punteggio maggiore.

In caso di parità si ripeteranno immediatamente i tiri finché non sarà determinato un vincitore. Ogni pezzo eliminato deve essere tolto dal gioco.

Se due o più pezzi sono contemporaneamente adiacenti ad un pezzo nemico, al numero ottenuto con il lancio del dado si sommeranno i valori di tutte le unità coinvolte nell'attacco.

Se nel combattimento si trovano coinvolti molti pezzi da entrambe le parti, lo scontro globale si scompone in tanti scontri più piccoli tale per cui il numero di scontri parziali è sempre uguale al numero di pezzi più basso fra le due parti.

**Come si vince**

Ogni pezzo eliminato dal gioco vale ai fini del punteggio finale 1 punto, ad eccezione del cannone che vale 2 punti.
Quando lo scarto di pezzi eliminati fra i due eserciti supera i 10 punti, l'esercito che in quel momento è in svantaggio ha diritto ad un ulteriore turno.
Se lo scarto di punti non è stato ancora ridotto, avrà perso la battaglia, altrimenti si continuerà la partita con il nuovo ordine di turni.
Nel caso che uno dei due eserciti sia arrivato a metà degli effettivi in campo senza che nessuno dei due contendenti abbia raggiunto la quota di punteggio sopra citata, lo scarto da raggiungere viene ridotto a 6 punti.
Se lo scarto non viene comunque raggiunto entro 2 turni per parte la battaglia termina in parità.

**Ostacoli e vantaggi**

- Fiumi e laghi: Qualsiasi corso d'acqua non è guadabile se la sua larghezza è superiore alla diagonale di un quadretto. Bisognerà quindi utilizzare gli appositi ponti. Negli altri casi i corsi d'acqua sono guadabili con la perdita di 2 quadretti di velocità per la cavalleria e di un quadretto per la fanteria.
- Strade: Ogni pezzo guadagna un quadretto di velocità se procede per almeno due quadretti dove sono segnate delle strade. I cannoni guadagnano due quadretti poiché si suppone siano trainati da cavalli.
- Montagne: nel salire sui rilievi montagnosi ogni pezzo perde 1 quadretto di velocità. Se due pezzi sono separati da un rilievo montuoso non si possono colpire reciprocamente a distanza. Solo il tiro dell'artiglieria può raggiungere il bersaglio.
- Case e fortificazioni: nel caso di attacco ad un pezzo appiedato o di artiglieria collocato in un quadretto dove è disegnata una casa, questo aumenta il proprio valore di un punto. All'interno di fortificazioni (come ad esempio nel castello di Hougoumont) i pezzi aumentano di mezzo punto e sono immuni dai colpi della fanteria e dell'artiglieria.
  Tali fortificazioni non sono occupabili dalla cavalleria.

- Bosco: Nei quadretti occupati da bosco, cioè piante o macchie segnate con colore più scuro rispetto alle altre, non si possono collocare pezzi né vi si può transitare. Le restanti piante sparse per il campo e non contraddistinte da particolare colore, non costituiscono ostacolo di sorta.

## 3.4 Il passaggio alla versione digitale

Le regole e le istruzioni del precedente paragrafo sono quelle ufficiali allegate all'edizione più recente rilasciata nel '99 sulla rivista *Sandokan*, versione di per sé già leggermente modificata rispetto quella originale del '71 della rivista *linus.*
Alcune regole ad esempio risultano superflue, come l'impossibilità di guadare i tratti dei fiumi la cui larghezza sia maggiore della diagonale del quadratino, non essendo più presenti nella mappa ne tratti di fiume così larghi, ne ponti per guadare, dato che la mappa è stata sostituita con una versione dai disegni diversi e dal numero inferiore di quadratini.
Le precedenti istruzioni e regole del gioco sono in alcuni punti poco chiare ed addirittura lasciano spazio ad interpretazioni dei giocatori, interpretazioni che di fatto modificano in maniera più o meno marcata il gameplay del gioco.
Tutta la sezione dedicata agli *ostacoli ed ai vantaggi* è molto vaga poiché ad esempio discriminare i quadrettini di bosco dal colore verde più o meno scuro, può portare ad opinioni discordanti fra i giocatori, così come anche definire quali sono i quadretti di montagna e quelli da considerare effettivamente strada.
I disegni degli elementi del campo di battaglia infatti non sono delimitati dalla quadratura delle celle ed in molti casi questi sconfinano nelle caselle adiacenti, lasciando il legittimo dubbio se considerarli ad esempio come una casella di montagna o di strada oppure no, scelte che vengono lasciate fare autonomamente ai giocatori una volta che si avviano a giocare.
Ad esempio questo fenomeno lo si riscontra nel famosissimo gioco da tavolo del Monopoly: una volta soddisfatta la regola del dover raccogliere tutti i terreni dello stesso colore per potervi costruire sopra, è usanza di molti giocatori aspettare di trovarsi con la propria pedina su uno dei terreni sul quale si desideri posizionare una "casetta", ma in realtà questa regola non è specificata da nessuna parte del manuale, anzi non vi è nessuna regola che determini il

momento esatto in cui si possa costruire o no, e dunque la costruzione può avvenire in qualsiasi momento, anche durante il turno di un avversario. Per quanto non venga stravolto il senso generale del gioco, ciò comporta invece un approccio al gameplay totalmente differente rispetto le regole originali.

Se nella versione cartacea può comunque sussistere un accordo fra i giocatori ad adottare, modificare o annullare una regola, ciò non è assolutamente ammissibile per la versione digitale, per la quale invece è stato necessario revisionare tutte le regole e rimuovere le eventuali fonti d'ambiguità.

La prima è più importante revisione è quella fatta alla successione e definizione delle azioni possibili all'interno di un turno;

Nel regolamento originale viene specificato che i pezzi di fanteria possono sparare o muoversi ma non fare entrambe le cose e le unità coinvolte in un combattimento ravvicinato devono forzatamente risolvere il combattimento altrimenti non possono muovere, ma non viene detto o specificato altro.

Le principali fonti di dubbio del gioco cartaceo sono:

- Un'unità che muove solo di alcuni quadretti rispetto al totale a sua disposizione, può successivamente, nello stesso turno, muoversi dei restanti quadretti rimastigli a disposizione oppure nella regola è sottointeso che i restanti punti movimento a disposizione sono annullati?
- Se un pezzo si trova coinvolto in un combattimento ravvicinato, lo vince, e non ha ulteriori pezzi nemici confinanti, può muoversi per le caselle da lui raggiungibili?
- Se può muoversi, può attaccare ancora una volta?
- Dato che i combattimenti più complessi vengono suddivisi in combattimenti che coinvolgono meno unità, può accadere che unità nemiche, risultanti vincitrici dei loro rispettivi combattimenti, si trovino ad essere ora confinanti fra loro e dunque ad essere nuovamente coinvolte in un combattimento ravvicinato, devono risolverlo subito o nel turno successivo?

- Nel caso si tratti di un'unità appiedata, non essendosi ancora spostata poiché coinvolta in un combattimento ravvicinato dal turno precedente, può sparare o muoversi?
- Un pezzo di artiglieria può ruotare la sua direzione e sparare nello stesso turno, può farlo indipendentemente dall'ordine in cui esegue le due azioni oppure una volta sparato non può ruotare?
- Se due fanti decidono di far fuoco su un pezzo, nel caso in cui il primo elimini l'obiettivo, il secondo pezzo potrà muoversi o far fuoco su un altro obiettivo?

Ognuna di queste domande ha una risposta che influenzerebbe il gameplay in maniera più o meno forte, basti pensare ad un pezzo di cavalleria che dopo un combattimento, potendosi ancora muovere può andare ad aiutare i suoi alleati in qualche altro combattimento ravvicinato ancora non risolto.

Nella fase di analisi del passaggio alla versione digitale per tablet sono stati analizzati i possibili effetti che le soluzioni proposte avrebbero avuto sul gameplay.

L'obbiettivo principale di questa fase iniziale è stata la risoluzione delle incongruenze nel regolamento senza però stravolgere il gameplay originale della versione cartacea.

Le azioni a disposizione del giocatore per ogni turno sono quattro:

1. **Movimento delle unità**

   Questo comando può essere chiamato più volte durante lo stesso turno.

   Tutte le unità che non siano coinvolte in un combattimento ravvicinato (alcune potrebbero essere contigue ad una o più unità nemiche a causa di un combattimento complesso del turno precedente), che non abbiano già mosso e che non abbiano fatto fuoco, possono essere selezionate e spostate in una qualsiasi casella da loro raggiungibile.

   Una volta che il movimento viene finalizzato, l'unità non avrà più modo di muoversi fino al turno successivo, anche se non avrà usufruito di tutti i suoi punti spostamento.

   Alternativamente alla finalizzazione del movimento è possibile resettare gli spostamenti delle unità mosse.

**2. Risoluzione dei combattimenti ravvicinati**

Questo comando può essere chiamato più volte durante lo stesso turno.

Ogni qual volta che verrà selezionato verranno evidenziate tutte le unità coinvolte in un combattimento, semplice o complesso, che devono scegliere un obiettivo con cui combattere.

Nei casi di scelte forzate, come ad esempio una sola unità contro soltanto un'altra unità nemica, la selezione, essendo superflua, sarà fatta implicitamente.

Ogni pezzo può scegliere un solo avversario e dovranno essere soddisfatte le regole enunciate nel precedente paragrafo.

Se tali regole non sono soddisfatte, o il giocatore volesse riassegnare le unità in un differente modo, potrà annullare tutto e ripetere la selezione.

**3. Fuoco con pezzi di artiglieria**

Questo comando può essere utilizzato una sola volta per turno.

Il giocatore sarà chiamato a scegliere per ogni pezzo di artiglieria abile a sparare l'area di fuoco che si desidera tentare di colpire, oppure può passare al pezzo successivo senza far fuoco.

I pezzi che non fanno fuoco possono essere successivamente mossi.

Terminati i pezzi di artiglieria il comando sarà disattivato fino al turno successivo.

**4. Fuoco con la fanteria**

Questo comando evidenzia tutte le unità di fanteria che possono colpire un bersaglio valido. Il pezzo preso di mira potrà essere scelto da una o più unità, ma ognuna di esse potrà scegliere un solo bersaglio per turno. Il giocatore potrà cambiare gli obiettivi di ogni unità finché non finalizza il comando di fuoco. Una volta finalizzato il comando, tutte le unità che hanno scelto un bersaglio, sia che hanno colpito il loro obiettivo, o no, non potranno fare ulteriori azioni fino al turno successivo e non sarà possibile far fuoco con altri fanti che non siano stati precedentemente scelti, i quali avranno comunque ancora la possibilità di muoversi.

Alternativamente, se il giocatore volesse far fuoco in un secondo momento, potrà resettare tutte le selezioni già fatte.

Quando il giocatore avrà terminato le sue scelte, potrà passare il turno all'altro giocatore.

L'introduzione di questi comandi, se da una parte sembra forzare e ridurre la libertà del giocatore, dall'altra parte rimuove tutte quelle situazioni vaghe di cui si è precedentemente discusso, riduce la complessità delle azioni e soprattutto riduce la durata media di un turno, rendendo il gioco più fluido senza che il giocatore inattivo passi troppo tempo ad aspettare.

Ulteriori modifiche introdotte sono state la standardizzazione della mappa, definendo effettivamente quali sono le strade, i corsi d'acqua, le fortificazioni, gli ostacoli ed i rilievi montuosi.

# Capitolo 4

## Implementazione

Il primo paragrafo di questo capitolo è dedicato alla presentazione degli strumenti hardware e software utilizzati per lo sviluppo della versione digitale. Nel secondo paragrafo è illustrata la prima implementazione basata sull'architettura Model View Controller (MVC), ponendo particolare attenzione sui principali oggetti che compongono il gioco e nel paragrafo finale verranno mostrate le tecniche ed i framework utilizzati per l'ottimizzazione delle prestazioni.

## 4.1 Presentazione degli strumenti utilizzati

Si è scelto come target, data la complessità del titolo, di sviluppare la prima versione per i sistemi iOS,nello specifico per iPad (versione 1) data la soddisfacente potenza di calcolo e la necessaria ampiezza dello schermo richiesta dal gioco stesso.

Per lo sviluppo del codice del gioco è stato scelto il game engine *Unity3D*[11] utilizzando *C#,* linguaggio di programmazione orientato agli oggetti sviluppato da Microsoft, che, come si vedrà successivamente, nell'ambito di Unity3D viene utilizzato come linguaggio di scripting.

Questo game engine offre un ambiente di sviluppo integrato con gerarchia degli oggetti, editor visuale, un dettagliato visualizzatore delle proprietà degli elementi in scena ed un'anteprima dal vivo del gioco, nonché la possibilità di poter sviluppare, utilizzando lo stesso codice sorgente, per molte altre piattaforme (tra cui importanti piattaforme mobile come *Android*, *Windows phone 8* e *BlackBerry 10* per eventuali future versioni).

Unity3D inoltre offre la possibilità di importare modelli dai principali tool di sviluppo 3D come *Maya, Blender, 3D Studio Max, Photoshop*.

Come tool per la modellazione degli oggetti 3D è stato utilizzato *Blender v2.68*[12] e *Photoshop CC*[13] per la grafica 2D.

Sono stati utilizzati i framework NGUI[14], TexturePacker[15] ed Orthello[16] per realizzare rispettivamente la parte di UI e del piano bidimensionale di gioco, nonché ottimizzare le prestazioni globali del gioco. Per la parte di gestione delle interazioni con la superficie del tablet ( o gesture ) è stato utilizzato il framework TouchScript[17]. Di questi framework si parlerà più specificatamente nel paragrafo 4.3.

## 4.2 MVC e prima implementazione

La struttura gerarchica di Unity è organizzata nel seguente modo: L'oggetto da cui derivano tutti gli elementi di gioco è detto GameObject ed a sua volta fornisce una gamma di classi (a loro volta modificabili) specializzate nella gestione di alcuni aspetti degli oggetti stessi quali posizione nello spazio, dimensione, rotazione, resa grafica, componenti per l'interazione fisica. A questi elementi possono essere affiancati ulteriori GameObject così da riproporre un ulteriore livello gerarchico dedicato ad un solo oggetto, denominato figlio, ed istruzioni di codice racchiuso nei file di scripting scritti in C# (da quì il perchè anche se orientato agli oggetti, C# viene usato come linguaggio di scripting). Ogni script può accedere alle proprietà di un gameObject e modificarli in accordo ai risultati che il programmatore vuole ottenere.

Il pattern architetturale Model-View-Controller prevede l'organizzazione del software in tre macro aeree con il fine di separare la parte di presentazione visiva, cioè la parte grafica e di interazione con l'utente ( pulsanti, campi testuali, interazioni, ecc, ecc) dalla parte logica e dalla parte dati. Lo scopo principale del pattern è quello di poter così mantenere le parti distinte fra loro in maniera da poter eventualmente cambiare solo alcuni elementi senza dover manipolare tutto il progetto.

Il principale motivo per cui si è scelto di adottare questo pattern, anche se in un contesto di scripting, è legato infatti allo sviluppo contemporaneo di due grafiche, una 2D ed una 3D, che potessero lavorare contemporaneamente utilizzando lo stesso codice.

La scelta di voler separare le due grafiche è stata fatta per mettere in risalto due aspetti fondamentali del gioco: da una parte, essendo un wargame strategico da utilizzare su tablet, è stato necessario fornire al giocatore una visuale strategica senza però voler rinunciare all'originale effetto del gioco cartaceo in cui le unità, disegnate su carta svettavano in rilievo dal campo di gioco.

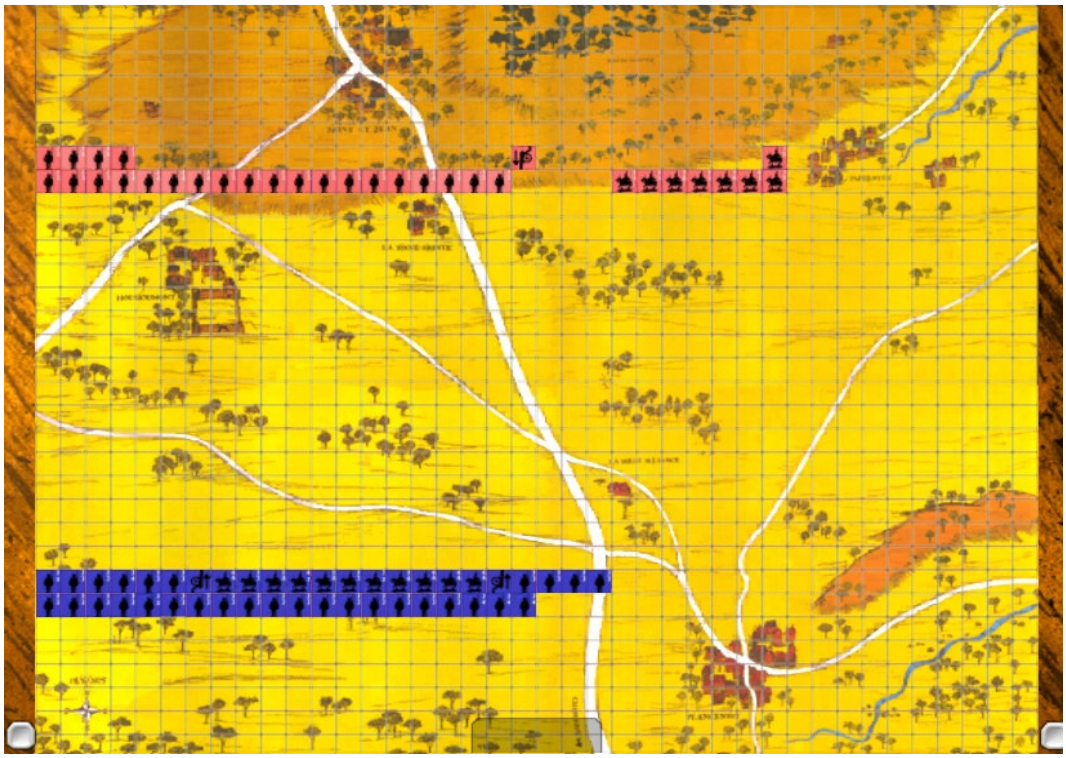

*Figura 4.1: Visuale strategica.*

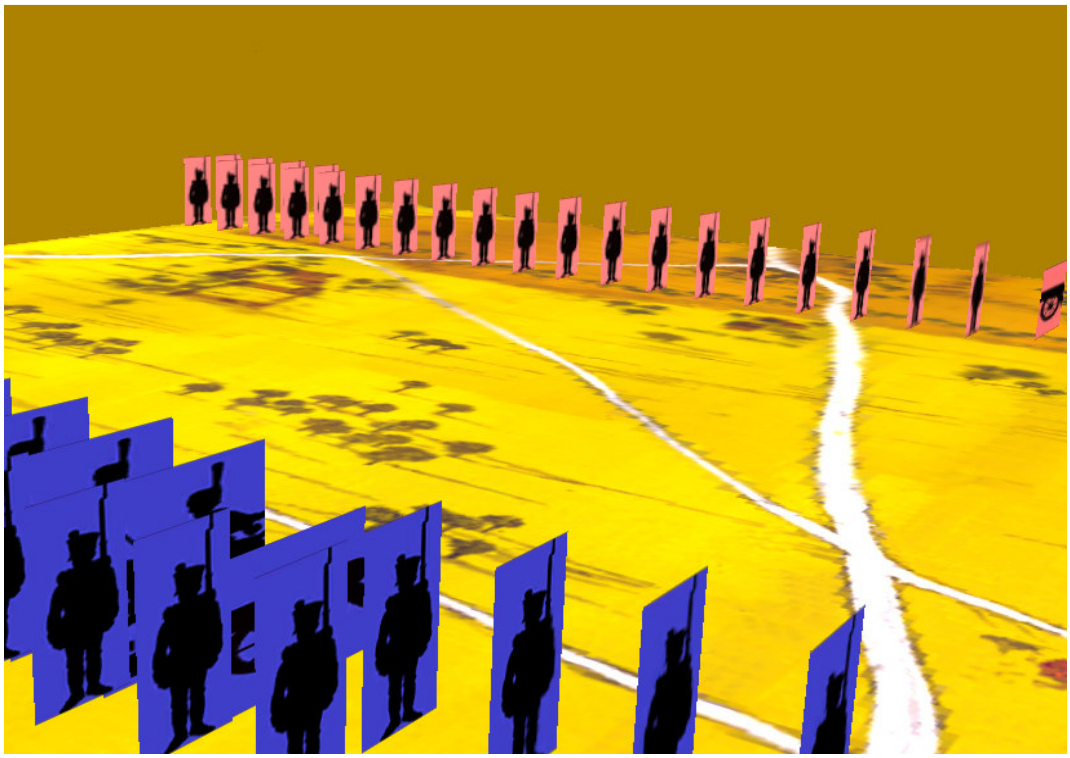

*Figura 4.2: Visuale 3D.*

Nel momento di stesura della tesi, non essendo ancora pronto il materiale originale di Guido Crepax digitalizzato, i disegni delle unità sono sostituiti con

materiale temporaneo creato amatorialmente così da poter fornire una bozza visiva del gioco e poter sostituire in futuro le texture.

### 4.2.1 Model Side

All'interno della macroaerea dei dati e dei modelli vi sono le classi che modellizzano gli oggetti principali del gioco. Le principali sono:

- Unit: Un oggetto Unit rappresenta una qualsiasi unità del gioco. I principali attributi di questa classe sono di conseguenza i valori che contraddistinguono ogni unità: valore di movimento, valore di combattimento, tipo di unità, schieramento militare, nome dell'unità, Tile su cui si trova, più altre variabili che servono per tracciare lo stato dell'unità durante la partita, come ad esempio se l'unità ha sparato, si è mossa, qual è la futura tile su cui muoverà o se ha attaccato. Da questa classe estende un'altra classe, Cannon, sempre strutturata all'interno della parte dei modelli, che rappresenta invece i pezzi di artiglieria. Questa classe aggiunge alle variabili e funzionalità della classe Unit quelle caratteristiche che appartengono ai soli pezzi di artiglieria, ovvero la direzione del pezzo e le indicazioni necessarie a calcolare l'area di fuoco, cioè se spara usando un dado o due, più eventualmente se il colpo è diretto verso destra/sinistra con il lancio di uno/due dadi.

- Tile: Un oggetto Tile rappresenta invece le informazioni necessarie per modellizzare le caselle di gioco: le coordinate, X e Y, che rappresentano riga e colonna nel campo di combattimento, l'Unit che vi risiede, le caratteristiche del terreno.

- WarGame_Enumerations: Questo oggetto contiene tutte le enumeration usate dal gioco:
    - Tipi di unità: Cavalry, Infantry, Artillery.
    - Armata: Francia, Inghilterra.
    - Direzione Rotazione: Nord, Sud, Est, Ovest.
    - Bonus/malus spostamento su una tile: Road,River,Rilievi.

- Bonus/malus combattimenti: Casa, Forte, Normale.
- Altezze: Altezza 1, Altezza 2.

Unit, Cannon e Tile sono gli elementi che da soli rappresentano interamente tutto il gioco. Questi elementi sono soltanto la rappresentazione virtuale del gioco, mentre gli oggetti mostrati graficamente sono altri.

WargameEnumeration è invece in questa macroaerea poichè fornisce dei valori costanti utilizzati da molti script, ricoprendo quindi un ruolo di dati da consultare. Inoltre, localizzando in un unico oggetto tutte le enumeration, si vuole fornire la possibilità di creare un file unico di configurazioni da poter estendere e modificare esternamente il gioco così da poter in futuro riutilizzare la stessa base di codice per gli altri wargame sviluppati da Crepax sempre sulla scia delle campagne napoleoniche.

Il lato model del pattern MVC in questa maniera espone tutti i metodi necessari ad accedere e modificare i dati. Gli oggetti all'interno del lato model seguono i principi della programmazione orientata agli oggetti, rispettando quindi anche il principio dell'encapsulation.

### 4.2.2 Controller Side

Il lato Controller contiene tutta la logica applicativa del gioco. Il suo ruolo, come specificato dal pattern MVC è quello di modificare ed utilizzare i dati del lato model in accordo alle azioni svolte dal giocatore nel lato view ed eventualmente aggiornarlo.

Al suo interno si trovano i seguenti script:

- GameLauncher: Questo script è il punto d'ingresso al gioco. Il suo scopo è infatti inizializzare il gioco instanziando gli oggetti che contengono gli script che realizzano la logica applicativa.

```
void Awake(){
      this.mapInformation = (MapInformation)
            this.GetComponent<MapInformation>();
```

```
        //taking gameObjectsManager

        this.gom = (GameObjectsManager)
              this.GetComponent<GameObjectsManager>();

        //taking camera controller
        this.cmc = (CameraMovementController)
        GameObject.FindGameObjectWithTag("CameraContainer").
        GetComponent<CameraMovementController>();

        //taking NGUI controller
        this.nguiContr = (NGUIController)
        this.GetComponent<NGUIController>();

        //taking tile matrix graphic controller
        this.tileGraphic = (TileMatrixGraphic)
        GameObject.FindGameObjectWithTag("TileMatrix").
        GetComponent<TileMatrixGraphic>();
}
```

- MapInformation: Questo script è il cuore di tutto il software, dal momento che contiene tutta la logica applicativa. Ricevendo dal lato view le azioni del giocatore, le trasforma e modifica i dati del lato model in accordo con le regole del gioco. Al suo interno vi sono tutti i metodi utilizzati per il calcolo delle aeree di combattimento, di fuoco del cannone, di selezione delle unità di combattimento e di risoluzione delle battaglie. Il suo scopo e di fungere da punto di raccordo fra gli altri script della sezione controller che si occupano di specifiche aree del gioco. Dato il suo ruolo centrale, mantiene anche tutte le informazioni necessarie allo sviluppo della partita, come punteggi, turni, giocatore attivo, unità in campo, e tutte le variabili necessarie.

  E' di seguito riportato il metodo che inizializza le variabili utilizzate per realizzare la partita:

```
public void setMapInformation(){
       this.map = new Tile[column,row];
       this.rnd = new System.Random();

       for(int i=0; i < this.column ; i++){
              for(int j=0; j< this.row ; j++){
                     map[i,j] = new Tile(i,j);
              }
       }

       //set tiles
       MapSettingsReader msr = new MapSettingsReader();
       msr.setMapTiles(this.map);
```

```
        //createMovementGraph
        this.mg = new MovementGraph(this);

        //setup variables
        this.ur =  new UnitReader(this);
        this.frenchFormation = new Formation(ARMY.FRENCH);
        this.englandFormation = new Formation(ARMY.ENGLAND);
        this.firstPrussianFormation = new Formation(ARMY.FIRST_PRUSSIAN);
        this.secondPrussianFormation = new
        Formation(ARMY.SECOND_PRUSSIAN);

        //Armies
        this.englandArmyUnits = new LinkedList<Unit>();
        this.frenchArmyUnits = new LinkedList<Unit>();
        this.PrussianArmyUnits = new LinkedList<Unit>();

        //prussian variables
        this.firstPrussianArmy = false;
        this.firstPrussianEntering = 7;
        this.firstPrussianEntering += this.rnd.Next(-3,4);
        this.secondPrussianArmy = false;
        this.secondPrussianEntering = 9;
        this.secondPrussianEntering += this.rnd.Next(-4,5);
        this.lastColumnOpen = false;

        //LinkedList of support
        this.tilePrecedentlySelected = null;
        this.unitAbleToMove = new LinkedList<Tile>();
        this.movementZone = new LinkedList<Tile>();
        this.movingAction = false;
        this.unitAbleToFire = new LinkedList<Tile>();
        this.unitDistantTargets = new LinkedList<Tile>();
        this.cannonFireZone = new LinkedList<Tile>();
        this.firingAction = false;
        this.unitAbleToMelee = new LinkedList<Tile>();
        this.defendersShared = new LinkedList<Tile>();
        this.feasableEnemies = new LinkedList<Tile>();
        this.meleeCombats = new LinkedList<SingleCombat>();
        this.meleeCombatsToResolve = new LinkedList<ComplexCombat>();
        this.meleeAction = false;
        this.meleeActionAI = false;

        //Turns variables
        this.turns = 2;
        this.turnLeft = 0;
        this.differenceLimit = 10;
        this.lastTurnToDraw = false;
        //Score variables
        this.frenchScore = 0;
        this.frenchStartPoints = 0;
        this.englandScore = 0;
        this.englandStartPoints = 0;
}
```

- GameObjectsManager: Le unità in 3D sono dei gameObject gestiti da questo script, il quale espone tutti i metodi necessari per la loro creazione, distruzione e trasformazione. Ciò permette di mantenere separata la logica di gestione degli oggetti di gioco da quella che è invece la logica che gestisce le meccaniche di gioco.

```
public void destroyAllGoUnits(){...}
public void createGoUnit(...){...}
public void setGoUnitNextPosition(...){...}
public void undoGoUnitNextPosition(...){...}
public void finalizeGoUnitMovement(...){...}
public void destroyGoUnitEliminated(...){...}
public void destroyGoUnitEliminated(...){...}
```

- CameraMovementController: Come le unità in 3D, anche la Camera è un GameObject ed è in particolare l'oggetto che realizza la visione del mondo virtuale di gioco. Essendo anche questo componente logicamente separato dalle meccaniche di gioco, i metodi che modificano e controllano la posizione della camera sono gestiti all'interno di questo script.

```
public void resetCamera(..){...}
public void setFrenchSetup(...){...}
public void setEnglandSetup(...){...}
public void setFirstPrussianSetup(){...}
public void setSecondPrussianSetup(){...}
```

- NGUIController: La parte relativa alla UI è gestita, come meglio illustrato nel prossimo paragrafo, dal framework NGUI. Questo script funge da giunzione tra il lato controller che richiama, tramite MapInformation, i metodi per gestire i menu grafici e la UI implementati nel lato View.

- AIManager: Il compito di questo script è quello di fornire i metodi e le variabili necessarie alla realizzazione della IA.

```
public void AITurn(...){...}
public void endAITurn(){...}
```

```
private void setupPrussianArmy(...){...}

private void selectInfantryAbleToFire(){...}

private void selectMovableUnits(){...}

private void selectArtilleryAbleToFireMove(){...}

private void fireWithArtillery(...){...}
```

Oltre agli script elencati quì sopra, ne sono presenti altri dedicati alla realizzazione di particolari funzionalità, come la lettura ed il salvataggio delle formazioni in fase di setup o la lettura dei file di configurazione, ed altri script che forniscono invece supporto alle operazioni svolte da MapInformation, utilizzati soprattutto per alleggerirne il codice contenuto.

### 4.2.3 View Side

All'interno della Vista sono presenti gli script che realizzano gli aspetti grafici e di UI insieme agli script che gestiscono le gesture dei giocatori sulla superficie del tablet. I principali script sono:

- TileMatrixGraphic: Questo script gestisce l'oggetto grafico che realizza, tramite il framework Orthello, il piano bidimensionale. Viene modificato in base alle chiamate fatte dallo script MapInformation.

```
public void setTransparent(...){...}

public void highlightCannonTargetArea(...){...}

public void setExplosion(...){...}

public void setUnit(...){...}

public void highlightUnits(...){...}

public void highlightUnits(...){...}

public void setTileAsPossibleTarget(...){...}

public void setTileAsPossibleTarget(...){...}

public void setTileAsTarget(...){...}

public void highlightMovementZones(...){...}
```

```
public void SetupOnTile(...){...}
```

- TapGestureManager, CameraScaleMovement, CameraPanMovement: Questi script, realizzati con il framework TouchScript, raccolgono i gesti fatti dal giocatore sulla superficie del tablet e realizzano le interazioni con il gioco, come il tocco per selezionare le unità, lo zoom della mappa strategica nonché lo scorrimento della visuale su di essa.

Gli altri script presenti nel lato Vista sono relativi alla costruzione dei menu di gioco, realizzati con NGUI, e gestiti tramite lo script NGUIController.

## 4.3 Framework ed ottimizzazioni

Una volta creato il primo prototipo si è passati ad una fase di testing su tablet per analizzare, oltre il corretto funzionamento del software e la corretta implementazione delle regole di gioco, il gameplay ottenuto dalle modifiche apportate alle regole originali del gioco cartaceo ed ad analizzare le sezioni di software da ottimizzare per migliorarne le prestazioni.
Infatti, essendo il gioco destinato ad un target mobile, è stato molto importante mantenere la richiesta di risorse computazionali al minor livello possibile onde evitare rallentamenti durante l'esperienza di gioco.
Le seguenti parti sono state individuate come sensibili in termini di prestazioni richieste:

- Complessità degli algoritmi per costruire le zone di spostamento o la ricerca delle possibili unità nemiche da colpire a distanza con le unità di fanteria.
- L'interfaccia utente mal gestita con gli elementi nativi di Unity3d.
- Le unità, ognuna con la propria coppia di disegni (uno per lato) della versione 3D ed in generale il numero di GameObject mostrati.

Ogni unità che può muoversi, nel momento in cui viene selezionata, evidenzierà le tile che può raggiungere e che non siano occupate da unità o ostacoli,

tenendo conto dei bonus/malus guadagnati nell'attraversare particolari tile come strade o fiumi o montagne.

Sono di seguito presentati due esempi di zone di spostamento:

*Figura 4.3: Esempi di zone di spostamento.*

Le zone in rosso chiaro sono le caselle raggiungibili con i punti base di movimento mentre le zone rosse sono raggiunte grazie al bonus guadagnati nell'eventualità si percorrano le strade (ogni 2 caselle di strade percorse se ne guadagna una casella di spostamento).
Con 2 punti di movimento è necessario controllare 5x5 caselle mentre con 7 punti di movimento, il massimo, sono da controllare 15x15 caselle.
Dalla casella dell'unità vengono controllate tutte quelle adiacenti ed aggiunte se valide per lo spostamento. Per ogni casella aggiunta, se sono ancora disponibili punti spostamento, viene ripetuto il procedimento. La visita delle caselle procede dunque per profondità fino ad esaurimento dei punti.
Ogni casella valida viene aggiunta ad una lista e memorizza il valore di punti movimento con i quali la si è raggiunta, se nelle successive iterazioni dell'algoritmo si ritroverà una casella di questa lista, si continuerà l'algoritmo solo se i punti movimento a disposizione sono maggiori di quelli memorizzati nella casella, cioè dunque se l'unità ha modo di raggiungere quella casella più velocemente.

Le unità di fanteria che possono fare fuoco devono invece calcolare quali unità sono effettivamente raggiungibili e quali coperte da ostacoli.
Unity3D offre meccanismi per calcolare le intersezioni fra gli oggetti, ma da un punto di vista computazionale sono molto onerosi, pertanto si è deciso di costruire un sistema simile basato soltanto sui necessari calcoli matematici. Tutti gli oggetti che fungono da ostacoli e le unità amiche e nemiche vengono aggiunte, dalle più vicine alle più lontane ad una lista. Ogni nuovo elemento viene confrontato con quelli già presenti per vedere se questi possono essere di ostacolo al nuovo elemento aggiunto.
Dalla retta che congiunge l'unità che fa fuoco al possibile bersaglio, viene calcolata la distanza del possibile ostacolo. Se la distanza ottenuta è sotto un valore limite, allora il bersaglio non è raggiungibile.
Questi due algoritmi hanno permesso notevolmente di abbassare il numero di risorse richieste, così da aumentare le prestazioni e la resa anche in ambiente mobile.
Per quanto riguarda invece la UI, la gestione nativa di Unity si è rivelata essere uno dei punti più deboli di tutto il game engine.
Anche un semplice menu composto da pochi pulsanti e da pochi label richiede infatti un numero di risorse spropositato rispetto alla funzionalità che svolge, alle volte anche dell'ordine di decine di drawcall.
Ogni drawcall rappresenta in termini computazionali un grosso carico di lavoro per la cpu, dovendo calcolare tutta la fisica ed i processi di renderizzazione necessari a disegnare un oggetto. Il numero di drawcall aumenta ogni qual volta viene introdotto un nuovo oggetto con un nuovo material (componente responsabile di definire texture e shader di un oggetto), dunque N oggetti con lo stesso material e la stessa forma corrispondono ad una drawcall mentre M oggetti tutti di forma diversa o con material diversi corrispondono a M drawcall.
Purtroppo Unity considera ogni panello della UI ed ogni lettera delle label una singola drawcall, e mentre le comuni risorse di un PC permettono di avere un numero di drawcall relativamente vasto, dell'ordine delle centinaia a seconda delle configurazioni hardware, per un ambiente mobile il limite di drawcall ammesse e tra le 10 e le 15 all'incirca. Oltre questo limite le performance

calano così drasticamente da causare vistosi rallentamenti durante l'esperienza di gioco. Vi sono altri fattori che determinano il numero di drawcall, ma essendo stati evitati o creati da particolari funzionalità che non intervengono nel lavoro di tesi, i principali fattori rimangono quelli elencati precedentemente.

Il framework NGUI ed Orthello si sono rivelati fondamentali nel processo di ottimizzazione delle performance dal momento che riducono, il primo la parte di UI, il secondo il piano di gioco bidimensionale, a 2 sole drawcall.

Entrambi i framework riescono a compiere questa sostanziale riduzione del numero di drawcall grazie all'uso delle *Sprite*.

Una sprite è semplicemente un'unica texture che racchiude all'interno molte altre texture. In questa maniera viene creato un unico material che, tramite un sistema di trasformazioni delle coordinate bidimensionali del material a quelle tridimensionali dei GameObject, permette di utilizzare soltanto alcune porzioni del material. In questa maniera viene caricata nella cpu un unico oggetto che verrà riutilizzato da più elementi senza violare  le regole sopracitate.

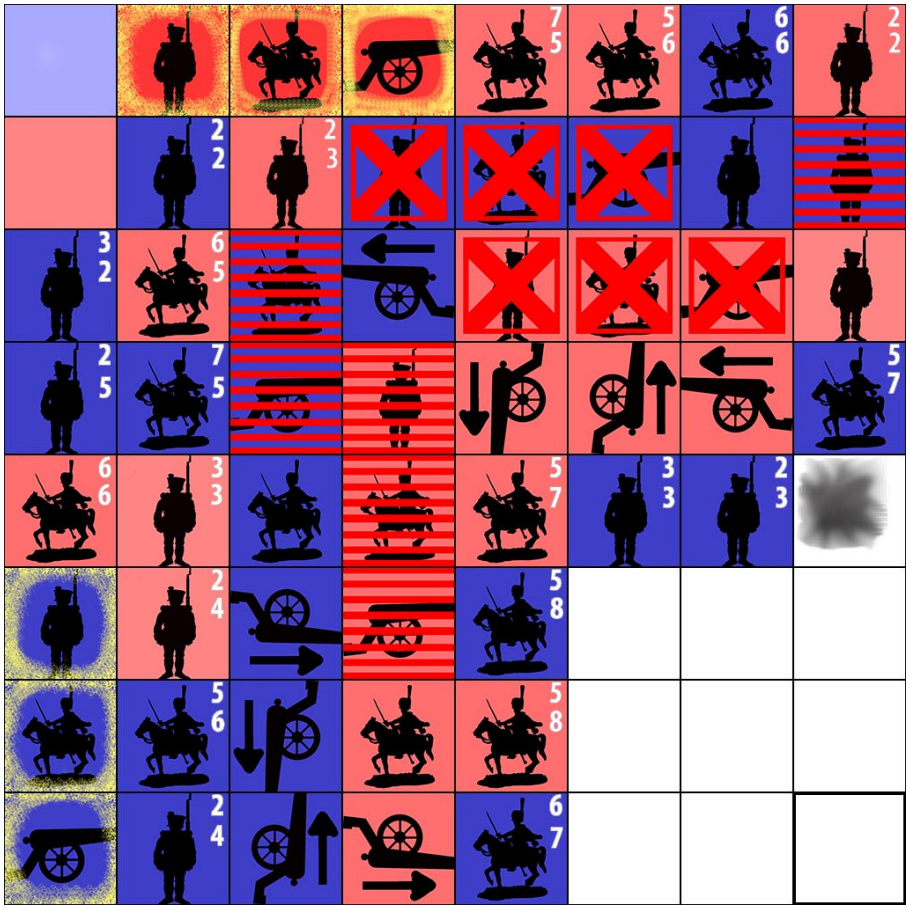

*Figura 4.4: Sprite delle tile della visuale strategica.*

Per quanto riguarda invece gli oggetti in 3D, in particolar modo per quanto riguarda le unità (anche il piano di gioco è un GameObject ma influisce al totale sempre con una sola drawcall) viene utilizzato il framework TexturePacker che agisce fondamentalmente nello stesso modo degli altri framework ma non può scendere al di sotto delle tre drawcall, essendo le unità, cavalleria, fanteria ed artiglieria, di tre dimensioni diverse.
Il numero finale di drawcall ottenute si aggira da un minimo di 4 della visione bidimensionale, a 7 quando viene aggiunta la visuale in 3D, riuscendo così a mantenere il gioco non solo al di sotto dei limiti imposti, ma anche a mantenere un margine per eventuali miglioramenti grafici da introdurre in futuro.
Bisogna infine citare il framework TouchScript usato per gestire le gesture su tablet. Anche se non introduce  dei miglioramenti in termini di prestazioni, fornisce una serie di meccanismi già costruiti ( e comunque modificabili liberamente essendo il codice sorgente OpenSource) per catturare i tocchi sullo schermo, permettendo di velocizzare lo sviluppo di questa parte di progetto che altrimenti avrebbe richiesto tempistiche implementative e di testing decisamente più lunghe ed onerose.

## 4.4 Svolgimento di una partita

Una partita fra due giocatori umani si svolge nel seguente modo.
La prima fase è quella di setup in cui i giocatori schierano le truppe che formano l'esercito che hanno scelto.
Le zone ammissibili sono evidenziate tramite il colore di fazione in accordo alle regole del gioco:

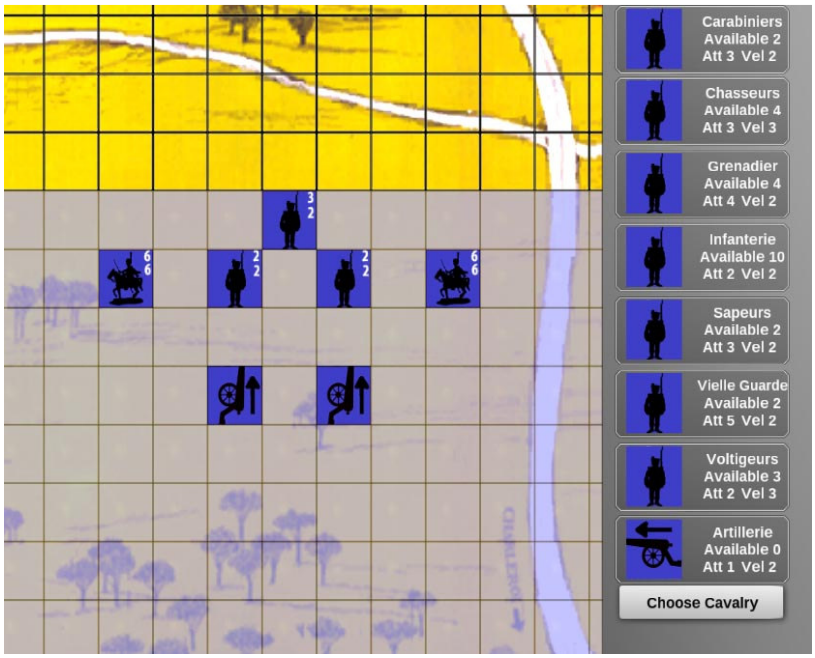


*Figura 4.5: Fase di setup.*

Il giocatore può salvare fino a 5 setup per formazione per poter velocemente caricare nelle future partite. Schiera per primo sempre il giocatore che ha scelto lo schieramento francese. La visuale può scorrere soltanto sulla zona di setup, così che il giocatore che ha scelto lo schieramento inglese non possa spiare il setup francese.

Una volta che entrambi i giocatori hanno compiuto le operazioni di schieramento, verrà scelto a caso chi comincerà a muovere.

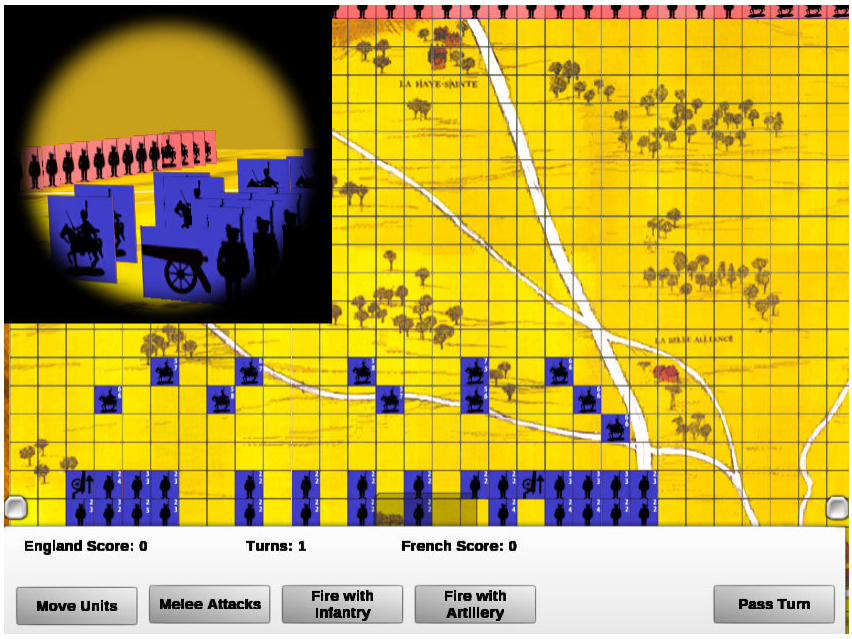

*Figura 4.6: Fase di gioco con entrambe le visuali.*

L'interfaccia di gioco è composta nella parte superiore dal punteggio dei giocatori e dal numero di turni trascorsi dall'inizio.
La parte inferiore è composta da quattro bottoni che corrispondono a 4 azioni:

- **Move Units**: Evidenzia con un bordo giallo tutte le unità che possono muovere. Per selezionare un'unità è sufficiente un doppio tap. Selezionata un'unità verranno evidenziate le caselle valide su cui può muovere e con un doppio tap su una di esse l'unità si sposterà.
  Quando il giocatore avrò terminato gli spostamenti potrà scegliere se finalizzare i movimenti oppure risistemare tutte le unità nella loro posizione iniziale.
- **Melee Attacks**: Tutte le unità che possono attaccare un nemico vicino vengono evidenziate di giallo. Con un doppio tap su un'unità evidenziata, le unità che questa può attaccare vengono evidenziate di rosso. Nell'esempio mostrato in figura4.7 si è selezionato il cavaliere inglese sull'estrema sinistra, che può attaccare solo il cavaliere francese adiacente. Una volta che tutte le unità (sia attaccanti che difensori, quando sono in numero maggiore degli aggressori) avranno selezionato il loro obbiettivo, se non vi saranno altri combattimenti da settare, l'attaccante potrà finalizzare l'attacco, se le regole dei combattimenti sono soddisfatte.

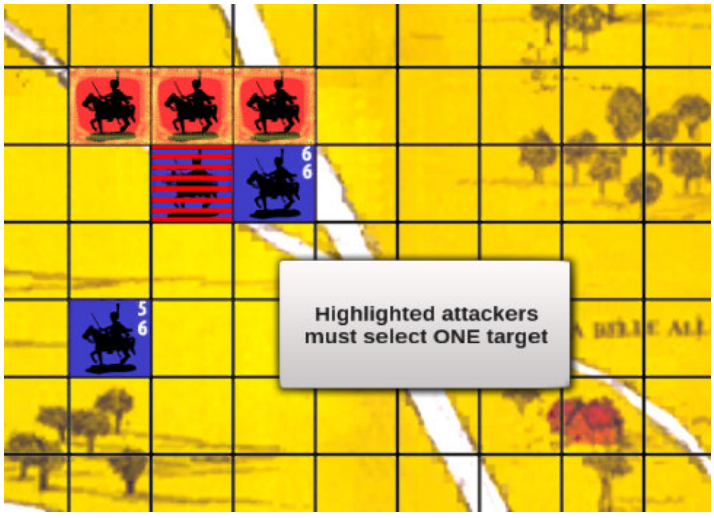


*Figura 4.7: Esempio combattimento.*

In caso contrario il giocatore sarà chiamato ad effettuare nuovamente la selezione.

Eventualmente può sempre decidere di rimandare l'attacco e tornare al menu precedente.

- **Fire With Infantry:** Anche in questo caso verranno evidenziate soltanto le unità che possono fare fuoco su un nemico. Tramite il doppio tap su un'unità evidenziata, verranno mostrate i bersagli validi . Con un doppio tocco su una di queste unità si selezionerà il bersaglio che si desidera colpire.

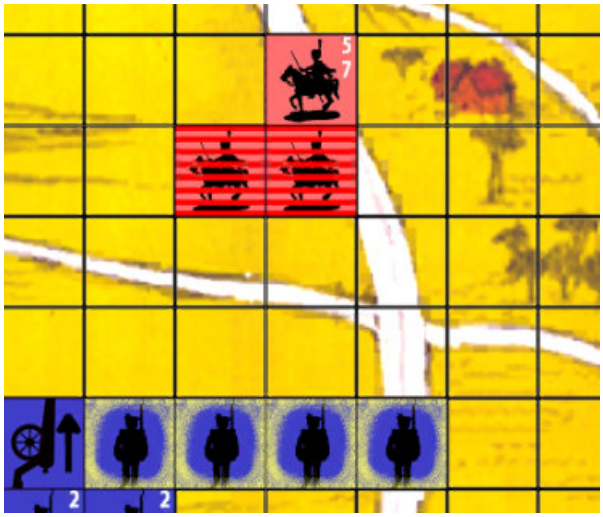


*Figura 4.8: Esempio fuoco a distanza.*

Questa azione può essere fatta una sola volta per turno. I pezzi di fanteria che non hanno sparato, dovranno aspettare il turno successivo. Il giocatore può naturalmente annullare l'ordine di fuoco per eseguirlo in un momento successivo.

- **Fire with Artillery:** La selezione di questa azione abiliterà il menu di fuoco con i pezzi di artiglieria, tramite il quale si potrà scegliere la zona in cui fare fuoco.

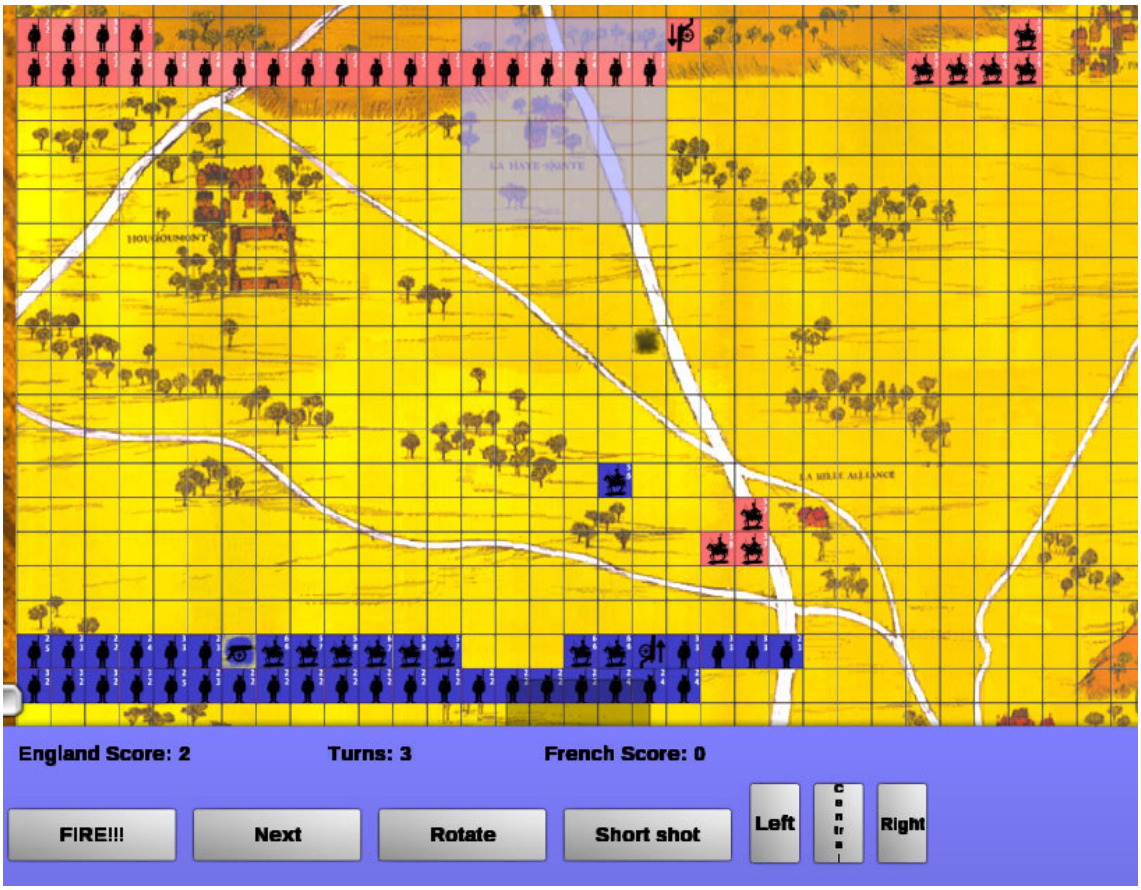


*Figura 4.9: Esempio di fuoco con l'artiglieria.*

Se un giocatore non desidera sparare con un particolare pezzo di artiglieria potrà saltare alla successiva unità tramite il tasto next, altrimenti fare fuoco con il tasto fire. Questa azione non può essere annullata, anche nel caso in cui si passi al pezzo successivo senza far fuoco, e può essere richiamata una volta soltanto per turno.

Quando il giocatore avrà terminato il suo turno, premendo il tasto in fondo a destra potrà passare il turno all'avversario. Il bottone in alto a sinistra serve per abilitare la visuale in 3D.

# Capitolo 5

## Conclusioni ed eventuali sviluppi

L'obbiettivo di questo lavoro di tesi è stato quello di creare tramite Unity3D un titolo per tablet partendo dal board game di Guido Crepax. Tutta la struttura del software è stata sviluppata da zero e sono state completate tutte le più importanti funzionalità, arrivando al deployment su tablet iPad di prima generazione una versione completamente giocabile ed altamente performante in termini di prestazioni. L'adattamento di un board game già esistente senza voler però rinunciare alle sue componenti caratteristiche si è rivelato essere sia un punto di riferimento sia un limite poichè le soluzioni adottate dovevano mantenere il gameplay originale. Questo fattore, unito alla necessità di approfondire gli aspetti tecnici legati al game engine ed ai framework utilizzati, hanno comportato in alcuni casi fare delle implementazioni da dover successivamente scartare perchè poco performanti oppure rivelatesi dal punto di gameplay inadeguate alla fruizione del titolo, aumentando i tempi di sviluppo. Queste soluzioni anche se scartate hanno però contribuito a comprendere le problematiche da affrontare, fino a quel momento impossibili da valutare a priori. Il titolo è dunque attualmente in uno stato molto avanzato di sviluppo e le parti rimaste parzialmente incomplete sono quelle relative al lato grafico ed all'implementazione di un Intelligenza Artificiale più competitiva di quella fin ora sviluppata. Essendo la logica di gioco già implementata, si trattano di migliorie facilmente implementabili in futuro. Soprattutto dal punto di vista dell'Intelligenza Artificiale il titolo sviluppato può fungere da base per testare le tecniche classiche usate per i videogiochi sulle piattaforme di gioco tipiche (PC e console) o implementarne di nuove su ambienti mobile che ad oggi risultano essere ancora marginalmente esplorati. Una volta completati questi step, un ulteriore lavoro che può essere fatto è l'inserimento di meccanismi attui ad accogliere, tramite semplici file di configurazioni in xml e le corrispondenti immagini delle unità, gli altri lavori di Guido Crepax e poter creare facilmente altri titoli da giocare.

# Bibliografia

[1] Russell S., Norvig P., *Artificial Intelligence: A Modern Approach (3rd Edition),* Prentice Hall, 2009.

[5] Rabin S., *AI Game Programming Wisdom,* Charles River Media, 2002.

[6] Millington I, *Artificial Intelligence for Games,* Morgan Kaufmann, 2009, pp 652.

[9] Sandokan, n°3 dell'anno III, Aprile 1999.

Peterson J., *Playing at the World: A History of Simulating Wars, People, and Fantastic Adventure from Chess to Role-Playing Games*, Unreason Press, 2012.

# Sitografia

[2] Turing A. M., *Computing machinery and intelligence*, in *Mind*, 59, pp. 433-460, 1950. URL consultato il 07-02-2014.

[3] Wikipedia, l'enciclopedia libera. Tennis for two - http://it.wikipedia.org/wiki/Tennis_for_Two- URL consultato l'ultima volta il 22/03/14.

[4] Wikipedia, l'enciclopedia libera. Scacchi - it.wikipedia.org/wiki/Scacchi - URL consultato l'ultima volta il 05/02/14.

[6] http://sander.landofsand.com/publications/Monte-Carlo_Tree_Search_-_A_New_Framework_for_Game_AI.pdf - URL consultato l'ultima volta il 10/04/14.

[7] http://webdocs.cs.ualberta.ca/~jonathan/publications/ai_publications/svsk.pdf - URL consultato l'ultima volta il 10/04/14.

[8]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.3015&rep=rep1&type=pdf - URL consultato l'ultima volta il 10/04/14.

[10] Wikipedia, l'enciclopedia libera. Spartaco Albertarelli - http://it.wikipedia.org/wiki/Spartaco_Albertarelli - URL consultato l'ultima volta il 07/03/14.

[11] www.unity3d.com - URL consultato l'ultima volta il 17/02/14.

[12] http://www.blender.org/ - URL consultato l'ultima volta il 17/02/14.

[13] http://www.adobe.com/ - URL consultato l'ultima volta il 17/02/14.

[14] http://www.tasharen.com/?page_id=140 - URL consultato l'ultima volta il 06/01/2014.

[15] http://www.codeandweb.com/texturepacker - URL consultato l'ultima volta il 12/01/2014.

[16] http://www.wyrmtale.com/orthello - URL consultato l'ultima volta il 12/01/2014.

[17] http://interactivelab.github.io/TouchScript/ - URL consultato l'ultima volta il 15/12/13.

http://faculty.virginia.edu/setear/students/wargames/home.html - Wargame - URL consultato l'ultima volta il 15/02/14.