Università degli Studi di Modena e Reggio Emilia
Facoltà di Ingegneria “Enzo Ferrari” di Modena

---

Corso di Laurea in Ingegneria Informatica

# L'INTELLIGENZA ARTIFICIALE NEI GIOCHI: IL CASO WORLD OF WARCRAFT

Relatore:
Dott. Ing. Laura Po

Corelatore:
Chiar.mo Prof. Sonia Bergamaschi

Candidato:
Davide Setti

---

Anno Accademico 2010/2011

# Indice generale

# Introduzione

Al giorno d'oggi una buona fetta del mercato dell'informatica, in continua espansione, è costituita dai videogiochi, di varia natura, che vogliono offrire all'utente esperienze sempre più avvincenti e realistiche.

Per arrivare a tale risultato si è passati attraverso una fase di evoluzione dei videogiochi, sia come generi che come marchi, da parte delle case produttrici. Questa evoluzione si può osservare essere sostanzialmente divisibile in tre fasi:

- inizialmente, la prima fase ha visto lo sviluppo dei motori grafici e audio dei videogiochi, che permettono di fornire un ambiente sempre più convincente ed accattivante, e che riescono a dare una sensazione piacevole al giocatore sia al primo impatto che durante il gioco;
- la seconda fase ha visto lo sviluppo del così detto "*gameplay*", ovvero dei contenuti in termini di sfide e prove che il giocatore deve affrontare per proseguire nel gioco e di quelle che il giocatore si trova ad affrontare a fine gioco, il così detto e*ndgame-content*, a questa parte è sostanzialmente affidato il successo o meno di un videogioco sul mercato;
- infine, l'ultima fase ha visto l'introduzione di funzionalità dell'intelligenza artificiale (IA o AI, *artificial intelligence*) nei videogiochi.

Questo ultimo campo è essenziale nei videogiochi che offrono soltanto la modalità giocatore singolo, *single player*, poiché il giocatore si troverà a doversi confrontare sempre contro l'AI del gioco, ma è altrettanto importante anche nei videogiochi multi giocatore, *multiplayer*, sia per controllare i fenomeni ambientali, in questo caso più che una vera intelligenza abbiamo un così detto motore fisico/ambientale, sia per controllare e far interagire i personaggi non controllabili dai giocatori, detti in gergo personaggi non giocabili o NPC (*non palyable character*), con i personaggi dei giocatori o i giocatori stessi.

Nel primo capitolo di questa tesi tratteremo l'evoluzione dei videogiochi e delle relative intelligenze artificiali negli ultimi cinquant'anni, ed esamineremo da vicino tre tipologie di videogioco ed i compiti che è chiamata a svolgere l'AI in essi.

Nel secondo capitolo ci focalizzeremo su uno dei videogiochi di maggior successo dell'ultimo decennio: *World of Warcraft* (*WoW*), partendo dalle origini del marchio e della saga di Warcraft, racconteremo la sua storia, dal gioco di strategia in tempo reale, da cui ebbe inizio, fino al successo planetario del gioco di ruolo di massa online, suo ultimo capitolo; tratteremo sia la trama di eventi

che ne costituisce la base narrativa che le sue caratteristiche principali come gioco. Infine tratteremo del successo sbalorditivo avuto da tale videogioco. Abbiamo scelto questo videogioco in particolare poiché rappresenta un valido esempio di successo sia commerciale (con oltre 15 milioni di copie vendute ed un picco di più di 12 milioni di utenti attivi) che culturale.

Nel terzo capitolo, rimanendo nel contesto di *WoW*, siccome si tratta di un gioco proprietario il cui codice sorgente non è di pubblico dominio, per vedere come sono implementate le funzionalità di AI all'interno del gioco tratteremo del fenomeno dell'emulazione dei server ufficiali di *WoW*, in particolare parleremo di come tre progetti di emulatori (*ArcEmu*, *MaNGOS* e *TrinityCore*) gestiscano ed implementino le AI dei vari personaggi che popolano il vasto mondo di *WoW* e ne faremo un breve confronto.

Nel quarto capitolo illustreremo alcuni esempi di script di intelligenza artificiale da noi implementati all'interno di un emulatore basato su *MaNGOS*, e vedremo come l'AI non sia unicamente utilizzata per gestire NPC propriamente detti, ma anche per gestire NPC-oggetti.

# Capitolo 1 Storia dell'Intelligenza Artificiale e dei Videogiochi

## *1.1 Definizione di Intelligenza Artificiale*

Con il termine Intelligenza Artificiale (IA o AI, *Artificial Intelligence*) si intende in genere la capacità di una macchina, spesso identificata con un computer, di svolgere funzioni, ragionamenti e comportamenti tipici della mente umana.

L'espressione “Intelligenza Artificiale” (Artificial Intelligence) fu coniata nel 1956 dal matematico americano John McCarthy, durante uno storico seminario interdisciplinare svoltosi nel New Hampshire. Secondo le parole di Marvin Minsky, uno dei pionieri dell'AI, lo scopo di questa nuova disciplina sarebbe stato quello di “far fare alle macchine delle cose che richiederebbero l'intelligenza se fossero fatte dagli uomini”.[1]

Lo scopo ultimo di questa disciplina è dunque quello di realizzare un computer in grado di emulare la mente umana fino ad arrivare al punto in cui non sia più possibile distinguere una mente “artificiale” da una “naturale”.

E' facile intuire quanto sia grande lo sforzo che viene speso in questa disciplina, poiché la mente umana è essa stessa un sistema estremamente complesso, sviluppatosi in milioni di anni di evoluzione della nostra specie, a partire dai primi ominidi fino ad arrivare all'uomo moderno, adattandosi ad interagire con un ambiente le cui leggi fondamentali sono quelle della fisica e che può presentare un'enorme, se non infinita, gamma di scenari da affrontare.

Il campo dei giochi, e successivamente dai videogiochi, si rivela un ottimo terreno da cui sviluppare un'AI, poiché un gioco costituisce un ambiente relativamente semplice e limitato, in cui quindi sono in gioco pochi parametri da osservare e valutare. Partendo da questa base si sono poi potute AI sempre più complesse per adattarsi ai vari generi di gioco ed ai singoli titoli che presentano varianti sul proprio genere.

Ma non è stata un'influenza monodirezionale, anche i videogiochi hanno tratto vantaggio dagli sviluppi avvenuti nel campo dell'AI in altri ambiti per poter sviluppare a loro volta nuove caratteristiche.

Oltre al mondo dei videogiochi, infatti, esistono svariati campi in cui viene impiegata l'AI per vari scopi. Ad esempio gli istituti finanziari e le aziende sanitarie utilizzano software basati sull'AI per organizzare e gestire parti dell'attività, come l'assegnazione dei turni di lavoro, in campo aeronautico l'AI viene utilizzata a bordo dei veivoli dal pilota automatico ed a terra per assistere i controllori di volo nella sorveglianza del traffico aereo, alcune imprese utilizzano l'AI per realizzare

dei sistemi di assistenza-automatica dei clienti, ecc.

Per cui l'AI, così come la mente umana che vuole emulare, non è rilegata ad un unico ambito, ma possiede un enorme campo d'applicazione a cui è collegato un mercato altrettanto vasto.

## *1.2 Sviluppo dell'AI e dei primi Videogiochi*

Come abbiamo detto in precedenza lo sviluppo dell'intelligenza artificiale (AI) è stato influenzato dallo sviluppo dei videogiochi, che a loro volta hanno ricevuto benefici dai suoi sviluppi avvenuti in altri campi, tanto che si può quasi dire che questi due campi di sviluppo siano andati di pari passo. Vediamo dunque alcuni videogiochi storici e cosa erano in grado di fare le AI in essi implementate.

### 1.2.1 Anni '50

I primi prototipi di videogiochi risalgono alla fine degli anni '40, ma i primi videogiochi fecero la loro comparsa sul mercato negli anni '50.

Nel 1952, nell'Università di Cambridge, A.S. Douglas nell'ambito della sua tesi sull'interazione uomo-macchina sviluppa *OXO*, una versione grafica del gioco del tris, in cui l'avversario del giocatore umano era costituito proprio da un'AI seppur semplice, infatti non era prevista la sfida tra due giocatori umani.

Nel 1958 è la volta di *Tennis For Two*, creato da William Higinbotham per intrattenere i visitatori del Brooklin National Laboratory di New York, come si evince del nome si tratta di una versione elettronica del tennis in cui due giocatori umani devono sfidarsi, non era quindi implementata una vera e propria AI di giocatore, tuttavia questo gioco presentava al suo interno un simulatore di forza di gravità (considerabile come AI del tipo dei motori fisici) ed è il primo videogioco ad avere una grafica “dinamica” con cui il giocatore deve interagire.

### 1.2.2 Anni '60

Negli anni ’60 inizia a delinearsi quello che sarà il concetto di videogioco come mezzo di intrattenimento di massa, tipico dei decenni successivi. In questi anni fanno la loro comparsa vari programmi grafici interattivi sviluppati al MIT (*Massachussets Institute of Technology*), tra cui ad esempio *Mouse in the Maze*, gioco che permetteva al giocatore umano di costruire un labirinto con all'interno vari premi ed ostacoli e rilasciare poi un topo virtuale, guidato dall'AI del gioco, che avrebbe dovuto attraversare il labirinto in cerca dei premi; un altro esempio è *Tic-Tac-Toe*, il

classico gioco del tris in cui l'avversario era sempre l'AI del gioco.

Nel 1961, sempre al MIT, un gruppo di studenti tra cui Steve Russel realizzò *Spacewar!*, uno dei videogiochi più influenti di quegli anni, in cui due giocatori umani dovevano affrontarsi al comando di due navette spaziali, provviste di vari armamenti, attorno ad una stella attorno alla quale potevano orbitare vari corpi celesti e che costituiva il centro di gravità del gioco.

Nel 1966 Ralph Baer realizzò un semplice gioco chiamato *Chase*, a cui seguì un anno più tardi *Bucket Filling Game*, l'innovazione di questi videogiochi fu che essi potevano essere visualizzati su normali schermi televisivi, quindi essi furono l'archetipo delle moderne console.

Nel 1969 il programmatore dell'AT&T Ken Thompson scrisse *Space Travel*, gioco che simulava il sistema solare con i suoi vari corpi celesti ed i loro movimenti, implementando così un discreto motore fisico, la sfida del giocatore era quella di far viaggiare la sua navicella spaziale attraverso il sistema solare, atterrando sui vari pianeti per poi ripartire alla volta di quello successivo. Parte del codice di *Space Travel* andrà poi a costituire una delle basi del sistema operativo UNIX, realizzato dallo stesso Thompson assieme a Dennis Ritchie in quel periodo.

### 1.2.3 Anni '70

Nel 1971 fanno la loro comparsa due versioni arcade, che tra l'altro saranno i precursori di questa tipologia di videogiochi, di *SpaceWar!*: *Galaxy Game*, sviluppato alla Standford University e realizzato in un singolo esemplare, e *Computer Space*, sviluppato da Nolan Bushwell e Ted Dabney e realizzato dalla Nutting Associates, mentre nel primo titolo ritroviamo ancora tutte le caratteristiche del gioco originale nel secondo invece compare la possibilità di giocare contro un avversario controllato da un'AI, infatti il giocatore deve affrontare con la propria navicella ben due dischi volanti controllati dall'AI di gioco, cercando di eseguire più abbattimenti di quanti ne riesca all'AI per continuare a giocare in maniera indefinita, tuttavia non ebbe un grande successo per via dell'elevata difficoltà della sfida.

Nel 1972 Bushwell fondò la Atari, per poter produrre in proprio videogiochi. In quello stesso anno uscì il primo videogioco firmato Atari, *PONG*, e fu subito un grande successo. *PONG*, come suggerisce il nome, era una versione elettronica del gioco del ping-pong in cui il giocatore controllava una barretta, rappresentante la racchetta da ping-pong, e doveva impedire ad una pallina di oltrepassare la propria barretta, facendola rimbalzare su di essa e lanciandola contro l'avversario, che poteva essere un altro giocatore umano o un'AI, la quale agiva come un vero giocatore cercando di posizionarsi dove sarebbe andata la palla ma con il vantaggio di poter effettivamente sapere dove

questa sarebbe andata, cosa che attualmente è definito come *cheating* o imbroglio dell'AI verso il giocatore.

Sempre nel 1972 Gregory Yob scrisse il gioco, in stile nascondino, *Hump the Wumpus*, in cui il giocatore umano si trova a dare la caccia ad una creatura, chiamata Wumpus, all'interno di un reticolo di stanze.

Tra il 1975 ed il '76 fanno la loro comparsa i primi videogiochi di ruolo (GDR o RPG), ispirati dall'allora appena comparso *Dungeons & Dragons*, uno tra i primi videogiochi di questo genere fu *Dungeon*, sviluppato da Don Daglow, che fu pioniere nell'introdurre varie caratteristiche poi rivelatesi importanti anche per altri tipi di videogioco come la line of sight (letteralmente linea visuale, LoS), ossia una simulazione del campo visivo del personaggio utilizzata sia per i personaggi giocati (PC) che per i personaggi e le creature gestite dall'AI di gioco (NPC); altri videogiochi precursori del genere furono *dnd*, uscito nel '75, *Telengard*, uscito nel '76, e *Zork*, uscito nel '77.

Sempre sul filone dei giochi di ruolo e dei dungeon, nel 1978 uscì *Multi-User Dungeon* (MUD I), primo gioco a realizzare un mondo virtuale multi utente a cui si accedeva tramite la rete, tale gioco fu il capostipite del genere MUD, che appunto prende il nome da questo gioco, da cui poi si svilupperà il genere MMORPG (Massively Multiplayer Online Role Playing Game).

Il 1978 fu anche l'anno d'oro dei videogiochi arcade, nel quale uscirono molti titoli destinati a fare la storia dei videogiochi e a diventare dei cult, come ad esempio *Space Invaders* e *Asteroids*, della Atari, seguiti nel '79 da *Galaxian*, della Namco.

### 1.2.4 Anni '80

Con l'avvento degli home computer, antenati dei personal computer e precursori delle console di gioco moderne, si diffuse anche la possibilità per i giocatori di possedere uno strumento relativamente semplice per poter giocare comodamente nella propria casa; agli inizi del decennio i maggiori fornitori di home computer erano la Apple Computer e la Commodore International, che si contendevano sia il mercato degli home computer sia quello della distribuzione dei videogiochi.

In questo decennio inoltre comparvero e iniziarono a definirsi molti dei moderni generi di videogiochi:

- Azione-Avventura-Roleplay, con *Zork* (1980), *Dragon Slayer II: Xandau* (1985), i primi capitoli della serie *The Legend of Zelda* (1986 e 1987);

- Platform, con *Space Panic* (1980), *Donkey Kong* (1980), *Mario Bros* (1983), *Metroid* (1986) e *Prince of Persia* (1989);
- Maze Game, con *Pac Man* (1980), 3*D Monster Maze* (1981) e *Dungeons od Daggorath* (1982);
- Strategia in Tempo Reale, con *Herzog Zwei* (1989);
- Roleplay, con *Akalabeth* (1980), *Ultima* (1981), *Dragon Warrior* (1986), *Phantasy Star* (1987) e *Final Fantasy* (1987);
- Sparatutto in Prima Persona, con *Astron Belt* (1983), *MIDI Maze* (1987) e *Interphase* (1989);
- e molti altri ancora.

I giochi sviluppati negli anni ’80 hanno visto, oltre ad un progressivo miglioramento della grafica e dell'interfaccia di gioco, una costante evoluzione dell'AI di gioco, chiamata a svolgere sempre più compiti e funzioni per simulare un avversario sempre più competitivo e agguerrito. Una nuova caratteristica introdotta nei videogiochi di questi anni è la “difficoltà”, mentre nei primi era presente un solo livello di difficoltà della sfida, ora ma a cui il giocatore può indicare il livello di ostilità desiderato. Questa caratteristica diviene essenziale alla longevità di un videogioco ed alla sua diffusione, permettendo alle case produttrici di non perdere clienti, evitando che i giocatori perdano interesse per il gioco poiché ritenuto “troppo difficile” o “troppo facile”.

### 1.2.5 Anni '90 e 2000

Con l'avvento dei personal computer, sempre più performanti sia sotto l'aspetto della grafica che sotto l'aspetto della potenza di calcolo, tra la fine degli anni '80 e l'inizio degli anni '90 e lo sviluppo delle console, con prestazioni grafiche molto superiori ai personal computer, il declino dei giochi arcade era ormai segnato, tanto che molti videogiochi nati come arcade vennero implementati o riadattati per essere eseguiti sulle moderne piattaforme di gioco.

Tuttavia questa evoluzione delle prestazioni dei calcolatori permise anche di poter implementare sempre più meccanismi all'interno delle AI di gioco, come la possibilità di prendere decisioni e la capacità di gestire delle risorse, ma anche imperfezioni tipicamente umane come la possibilità di commettere errori durante l'esecuzione di un compito.

In questo periodo fanno la loro comparsa centinaia e centinaia di titoli divenuti famosi e non nei

vari generi, che a loro volta assumono le caratteristiche attuali ed iniziano a differenziarsi ed a mischiarsi in sottogeneri, talmente tanti e tanto famosi da meritare una trattazione a parte.

Passiamo ora ad esaminare alcuni generi di videogioco molto popolari, e che rappresentano una larga parte del mercato dei videogiochi per computer, in cui l'AI assume un ruolo molto importante per il loro funzionamento, influenzando molto il gameplay.

## *1.3 AI dei Videogiochi di Strategia in Tempo Reale (RTS)*

### 1.3.1 Definizione di RTS

In un videogioco di strategia in tempo reale (*real time strategy*, RTS) si affrontano generalmente due sfide contemporaneamente: la prima è quella di creare e gestire un proprio centro di produzione unità e di raccolta delle risorse, la seconda è quella di gestire il proprio esercito per affrontare ed eliminare gli eserciti nemici ed i loro centri produttivi, divenendo così il dominatore della mappa di gioco. Sostanzialmente quindi è una simulazione abbastanza fedele, ovviamente nei limiti di un gioco, di una vera e propria guerra dove può anche capitare di dover gestire intricati sistemi di alleanze.

Il gioco è improntato ad un contesto multi-giocatore, i giocatori avversari e alleati possono quindi essere sia altri utenti che giocatori controllati dall'AI di gioco, a cui può essere assegnato un grado di difficoltà.

Esempi di questo genere sono: *Caesar*, *Warcraft: Orcs and Humans*, *Age of Empires*, *Starcraft*, *Dune II: Battle for Arrakis* e *The Settlers*.

### 1.3.2 Compiti dell'AI di un RTS

In questo genere di gioco si possono distinguere due AI distinte: la prima è quella che simula il comportamento del giocatore umano, mentre la seconda è quella che gestisce le azioni automatiche delle unità e degli edifici presenti nel gioco, sia che siano o meno possedute dal giocatore umano.

La AI che simula il giocatore umano deve prendere decisioni di genere economico, militare e strategico, deve cioè saper gestire ed amministrare le sue risorse iniziali per accumularne quantitativi sempre maggiori in modo da poter sostenere lo sforzo bellico richiesto dalla partita, deve saper costituire e gestire il suo esercito, difendendo le posizioni strategicamente importanti come i propri impianti di estrazione e quelli produttivi ed attaccando sistematicamente quelli nemici. Inoltre, se il gioco implementa un sistema di diplomazia deve essere in grado di svolgere un

gioco di squadra con i propri alleati o avere le facoltà per valutare un'offerta d'alleanza o un ultimatum di resa; oltre a questi compiti deve essere anche in grado di instaurare un certo livello di comunicazione con il giocatore umano, amico o nemico, che si trova di fronte.

Mentre le funzioni decisionali utilizzano spesso raffinati algoritmi comportamentali e decisionali, le funzioni di comunicazione spesso sono decisamente semplici e rispondono ad una gamma di comandi abbastanza ristretta con azioni semplici. I parametri che questo genere di AI utilizza per le sue decisioni possono essere influenzati dal giocatore umano in fase di inizializzazione e definizione della singola partita tramite i parametri di difficoltà dell'AI, mentre solitamente ai livelli di difficoltà “bassi” si rimane su un gioco abbastanza bilanciato tra giocatore e AI, spesso i livelli di difficoltà “alti” ammettono che l'AI possa barare in vari modi a seconda del gioco, rendendo la sfida davvero molto ardua.

L'AI adibita a controllare le azioni/reazioni automatiche delle unità è un componente fondamentale per il buon funzionamento di un RTS, che viene messa a supporto del giocatore umano, poiché egli non sarebbe in grado di manovrare simultaneamente tutte le unità che vanno a costituire il suo popolo senza questo supporto. Questo genere di AI è decisamente più semplice di quella precedente, poiché si limita ad eseguire una serie di azioni/reazioni seguendo uno schema a stati finiti che spesso risulta estremamente semplice. Anche questo genere di comportamento può essere modificato dal giocatore secondo alcuni schemi, nel corso della partita sarà quindi possibile ordinare ad un'unità di essere “aggressiva”, attaccando il nemico con cui entra in contatto ed inseguendolo fino alla sua sconfitta, “stazionaria” o “difensiva”, mantenendo una posizione senza spostarsi ed attaccando il nemico solo finché esso è a portata ma senza inseguirlo, o “neutrale”, non effettuerà alcun azione offensiva se non dietro diretto ordine del giocatore.

## *1.4 AI dei Videogiochi Sparatutto in Prima Persona (FPS)*

### 1.4.1 Definizione di FPS

In un videogioco sparatutto in prima persona (f*irst person shooter*, FPS) si interpreta un personaggio, solitamente un soldato o un mercenario, che deve affrontare una serie di sfide e missioni combattendo prevalentemente con armi da fuoco. La caratteristica principale di questo genere di sparatutto è che la visuale di gioco è quella del personaggio che si sta utilizzando, da cui il nome del genere. Questo genere rientra nella categoria delle simulazioni di guerra, in cui solitamente si controlla solo il proprio personaggio o al massimo una piccola unità di combattimento.

Il gioco è improntato al contesto multi-giocatore, siano essi umani o controllati dall'AI di gioco a cui può essere assegnato un grado di difficoltà.

Esempi di questo genere sono: *Quake*, *Doom*, *Medal of Honor*, *Call of Duty*, *Halo: Combat Evolved*, *Unreal Tournament* e *Half Life*.

### 1.4.2 Compiti dell'AI di un FPS

In questo genere di videogioco abbiamo sostanzialmente un solo tipo di AI, quella dei personaggi non giocabili, che non sono cioè controllati direttamente dal giocatore, anche se in alcuni casi esiste anche una semplice AI per i proiettili e gli esplosivi.

L'AI di un personaggio deve quindi poter eseguire la stessa gamma di operazioni che può eseguire un giocatore.

Innanzitutto deve sapere come muoversi, cioè saper scegliere se per spostarsi sia meglio rimanere in piedi e correre o accovacciarsi e camminare o addirittura sdraiarsi e strisciare dopo aver valutato la situazione. Deve saper valutare se sia il caso di cercare un riparo per defilarsi dalla visuale nemica o se invece sia il caso di rimanere allo scoperto. Deve essere consapevole dell'ambiente attorno a se, avendo consapevolezza del proprio campo visivo ed a volte anche uditivo, in modo da poter e sapere in che modo reagire ad eventuali minacce od attacchi nemici.

Deve sapere combattere, valutare il tipo di armi in suo possesso e le relative scorte di munizioni e scegliere quale sia meglio usare contro il nemico attuale, deve anche sapere prendere la mira con le proprie armi, ma in maniera umana, cioè non deve avere una mira perfetta, che potrebbe facilmente avere essendo parte del software di gioco e potendo mirare con le coordinate invece che con la vista umana, per cui vengono inseriti dei meccanismi per rendere la mira dell'AI imperfetta.

Infine deve essere in grado di cooperare con altre AI o giocatori umani per svolgere la propria missione. La precisione e l'accuratezza con cui l'AI esegue questi compiti è fortemente influenzata dai parametri di difficoltà del gioco, che qui si può seriamente considerare pari al livello di intelligenza utilizzato dall'AI.

Parlando dell'AI che gestisce proiettili ed esplosivi, in genere mine, si tratta quindi di entità all'apparenza semplici, ma che per alcuni tratti risultano complesse. La semplicità risiede nel fatto che deve gestire ed interpretare un minor numero di dati e operazioni, infatti le uniche operazioni che svolge questo tipo di AI sono un controllo di prossimità, per stabilire se ha colpito o meno qualcosa, un controllo sulla propria energia cinetica per stabilire la traiettoria ed eventuali perforazioni, ed infine un conteggio del danno inflitto ai personaggi o alle strutture, e la complessità

di questo genere di operazioni costituisce la complessità di questo genere di AI, che tuttavia rimane meno sofisticato di quello precedente e che a volte non viene neppure implementato, lasciando le operazioni sopracitate ad altre parti del gioco.

## *1.5 AI dei Videogiochi di Ruolo (RPG, MMORPG)*

### 1.5.1 Definizione di RPG, MMORPG

In un videogioco di ruolo (*role play game*, RPG), e nella sua variante multi-giocatore di massa (*massively multiplayer online role play game*, MMORPG), il giocatore interpreta letteralmente il proprio personaggio, detto avatar, all'interno di un mondo più o meno fantastico nel quale l'avatar dovrà affrontare vari tipi di sfide per poter salire di livello, evolvere e potenziarsi, tali sfide si differenziano sostanzialmente in due macro-categorie:

- Player vs Player (PvP): in cui gli avatar si affrontano tra di loro, sia singolarmente che a gruppi, per dimostrare quale sia il più potente;
- Player vs Enviroment (PvE): in cui gli avatar devono affrontare sfide poste dall'ambiente di gioco, sia singolarmente che in gruppi, combattendo contro i personaggi e le creature controllati dall'AI di gioco o svolgendo missioni (*quest*) fornite da alcuni personaggi controllati dall'AI.

Il gioco è fortemente improntato al contesto multi-giocatore.

Esempi di questo genere sono: *EverQuest*, *Ultima Online*, *World of Warcraft*, *Guild Wars* e *Lineage II*.

### 1.5.2 Compiti dell'AI di un RPG, MMORPG

In un videogioco RPG, o MMORPG, l'AI ha il compito di gestire ogni singolo personaggio o creatura non controllabile direttamente da un giocatore, detti NPC (*non playable character*).

Questi NPC si differenziano in svariate categorie a seconda del proprio ruolo nel gioco e delle loro reazioni nei confronti dei giocatori, per cui ogni categoria deve avere una sua AI con un grado di consapevolezza via via diverso a seconda dei propri compiti nel gioco. In generale però tutte le categorie di AI che si possono incontrare in un videogioco di questo tipo condividono un set di operazioni basilari: devono sapere in che maniera muoversi e devono saper riconoscere le unità nemiche ed affrontarle in combattimento.

A seconda della categoria dell'NPC tali funzioni vengono quindi raffinate ed affiancate ad altre, come ad esempio la possibilità di offrire quest ai giocatori, o la possibilità di dialogare, anche se con testi predefiniti, coi giocatori per offrire informazioni o narrare una storia, o ancora la possibilità di usare oggetti o lanciare magie, la possibilità di evocare altri NPC, la capacità di teletrasportare i giocatori in determinati luoghi, la possibilità di scatenare eventi sotto certe condizioni, la capacità di cooperare e comunicare con altri NPC, ecc.

Oltre all'AI degli NPC veri e propri, spesso si trova anche un'AI addetta a controllare le creature che i giocatori possono evocare al proprio fianco per combattere. Queste creature spesso sono quasi completamente controllabili dal giocatore che le evoca, ma alcuni loro comportamenti possono essere lasciati in gestione ad un'AI configurabile durante il gioco attraverso un'apposita interfaccia. I compiti di questo tipo di AI sono in gran parte simili a quelli di un qualsiasi NPC, se non che alcuni possono essere forzati dal giocatore stesso.

Per cui in un videogioco RPG, o MMORPG, non abbiamo un numero generico e predefinito di AI, ma abbiamo una famiglia di AI, dipendente dal singolo titolo di gioco, a cui è demandata una grande quantità di compiti.

# Capitolo 2 Warcraft

## *2.1 Storia del Videogioco e del Marchio*

Analizziamo ora un titolo che si è sviluppato da "semplice" videogioco di strategia in tempo reale (RTS) fino a diventare un successo mondiale come gioco di ruolo di massa online (MMORPG), entrando tra i 10 più influenti videogiochi di sempre.[2]

### 2.1.1 Warcraft I: Orcs and Humans

Immagine 2.1.1: logo del gioco

*Warcraft: Orcs and Humans*, anche noto come *Warcraft I*, è un videogioco strategico in tempo reale (RTS) sviluppato da Blizzard Entertaiment nel 1994.[3]

Questo capitolo, capostipite della saga di Warcraft, fu uno degli RTS più popolari dei suoi tempi e divenne una pietra miliare del genere. Poteva essere installato sia su sistemi Microsoft che Macintosh.

Prevedeva una modalità multiplayer, le cui specifiche di connessione erano indicate nel manuale di istruzioni del gioco.

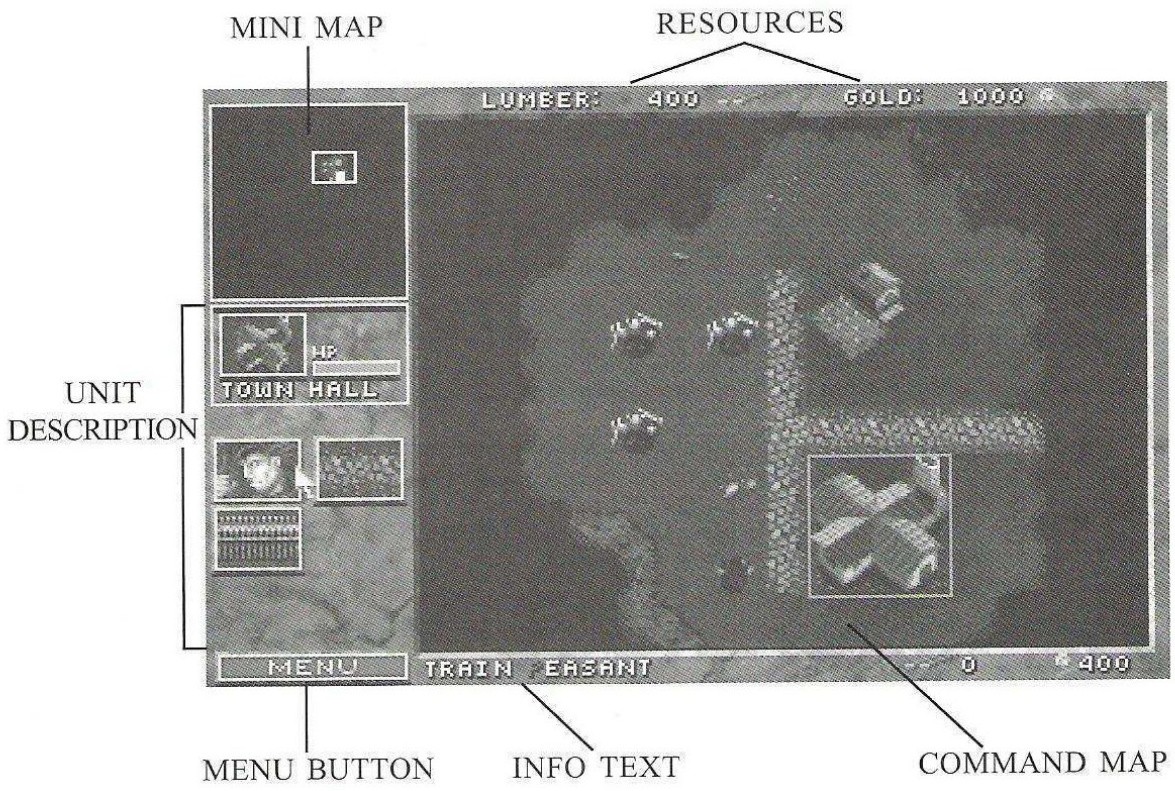

Figura 2.1.1: schermata di gioco, presa dal "Manuale di Gioco"

Sebbene la grafica fosse ancora abbastanza primitiva, poco improntata alla tridimensionalità, nella schermata di gioco possiamo già vedere tutti gli elementi che andranno a caratterizzare molti altri titoli dello stesso genere:

vediamo infatti una casella dedicata alla minimappa (mini map), uno spazio per la visione delle risorse di gioco (resources) a disposizione, in questo caso oro (gold) e legname (lumber), un riquadro per la descrizione dell'unità/edificio selezionato (unit description), una parte informativa delle abilità degli edifici (info text), ed al centro la mappa vera e propria su cui si svolge l'azione del gioco (command map).

In questo gioco si poteva scegliere di affrontare partire brevi contro degli avversari che potevano essere sia avversari controllati dall'AI di gioco, sia avversari umani o avversari e due i tipi. In queste partite era possibile giocare comandando di una delle due razze presenti nel gioco: Orchi o Umani.

Inoltre, ma solo in modalità giocatore singolo, era possibile affrontare una campagna di scenari in cui si controlla a seconda dello scenario una delle due razze presenti. Durante la campagna si assiste alla narrazione della storia dell'universo di Warcraft, quindi all'esperienza del gioco in se si affianca anche una componente narrativa molto coinvolgente.

I fatti narrati nella campagna di *Warcraft I* e nel suo manuale, nella cronologia dell'universo di Warcraft, sono quelli della così detta Prima Guerra, riguardanti l'arrivo dell'Oda degli orchi, originari del pianeta Draenor e corrotti dalla Legione Ardente, sul pianeta di Azeroth attraverso il Portale Oscuro aperto dal guardiano Medivh, posseduto dal titano malvagio Sargeras, e dallo stregone Gul'dan. Dopo un primo attacco infruttuoso contro la città di Stormwind, l'Orda riorganizzatasi, sotto la guida del warchief Blackhand, dilaga nelle terre attorno al Portale Oscuro, distruggendo numerosi villaggi del regno di Stormwind, dopo di che il generale dell'Orda Orgrim assassina Blackhand e ne prende il posto a capo dell'Orda e guida un nuovo attacco a Stormwind che si conclude con la distruzione della città e con l'assassinio del suo re Llane. Nel frattempo un gruppo di umani, guidati dal paladino Lothar e dal mago Khadgar, scopre del tradimento di Medivh e decidono di attaccare la sua torre, durante lo scontro Medivh viene ucciso e lo spirito del titano fugge. Intanto il capo del clan Frostwolf, Durotan, informa Orgrim che Gul'dan ha intenzione di tradire l'Orda, ma poco dopo viene ucciso con la sua compagna dagli orchi fedeli a Gul'dan, lasciando così orfano il figlio, che viene trovato da un gruppo di umani che gli danno il nome di Thrall. Scoperto del tradimento di Gul'dan, Orgrim ordina ai suoi soldati migliori di ucciderlo assieme al suo Concilio Oscuro, ma Gul'dan, in coma dopo aver cercato di recuperare informazioni sulla tomba di Sargeras, riesce a scappare con alcuni stregoni e si rifugia nuovamente su Draenor. Dopo la caduta di Stormwind, Lothar decreta il regno ormai perduto e si dirige a nord verso il regno

di Lordaeron. Si conclude così la Prima Guerra.

### 2.1.2 Warcraft II: Tides of Darkness & Beyond the Dark Portal

*Warcraft II: Tides of Darkness* è stato pubblicato nel novembre del 1995 dalla Blizzard Entertainment. Nell'aprile del 1996 la Blizzard ha pubblicato l'espansione *Warcraft II: Beyond the Dark Portal*, e nel settembre del 1999 ha pubblicato una nuova versione intitolata *Warcraft II Battle.net Edition*, che permette di giocare in multiplayer utilizzando il servizio gratuito di gioco online della Blizzard, Battle.net.[4]

Immagine 2.1.2.1: logo del gioco

Il gioco ricalca gli aspetti salienti del primo capitolo, migliorando la grafica, ed inserendo nuove unità agli eserciti delle due fazioni.

Come abbiamo detto in precedenza la grafica è stata migliorata ed improntata verso la tridimensionalità dei modelli, per dare una sensazione più realistica agli scenari, anche se lo stile ed i colori utilizzati rimangono quelli tipici dei cartoni animati per dare maggiormente l'idea di un mondo di fantasia.

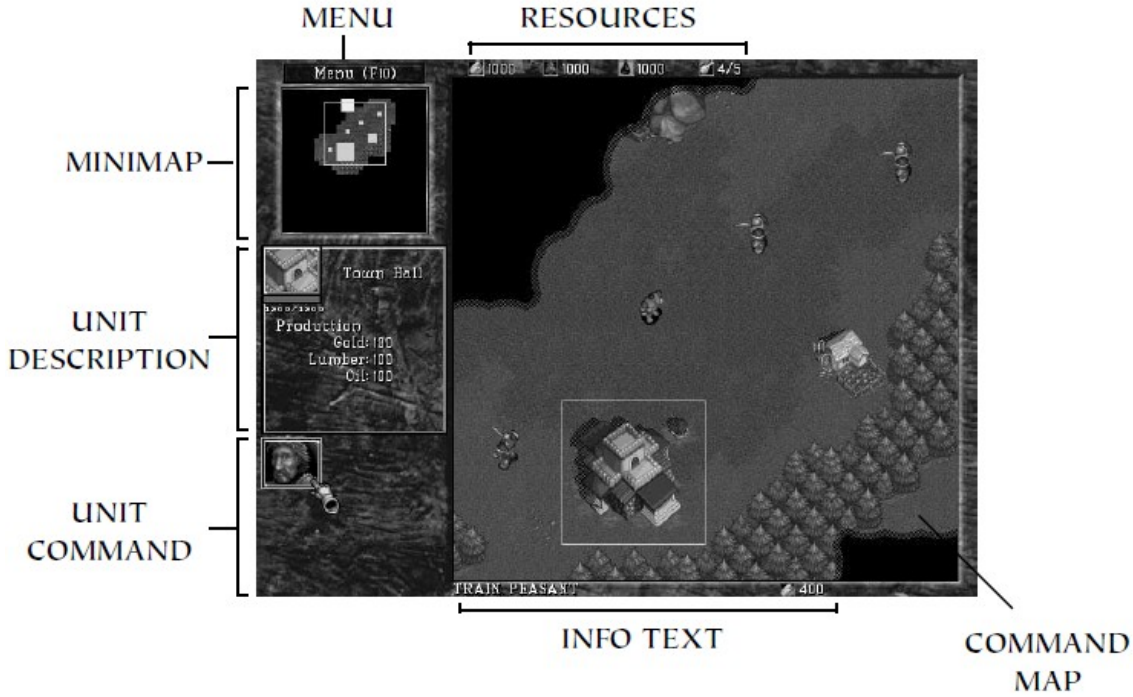

Figura 2.1.2: schermata di gioco, presa dal "Manuale di Gioco"

In questo capitolo fanno la loro comparsa nuove razze, inserite nelle due fazioni controllabili nel

gioco: da una parte abbiamo l'Alleanza di Lordaeron, costituita da umani, nani, gnomi ed elfi alti; dall'altra abbiamo l'Orda, costituita da orchi, goblin, troll e ogre. Inoltre nelle campagne sono presenti delle unità speciali, dette eroi, più potenti delle normali unità e che rappresentano personaggi importanti per la storia.

Tutta la storia narrata nelle campagne di questo capitolo costituisce nella cronologia di Warcraft la Seconda Guerra.

Nelle campagne e nel manuale di *Tides of Darkness* si narra della fondazione dell'Alleanza di Lordaeron, per contrastare e respingere l'Orda; la storia si svolge a sei anni di distanza dalla fine della Prima Guerra, gli orchi, guidati da Orgrim, si sono maggiormente organizzati rispetto al precedente conflitto e mirano a distruggere tutti i regni umani rimasti, in questi anni hanno guadagnato l'appoggio della razza dei goblin e si sono impossessati della Demon Soul, con la quale hanno soggiogato Alextrasza e la sua prole dei draghi rossi; mentre Lothar, con l'aiuto del re di Lordaeron, Terenas, è riuscito a stabilire un'alleanza, che sarà conosciuta come Allenza di Lordaeron, tra i regni umani di Lordaeron, Gilneas, Stormgarde, Dalaran, Alterac e Kul Tiras, i nani di Ironforge e Aerie Peak, gli gnomi di Gnomeregan e gli elfi alti di Quel'talas. Inizialmente l'Orda lancia due assalti simultanei: il primo nella regione delle montagne Blackrock, diretto a minacciare il regno nanico di Khaz Modan; il secondo invece nella zona insulare al largo delle rovine della città di Stormwind, per impadronirsi della basi navali dell'area e lanciare un successivo attacco ai regni del nord, nelle regioni delle Wetlands negli altipiani di Arathi e soprattutto sulle coste meridionali di Lordaeron. Durante il primo assalto alle coste vennero distrutti tutti i villaggi costieri degli umani e l'Orda potè inviare un gran numero di truppe nelle terre di Lordaeron, grazie anche all'aiuto del regno di Alterac. Durante questo assalto l'Orda entra in contatto con Zul'jin ed i troll del regno di Zul'aman, prigionieri degli umani, con i quali stringono un'alleanza per distruggere prima gli umani, nemici dell'Orda, e poi gli elfi alti, nemici ancestrali dei troll. Sull'altro fronte le forze dell'Orda costringono i nani e gli gnomi e rifugiarsi nella città-fortezza di Ironforge, scavata nella montagna, dove riescono a resistere grazie ai rifornimenti aerei portati dai grifoni, così l'Orda però ottiene una seconda via d'accesso ai territori settentrionali e stabilisce la sua base operativa a Grim Batol. Con un attacco diversivo contro Aerie Peak che tiene impegnato l'esercito di Lothar, un esercito dell'Orda, guidato dai troll di Zul'jin, riesce a penetrare nelle terre precedentemente in possesso dei troll ed ora sotto il controllo degli elfi alti della foresta di Eversong, prima dell'arrivo dei rinforzi dell'Alleanza. Oltraggiati dall'aggressione ai propri territori gli elfi alti schierano finalmente la loro completa forza militare in campo, inizia così la controffensiva dell'Alleanza. Il comandante Lothar decide quindi di dividere le sue forze per sconfiggere l'Orda, una parte comandata dal suo secondo

andrà nei territori degli elfi alti, mentre l'altra comandata da Lothar in persona rimarrà nelle Hinterlands di Aerie Peak per sconfiggere le truppe dell'Orda ancora nella zona. Nonostante lo stregone Gul'dan riesca ad impadronirsi di un artefatto magico degli elfi, l'Alleanza riesce a respingere l'Orda dai suoi territori settentrionali, costringendo la sua flotta ad abbandonare le rive di Lordaeron. Inizia così la campagna per la riconquista dei territori meridionali, con la battaglia per il controllo del ponte di Thandol Span , che collega le due metà del continente, con la sua conquista l'Alleanza ottiene una via d'accesso ai regni meridionali, che permetterà in poco tempo di liberare Khaz Modan e di infliggere gravi perdite all'Orda. Nel frattempo si scopre che il regno di Alterac stava tramando per distruggere l'Alleanza dall'interno, per rappresaglia quindi le sue forze distruggono il piccolo regno traditore, che però ha già permesso ad un esercito dell'Orda di infiltrarsi nei suoi territori, arrivando a minacciare la vicina capitale di Lordaeron, assediandola. Nel frattempo Gul'dan con due clan orcheschi a lui fedeli è partito per la tomba del titano Sargeras, dove però verrà ucciso dai demoni che la abitano, saputo della spedizione di Gul'dan, il capo dell'Orda Orgrim decide di ritirare parte delle sue truppe per fermare lo stregone. Con le forze dell'Orda così indebolite per l'Alleanza è facile rompere l'assedio della capitale e dilagare verso sud, fino alla fortezza di Blackrock, qui Orgrim e Lothar dopo molto tempo si incontrano di persona per la prima volta e si affrontano in duello, nel quale dopo molte peripezie ha la meglio l'orco, che però viene catturato dal vice comandante del defunto Lothar, senza il loro comandante l'Orda è costretta a ritirarsi fino al Portale Oscuro che verrà difeso con una grinta straordinaria dagli orchi, ma alla fine il mago Khadgar, protetto dai prodi guerrieri dell'Alleanza, riuscirà a distruggere il Portale Oscuro, ponendo fine alla resistenza degli orchi ed alla Seconda Guerra

Immagine 2.1.2.2: logo dell'espansione

Nelle campagne e nel manuale di *Beyond the Dark Portal* si narra del tentativo dell'orda di ritornare su Azeroth dal loro pianeta di origine, Draenor conosciuto poi come le Terre Esterne. Su Draenor, dopo la sconfitta di Orgrim e la morte di Gul'dan, il capo Ner'zul ha preso con la forza il controllo delle forze dell'Orda che vi erano rimaste e pianifica una nuova invasione di Azeroth ed una riapertura del Portale Oscuro, ma per il suo piano ha bisogno di recuperare alcuni artefatti in possesso degli umani. Parte così una spedizione su Azeroth che, aiutata dalle forze ancora libere dell'Orda sul pianeta, attacca le città di Stormwind e Dalaran dove erano custoditi tali artefatti e riesce a trafugarli su Draenor. L'Alleanza, ora orfana degli elfi alti, che hanno ritenuto estinto il loro debito nei confronti degli umani, e dei regni umani di Gilneas e Stormgarde, sfavorevoli alla politica di salvaguardia degli orchi rimasti all'interno dei

campi di prigionia, viene così a conoscenza che il passaggio del Portale Oscuro è rimasto aperto, per contrastare la minaccia decide quindi di inviare una spedizione su Draenor, per recuperare artefatti trafugati e per chiudere definitivamente il passaggio che collega i due mondi. Di fronte alla controinvasione dell'Alleanza, Ner'zul è costretto ad accelerare i suoi piani per la riapertura definitiva del passaggio tra i mondi e così apre numerosi passaggi nel Twisting Nether, la dimensione che collega tutti i mondi, che però portano Draenor al collasso, causando anche l'invasione di molte specie di demone, riducendo il pianeta ad un ammasso di terre fluttuanti, che verrà chiamato Terre Esterne, la spedizione dell'Alleanza decide però di rimanere sui resti del pianeta per distruggere il Portale Oscuro li presente, o quantomeno controllarne l'accesso. Ma il capo Ner'zul e alcuni sui fedelissimi riescono a sfuggire attraverso uno dei tanti passaggi aperti. Con questi eventi si conclude definitivamente la Seconda Guerra.

### 2.1.3 Warcraft III: Reign of Chaos & The Frozen Throne

*Warcraft III: Reign of Chaos* è stato pubblicato nel luglio del 2002 dalla Blizzard Entertaiment. Un anno dopo, nel luglio 2003 viene pubblicata l'espansione *Warcraft III: The Frozen Throne*. In entrambe le versioni è già incluso il supporto alla modalità multiplayer attraverso il portale Battel.net della Blizzard.

Immagine 2.1.3.1: logo del gioco

Figura 2.1.3: schermata di gioco

In questo capitolo troviamo introdotte varie novità e molte caratteristiche più tipiche di un gioco di ruolo che di uno strategico. Innanzitutto è introdotto un nuovo tipo di unità: l'eroe, a cui viene dato un nome proprio casuale, più potente della maggior parte delle unità normali e con vari poteri a disposizione, a differenza delle altre unità può acquisire esperienza e salire di livello, guadagnando bonus aggiuntivi per le proprie abilità, inoltre può possedere, racchiusi nel suo inventario, ed utilizzare fino a sei oggetti di varia natura, ancora, l'eroe una volta morto può essere riportato in vita ad un costo variabile. Essendo un'unità tanto potente e pregiata ne viene limitato l'uso ad un massimo di tre per giocatore nelle partite normali, mentre nelle campagne o negli scenari questo numero può essere superato, inoltre si può avere un solo eroe di una certa classe. Altra novità del gioco è il ciclo giorno-notte, che influisce sulle abilità delle unità e sui loro campi visivi. Compaiono nelle mappe i "nemici neutrali", cioè coloro che non sono alleati di nessuno, ma che comunque reagiscono in maniera ostile a qualunque unità di un giocatore che entri nella loro zona. Una volta uccisi, questi avversari lasceranno una piccola quantità d'oro ed a volte oggetti per l'eroe, spesso queste creature di notte sono "addormentate" e non reagiscono se non attaccate.

In questo capitolo sono presenti ben quattro razze giocabili: Umani e Orchi, già presenti nei precedenti capitoli, e Non-morti ed Elfi della Notte, nuove razze introdotte nella storia. Ogni razza ha una sua selezione di unità da combattimento ed edifici, inoltre ogni razza ha una sua serie di poteri speciali ed un suo particolare sistema di estrazione delle risorse, ancora ogni razza possiede diverse classi di eroi, tre originariamente in *Reign od Chaos* e poi quattro in *Frozen Throne*.

Ovviamente sia la grafica del gioco che il livello dell'AI sono stati fortemente potenziati, adesso la grafica ha un aspetto completamente tridimensionale ed è possibile sia ruotare la visuale sia ingrandire la prospettiva classica dall'alto fino alla prospettiva delle unità, con la quale sono realizzati alcuni filmati delle campagne. Sono inoltre presenti filmati cinematici per ogni campagna, ciò migliora di molto l'impianto narrativo del gioco, mentre l'AI di gioco diviene molto agguerrita e competitiva, in grado di offrire un avversario valido già dal livello principiante.

Nel manuale di gioco troviamo raccolte le storie delle quattro razze presenti, di cui viene riassunto l'accaduto dalla loro ultima comparsa nella saga, inoltre in questo capitolo è presente anche la storia della Legione Ardente, che sarà il vero antagonista della storia qui narrata.

In ognuna delle due versioni del gioco abbiamo quattro campagne, una per ogni razza, inoltre in *Reign of Chaos* è presente una campagna introduttiva al gioco con gli orchi.

I fatti narrati dalle campagne di *Reign of Chaos*, nella cronologia dell'universo di Warcraft, sono quelli della Terza Guerra, in cui le razze di Azeroth si troveranno unite a combattere contro

l'invasione della Legione Ardente e dei suoi servi, tutto inizia con un sogno premonitore dello sciamano Thrall, in cui un profeta gli dice che deve radunare l'Orda per dirigersi verso il continente di Kalimdor, usando il suo carisma ed i suoi poteri lo sciamano riuscirà quindi a radunare ciò che resta dell'Orda degli orchi e a condurla verso questa nuova terra. Nel frattempo il regno umano di Lordaeron si trova alle prese con un nuovo nemico, il Flagello dei non-morti, che sta rapidamente dilagando nei territori più settentrionali, il principe Arthas inizia così una sua campagna per sconfiggere questo nuovo nemico e trovarne l'origine, in questa sua ricerca però il giovane principe perde gradualmente il senno e la ragione, arrivando a sterminare un'intera città per evitare che i suoi cittadini, ormai infettati dal morbo che li avrebbe trasformati in non-morti, si uniscano alle schiere del nemico, durante questa "epurazione" Arthas incontra il signore delle tenebre Mal'ganis, che controlla i non-morti, che però riesce a sfuggire alla cattura rifugiandosi nel continente di Northrend. Il principe prende quindi il suo esercito e fa rotta per il continente artico di Northrend, dove anche l'ultima parte di sanità del principe svanirà assorbita dalla sua sete di vendetta nei confronti del Flagello, che lo porterà ad impugnare la spada runica Frostmourne per poter uccidere Mal'ganis, tuttavia tale arma è in realtà legata al vero comandante del flagello, il Re dei Lich, che così riesce a trasformare il principe Arthas nel suo primo cavaliere della morte. Arthas fa così ritorno nella sua terra natale, dove uccide il padre e scioglie gli eserciti di Lordaeron, eliminando una delle maggiori minacce per il Flagello, dopo di che riceve ordine di riportare in vita il negromante Kel'thuzad, il quale ha il compito di aprire il portale per permettere alla Legione Ardente di entrare su Azeroth, nel compiere questi obiettivi Arthas distruggerà il regno elfico di Quel'talas ed il regno magico di Dalaran, solo pochi superstiti riusciranno a mettersi in salvo attraversando il mare o rifugiandosi nei regni di Khaz Modan e Stormwind. Dall'altra parte del mare, nel continente di Kalimdor, l'Orda è riuscita a sbarcare e si è alleata con la locale razza dei nomadi tauren, ma l'Orda quando inizia ad addentrarsi nel continente viene anche a scontrarsi contro le Sentinelle degli elfi della notte e con i superstiti di Lordaeron, ma questi conflitti vengono presto fermati dal profeta Medivh che annuncia ai comandanti dei tre eserciti che presto la Legione sarebbe arrivata li per impossessarsi della fonte magica del pianeta, l'albero del mondo Nordrassil, così viene stabilita una tregua ed un'insolita alleanza tra le opposte fazioni per contrastare il nemico comune. La Legione sferra quindi il suo attacco in forze, comandato dal signore della legione Archimonde, per catturare Nordrassil, però grazie agli eserciti riuniti di umani e orchi e tauren e elfi della notte che riescono a trattenere le forze della Legione ai piedi del monte Hijal, sulla cui cima sorge l'albero del mondo, il druido Malfurion riesce a trovare un modo per distruggere i seguaci della Legione giunti sul pianeta sacrificando l'albero del mondo. Con la sconfitta della Legione si conclude quindi la Terza Guerra.

Immagine 2.1.3.2: logo dell'espansione

I fatti narrati dalle campagne di *Frozen Throne* narrano dell'ascesa del cavaliere della morte, Arthas, fino alla sua fusione col Re dei Lich. La storia inizia con l'inseguimento da parte della guardiana degli elfi della notte Maiev dell'ex prigioniero Illidan, liberato durante la Terza Guerra per combattere la Legione, poiché convinta che egli, ora alleato dei naga una razza di elfi mutati in uominiserprente, abbia intenzione di distruggere il mondo, alla fine si rivelerà che il piano di Illidan era mirato alla distruzione del Trono Ghiacciato nel quale è rinchiuso il Re dei Lich, e per questo il druido Malfurion decreterà la sua libertà definitiva lasciandolo autoesiliarsi nelle Terre Esterne, dove però la guardiana Maiev, accecata dalla vendetta verso Illidan, lo inseguirà per catturarlo. Nel frattempo gli esuli del regno elfico di Quel'talas, comandati dal principe Kael'tas, si sono uniti alle forze di resistenza di Lordaeron che hanno l'obiettivo di riconquistare i territori controllati dal Flagello, tuttavia il comandante supremo Garithos disprezza gli elfi e quando scopre che essi hanno accettato l'aiuto in battaglia dei naga, fa rinchiudere la maggior parte di loro in prigione, qui la regina dei naga lady Vashj aiuta Kael'tas e il suo popolo ad evadere e a fuggire nelle Terre Esterne, dove diverranno alleati di Illidan, che con il loro aiuto riuscirà a sconfiggere il signore del Black Temple, Maghteridon ed impadronirsi delle sue terre, liberando dall'oppressione la popolazione locale dei draenei. Tuttavia qui Illidan viene raggiunto dal signore della legione Kil'jaeden, che gli intima di ritornare su Azeroth col suo esercito per distruggere definitivamente il Re dei Lich. Nel frattempo nelle terre di Lordaeron, Arthas a capo del Flagello inizia a perdere lentamente i suoi poteri ed il controllo sui non-morti, tanto che i tre signori delle tenebre rimasti e la banshee Silvanas gli si ribellano contro e cercano di spodestarlo, preoccupato per il Re dei Lich, fonte dei suoi poteri, Arthas lascia quindi Lordaeron alla volta di Northrend dove si trova il Trono Ghiacciato, purtroppo durante la sua assenza la ribellione si estende ed alla fine Silvanas, dopo aver sconfitto i signori delle tenebre che non si sono piegati al suo volere, si proclama signora dei Reietti, i non-morti che s son separati dal Flagello. Nel continente di Northrend Arthas incontra da subito l'esercito di Illidan, che è già in marcia verso il giacciaio Icecrown, al cui centro si trova il Trono Ghiacciato, per superarlo in rapidità decide quindi di attraversare il regno sotterraneo di Azjol'nerub, distrutto all'arrivo del Flagello in quelle terra, guidato dal signore delle cripte Anub'arak, inviatogli dal Re dei Lich in persona, una volta giunto ad Icecrown Arthas organizza tutte le truppe non-morte presenti per respingere l'attacco di Illidan. Si scatena una feroce battaglia per il controllo dell'accesso al Trono Ghiacciato che termina con un duello tra Illidan e Arthas, che si conclude con la vittoria del cavaliere della morte e la definitiva cacciata di Illidan da Azeroth, avendo eliminato tutte le possibili minacce al suo padrone, Arthas si

incammina verso il Trono Ghiacciato che per ordine del Re dei Lich stesso frantuma con la spada runica Frostmourne, liberando lo spirito del Re dei Lich racchiuso nella sua armatura intrappolata nel Trono Ghiacciato, a questo punto Arthas indossa l'elmo del Re dei Lich e così facendo si fonde definitivamente con esse, diventando lui stesso il Re dei Lich.

L'ultima campagna di *Frozen Throne* narra invece della fondazione della nuova patria degli orchi nel continente di Kalimdor a fianco del regno dei superstiti di Lordaeron sull'isola di Theramor, raccontando come Thrall riesca a dare dopo molti anni una patria agli orchi, dopo aver combattuto con l'ammiraglio Proudmore, superstite di Lordaeron e veterano delle prime due guerre.

## *2.2 World of Warcraft*

### 2.2.1 Panoramica del Videogioco

Immagine 2.2.1: logo del gioco

*World of Warcraft* (letteralmente "il mondo di Warcraft", spesso abbreviato in *WoW*) è un videogioco fantasy tridimensionale di tipo MMORPG, giocabile esclusivamente online, previo il pagamento di un canone. Sviluppato dalla Blizzard Entertainment, è stato pubblicato il 23 novembre 2004. ***[5]***

L'ambientazione del gioco è il mondo in cui si sono svolti i capitoli precedenti della saga di Warcraft, cioè l'universo di Warcraft. Il periodo in cui si svolge la storia di *WoW* è successivo ai fatti narrarti in *Warcraft III: The Frozen Throne* di circa quattro anni, il mondo di Azeroth ha subito dei cambiamenti di natura politica che hanno portato alla nascita, ed all'espansione, delle due grandi fazioni che coesistono e si contendono il dominio sul pianeta: l'Alleanza e l'Orda. Che si ritrovano ad affrontare varie nuove minacce come i draghi neri Onyxia e Nefarian, figli di Neltharion, o la ricomparsa dei troll Gurubashi nella loro città di Zul'gurub guidati dal loro dio sanguinario Hakkar, o la Guerra delle Sabbie condotta dai quiraj e dal loro antico dio C'Thun, o ancora la ricomparsa delle truppe del flagello comandate da Kel'thuzad nella cittadella della morte di Naxxramas.

Un giocatore può scegliere tra quattro tipi diversi di server su cui giocare:

- Normal (PvE), in cui il gioco è incentrato nelle sfide con l'ambiente;
- RP, in cui ci si focalizza sulla componente roleplay;
- PvP, in cui il gioco è incentrato nella sfida dell'altra fazione;

- RPPvP, che ha le caratteristiche sia di un server RP che di server PvP.

Nella versione base del gioco si può scegliere di giocare in una delle due opposte fazioni del gioco, l'Alleanza o l'Orda, per ogni fazione sono disponibili quattro diverse razze con diversi poteri e classi disponibili.

Le razze dell'Alleanza sono:

- umani (*humans*), con capitale a Stormwind, ricostruita dopo la Seconda Guerra, che hanno accesso alle classi di guerriero (*warrior*), paladino (*paladin*), ladro (*rogue*), sacerdote (*priest*), mago (*mage*) e stregone (*warlock*);
- elfi della notte (*night elves*), con capitale a Darnassus, dove si sono stabiliti dopo la battagli del monte Hijal, che hanno accesso alle classi di guerriero (*warrior*), druido (*druid*), ladro (*rogue*), cacciatore (*hunter*) e sacerdote (*priest*);
- nani (*dwarves*), con capitale ad Ironforge, che hanno accesso alle classi di guerriero (*warrior*), paladino (*paladin*), ladro (*rogue*), cacciatore (*hunter*) e sacerdote (*priest*);
- gnomi (*gnomes*), che esuli dalla loro capitale Gnomeregan si sono stabiliti ad Ironforge, che hanno accesso alle classi di guerriero (*warrior*), ladro (*rogue*), mago (*mage*) e stregone (*warlock*).

Le razze dell'Orda sono:

- orchi (*orcs*), con capitale a Orgrimmar, fondata da poco dal capo Thrall, che hanno accesso alle classi di guerriero (*warrior*), sciamano (*shaman*), ladro (*rogue*), cacciatore (*hunter*) e stregone (*warlock*);
- troll (*trolls*), esuli dalla loro terra si sono stabiliti poco distanti da Orgrimmar a cui fanno riferimento come capitale, che hanno accesso alle classi di guerriero(*warrior*), sciamano (*shaman*), ladro (*rogue*), cacciatore (*hunter*), sacerdote (*priest*) e mago (*mage*);
- tauren (*taurens*), con capitale a Thunder Bluff, creata come centro di aggregazione per le carovane, che hanno accesso alle classi di guerriero(*warrior*), sciamano (*shaman*), druido (*druid*) e cacciatore (*hunter*);
- non-morti reietti (*forsaken undead*), con capitale ad Undercity, che altro non è che i resti della fortezza sotterranea costruita sotto la vecchia capitale di Lordaeron, che hanno accesso alle classi di guerriero (*warrior*), ladro (*rogue*), sacerdote (*priest*), mago (*mage*) e stregone (*warlock*);

Oltre alla fazione, alla razza ed alla classe è data la possibilità al giocatoredi scegliere per il suo personaggio fino a due delle nove professioni "primarie" disponibili:

- erobristeria (*herbalism*), che permette di raccogliere erbe e piante che si troveranno durante i propri viaggi, per poterle rivendere o utilizzarle come materia prima per altre professioni;
- estrazione (*mining*), che permette di estrarre minerali e gemme dalle vene che si trovano sparse nel mondo di Azeroth, inoltre consente di trasformare i minerali grezzi in barre del relativo metallo, per poterle rivendere o utilizzarle come materia prima per altre professioni;
- scuoiatura (*skinning*), che permette di prelevare le pelli dai corpi degli animali uccisi, per poterle rivendere o utilizzarle come materia prima per altre professioni;
- alchimia (*alchemy*), che consente di creare pozioni e oli, che daranno bonus al personaggio che li utilizza, e trasmutare elementi, che serviranno come materia prima per altre professioni, la materia prima primaria che richiede questa professione sono le erbe e le piante;
- forgiatura (*blacksmithing*), che consente di creare armature in piastre o in cotta di maglia, oltre che vari tipi di armi, oltre ad utensili che possono dare bonus al personaggio che li utilizza, la materia prima primaria che richiede questa professione sono le barre di metallo;
- conciatura (*leatherworking*), che consente di creare armature di pelle o in cotta di maglia e vari rinforzi per esse, ma anche faretre, la materia prima primaria che richiede questa professione sono le pelli;
- ingegneria (*engineering*), che consente di creare una gran varietà di oggetti particolari, molti dei quali esplosivi, fucili e proiettili, richiede molti tipi di materia prima, anche se quella prevalentemente utilizzata sono le barre di metallo;
- sartoria (*tailoring*), che consente di creare armature di stoffa e borse, la materia prima primaria che richiede sono le stoffe, che possono essere trovate addosso alla maggior parte delle creature umanoidi dopo averle sconfitte;
- incantamento (*enchanting*), che consente di creare bacchette magiche e di incantare i vari componenti del proprio equipaggiamento con vari effetti, la materia prima richiesta per questa professione è il materiale magico, che si può ottenere distruggendo pezzi di equipaggiamento di qualità medio-alta.

E' possibile in qualsiasi momento abbandonare una professione "primaria" e perdere tutti i progressi

fatti con essa, per poterne imparare un'altra.

Oltre alle professioni “primarie”, il personaggio del giocatore può imparare anche le tre professioni “secondarie”:

- pronto soccorso (*first aid*), che consente di creare bende ed antidoti per curare i personaggi, la materia prima che richiede questa professione sono le stoffe;
- cucina (*cooking*), che consente di creare cibi ed alcune bevande che possono dare bonus al personaggio che se ne ciba, la materia prima che richiede questa professione sono le carni degli animali ed i pesci;
- pesca (*fishing*), che consente di pescare in fiumi, laghi e nel mare, catturando così i pesci richiesti dalla professione cucina.

Un personaggio una volta giunto al livello 10 ottiene dei “punti talento” che può spendere per acquistare potenziamenti o abilità supplementari per la propria classe, scegliendo da tre diversi rami di specializzazione, tale scelta influirà poi sul ruolo svolto dal personaggio.

I personaggio possono unirsi in organizzazioni, chiamate gilde, per darsi reciproco aiuto durante il gioco. Ogni gilda può creare un suo stendardo (*guild tabard*) che i suoi membri possono indossare per farsi riconoscere. La struttura di base delle gilde è abbastanza gerarchica, al cui vertice sta il *guild master*, solitamente colui che ha deciso di fondare la gilda, circondato dai suoi ufficiali e poi vengono tutti gli altri membri divisi per gradi a discrezione del guild master. Per fondare una gilda è necessario procurarsi il consenso di almeno 10 personaggi (uno dei quali diventerà *guild master*) e fargli “firmare” la carta di fondazione della gilda, che si può acquistare a poco prezzo in qualsiasi capitale da un mastro delle gilde, ovviamente si può entrare a far parte di una sola gilda alla volta e si può essere invitati in una gilda già creata.

Per combattere effettivamente uno affianco all'altro i giocatori devono unire i propri personaggi in gruppi, detti *party* che possono contenere al massimo 5 personaggi o *raid* che ne possono contenere fino a 40. A capo di un gruppo c'è sempre un *leader*, che solitamente è il primo personaggio che invita a far parte del gruppo gli altri personaggi, che può decidere varie regole per il gruppo stesso e può in caso sanzionare chi le infrange, nei gruppi *raid* il *leader* può concedere ad alcuni membri il ruolo di assistente (*assist*), permettendogli così di poter invitare/rimuovere personaggi nel guppo e modificare la composizione dei sottogruppi.

Nel mondo sono disponibili varie zone istanziate, dette appunto *istance*, in cui un personaggio può interagire solo con l'ambiente circostante ed i componenti del suo gruppo, che si suddividono in due

macro-categorie: *istance*, se ci si può entrare anche da soli o con un gruppo *party*, e *raid*, se ci si può entrare solo con un gruppo *raid*. A loro volta i *raid* si distinguono in due categorie, a seconda del numero dei giocatori che possono avervi accesso contemporaneamente, *20man* e *40man*.

Dal livello 40 il personaggio può acquistare la capacità di cavalcare creature dette *mount*, per incrementare la velocità dei suoi spostamenti.

In questa prima versione, nota poi come *Vanilla WoW*, il livello massimo raggiungibile era il 60. Ad essa vengono effettuate molte aggiunte ed aggiornamenti, detti *patch*, dal momento del suo rilascio, si passa dalla versione 1.1.0, al momento del rilascio, fino ad arrivare alla versione 1.12.2, ultima versione pre-2.x, ed alla 2.0.1 prima del rilascio della prima espansione.

Nel 2007 è stata pubblicata la prima espansione di *World Of Warcraft*, sottotitolata *The Burning Crusade*. [5]

Immagine 2.3.3: logo della prima espansione

I fatti che si svolgono in questa espansione avvengono a circa un anno di distanza dall'inizio della versione base di *WoW*, secondo la cronologia dell'univero di Warcraft. Tutto comincia con la riapertura del Portale Oscuro da parte del signore dei demoni Kazzak, che scatena da entrambe le fazioni una crociata per la sconfitta dei demoni su Azeroth e nelle Terre Esterne, tuttavia la rivalità tra le due fazioni porta i due eserciti a scontrarsi varie volte, nel frattempo Illidan, preoccupato per la sua posizione, decide di attaccare entrambi gli eserciti delle due fazioni per aver invaso i suoi domini ed attaccato i suoi alleati, così però crea le condizioni per una tregua tra le due fazioni che si uniranno per affrontarlo di fronte alla sua cittadella del Black Temple; altri nemici che compariranno in questa espansione saranno i troll dell'impero Amani di Zul'aman, e Kil'jaeden signore della legione.

Con questa espansione vengono introdotte molte novità.

Innanzi tutto il livello massimo raggiungibile è incrementato al 70.

Una nuova professione “primaria”, creatore di gioielli (*jewelcrafting*), che consente di creare anelli, collane, ninnoli e gemme per potenziare l'equipaggiamento dei personaggi, richiede come materie prima sia i metalli grezzi che le barre.

Un nuovo continente, le Terre Esterne (*Outland*), e nuove zone su Azeroth, due per ognuno dei continenti presenti.

Due nuove razze:

- draenei (*draenei*), a fianco dell'Alleanza, con capitale a Exodar, che hanno accesso alle classi di di guerriero(*warrior*), sciamano (*shaman*), paladino (*paladin*), cacciatore (*hunter*), sacerdote (*priest*) e mago (*mage*);
- elfi del sangue (*blood elves*), a fianco dell'Orda, con capitale a Silvermoon ricostruita dopo la Terza Guerra, che hanno accesso alle classi di paladino (*paladin*), ladro (*rogue*), cacciatore (*hunter*), sacerdote (*priest*), mago (*mage*) e stregone (*warlock*);

Un nuovo sistema di potenziamento dell'equipaggiamento basato sui supporti (*socket*) per le gemme prodotte dai creatori di gioielli, che si affianca al precedente sistema di incanti.

La possibilità di possedere ed utilizzare cavalcature volanti nelle Terre Esterne, una volta raggiunto il livello massimo.

Ed ovviamente nuove *istance*, che risultano decisamente diverse dalle *istance* classiche, innanzitutto sono state decisamente ridotte in dimensioni e ne è stata semplificata la pianta, viene anche introdotta una modalità "eroica" per le *istance*, sbloccabile una volta raggiunto il livello 70 ed acquistata una particolare chiave; anche i *raid* nuovi vengono ridimensionati, sia nella pianta che nel numero di partecipanti, rispetto ai precedenti, infatti sono presenti solo *raid* di tipo *10man* o *25man*.

Viene inoltre aggiunto un nuovo *battleground*, zone istanziate in cui i personaggi delle due fazioni possono affrontarsi in giochi di gruppo a squadre facendo anche PvP introdotti nella versione 1.5.0, con ambientazione nelle Terre Esterne.

Anche questa espansione subisce molti cambiamenti con le *patch*, si passa infatti dalla versione 2.0.3, al momento del lancio, alla 2.4.3, ultima versione pre-3.x, fino alla 3.0.2 prima del lancio della seconda espansione, che ne anticipa alcuni contenuti.

Immagine 2.2.3: logo della seconda espansione

*Wrath of The Lich King* è la seconda espansione di World of Warcraft, pubblicata in contemporanea mondiale il 13 novembre 2008.[5]

I fatti narrati da questa espansione si svolgono poco dopo la caduta di Illidan e la sconfitta di Kil'jaeden, il Re dei Lich, Arthas, si è finalmente risvegliato dal suo sonno sul Trono Ghiacciato e mira a sottomettere tutte le razze di Azeroth sotto il dominio del Flagello. Per il suo scopo Arthas fonda un nuovo ordine di cavalieri della morte, la Spada d'Ebano, per contrastare i paladini dell'ordine della Mano d'Argento,

tuttavia l'assalto contro la roccaforte dei paladini si risolverà in una disfatta delle truppe del Re dei Lich, che verranno in gran parte catturarti e liberati dal suo controllo mentale ed uniranno le loro forze a quelle dell'Orda e dell'Alleanza. Le due fazioni invieranno così le proprie truppe nel continente artico di Northrend, per fermare la minaccia del ritorno del Flagello. In questa espansione gli avversari che i giocatori sono molteplici, prima tra tutte il ritorno delle cittadella della morte di Naxxramas nelle terre di Northrend per riorganizzare l'esercito del Flagello ed annientare le spedizioni dell'Orda e dell'Alleanza, il ritorno dei draghi neri con Sartharion, prima, e Onyxia, dopo, la guerra del Nexus condotta dal signore dei draghi blu Malygos contro ogni essere non draconico che utilizzi la magia, il risveglio dell'antico dio della morte Yogg Saron dalla sua prigione di Ulduar, la chiamata alla crociata contro il Re dei Lich e le prove per essere degni di parteciparvi che si svolgono nel Colosseo della Crociata, ed infine, naturalmente, l'assalto alla fortezza della cittadella di Icecrown dove si trova il Re dei Lich.

Anche questa espansione, assieme alle sue patch, aggiunge varie novità al gioco.

In primissimo luogo il livello massimo viene innalzato all'80.

Un nuovo continente su Azeroth, Northrend con una zona completamente dedicata al PvP di massa.

Una nuova classe, indicata come classe eroica, il cavaliere della morte (*death kinight*), disponibile per tutte le razze, per poter creare un personaggio di questa classe deve essere presente nell'account almeno un personaggio che abbia raggiunto il livello 55, il cavaliere della morte inizia a crescere da tale livello in una zona istanziata dove, attraverso una serie di quest, ripercorre la storia della Spada d'Ebano. E' possibile possedere un unico cavaliere della morte per ogni reame (server) di gioco su uno stesso account.

Una nuova professione “primaria”, inscrizione (*inscription*), che consente di creare pergamene (*scroll*), che danno potenziamenti temporanei, e glifi (*glyph*), che modificano alcune abilità di classe del personaggio come i talenti, richiede come materia prima le erbe e le piante. (presente già nel pre-rilascio della *patch* 3.0.2)

L'introduzione dei veicoli, macchine con abilità proprie che un giocatore può controllare una volta che il suo personaggio vi è “entrato a prenderne il controllo”.

L'introduzione del sistema degli *Achievements*, che assegna dei punti di prestigio e a volte anche oggetti rari o titoli ad un personaggio in base ai traguardi da esso raggiunti nel gioco, per incentivare i giocatori ad esplorare tutti gli aspetti del gioco. (presente già nel pre-rilascio della *patch* 3.0.2)

L'inserimento delle botteghe dei barbieri (*barbershop*) nelle capitali, in cui i giocatori possono modificare taglio e colore della capigliatura dei propri personaggi. (presente già nel pre-rilascio della *patch* 3.0.2)

Nuove *instance*, seguendo lo stile dell'espansione precedente, presenti sia in modalità *normal*, il cui livello d'accesso varia a seconda dell'*istance*, sia in modalità *heroic*, accessibili dal livello 80; viene inoltre introdotta la possibilità di modificare anche il livello di difficoltà dei *raid*, inizialmente solo in base al numero di personaggi che vi potevano partecipare, scegliendo tra le modalità *10man normal* e *25man normal*, e successivamente anche in base alle capacità dei boss presenti, *10man heroic* e *25man heroic*. Viene così fornita una vasta gamma di incontri PvE.

Per il PvP sono aggiunti nuovi *battleground*, uno al momento del rilascio ed altri due con le *patch* successive.

Successivamente viene introdotta anche la così detta *dual talent specialization*, acquistabile dal livello 40, che consente di poter possedere contemporaneamente due diverse impostazioni di talenti e glifi su uno stesso personaggio, di cui però può esserne attiva una sola per volta, favorendo così la possibilità di svolgere più ruoli con lo stesso personaggio.

Anche questa espansione subisce molti cambiamenti con le *patch*, si passa infatti dalla versione 3.0.3, al momento del lancio, alla 3.3.5a, ultima versione pre-4.x, fino alla 4.0.3a prima del lancio della terza espansione, che contiene già i cambiamenti che saranno rilasciati al momento del lancio della terza espansione, che mantiene questo numero di versione.

Immagine 2.2.4: logo della terza espansione

*Cataclysm* è la terza espansione di *World of Warcraft*. È stata annunciata il 21 agosto 2009 dalla Blizzard, ed è stata pubblicata il 7 dicembre 2010.[5]

I fatti narrati in questa espansione si svolgono pochi anni dopo la caduta del Re dei Lich Arthas, ultimo grande evento avvenuto nella precedente espansione; tutto inizia con il ritorno del signore dei draghi neri Neltharion, meglio conosciuto come Deathwing, dal suo esilio nelle profondità do Azeroth, che con il suo ritorno porta un enorme cataclisma che sconvolge tutta la superficie del pianeta. Dopo il cataclisma compariranno molte minacce, come il ritorno di Nefarian, la ricomparsa Cho'Gall, che fu allievo dello stregone Gul'dan, manipolato da Sintharia, la prima consorte di Deathwing, la comparsa del dio dei venti del piano elementale Al'Akir, il ritorno del signore degli elementali di fuoco Ragnaros, ed infine Deathwing stesso.

Questa è l'espansione che, al momento, apporta più cambiamenti al gioco.

Per prima cosa viene modificato radicalmente l'aspetto dei due continenti iniziali di Kalimdor e degli Eastern Kingdoms, secondo gli sconvolgimenti del cataclisma, inoltre vengono aggiunte nuove zone ed ora è possibile utilizzare le cavalcature volanti in ogni spazio aperto, indipendentemente dal continente. (alcune di queste modifiche sono presenti già nel pre-rilascio della *patch* 4.0.3a)

Il livello massimo raggiungibile viene innalzato all'85.

Viene modificato radicalmente il sistema dei talenti, non si guadagna più un punto talento ogni livello dopo il livello 10, ma se ne guadagna uno ogni livello dispari tra il 10 e l'81, successivamente se ne guadagna nuovamente uno ogni livello, il primo talento acquisito e speso al livello 10 determina il ramo principale su cui andranno spesi i punti talento, finché non si arriverà ad avercene spesi almeno 31, dopo di che sarà possibile spendere i restanti punti su un altro ramo di talenti. (presente già nel pre-rilascio della *patch* 4.0.1)

Viene inserito il sistema di riforgiatura (*reforging*) dell'equipaggiamento, che consente ai giocatori di sostituire una statistica di un pezzo dell'equipaggiamento con un'altra non presente sul pezzo, andando da un riforgiatore arcano (*arcane reforger*), presente in ogni capitale. (presente già nel pre-rilascio della *patch* 4.0.1)

Sono inserite nuove combinazioni razza-classe. (presente già nel pre-rilascio della *patch* 4.0.3a)

Due nuove razze:

- worgen (*worgens*), a fianco dell'Alleanza, prima esuli a Darnasuss e poi con capitale a Gileas;
- goblin (*goblins*), a fianco dell'Orda, con capitale a Bilgewater Harbor;

Una nuova professione “secondaria”, archeologia (*archaeology*), che consente di ricercare e ricostruire artefatti in giro per il mondo e ricevere ricompense in base agli artefatti ritrovati.

Viene modificato sostanzialmente anche il sistema delle statistiche dei personaggi, riducendo il numero di esse e/o rimuovendole dall'equipaggiamento.

Viene introdotto un sistema di avanzamento per le gilde, che ora possono salire di livello, guadagnare *achievement* ed acquisire talenti, per dare dei bonus ai personaggi che ne fanno parte.

Sono aggiunte nuove *istance* e *raid*, che seguono lo schema inserito con la precedente espansione per i livelli di difficoltà, inoltre sono rivisitate alcune *istance* e *raid* delle precedenti versioni che

sono resi disponibili per la maggior parte come *istance heroic* per i personaggi di livello 85.

Sono anche aggiunti nuovi *battleground* per il PvP.

Anche questa espansione viene modificata da varie *pacth*, ma in modo relativamente meno significativo delle precedenti anche per via delle grandi modifiche che apporta con la sua introduzione all'intero gioco, si parte dalla versione 4.0.3a al momento del rilascio, all'attuale versione 4.2.2, è previsto il rilascio della *patch* 4.3.0 entro la fine del 2011 ed è possibile che vengano rilasciate altre *patch* a seguirsi.

La Blizzard ha da poco annunciato, il 21 ottobre 2011, che verrà prodotta anche una quarta espansione, denominata *Mists of Pandaria*, la cui uscita sembra prevista nel 2012.[6]

### 2.2.2 Successo del Prodotto

Il successo di *World of Warcraft* è stato un fenomeno che nemmeno la casa produttrice, la Blizzard, si aspettava, infatti, come ammesso anche da loro: "in un meeting in Blizzard sul possibile mercato di *World of WarCraft* speravamo di vendere un milione di copie".[7] A tuttora, il gioco ha venduto oltre 15 milioni di copie nel mondo, e sui server ufficiali si possono contare circa 10,3 milioni di iscritti attivi, mentre è più difficile stimare il quantitativo di giocatori presenti sui così detti *shard*, o server privati, dotati di emulatori dei server ufficiali.

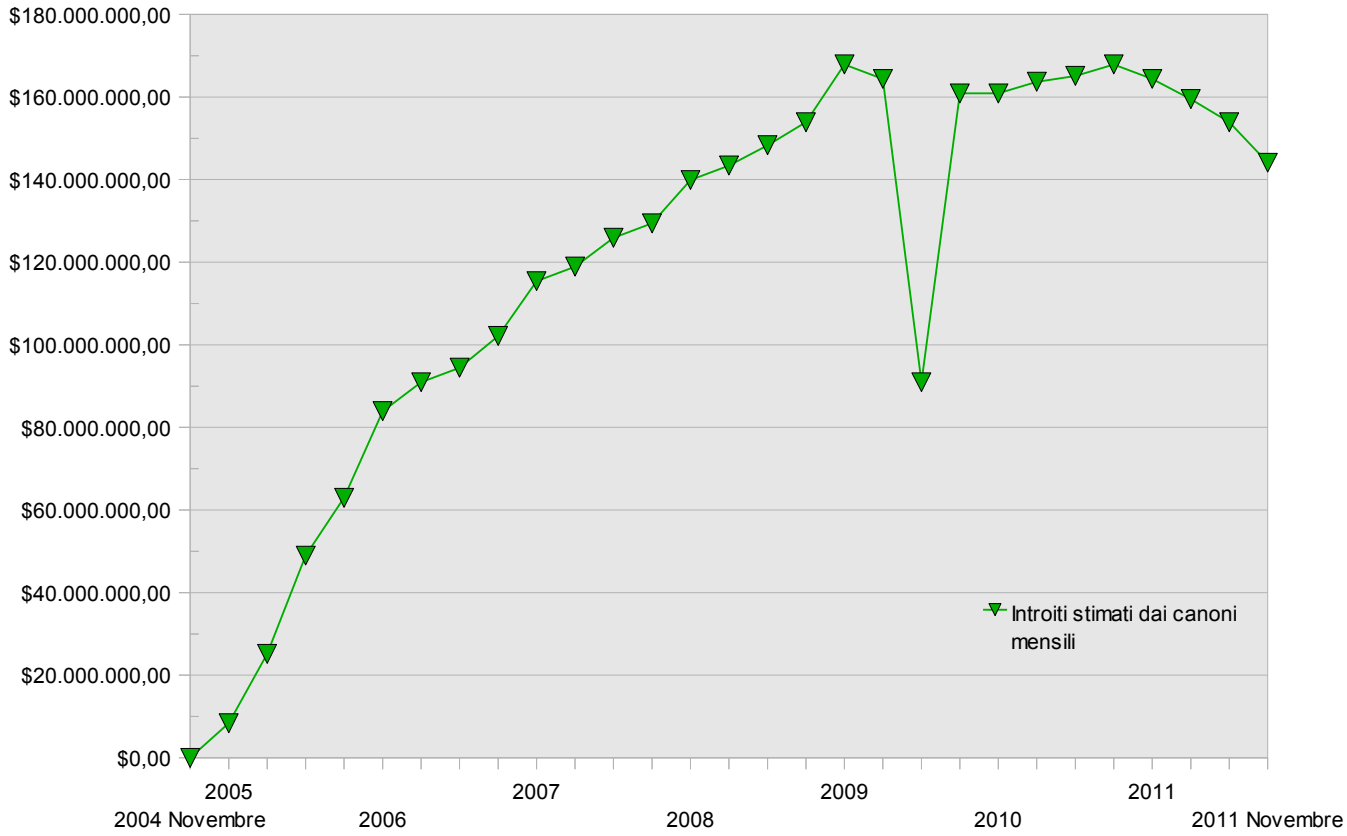


Grafico 2.2.2: introiti stimati dei canoni mensili di *WoW*

Ma ancora più grande è stato il suo successo economico, se si pensa che ogni utente versa un canone mensile medio di circa 13,99$ [8], i cui valori globali sono stimati e visibili nel *Grafico 2.2.2* (il picco negativo della metà del 2009 è dovuto ad un momentaneo blocco del mercato asiatico). Con un tali ricavi la Blizzard è riuscita non solo coprire i costi di realizzazione e di mantenimento e funzionamento dell'infrastruttura che sta alla base di *WoW*, così come lo sviluppo di nuove patch ed espansioni di *WoW*, ma anche nuovi videogiochi e altri progetti secondari. A questi ricavi vanno poi aggiunti quelle provenienti dalla vendita del gioco, delle sue espansioni e dei cofanetti speciali (gioco base + espansioni + guide/libri su Warcraft). Ancora, sempre legati direttamente al gioco, ci sono da considerare i ricavi derivanti dalle reward a pagamento presenti nel *Blizzard Store[9]*. Oltre a tutto questo è presente anche un enorme mercato basato sul merchandasing di World of Warcraft, che genera ulteriori ricavi per la Blizzard.

Sull'onda del successo di *WoW*, sono nati siti come *WoWWiki[10]*, che inizialmente raccoglieva solo spiegazioni sui contenuti del gioco, ma che poi si è espansa fino ad ospitare la più grande raccolta online sulla storia di Warcraft, e *WoWHead[11]*, che mantiene un database di tutto ciò che si può incontrare nel gioco, con informazioni abbastanza accurate dei contenuti a volte corredati da commenti dei giocatori stessi, e molti altri siti simili.

*World of Warcraft* negli anni ha ricevuto numerosi premi e riconoscimenti, sia dalla critica che dai giocatori, divenendo così uno dei giochi più conosciuti ed apprezzati di tutti i tempi.

Grazie al suo successo si è anche ampliata la narrativa su Warcraft in generale, con la pubblicazione di romanzi e fumetti, ma ha anche invaso il mondo dei giochi, uscendo dallo schermo del computer per diventare gioco da tavolo e gioco di carte collezionabili, diventando così una grossa fonte di introiti per la casa produttrice, la Blizzard Entertaiment, e creando una cultura di gioco sempre più radicata.

Complice del successo di questo gioco è anche la sua grande capacità di dare dipendenza e assuefazione per il forte grado di coinvolgimento nel gioco, tanto che tale dipendenza è considerata al pari di una dipendenza da droghe, ed è possibile trovare in rete una vasta gamma di siti che ne trattano i sintomi e le terapie per la “disintossicazione da Warcraft”.

# Capitolo 3 Emulatori dei Server di WoW

## *3.1 Il Fenomeno dell'Emulazione*

Il fenomeno dell'emulazione dei server di World of Warcraft ha iniziato a diffondersi poco tempo dopo il rilascio del gioco, come conseguenza della sua fama e del suo successo, motivati in parte dal suo costo di utilizzo.

I primi server così detti "privati", non gestiti dalla Blizzard, inizialmente puntavano sull'offrire ai giocatori la stessa esperienza dei server Blizzard, in termini di aggiornamenti e contenuti, con l'aggiunta di alcune caratteristiche custom per incentivare a giocare su di essi. Per questi motivi la casa produttrice e gestrice di *WoW* mal tollerava questa concorrenza, tanto da intentare nei loro confronti azioni persuasive e  a volte legali per porre termine alle loro attività di emulazione, poiché era basata sul codice proprietario della Blizzard Entertaiment e, a volte, su dati dei beta-test in corso.[12] [13]

Tuttavia, vista anche la grande quantità di questi server e la loro qualità di gioco, spesso molto inferiore a quella dei server ufficiali, successivamente la Blizzard ha deciso di tollerare l'esistenza dei server privati, purché essi non usino codice derivato da quello della Blizzard e rimangano sempre a versioni precedenti a quelle disponibili sui server ufficiali. [12] Probabilmente sotto a questa tolleranza si cela il fatto che molti giocatori inizino la propria carriera su questi server, che sono sostanzialmente gratuiti, e qui si appassionino al gioco, ma per le ovvie mancanze ed i difetti dell'emulatore, qualunque esso sia, poi decidano di iscriversi e giocare sui server Blizzard, dove la dipendenza dal gioco tende a diventare più forte per le succitate differenze tra l'emulatore e l'originale.

Di seguito focalizzeremo l'attenzione su tre dei maggiori progetti di emulazione Open Source ancora attivi dei server di *World of Warcraft*, descrivendo in che modo essi trattino l'AI che controlla e gestisce i personaggi presenti nel gioco.

## *3.2 ArcEmu*

Il primo emulatore che vediamo è *ArcEmu[14]*, nato nel giugno del 2008 dopo la chiusura del progetto *AcentEmu*.

*ArcEmu* è un server di *World of Warcraft* open source, è x86 compatibile (inclusi Pentium e Athlon), amd64 compatibile, e PowerPC compatibile. E' attualmente sviluppato e mantenuto da un team di

individui. Il server è scritto in C++ e offre networking avanzato, prestazioni, un anti-hack di sicurezza ingame e caratteristiche uniche che ad oggi mancano ancora in altri pacchetti software per server.[14]

*Arcemu* utilizza il linguaggio C++ per fornire prestazioni veloci. Al centro di *Arcemu* vi è un nucleo multithread, che sfrutta al meglio l'hardware di elaborazione eseguendo compiti in parallelo, garantendo nel contempo che i controlli più stringenti siano collocati nella memoria di protezione inter-thread per offrire maggiore stabilità.[14]

Attualmente *ArcEmu* offre supporto server per la versione 3.3.5a di *Wolrd of Warcraft: Wrath of the Lich King*, quindi non offre alcun contenuto in anteprima rispetto ai server ufficiali.

Utilizza come sistema di distribuzione del proprio codice e delle proprie revisioni il sistema di controllo di versione Subversion (SVN), tuttavia non distribuisce alcun file di progetto o soluzione per la compilazione, che vanno creati in locale utilizzando il programma CMake[15].

Utilizza anche tre database MySQL, che vengono distribuiti assieme al codice, ognuno dei quali con una diversa funzione:

- ***character***: contiene tutte le strutture dati riguardanti i personaggi creati dai giocatori sul server, quindi inizialmente questo database sarà vuoto e verrà riempito man mano che vengono creati personaggi dai processi del server;
- ***logon***: contiene tutte le strutture dati riguardanti gli account dei giocatori presenti sul server, anche questo database inizialmente sarà vuoto e verrà riempito dai processi del server;
- ***world***: contiene tutte le strutture ed i dati riguardanti l'ambiente di gioco, questo è l'unico database che è necessario popolare prima di poter avviare i processi del server, attualmente l'unico progetto di world database già compatibile con *ArcEmu* è WhyDB[16], ma sul sito del progetto è disponibile un programma per convertire world database compatibili con il progetto *MaNGOS* in modo che diventino compatibili con *ArcEmu*.

Come nella maggior parte degli emulatori di *WoW*, l'AI degli NPC è gestibile in due modi principali: da database o da codice, in ambedue i casi fa uso di script.

### 3.2.1 Scriptting da DB

Questo tipo di scripting basa il suo funzionamento sui valori contenuti in un set ti tabelle presenti nel database ***world*** del server (vedi *Figura 3.2.*1 e *Appendice-A*):

- `creature_names`: contiene parte dei dati di default riguardanti la nomenclatura di un NPC, finché esso non comparirà nel gioco;
- `creature_proto`: contiene parte dei dati di default riguardanti le caratteristiche di combattimento di un NPC, assieme alla tabella `creature_names` fornisce la descrizione generale di un NPC;
- `creature_spawns`: contiene i dati di dove e come gli NPC appariranno normalmente all'interno del gioco;
- `creature_waypoints`: contiene i dati delle locazioni per cui un NPC deve passare durante un evento;
- `creature_formations`: contiene le informazioni sulle disposizioni che gli NPC devono assumere in certi gruppi;
- `npc_mostersay`: contiene le frasi che le creature fornite di AI possono dire in risposta a particolari eventi;
- `ai_agents`: contiene le informazioni sulle abilità che un NPC può utilizzare, e costituisce quindi la vera AI di un NPC, a meno che non sia presente uno script da codice, che si sostituisce a questa tabella, in assenza di valori in questa tabella per un NPC esso eseguirà solo le funzioni base di attacco in mischia contro i personaggi ostili.

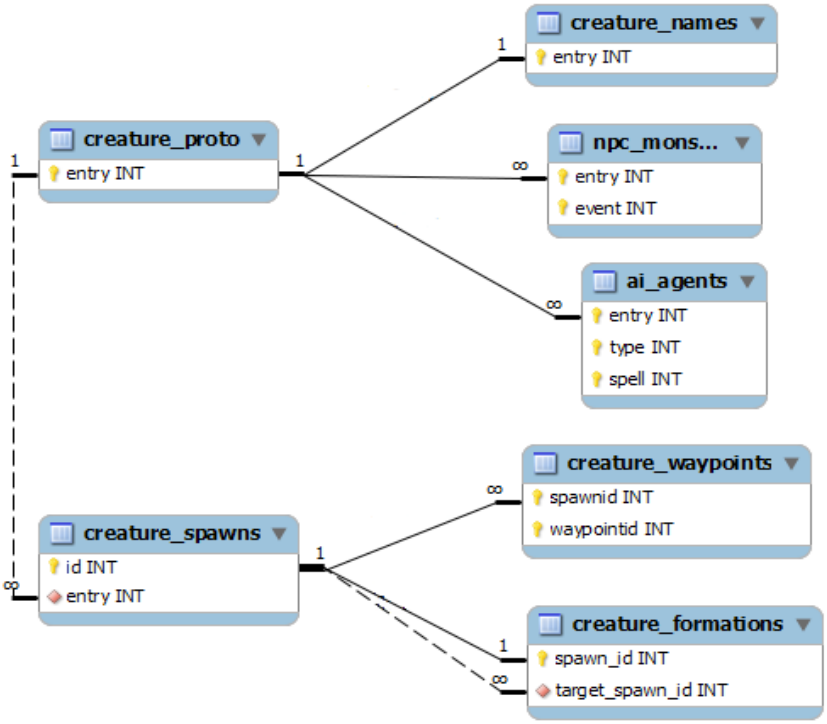


Figura 3.2.1: schema semplificato del database

Come si può vedere dalla struttura mostrata in *Figura 3.2.1* (e dagli script riportati in *Appendice-A*) delle due tabelle che maggiormente influiscono sull'AI di un NPC, `ai_agents` e `npc_monstersay`, questo tipo di approccio per la programmazione dell'AI è fortemente legato agli eventi specificati nel campo `event` di queste tabelle, che può assumere i valori riportati in *Tabella 3.2.1*, possiamo quindi dire che questo approccio è di tipo reattivo più che attivo, poiché un NPC dotato di questa AI non ha capacità di decisione attiva sui propri comportamenti, ma può solo rispondere a stimoli esterni, vediamo quindi una certa povertà nella consapevolezza che questo tipo di AI da all'NPC che la utilizza.

| event | Significato |
|---|---|
| 0 | Enter in Combat |
| 1 | Leave Combat |
| 2 | Damage Taken |
| 3 | Target Cast Spell |
| 4 | Target Parried |
| 5 | Target Dodgied |
| 6 | Target Blocked |
| 7 | Target Crit Hit |
| 8 | Target Died |
| 9 | Target Died |
| 10 | Unit Parried |
| 11 | Unit Dodgied |
| 12 | Unit Blocked |
| 13 | Unit Crit Hit |
| 14 | Unit Died |
| 15 | Assist Target Died |
| 16 | Follow Owner |

Tabella 3.2.1: event

Tuttavia, nonostante i difetti di questo approccio, esso viene utilizzato per lo scripting di eventi semplici, che non richiedono per l'appunto grande capacità ai NPC che vi prendono parte; inoltre non essendo implementata da codice ha il vantaggio che per rendere effettive le modifiche apportate a di questo tipo di AI è sufficiente riavviare il server che caricherà i valori aggiornati di queste tabelle.

L'associazione di questo tipo di AI avviene tramite vincoli referenziali sul database, grazie al campo `entry` o `creature_id` che consente di collegare tra loro le varie tabelle.

### 3.2.2 Scripting da codice

Per lo scripting da codice *ArcEmu* usa simultaneamente due meccanismi differenti e due linguaggi differenti.

Il primo si basa su uno scripting da codice di programmazione, scritto in C++, integrato nel codice del server. In questa modalità si utilizza il meccanismo dell'ereditarietà, ogni AI definita in questo modo è derivata in qualche modo dalla classe `SERVER_DECL CreatureAIScript` (vedi *Appendice-A8*), definita nel file di libreria ScriptMgr.h, da cui eredita una serie di metodi generali utilizzati per gestire il comportamento di un NPC.

La funzione più importante che questa classe fornisce è `AIUpdate()`, che viene chiamata in automatico dal server a seconda del parametro fornito alla funzione `RegisterAIUpdateEvent(uint32 frequency)` la cui frequenza può poi essere modificata invocando la funzione `ModifyAIUpdateEvent(uint32 newfrequency)`, questa funzione assieme alle altre funzioni di risposta agli eventi che compongono la classe consentono una grande flessibilità di questo tipo di AI, che così dispone di una miglior sistema di controllo, che permette di implementare comportamenti più raffinati degli NPC.

L'associazione tra script e NPC in questo caso avviene direttamente da codice, grazie alla classe `SERVER_DECL ScriptMgr` (vedi *Appendice-A9*) che assegna ad una `entry` del world database il relativo script.

Il vantaggio di questo tipo di script è la sua rapidità di esecuzione, essendo codice sorgente compilato. Tuttavia esso può causare facilmente errori nel processo server e portarlo a situazioni di crash, per cui va tenuta una maggiore attenzione durante la realizzazione dello script. Inoltre per rendere effettive le modifiche di uno script così scritto è necessario ricompilare il suo progetto per ricreare la libreria dinamica che viene richiamata dal processo server.

Il secondo meccanismo si basa su uno scripting engine, denominato *ALE* (Arcemu Lua Engine), questo è ancora lo script engine di default di *ArcEmu*, che è poi stato modificato e migliorato dando vita al *LuaBridge*, che come suggerisce il nome consentono di utilizzare script scritti nel linguaggio semi-interpretato Lua[17].

Gli script per l'AI scritti in Lua utilizzano sostanzialmente un sistema ad eventi, divisi in categorie a seconda di cosa li genera ed in che contesto, si distinguono così in *Quest Events*, *Creature Events*, *GameObject Events*, *Gossip Events*, *Instance Hooks* e *Server Hooks*, tuttavia tra questi eventi, nella fattispecie nei *Creature Events* e nei *GameObject Events*, è presente anche l'evento di AIUpdate, inoltre la serie di eventi che fornisce è del tutto simile alla suite di funzioni presenti nella classe `SERVER_DECL CreatureAIScript` e quindi ne può benissimo fare le veci.

Il *LuaBridge* aggiunge una maggiore chiarezza al codice di scripting utilizzato da *ALE*:

- distingue chiaramente i vari tipi di unità che si possono gestire e le relative funzioni;
- migliora l'incapsulamento degli script, impedendo che script completamente scorrelati possano interagire sulle loro variabili;
- implementa un sistema di caricamento condizionato degli script, in maniera che vengano caricati solo una volta quando necessario;
- consente di utilizzare le classi definite nel codice sorgente scritto in C++.

Tuttavia questo sistema funziona soltanto su piattaforme Windows 2008 Server / Vista o più recenti, il che ne riduce fortemente la sua portabilità e per questo continua ad esistere il sistema *ALE*.

Anche in questo caso l'associazione tra script e NPC avviene direttamente nel codice, mediante richiami di funzioni del Lua Engine.

Questo tipo di script è abbastanza veloce nell'esecuzione, semplice da scrivere e da gestire, ed il loro caricamento avviene tutto a carico del Lua Engine. Tuttavia, come per il metodo precedente richiede un alto grado di attenzione nella stesura del codice, per evitare di generare situazioni di crash nel processo server. In più richiede che ogni singolo comportamento definito da una funzione di risposta ad un evento venga associata con un sistema apposito nel codice dello script, per cui, se si definisce una funzione di risposta ma non la si associa questa non sarà poi utilizzata dall'NPC a cui viene assegnato lo script.

## *3.3 MaNGOS*

Il secondo emulatore che prendiamo in considerazione è *MaNGOS*, acronimo di Massively Network Game Object Server, nato nell'agosto del 2005 dopo la chiusura del progetto *WOWD* per problemi legali con la Blizzard. E' quindi uno dei progetti ancora attivi con più storia sulle sue spalle, inoltre fu anche il primo progetto server di *WoW* scritto in C++ ad essere open source.

Il progetto *MaNGOS* è una suite server completa per *World of Warcraft*, che include server per l'autenticazione, aggiornamento del client, world content e battleground. Altre caratteristiche includono strumenti per la costruzione e lo sviluppo dei contenuti di gioco. E' compatibile con la versione 3.3.5a del client.[18]

*MaNGOS* è un progetto educativo. Ciò significa che il suo scopo primario è di imparare ed insegnare il più possibile sullo sviluppo di un progetto in C++ su larga scala.[18]

Lo scripting e lo sviluppo dell'AI per *MaNGOS* sono affidati al progetto *ScriptDev2*, mentre per lo sviluppo e l'aggiornamento del world database si fa riferimento ai progetti *UDB* e *YTDB*.[18]

Inoltre *MaNGOS* mantiene attivo il supporto per le vecchie versioni di gioco, attraverso due branch:

- *MaNGOS*-one che offre supporto per la versione 2.4.3
- *MaNGOS*-zero che offre supporto per la versione 1.12.1

Inizialmente MaNGOS utilizzava come sistema di distribuzione del proprio codice e delle proprie revisioni il sistema di controllo di versione Subversion (SVN), ma dopo la scissione del progetto TrinityCore, è passato al sistema di revisione *git* ed all'hosting sul sito *github[19]*, dove attualmente risiede ed è visibile tutto il suo codice.

*MaNGOS* utilizza come DBMS MySQL o PostgreSQL, i dati risiedono su quattro database:

- ***characters***: contiene tutte le strutture dati riguardanti i personaggi creati dai giocatori sul

server, inizialmente questo database sarà vuoto e verrà riempito man mano che vengono creati personaggi dai processi del server;

- ***realmd***: contiene tutte le strutture dati riguardanti gli account dei giocatori presenti sul server, anche questo database inizialmente sarà vuoto e verrà riempito dai processi del server;
- ***mangos***: contiene tutte le strutture ed i dati riguardanti l'ambiente di gioco, questo database deve essere necessariamente popolato prima di poter avviare i processi del server. Le istanze per popolare tale atabase vengono fornite dai progetti *UDB* e *YTDB*;
- ***scriptdev2***: contiene tutte le strutture dati che gli script possono richiamare direttamente durante la loro esecuzione, anche questo database deve essere popolato per il corretto funzionamento degli script scritti in codice, i suoi dati sono forniti direttamente dal progetto *ScriptDev2*.

Come altri emulatori di *WoW*, l'AI degli NPC può essere gestita sia reperendo i dati dal database, tramite un sistema chiamato *ACID*, che da codice.

### 3.3.1 Scripting da DB: *ACID*

Il sistema *ACID* per il suo funzionamento fa riferimento ad un set di tabelle presenti nel database ***mangos*** (vedi *Figura 3.3.1* e *Appendice-B*):

- `creature_template`: contiene tutte le informazioni di default che definiscono uno specifico NPC, particolarmente importanti sono i suoi campi `AIName` e `ScriptName` che specificano se l'AI utilizzata proviene, rispettivamente, da database o da codice, solitamente la AI fornita da codice sovrascrive l'eventuale AI definita da database;
- `creature`: contiene i dati di dove e come gli NPC appariranno normalmente all'interno del gioco;
- `creature_(template)_addon`: contiengono informazioni addizionali su alcune caratteristiche che può possedere un NPC al momento della sua comparsa nel gioco, la tabella `creature_addon` fa riferimento ad uno specifico NPC presente nel gioco, quindi alla tabella `creature`, mentre la tabella `creature_template_addons` fa riferimento alla entry dell'NPC, quindi alla tabella `creature_template`;
- `creature_movement(_template)`: contengono informazioni sui percorsi che gli NPC devono percorrere, la tabella `creature_movement` fa riferimento al singolo spawn,

quindi alla tabella `creature`, mentre la tabella `creature_movement_template` fa riferimento alla entry dell'NPC, quindi alla tabella `creature_template`;

- `creature_movement_scripts`: specifica ulteriormente l'azione da eseguire una volta raggiunto il waypoint specificato in `creature_movement` o `creature_movement_template`;
- `creature_ai_scripts`: contiene la vera e propria AI di un NPC, in cui sia stato impostato il campo `AIName = `EventAI`` nella relativa entry della tabella `creature_template`, con le informazioni riguardanti le abilità che l'NPC può utilizzare e le condizioni per utilizzarle;
- `creature_ai_summons`: contiene informazioni di supporto per le azioni di evocazione indicate nella tabella `creature_ai_scripts`;
- `creature_ai_texts`: contiene informazioni sulle frasi che un NPC può pronunciare, fornisce quindi supporto per le azioni di dialogo indicate nella tabella `creature_ai_scripts`.

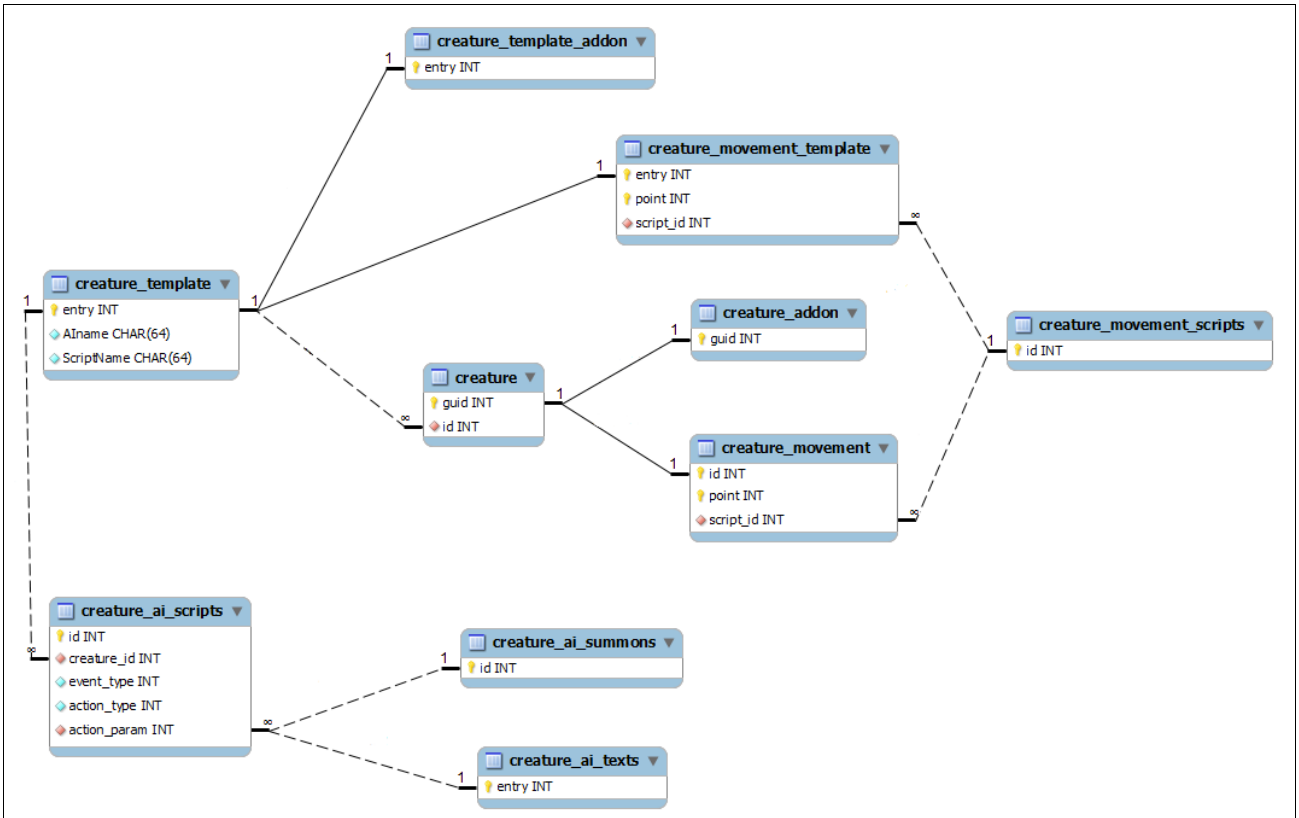

Figura 3.3.1: schema semplificato del database

Come è possibile intuire dal fatto che questo tipo di AI sia chiamata “EventAI” ed osservando la

struttura (vedi *Appendice-B6*) della tabella `creature_ai_scripts`, questo tipo di AI è legato ad una programmazione ad eventi, tuttavia grazie ad una vasta gamma di eventi messi a disposizione dei programmatori è possibile creare atteggiamenti attivi con questo tipo di AI mediante l'uso appropriato degli eventi di tipo `EVENT_T_TIMER` e `EVENT_T_TIMER_OOC`.

| event_type | Denominazione |
|---|---|
| 0 | EVENT_T_TIMER |
| 1 | EVENT_T_TIMER_OOC |
| 2 | EVENT_T_HP |
| 3 | EVENT_T_MANA |
| 4 | EVENT_T_AGGRO |
| 5 | EVENT_T_KILL |
| 6 | EVENT_T_DEATH |
| 7 | EVENT_T_EVADE |
| 8 | EVENT_T_SPELLHIT |
| 9 | EVENT_T_RANGE |
| 10 | EVENT_T_OOC_LOS |
| 11 | EVENT_T_SPAWNED |
| 12 | EVENT_T_TARGET_HP |
| 13 | EVENT_T_TARGET_CASTING |
| 14 | EVENT_T_FRIENDLY_HP |
| 15 | EVENT_T_FRIENDLY_CC |
| 16 | EVENT_T_MISSING_BUFF |
| 17 | EVENT_T_SUMMONED_UNIT |
| 18 | EVENT_T_TARGET_MANA |
| 19 | |
| 20 | |
| 21 | EVENT_T_REACHED_HOME |
| 22 | EVENT_T_RECIEVED_EMOTE |
| 23 | EVENT_T_BUFFED |
| 24 | EVENT_T_TARGET_BUFFED |
| 25 | EVENT_T_SUMMONED_JUST_DIED |
| 26 | EVENT_T_SUMMONE_JUST_DESPAWN |

Tabella 3.3.1: event_type

Come vediamo riportato in *Tabella 3.3.1*, la gamma di eventi utilizzati da questo tipo di AI è davvero ampia e le consente di poter interagire in varie situazioni di gioco in maniera abbastanza soddisfacente, nonostante basi il suo funzionamento esclusivamente su degli eventi.

Tuttavia, nonostante la sua buona capacità espressiva, non è esente da difetti e problemi di natura tecnica.

In primo luogo, fatica a sostenere condizioni molto complesse che possono presentarsi durante un combattimento, poiché non aggiunge alcuna vera variabile, se non i timer, al set di variabili che definisce l'NPC.

Ancora, è in grado di osservare e reagire solo ai comportamenti del proprio bersaglio, di alcuni altri NPC "amichevoli" vicini ad essa e la sorte degli NPC che essa stessa evoca, ma non è in grado di percepire in alcun modo i comportamenti delle altre entità che possono esserci.

Inoltre risulta essere abbastanza difficile da utilizzare, proprio per la sua capacità espressiva elevata che contempla numerose combinazioni dei valori dei campi e spesso un polimorfismo di alcuni campi che assumono una funzione diversa in base al tipo di evento e dall'azione che si vogliono usare.

Abbiamo detto in precedenza che questo tipo di AI, il cui comportamento è descritto nel database, viene considerata quando il campo `AIName` della tabella `creature_template` è impostato a `` `EventAI` ``, vediamo gli altri valori ammessi per tale campo, tralasciando la stringa vuota:

- `` `NullAI` ``: AI completamente vuota, un NPC con questa AI non esegue alcuna azione/reazione;

- `AggressorAI`: un NPC con questa AI aggredisce ogni avversario che entra nella sua zona di minaccia;
- `ReactorAI`: un NPC con questa AI aggredisce soltanto se viene attaccato per primo;
- `PetAI`: un NPC con questa AI si comporta come un assistente/compagno di un altro personaggio, solitamente controllato da un giocatore, a cui è legato;
- `GuardAI`: un NPC con questa AI ha un comportamento intermedio tra la `AggressorAI` e la `ReactorAI`;
- `TotemAI`: un NPC con questa AI può usare l'abilità specificata nel campo `spell1` della tabella `creature_template`, per il resto ha il medesimo comportamento della `NullAI`.

### 3.3.2 Scripting da codice

Per lo scripting da codice *MaNGOS* utilizza un dispositivo sviluppato dal progetto *ScriptDev2*, che fornisce una libreria dinamica da cui vengono richiamati i vari script che costituiscono le varie istanze della classe `Script` (vedi *Appendice-B8*) definita nel file di libreria ScriptMgr.h, che altro non è che una raccolta di puntatori a funzioni che definiscono lo script, tra cui quella che restituisce l'AI utilizzata, ed una stringa che ne costituisce l'identificatore.

Come abbiamo accennato prima, l'associazione tra script ed NPC avviene nel database, inserendo la stringa opportuna nel campo `ScriptName` della tabella `creature_template`, saranno poi il lo script manager e lo script loader che si faranno carico dell'effettiva ricerca e caricamento dello script a tempo di esecuzione.

Tutte le AI definite all'interno di *ScriptDev2* ereditano la propria struttura originaria dalla classe `MANGOS_DLL_SPEC CreatureAI` (vedi *Appendice-B9*) definita in *MaNGOS* nel file di libreria CreatureAI.h, questo per consentire un corretto interfacciamento tra i due progetti; inoltre, è doveroso notare che la classe `CreatureAI` possiede tutti i metodi di risposta necessari ai comportamenti di un'`EventAI` definita nel database, infatti la classe utilizzata dal codice per gestire tali AI, `MANGOS_DLL_SPEC CreatureEventAI` è derivata direttamente da essa. In questo modo per il nucleo (*core*) del server è possibile gestire tutti i tipi di AI attraverso un'unica interfaccia di base.

Tuttavia per differenziarsi dal core di *MaNGOS*, *ScriptDev2* utilizza un'ulteriore classe come progenitrice delle AI, `MANGOS_DLL_DECL ScriptedAI` (vedi *Appendice-B10*)che deriva direttamente da `CreatureAI`. Tale classe, oltre a fornire un'implementazione dei metodi della classe progenitrice,

aggiunge una serie di funzioni e variabili alla classe originaria, che le consentono così di gestire meglio i suoi eventi e dando all'NPC una maggiore consapevolezza sull'ambiente circostante.

Gli script dell'AI creati in *ScriptDev2* possiedono tutte le funzioni reattive disponibili anche per l'`EventAI`, ma ad esse affiancano una nuova serie di funzioni e la possibilità di creare condizioni molto complesse per gestire il comportamento dell'NPC.

La funzione principalmente usata per gestire il comportamento dell'NPC è `UpdateAI(const uint32 uiDiff)`, che viene periodicamente richiamata dal core del server che le fornisce come parametro il tempo trascorso dall'ultima chiamata di tale funzione espresso in millisecondi, rendendo così facile implementare all'interno di tale funzione dei controlli basati su dei timer, le altre funzioni a disposizione dell'AI invece, fungono principalmente da supporto a questa funzione attivando o disattivando alcune variabili booleane e/o timer in risposta ai relativi stimoli. Altra funzione di essenziale importanza per questo genere di AI è la funzione `Reset()`, che ha il compito di riportare l'NPC in uno stato di default dopo un combattimento, per questo quindi viene anche usata all'interno del costruttore per inizializzare alcune variabili, e che deve essere necessariamente implementata per poter istanziare classi derivate da `ScriptedAI`, poiché è l'unico membro di tale classe a non possedere una chiamata di default.

Come abbiamo detto in precedenza, esiste un database, detto appunto ***scriptdev2***, da cui gli script definiti in *ScriptDev2* possono attingere alcuni dati tramite delle funzioni globali del progetto. Le tabelle utilizzate di tale database sono:

- `script_texts` e `custom_texts`: che contengono informazioni sulle frasi che un NPC può pronunciare, a cui si può far accesso tramite le funzioni `DoScriptText` e `DoOrSimulateScriptTextForMap`, entrambe le tabelle hanno la medesima struttura e differiscono soltanto per il range di entry, sono quindi l'analogo della tabella `creature_ai_texts` contenuta nel database ***mangos***;
- `script_waypoint`: che contiene informazioni sui percorsi che gli NPC devono percorrere, utilizzata solo dalle AI derivate dalla classe `npc_escortAI`, definita in *ScriptDev2*, è quindi l'analoga delle tabelle `creature_movement(_template)` contenute nel database ***mangos***;

L'uso di queste tabelle è fatto per alleggerire il codice degli script, evitando così di dover caricare tutte le relative strutture ogniqualvolta viene caricato lo script. Anche se così facendo si perde un po' in termini di prestazioni specifiche per l'accesso alle strutture, si riesce a guadagnare in termini di prestazioni generali grazie al risparmio di memoria allocata che l'operazione comporterebbe se tali

dati fossero già presenti nel codice di ogni script.

Le considerazioni che abbiamo detto in precedenza a proposito dello scripting da codice per *ArcEmu*, sono valide anche per quanto riguarda *MaNGOS-ScriptDev2*. *ScriptDev2* rispetto ad *ArcEmu* ha il grande vantaggio di non dover ricorrere direttamente a chiamate a sistema per conoscere il tempo trascorso, poiché il server si preoccupa direttamente di fornirgli tale informazione, e ciò facilita molto la stesura del codice degli script.

Va infine fatto notare che il sistema di associazione degli script, siano essi implementati tramite *ACID* o *ScriptDev2*, utilizzato da *MaNGOS* fornisce agli sviluppatori un mezzo molto efficiente per risalire allo script una volta nota la entry dell'NPC, grazie appunto alle informazioni contenute in `creature_template`, il che risulta molto utile sia in fase di debug che in fase di sviluppo.

## *3.4 TrinityCore*

Il terzo emulatore che prendiamo in considerazione è *TrinityCore*, nato nell'estate del 2008 a partire dal progetto *MaNGOS* per un contrasto sulla politica di sviluppo di tale progetto, ritenuto troppo lento e poco performante, e per la sua frammentarietà, cioè la sua divisione tra database, core server e script. Come i precedenti progetti è anch'esso open source.

*TrinityCore* è un framework per MMORPG costruito in C++, che usa un back-end di MySQL per i contenuti del database.[20]

Nonostante una delle promesse dei suoi fondatori fosse di utilizzare da subito *git* come sistema di revisione, per differenziarsi da *MaNGOS* che allora usava ancora Subversion (SVN), è invece stato utilizzato prima il vecchio sistema SVN, poi il sistema Mercurial, e solo a fine 2010 il progetto è finalmente passato a *git*, approdando su *github* dove è visibile il suo codice attuale.

*TrinityCore* offre supporto server per la versione 3.3.5a di *World of Warcraft* (il nome specifico dell'emulatore riguardante i contenuti 3.x sarebbe *TrinityCore2*, tuttavia sono usati come sinonimi).

I dati di *TrinityCore* sono memorizzati su tre database:

- ***characters***: contiene tutte le strutture dati riguardanti i personaggi creati dai giocatori sul server, quindi inizialmente questo database sarà vuoto e verrà riempito man mano che vengono creati personaggi dai processi del server;
- ***world***: contiene tutte le strutture ed i dati riguardanti l'ambiente di gioco, questo database

deve essere necessariamente popolato prima di poter avviare i processi del server, i cui dati sono forniti direttamente da *TrinityCore*, e tale database è detto *TDB* (*Trinity DataBase*);

- ***auth***: contiene tutte le strutture dati riguardanti gli account dei giocatori presenti sul server, anche questo database inizialmente sarà vuoto e verrà riempito dai processi del server;

Come gli altri emulatori di *WoW*, anche *TrinityCore* consente di sviluppare le AI sia da database che da codice.

### 3.4.1 Scripting da DB

Essendo un derivato del progetto *MaNGOS*, anche *TrinityCore* ha al suo interno il sistema *ACID* per lo scripting da database, su cui non ci soffermeremo nuovamente.

Tuttavia ha introdotto un nuovo tipo di AI, che a conti fatti dovrebbe sostituire la `EventAI` di *ACID*, detta `SmartAI`. Questo tipo di AI basa il suo funzionamento sulle seguenti tabelle presenti nel database ***world*** (vedi *Figura 3.4.1* e *Appendice-C*):

- `creature_template`: quasi identica all'analoga di *MaNGOS*, e con le medesime funzioni;
- `creature`: anch'essa molto simile all'analoga di *MaNGOS*, e con le medesime funzioni;

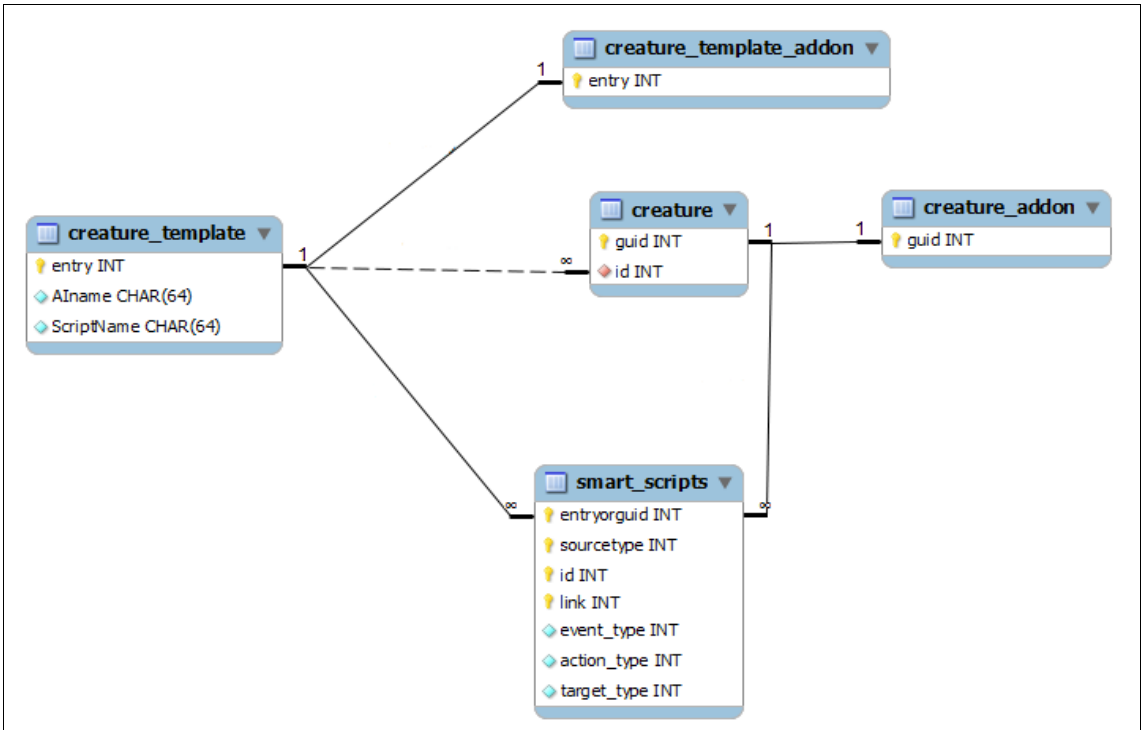


Figura 3.4.1: schema semplificato del database, focalizzato su smart_scripts

- `creature_(template)_addon`: anch'esse quasi identiche alle analoghe di *MaNGOS*, e con le medesime funzioni;
- `smart_scripts`: contiene i dati relativi alla vera e propria AI che verrà utilizzata da un NPC, in cui è stato impostato il campo `AIName = `SmartAI`` nella relativa entry della tabella `creature_template`, con le informazioni riguardanti le abilità che l'NPC può utilizzare e le condizioni per utilizzarle.

E' facile notare che l'uso che viene fatto della tabella `smart_scripts` è sostanzialmente identico all'uso che veniva fatto in *ACID* della tabella `creature_ai_scripts`. Ma questo è dovuto al fatto che tale tabella è stata concepita per sostituirsi al vecchio sistema dell' `EventAI` e creare un sistema di scripting da database che fosse quasi equivalente allo scripting da codice.

Una delle prime differenze possiamo vederla nell'uso del campo `entryorguid`, che consente di assegnare l'AI sia ad una `entry` in `creature_template` sia ad una `guid` in `creature`, a seconda del segno utilizzato, in caso di conflitto prevale l'AI assegnata alla `guid` .[21]

Altra differenza sta nella gamma di `event_type` disponibili, che è incrementata da 25 a 68, anche se al momento non tutti i tipi di evento son supportati.[21]

Un'altra differenza è l'uso di un solo campo azione, invece di tre, che però è maggiormente descritto sia in termini di varietà di tipi, 101 tipi contro i 43 dell'`EventAI`, sia grazie all'utilizzo di sei parametri, invece di tre.[21] [22]

Ancora, è dato un campo esplicito per il bersaglio dell'azione, che può essere specificato tra 25 possibili tipi, ed ulteriormente specificato mediante tre parametri ed una coordinata spaziale.

Tutto questo rende la `SmartAI` uno strumento molto flessibile, espressivo ed utile in termini di programmazione di AI, poiché in grado di descrivere il comportamento di un NPC quasi come da codice di programmazione.

Tuttavia questi pregi hanno un notevole prezzo in termini di manutenzione e realizzazione, legato proprio alla sua discendenza concettuale dall'`EventAI`. Come essa infatti mantiene alcuni campi con nomi abbastanza generali e ne demanda il significato a valori contenuti in altri campi, questo causava già problemi di comprensione nell'`EventAI` con un numero più limitato di tipi di eventi e azioni e dove il bersaglio di un'azione era specificato come un suo parametro, nella `SmartAI`, dove invece è presente una maggior varietà in termini di eventi e azioni e dove il bersaglio è un campo esplicito con i suoi relativi parametri, è quindi più facile che si crei confusione e si comprometta il corretto funzionamento dell'AI.

Ciononostante risulta un ottimo metodo di programmazione dell'AI, ma solo per un programmatore molto esperto ed abituato a districarsi nelle sue combinazioni di valori.

### 3.4.2 Scripting da codice

Per lo scripting da codice *TrinityCore* utilizza un dispositivo derivato da *ScriptDev2*, noto come *TrinityScript*, che è integrato nel core del server, per questo solitamente ci si riferisce ad esse sempre con il nome di *TrinityCore*.

Come *ScriptDev2*, anche *TrinityScript* fornisce una libreria dinamica da cui vengono richiamati i vari script, gli script degli NPC in questa libreria costituiscono le istanze di classi derivate dalla classe `CreatureScript` definita nel file di libreria ScriptMgr.h, che altro non è che una raccolta di puntatori a funzioni che definiscono lo script, tra cui quella che restituisce l'AI utilizzata, ed una stringa che ne costituisce l'identificatore.

Anche in *TrinityScript*, l'associazione tra script ed NPC avviene nel database, inserendo la stringa opportuna nel campo `ScriptName` della tabella `creature_template`, saranno poi il lo script manager e lo script loader che si faranno carico dell'effettiva ricerca e caricamento dello script a tempo di esecuzione.

All'interno di tali classi derivate è solitamente inserita la classe che costituisce l'AI vera e propria, che a sua volta è una classe derivata dalla classe `ScriptedAI`, definita nel file di libreria ScriptedCreature.h, a sua volta derivata dalla classe `CreatureAI`, definita nel file di libreria CreatureAI.h, a sua volta derivata dalla classe astratta `UnitAI`, definita nel file di libreria UnitAI.h.

In questo modo un'AI ha disposizione un gran numero di funzioni a cui attingere per sviluppare i propri comportamenti, proprio come avviene per le AI sviluppate in *ScriptDev2*, da cui *TrinityScript* è derivato.

Anche qui quindi la funzione di maggior rilievo per l'AI è `UpdateAI(const uint32 diff)`, che viene sempre richiamata dal server che le fornisce come parametro il tempo trascorso dall'ultima chiamata espresso in millisecondi, all'interno della quale si sviluppano tutte le condizioni, più o meno complesse, che determinano il comportamento effettivo dell'NPC, mentre la maggior parte delle altre funzioni hanno semplicemente il compito di supportare tale funzione attivando o disattivando alcune variabili booleane e/o timer in risposta ai relativi stimoli. Al contrario di *ScriptDev2*, in *TrinityScript* la funzione `Reset()` assume minor importanza, che qui ha una chiamata di default la quale non esegue alcun operazione.

Anche gli script definiti in *TrinityScript*, proprio come quelli di *ScriptDev2*, utilizzano il database per conservare alcune informazioni, in questo caso è il database ***world*** ad essere utlilizzato. Le tabelle utilizzate sono:

- `script_texts` e `custom_texts`: che contengono informazioni sulle frasi che un NPC può pronunciare, a cui si può far accesso tramite la funzione `DoScriptText`, entrambe le tabelle hanno la medesima struttura e differiscono soltanto per il range di entry;
- `script_waypoint`: che contiene informazioni sui percorsi che gli NPC devono percorrere, utilizzata solo dalle AI derivate dalla classe `npc_escortAI`, definita in *TrinityScript*;

Questa funzionalità è sostanzialmente ereditata da *ScriptDev2*, e viene mantenuta per i medesimi motivi, così come la struttura di tali tabelle.

Come in *ArcEmu* e *MaNGOS*, anche qui lo scripting da codice si rivela essere lo strumento migliore per generare un'AI, poiché offre una maggiore gamma di metodi, che possono essere anche riscritti caso per caso, per dare all'NPC una migliore consapevolezza di se e dell'ambiente che lo circonda.

Essendo *TrinityScript* derivato da *ScriptDev2*, condivide con esso tutti i vantaggi precedentemente descritti nei confronti di *ArcEmu*. Nel confronto diretto, invece, possiamo dire che si trovano abbastanza differenze tra i due progetti: per prima cosa notiamo che la gerarchia di classi impiegata da Trinity differenzia ulteriormente le AI, che ora hanno come capostipite la classe `UnitAI`, e gli script, che sono differenziati a seconda dell'entità a cui verranno associati; inoltre le classi adibite al funzionamento dell'AI hanno a disposizione un maggior numero di funzioni e variabili di default da utilizzare per gestire i propri comportamenti, alcuni dei quali non erano ben gestiti in *ScriptDev2*. Queste differenze rendono in un certo senso *TrinityScript* migliore di *ScriptDev2*, per la miglio capacità espressiva dell'AI e la maggior specializzazione degli script, tuttavia va fatto notare che alcune delle funzioni implementate dall'AI di TrinityScript altro non sono che chiamate a funzioni del membro `Creature* me` dell'AI, che rappresenta l'NPC a cui è assegnata l'AI, sempre riguardo a tale membro, notiamo che viene ridefinito in tutta la gerarchia, diventando addirittura un membro pubblico nella classe `ScriptedAI`! Per questo motivo non si può definire questo progetto migliore del suo progenitore.

Il maggior problema del progetto *TrinityCore*, in generale, è la consistenza del suo codice, infatti per rendere più rapido possibile il suo sviluppo *TrinityCore* ha inserito nel proprio codice sorgente

molti aggiornamenti provenienti da fonti diverse, che spesso ridefinivano funzioni già esistenti e funzionanti in punti diversi e con un formalismo diverso, trasformando così il suo codice in un così detto “spaghetti-code”, estremamente difficile da revisionare.

## *3.5 Confronto degli Emulatori*

Dopo aver visto come questi emulatori sviluppino le funzioni riguardanti l'AI, passiamo a vedere brevemente quali feature di *WoW* essi implementano (*Tabella 3.5*), e quanto queste siano accurate nei confronti dei server ufficiali della Blizzard.

| | *Blizzard** | *ArcEmu* | *MaNGOS* | *TrinityCore* |
|---|---|---|---|---|
| **Feature Originali:** | | | | |
| AI degli NPC | buona | discreta | più che discreta | più che discreta |
| Sistema delle Instance | buona | più che discreta | più che discreta | più che discreta |
| Dungeon Finder (1) | buona | mediocre | mediocre | discreta |
| Sistema PvP | buona | discreta | discreta | discreta |
| Abilità dei Personaggi | buona | mediocre | discreta | discreta |
| Sistema dei Veicoli (2) | buona | mediocre | mediocre | discreta |
| Sistema dei Pet (3) | buona | discreta | discreta | discreta |
| **Feature Aggiuntive:** | | | | |
| PlayerBot (4) | non presente | non presente | discreta | discreta |
| AuctionBot (5) | non presente | non presente | più che discreta | più che discreta |

Tabella 3.5: confronto feature
* riferito alla versione 3.3.5a

(1) meccanismo che permette di riunire automaticamente gruppi di giocatori per affrontare le sfide di un'Instance;

(2) sistema di oggetti utilizzabili come veicoli dai giocatori;

(3) sistema che consente ad alcune classi di personaggio di possedere un compagno (Pet)

semicontrollato dall'AI che combatte al proprio fianco;

(4) sistema che consente di richiamare altri personaggi dal proprio account di gioco, mettendoli sotto il controllo dell'AI, in maniera da poter utilizzarne contemporaneamente più di uno;

(5) sistema di popolamento della casa d'aste di oggetti nel gioco, con un NPC dedicato che può vendere/comprare vari oggetti all'asta.

Attualmente sono presenti migliaia di server privati, detti *shard*, la cui utenza media è solitamente tra i 100 ed i 400 utenti connessi contemporaneamente, mentre il numero complessivo di account attivi è più difficile da ricavare, poiché non tutti gli *shard* eliminano gli account inattivi e continuano a mantenerli nei contatori delle proprie statistiche o anche poiché molti *shard* non riportano pubblicamente tale statistica.

Nonostante il gran numero di *shard* attivi, risulta assai difficile stilare una statistica sull'impiego di questi emulatori da parte di essi, poiché l'informazione sulla natura dell'emulatore spesso non viene resa nota all'utenza, la quale è più interessata ad altri aspetti come la versione di gioco supportata, il tipo di server (PvE, PvP, RP, RPPvP), la probabilità con cui si posso trovare oggetti, ecc.

# Capitolo 4 Sviluppo di funzionalità di AI nel contesto *MaNGOS*

Gli emulatori visti nel *Capitolo 3* sono tutti gestiti all'interno di progetti open source. Il codice sorgente viene costantemente rilasciato, ed è l'intera comunità di volontari che contribuisce al miglioramento dell'emulatore, essi si ritrovano sui relativi forum per discutere il codice e le funzionalità dell'emulatore e prendono parte allo sviluppo del software.

Essendo progetti open source, chiunque può utilizzare, modificare e contribuire allo sviluppo di nuovo codice, ciascun progetto poi, a seconda delle sue politiche di aggiornamento, valuterà ed eventualmente inserirà tale codice nell'emulatore.

Durante lo svolgimento di questa tesi abbiamo avuto modo di sviluppare e modificare alcune funzionalità di AI nel contesto del progetto *Enturion's World RolePlay*[23] che utilizza come base il codice di *MaNGOS*, per cui abbiamo lavorato su del codice *MaNGOS*-compliant.

In questo capitolo mostreremo porzioni di codice di alcune funzionalità di AI che abbiamo implementato, spiegandone i comportamenti e le funzioni.

## *4.1 Scambio di Oggetti tra Giocatore e NPC*

Le prime funzionalità di AI a cui abbiamo lavorato e che mostriamo gestiscono due NPC coinvolti in una quest. L'obiettivo di questa quest è di prelevare degli oggetti dal primo NPC, che li consegna lanciandoli al giocatore quando questi è abbastanza vicino (*vedi Codice 4.1*), e consegnarli poi al

```
void MoveInLineOfSight (Unit* pWho)
{
    if (m_creature->GetDistance(pWho) > 10.0f)
        return;
    if (pWho->GetTypeId() != TYPEID_PLAYER)
        return;
    Player* pPlayer = m_pMap->GetPlayer(pWho->GetObjectGuid());
    if (!pPlayer)
        return;
    if (pPlayer->GetQuestStatus(QUEST_BACK_AGAIN_A) == QUEST_STATUS_INCOMPLETE)
    {
        if (pPlayer->HasItemCount(ITEM_PORTABLE_KEG, 1))
            return;
        ItemPosCountVec dest;
        uint8 msg = pPlayer->CanStoreNewItem(NULL_BAG, NULL_SLOT, dest, ITEM_PORTABLE_KEG, 1, NULL);
        if (msg == EQUIP_ERR_OK)
        {
            pPlayer->StoreNewItem(dest, ITEM_PORTABLE_KEG, 1, true);
            pPlayer->GetItemByEntry(ITEM_PORTABLE_KEG)->SendCreateUpdateToPlayer(pPlayer);
        }
    }
}
```

Codice 4.1: funzione per consegnare l'oggetto ad un giocatore che possiede la quest del primo NPC

secondo NPC, che li preleva al volo dal giocatore (vedi *Codice 4.2*).

```
void MoveInLineOfSight (Unit* pWho)
{
    if (m_creature->GetDistance(pWho) > 10.0f)
        return;
    if (pWho->GetTypeId() != TYPEID_PLAYER)
        return;
    Player* pPlayer = m_pMap->GetPlayer(pWho->GetObjectGuid());
    if (!pPlayer)
        return;
    if (pPlayer->GetQuestStatus(QUEST_BACK_AGAIN_A) == QUEST_STATUS_INCOMPLETE)
    {
        if (pPlayer->HasItemCount(ITEM_PORTABLE_KEG, 1))
        {
            pPlayer->DestroyItemCount(ITEM_PORTABLE_KEG, 1, true);
            pPlayer->KilledMonsterCredit(NPC_CREDIT_CREATURE,0);
        }
    }
}
```

Codice 4.2: funzione per la raccolta dell'oggetto dal giocatore del secondo NPC

## 4.2 NPC Invisibile con Timer

Questa funzionalità di AI, si occupa di gestire un NPC che si è deciso di tenere invisibile per un periodo di tempo dopo la sua comparsa, per poi tornare visibile (vedi *Codice 4.3*), in modo renderne difficile la ricerca.

```
void UpdateAI(const uint32 uiDiff)
{
    if (m_bNeedInvisible)
    {
        if (m_uiTime2Spawn <= uiDiff)
        {
            m_creature->RemoveFlag(UNIT_FIELD_FLAGS, UNIT_FLAG_NOT_SELECTABLE);
            m_creature->SetVisibility(VISIBILITY_ON);
            m_bNeedInvisible = false;
            m_bWasVisible = true;
        }
        else
            m_uiTime2Spawn -= uiDiff;
    }
}
```

Codice 4.3: funzione di aggiornamento dell'AI con timer per l'NPC

## 4.3 Sviluppo di un Boss NPC

Questa funzionalità di AI gestisce un NPC che costituisce un *boss* (cioè un avversario relativamente difficile da affrontare) di un evento. Questo boss si trova nel retrobottega di una taverna, intento ad osservare alcuni barili di birra, per poterlo affrontare i giocatori devono prima dialogarci ed affermare che la birra fatta da lui sia pessima (vedi *Codice 4.4*), dopo di che il *boss* chiamerà tre suoi assistenti (*Codice 4.4*) ed inizierà a discutere con essi (vedi *Codice 4.5*). Dopo che gli assistenti

```
bool GossipSelect_boss_coren_direbrew(Player* pPlayer, Creature* pCreature, uint32 uiSender, uint32 uiAction)
{
    if (pPlayer)
    {
        if (boss_coren_direbrewAI* pCorenAI = dynamic_cast<boss_coren_direbrewAI*>(pCreature->AI()))
        {
            if(pCorenAI->m_creature == pCreature)
            {
                Creature* temp;
                temp = pCreature->SummonCreature(MOB_ANTAGONIST,Coord[0][0],Coord[0][1],Coord[0][2],0,TEMPSUMMON_DEAD_DESPAWN,0);
                [...]
                temp = pCreature->SummonCreature(MOB_ANTAGONIST,Coord[1][0],Coord[1][1],Coord[1][2],0,TEMPSUMMON_DEAD_DESPAWN,0);
                [...]
                temp = pCreature->SummonCreature(MOB_ANTAGONIST,Coord[2][0],Coord[2][1],Coord[2][2],0,TEMPSUMMON_DEAD_DESPAWN,0);
                [...]
                pCorenAI->m_bEventStarted = true;
                pCorenAI->m_pInsulter = pPlayer;
            }
        }
    }
    return true;
}
```

Codice 4.4: funzione per gestire la comunicazione dei giocatori utilizzata dallo script

gli avranno confermato la qualità dei suoi prodotti e lo avranno incitato ad eliminare chi lo ha insultato, si scaglieranno tutti assieme sul gruppo di giocatori che hanno insultato il *boss* (vedi

```
if (m_bEventStarted)
{
    if (m_uiWalkTimer < uiDiff)
    {
        m_creature->SetSpeedRate(MOVE_WALK, 1.0f);
        switch(m_uiWalk)
        {
            case 0:
                m_creature->GetMotionMaster()->MovePoint(1,Coord[3][0],Coord[3][1],Coord[3][2]);
                m_uiWalk = 1;
                break;
            case 1:
                m_creature->GetMotionMaster()->MovePoint(2,Coord[4][0],Coord[4][1],Coord[4][2]);
                m_uiWalk = 0;
                break;
        }
        m_uiWalkTimer = 6000;
    }
    else m_uiWalkTimer -= uiDiff;
    if (m_uiSpeachTimer < uiDiff)
    {
        switch(m_uiSpeach)
        {
            case 0:
                [...]
            case 7:
                [...]
                m_creature->setFaction(16);
                m_creature->SetInCombatWithZone();
                m_creature->RemoveFlag(UNIT_FIELD_FLAGS, UNIT_FLAG_NON_ATTACKABLE);
                AttackStart(m_pInsulter);
                break;
        }
    }
    else m_uiSpeachTimer -= uiDiff;
}
```

Codice 4.5: parte della funzione UpdateAI che gestisce il discorso con gli assistenti

*Codice 4.6*), durante il combattimento poi interverranno anche altri due assistenti a fianco del *boss*.

```
if (!m_bAntagonistAttacked && m_creature->getVictim())
{
    Assault(m_Antagonist1Guid, m_creature->getFaction(), m_creature->getVictim());
    Assault(m_Antagonist2Guid, m_creature->getFaction(), m_creature->getVictim());
    Assault(m_Antagonist3Guid, m_creature->getFaction(), m_creature->getVictim());
    m_bAntagonistAttacked = true;
}
```
Codice 4.6: parte della funzione UpdateAI che gestisce l'assalto degli assistenti

## *4.4 NPC che Rappresenta un Fuoco*

Questa funzionalità di AI gestisce un NPC che rappresenta un fuoco, presente durante un evento in varie località del gioco. Durante tale evento al giocatore veniva chiesto, tramite un'apposita quest di spegnere questi fuochi con dei secchi d'acqua (vedi *Codice 4.7*), una volta colpito dall'acqua il

```
void SpellHit(Unit* pWho, const SpellEntry* pSpell)
{
    if (pWho->GetTypeId() != TYPEID_PLAYER)
        return;
    if (pSpell->Id == SPELL_BUCKET_LANDS)
        m_bBucketReceived = true;
}
```
Codice 4.7: funzione per gestire il ricevimento del secchio d'acqua

fuoco doveva stabilire in quale zona si trovasse, partendo da una serie di zone note in cui avrebbe potuto trovarsi (vedi *Codice 4.8*), dopo di che richiamava un NPC ad affrontare il giocatore che

```
if (m_bBucketReceived)
{
    uint8 uiIndex;
    bool bFind = false;
    for (uiIndex = 0; uiIndex < 6; uiIndex++)
    {
        if (m_uiMapId == asLocation[uiIndex].map && m_uiZoneId == asLocation[uiIndex].zone && m_uiAreaId ==
asLocation[uiIndex].area)
        {
            bFind = true;
            break;
         }
    }
    if (bFind)
    {
        Creature* pHorseman = m_creature->SummonCreature(NPC_BUNNY_HORSEMAN, asLocation[uiIndex].x, asLocation[uiIndex].y,
asLocation[uiIndex].z, asLocation[uiIndex].o, TEMPSUMMON_CORPSE_TIMED_DESPAWN, 5000);
        pHorseman->RemoveFlag(UNIT_FIELD_FLAGS, UNIT_FLAG_OOC_NOT_ATTACKABLE);
        pHorseman->RemoveFlag(UNIT_FIELD_FLAGS, UNIT_FLAG_PASSIVE);
        pHorseman->RemoveFlag(UNIT_FIELD_FLAGS, UNIT_FLAG_NON_ATTACKABLE);
        pHorseman->setFaction(14);
        pHorseman->SetPhaseMask(2,true);
    }
    m_creature->DealDamage(m_creature,m_creature->GetHealth(),NULL, DIRECT_DAMAGE,SPELL_SCHOOL_MASK_NORMAL,NULL,false);
    m_creature->ForcedDespawn();
}
```
Codice 4.8: parte della funzione UpdateAI che gestisce la reazione all'acqua ricevuta

l'avesse colpita ed infine si spegneva, sparendo (*Codice 4.8*).

Oltre alle funzionalità di AI fin qui mostrate abbiamo lavorato su numerose altre classi di codice che implementano funzionalità di AI, principalmente revisionandole e correggendo alcuni

comportamenti errati, ma anche implementando alcune caratteristiche comportamentali mancanti, che tuttavia non mostriamo per non dilungare ulteriormente questa tesi.

# Conclusioni

Il ruolo svolto dall'intelligenza artificiale nel mondo dei videogiochi è stato fondamentale per il suo sviluppo ed il suo successo sia culturale ed economico. A sua volta, il mondo dei videogiochi ha contribuito allo sviluppo ed al miglioramento dell'AI, fornendole una rappresentazione semplificata del mondo reale, in maniera simile a come i giochi “tradizionali” aiutano noi esseri umani nello sviluppare la nostra intelligenza.

Il videogioco *World of Warcraft*, realizzato dalla Blizzard Entertaiment, ed in generale l'intera saga di Warcraft, ben rappresenta quanto questo mercato sia vasto, sia per numero di utenti che per guadagni, e quale influenza abbia sulla cultura del nostro tempo. Nonostante *WoW* stia attualmente attraversando un periodo di crisi.

I vari progetti di emulazione di *World of Warcraft* consentono di vedere un metodo con cui si possa implementare e gestire un'AI all'interno di un videogioco, e di come si possano utilizzare vari metodi per raggiungere tale scopo: realizzando codice specifico per il comportamento di ciascuna funzionalità di AI presente o usando una classe generica che mappa i propri comportamenti nel database. Inoltre abbiamo visto che tali progetti utilizzano l'AI anche per creare feature aggiuntive, come ad esempio i PlayerBot, rispetto al gioco originale, per venire incontro al minor livello di utenza dei server che li utilizzano, distaccandosi così dalla mera emulazione di *WoW*, portando l'AI ad assistere direttamente i giocatori nel gioco.

Implementando funzionalità di AI all'interno dell'ambiente del progetto *Enturion's World RolePlay*, che utilizza le basi sviluppate dal progetto *MaNGOS*, abbiamo visto come sia possibile utilizzare gli NPC sia per implementare veri e propri personaggi con vari ruoli, sia come sia possibile utilizzare NPC per creare oggetti inanimati. Durante questa esperienza, inoltre, abbiamo potuto vedere come, nell'ambiente degli emulatori open source, capiti spesso che il codice inizialmente sviluppato su un emulatore venga in seguito adattato o usato come base per lo sviluppo della medesima feature per un altro emulatore.

# Bibliografia e Sitografia


1. Intelligenza Artificiale - Wikipedia [Internet]. [cited 2011 Oct 13];Available from: http://it.wikipedia.org/wiki/Intelligenza_artificiale

2. CHAPLIN H. Is That Just Some Game? No, It's a Cultural Artifact [Internet]. New York Times. 2007 Mar 12 [cited 2011 Oct 14];Available from: http://www.nytimes.com/2007/03/12/arts/design/12vide.html?ex=1331352000&en=380fc9bb18694da5&ei=5124&partner=permalink&exprod=permalink

3. Warcraft: Orcs and Humans - Wikipedia [Internet]. [cited 2011 Oct 13];Available from: http://it.wikipedia.org/wiki/Warcraft:_Orcs_and_Humans

4. Warcraft II: Tides of Darkness - Wikipedia [Internet]. [cited 2011 Oct 13];Available from: http://it.wikipedia.org/wiki/Warcraft_II:_Tides_of_Darkness

5. World of Warcraft - Wikipedia [Internet]. [cited 2011 Oct 13];Available from: http://it.wikipedia.org/wiki/World_of_Warcraft

6. Blizzard Entertaiment. Svelato World of Warcraft: Mists of Pandaria - Blizzcon [Internet]. [cited 2011 Oct 27];Available from: http://eu.battle.net/blizzcon/it/blog/3167714/Svelato_World_of_Warcraft_Mists_of_Pandaria-21_10_2011#blog

7. Grasso R. Roper: speravamo che WoW vendesse un milione di copie [Internet]. GameMag. 2007 Dicembre [cited 2011 Oct 31];Available from: http://www.gamemag.it/news/roper-speravamo-che-wow-vendesse-un-milione-di-copie_23464.html

8. Blizzard Support [Internet]. [cited 2011 Nov 14];Available from: http://us.blizzard.com/support/article.xml?locale=en_US&articleId=21450

9. Blizzard Store [Internet]. [cited 2011 Nov 23];Available from: http://us.blizzard.com/store/

10. World of Warcraft universe guide - WoWWiki [Internet]. [cited 2011 Nov 23];Available from: http://www.wowwiki.com/Portal:Main

11. Wowhead: Use it to find Mankrik's wife. [Internet]. [cited 2011 Nov 23];Available from: http://www.wowhead.com/

12. Emulated server [Internet]. WoWWiki. [cited 2011 Nov 2];Available from: http://www.wowwiki.com/Emulated_server

13. Blizzard legal targets private servers [Internet]. [cited 2011 Nov 2];Available from: http://wow.joystiq.com/2008/12/05/blizzard-legal-targets-private-servers/

14. Welcome to ArcEmu [Internet]. ArcEmu. [cited 2011 Oct 13];Available from: http://www.arcemu.org/about.php

15. CMake - Cross Platform Make [Internet]. [cited 2011 Nov 23];Available from: http://www.cmake.org/

16. WhyDB 2011 [Internet]. [cited 2011 Nov 23];Available from: http://www.whydb.org/

17. The Programming Language Lua [Internet]. [cited 2011 Nov 23];Available from: http://www.lua.org/

18. mangos foundation — the free, open source World of Warcraft server [Internet]. [cited 2011 Oct 13];Available from: http://getmangos.com/

19. Your Dashboard - GitHub [Internet]. [cited 2011 Nov 24];Available from: https://github.com/

20. TrinityCore MMoRPG Framework [Internet]. [cited 2011 Oct 13];Available from: http://trinitycore.github.com/

21. Smart scripts tc2 - TrinityCore Wiki [Internet]. [cited 2011 Oct 13];Available from: http://www.trinitycore.info/Smart_scripts_tc2

22. Creature ai scripts tc2 - TrinityCore Wiki [Internet]. [cited 2011 Nov 7];Available from: http://www.trinitycore.info/Creature_ai_scripts_tc2

23. eWorldRP's Profile - GitHub [Internet]. [cited 2011 Nov 28];Available from: https://github.com/eWorldRP

24. History of video games - Wikipedia, the free encyclopedia [Internet]. [cited 2011 Oct 14];Available from: http://en.wikipedia.org/wiki/History_of_video_games

25. Storia dei videogiochi - Wikipedia [Internet]. [cited 2011 Oct 18];Available from: http://it.wikipedia.org/wiki/Storia_dei_videogiochi

26. Blizzard Entertaiment. Manuale World of Warcraft - Manuale di Gioco. 2004.

27. Blizzard Entertaiment. Warcraft II - Battle.net edition - Manual. 1999.

28. Blizzard Entertaiment. Warcraft III - Manual. 2002.

29. Blizzard Entertaiment. Warcraft: Orcs & Humas - Manual. 1994.

30. MMOData.net: World of Warcraft in decline, but perhaps not for long ... [Internet]. [cited 2011 Nov 14];Available from: http://mmodata.blogspot.com/2010/02/world-of-warcraft-in-decline-but.html

31. Timeline (World of Warcraft) - WoWWiki - Your guide to the World of Warcraft [Internet]. [cited 2011 Oct 28];Available from: http://www.wowwiki.com/Timeline_(World_of_Warcraft)

# Appendice

# Appendice-A ArcEmu

## A1 Struttura tabella `creature_names` (*ArcEmu*):

```
CREATE TABLE `creature_names` (
  `entry` int(10) unsigned NOT NULL default '0',
  `name` varchar(100) NOT NULL,
  `subname` varchar(100) NOT NULL,
  `info_str` varchar(500) NOT NULL,
  `flags1` int(10) unsigned NOT NULL default '0',
  `type` int(10) unsigned NOT NULL default '0',
  `family` int(10) unsigned NOT NULL default '0',
  `rank` int(10) unsigned NOT NULL default '0',
  `killcredit1` int(10) unsigned NOT NULL default '0',
  `killcredit2` int(10) unsigned NOT NULL default '0',
  `male_displayid` int(30) unsigned NOT NULL default '0',
  `female_displayid` int(30) unsigned NOT NULL default '0',
  `male_displayid2` int(30) unsigned NOT NULL default '0',
  `female_displayid2` int(30) unsigned NOT NULL default '0',
  `unknown_float1` float NOT NULL default '1',
  `unknown_float2` float NOT NULL default '1',
  `leader` tinyint(3) unsigned NOT NULL default '0',
  `questitem1` int(11) unsigned NOT NULL default '0',
  `questitem2` int(11) unsigned NOT NULL default '0',
  `questitem3` int(11) unsigned NOT NULL default '0',
  `questitem4` int(11) unsigned NOT NULL default '0',
  `questitem5` int(11) unsigned NOT NULL default '0',
  `questitem6` int(11) unsigned NOT NULL default '0',
  `waypointid` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY  (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Creature System';
```

## A2 Struttura tabella `creature_proto` (*ArcEmu*):

```
CREATE TABLE `creature_proto` (
  `entry` int(30) unsigned NOT NULL DEFAULT '0',
  `minlevel` int(30) unsigned NOT NULL,
  `maxlevel` int(30) unsigned NOT NULL,
  `faction` int(30) unsigned NOT NULL DEFAULT '0',
  `minhealth` int(30) unsigned NOT NULL,
  `maxhealth` int(30) unsigned NOT NULL,
  `mana` int(30) unsigned NOT NULL DEFAULT '0',
  `scale` float NOT NULL DEFAULT '0',
  `npcflags` int(30) unsigned NOT NULL DEFAULT '0',
  `attacktime` int(30) unsigned NOT NULL DEFAULT '0',
  `attacktype` int(4) NOT NULL DEFAULT '0',
  `mindamage` float NOT NULL DEFAULT '0',
  `maxdamage` float NOT NULL DEFAULT '0',
  `can_ranged` int(11) unsigned NOT NULL DEFAULT '0',
  `rangedattacktime` int(30) unsigned NOT NULL DEFAULT '0',
  `rangedmindamage` float unsigned NOT NULL DEFAULT '0',
  `rangedmaxdamage` float unsigned NOT NULL DEFAULT '0',
  `respawntime` int(30) unsigned NOT NULL DEFAULT '0',
  `armor` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance1` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance2` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance3` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance4` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance5` int(30) unsigned NOT NULL DEFAULT '0',
  `resistance6` int(30) unsigned NOT NULL DEFAULT '0',
  `combat_reach` float NOT NULL DEFAULT '0',
  `bounding_radius` float NOT NULL DEFAULT '0',
  `auras` longtext NOT NULL,
  `boss` int(11) unsigned NOT NULL DEFAULT '0',
  `money` int(30) NOT NULL DEFAULT '0',
  `invisibility_type` int(30) unsigned NOT NULL,
  `walk_speed` float NOT NULL DEFAULT '2.5',
  `run_speed` float NOT NULL DEFAULT '8',
  `fly_speed` float NOT NULL DEFAULT '14',
  `extra_a9_flags` int(30) NOT NULL DEFAULT '0',
  `spell1` int(30) unsigned NOT NULL DEFAULT '0',
  `spell2` int(30) unsigned NOT NULL DEFAULT '0',
  `spell3` int(30) unsigned NOT NULL DEFAULT '0',
  `spell4` int(30) unsigned NOT NULL DEFAULT '0',
  `spell5` int(30) unsigned NOT NULL DEFAULT '0',
  `spell6` int(30) unsigned NOT NULL DEFAULT '0',
```

```
  `spell7` int(30) unsigned NOT NULL DEFAULT '0',
  `spell8` int(30) unsigned NOT NULL DEFAULT '0',
  `spell_flags` int(30) NOT NULL DEFAULT '0',
  `modImmunities` int(30) unsigned NOT NULL DEFAULT '0',
  `isTrainingDummy` int(10) unsigned NOT NULL DEFAULT '0',
  `guardtype` int(10) unsigned NOT NULL DEFAULT '0',
  `summonguard` int(10) unsigned NOT NULL DEFAULT '0',
  `spelldataid` INT UNSIGNED DEFAULT '0' NOT NULL,
  `vehicleid` int(30) unsigned NOT NULL DEFAULT '0',
  `rooted` int(30) unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Creature System';
```

## A3 Struttura tabella `creature_spawns` (*ArcEmu*):

```
CREATE TABLE `creature_spawns` (
  `id` int(11) unsigned NOT NULL auto_increment,
  `entry` int(30) NOT NULL,
  `map` int(30) NOT NULL,
  `position_x` float NOT NULL,
  `position_y` float NOT NULL,
  `position_z` float NOT NULL,
  `orientation` float NOT NULL,
  `movetype` int(30) NOT NULL default '0',
  `displayid` int(30) unsigned NOT NULL default '0',
  `faction` int(30) NOT NULL default '14',
  `flags` int(30) NOT NULL default '0',
  `bytes0` int(30) NOT NULL default '0',
  `bytes1` int(30) NOT NULL default '0',
  `bytes2` int(30) NOT NULL default '0',
  `emote_state` int(30) NOT NULL default '0',
  `npc_respawn_link` int(30) NOT NULL default '0',
  `channel_spell` int(30) NOT NULL default '0',
  `channel_target_sqlid` int(30) NOT NULL default '0',
  `channel_target_sqlid_creature` int(30) NOT NULL default '0',
  `standstate` int(10) NOT NULL default '0',
  `mountdisplayid` int(10) unsigned NOT NULL default '0',
  `slot1item` int(10) unsigned NOT NULL default '0',
  `slot2item` int(10) unsigned NOT NULL default '0',
  `slot3item` int(10) unsigned NOT NULL default '0',
  `CanFly` smallint(3) NOT NULL default '0',
  `phase` int(10) unsigned NOT NULL default '4294967295',
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COMMENT='Spawn System' AUTO_INCREMENT=6172830;
```

## A4 Struttura tabella `creature_waypoints` (*ArcEmu*):

```
CREATE TABLE `creature_waypoints` (
  `spawnid` int(10) unsigned NOT NULL default '0',
  `waypointid` int(10) unsigned NOT NULL default '0',
  `position_x` float NOT NULL default '0',
  `position_y` float NOT NULL default '0',
  `position_z` float NOT NULL default '0',
  `waittime` int(10) unsigned NOT NULL default '0',
  `flags` int(10) unsigned NOT NULL default '0',
  `forwardemoteoneshot` tinyint(3) unsigned NOT NULL default '0',
  `forwardemoteid` int(10) unsigned NOT NULL default '0',
  `backwardemoteoneshot` tinyint(3) unsigned NOT NULL default '0',
  `backwardemoteid` int(10) unsigned NOT NULL default '0',
  `forwardskinid` int(10) unsigned NOT NULL default '0',
  `backwardskinid` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY  (`spawnid`,`waypointid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Creature System';
```

## A5 Struttura tabella `creature_formations` (*ArcEmu*):

```
CREATE TABLE `creature_formations` (
  `spawn_id` int(10) unsigned NOT NULL default '0',
  `target_spawn_id` int(10) unsigned NOT NULL default '0',
  `follow_angle` float NOT NULL default '0',
  `follow_dist` float NOT NULL default '0',
  PRIMARY KEY  (`spawn_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Creature System';
```

## A6 Struttura tabella `npc_monstersay` (*ArcEmu*):

```
CREATE TABLE `npc_monstersay` (
  `entry` int(10) unsigned NOT NULL default '0',
  `event` int(10) unsigned NOT NULL default '0',
  `chance` float NOT NULL default '0',
```

```
  `language` int(10) unsigned NOT NULL default '0',
  `type` int(10) unsigned NOT NULL default '0',
  `monstername` longtext character set utf8 collate utf8_unicode_ci,
  `text0` longtext character set utf8 collate utf8_unicode_ci,
  `text1` longtext character set utf8 collate utf8_unicode_ci,
  `text2` longtext character set utf8 collate utf8_unicode_ci,
  `text3` longtext character set utf8 collate utf8_unicode_ci,
  `text4` longtext character set utf8 collate utf8_unicode_ci,
  PRIMARY KEY  (`entry`,`event`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='NPC System';
```

## A7 Struttura tabella `ai_agents` (*ArcEmu*):

```
CREATE TABLE `ai_agents` (
  `entry` int(11) unsigned NOT NULL default '0',
  `instance_mode` int(10) unsigned NOT NULL default '4',
  `type` smallint(5) unsigned NOT NULL default '0',
  `event` int(11) unsigned NOT NULL default '0',
  `chance` int(11) unsigned NOT NULL default '0',
  `maxcount` int(11) unsigned NOT NULL default '0',
  `spell` int(11) unsigned NOT NULL default '0',
  `spelltype` int(11) unsigned NOT NULL default '0',
  `targettype_overwrite` int(11) NOT NULL default '-1',
  `cooldown_overwrite` int(11) NOT NULL default '-1',
  `floatMisc1` float NOT NULL default '0',
  `Misc2` int(11) unsigned NOT NULL default '0',
  PRIMARY KEY  (`entry`,`type`,`spell`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='AI System';
```

## A8 Classe `SERVER_DECL CreatureAIScript` (*ArcEmu*):

```
class SERVER_DECL CreatureAIScript
{
        public:
                CreatureAIScript(Creature* creature);
                virtual ~CreatureAIScript();
                virtual void OnCombatStart(Unit* mTarget) {}
                virtual void OnCombatStop(Unit* mTarget) {}
                virtual void OnDamageTaken(Unit* mAttacker, uint32 fAmount) {}
                virtual void OnCastSpell(uint32 iSpellId) {}
                virtual void OnTargetParried(Unit* mTarget) {}
                virtual void OnTargetDodged(Unit* mTarget) {}
                virtual void OnTargetBlocked(Unit* mTarget, int32 iAmount) {}
                virtual void OnTargetCritHit(Unit* mTarget, int32 fAmount) {}
                virtual void OnTargetDied(Unit* mTarget) {}
                virtual void OnParried(Unit* mTarget) {}
                virtual void OnDodged(Unit* mTarget) {}
                virtual void OnBlocked(Unit* mTarget, int32 iAmount) {}
                virtual void OnCritHit(Unit* mTarget, int32 fAmount) {}
                virtual void OnHit(Unit* mTarget, float fAmount) {}
                virtual void OnDied(Unit* mKiller) {}
                virtual void OnAssistTargetDied(Unit* mAssistTarget) {}
                virtual void OnFear(Unit* mFeared, uint32 iSpellId) {}
                virtual void OnFlee(Unit* mFlee) {}
                virtual void OnCallForHelp() {}
                virtual void OnLoad() {}
                virtual void OnDespawn() {}
                virtual void OnReachWP(uint32 iWaypointId, bool bForwards) {}
                virtual void OnLootTaken(Player* pPlayer, ItemPrototype* pItemPrototype) {}
                virtual void AIUpdate() {}
                virtual void OnEmote(Player* pPlayer, EmoteType Emote) {}
                virtual void StringFunctionCall(int) {}
                void RegisterAIUpdateEvent(uint32 frequency);
                void ModifyAIUpdateEvent(uint32 newfrequency);
                void RemoveAIUpdateEvent();
                bool IsAlive();
                virtual void Destroy() { delete this; }
                Creature* GetUnit() { return _unit; }
                CreatureAIScript* GetLinkedCreature() { return linkedCreatureAI; }
                void SetLinkedCreature(CreatureAIScript* creatureAI);
                void LinkedCreatureDeleted();
                virtual void OnEnterVehicle(){}
                virtual void OnExitVehicle(){}
                virtual void OnFirstPassengerEntered(Unit *passenger){}
                virtual void OnVehicleFull(){}
                virtual void OnLastPassengerLeft(Unit *passenger){}
        protected:
```

```
		Creature* _unit;
	private:
		CreatureAIScript* linkedCreatureAI;
};
```

## A9 Classe `SERVER_DECL ScriptMgr` (*ArcEmu*):

```
class SERVER_DECL ScriptMgr : public Singleton<ScriptMgr>
{
	public:
		ScriptMgr();
		~ScriptMgr();
		friend class HookInterface;
		void LoadScripts();
		void UnloadScripts();
		void DumpUnimplementedSpells();
		CreatureAIScript* CreateAIScriptClassForEntry(Creature* pCreature);
		GameObjectAIScript* CreateAIScriptClassForGameObject(uint32 uEntryId, GameObject*
pGameObject);
		InstanceScript* CreateScriptClassForInstance(uint32 pMapId, MapMgr* pMapMgr);
		bool CallScriptedDummySpell(uint32 uSpellId, uint32 i, Spell* pSpell);
		bool HandleScriptedSpellEffect(uint32 SpellId, uint32 i, Spell* s);
		bool CallScriptedDummyAura(uint32 uSpellId, uint32 i, Aura* pAura, bool apply);
		bool CallScriptedItem(Item* pItem, Player* pPlayer);
		void register_creature_script(uint32 entry, exp_create_creature_ai callback);
		void register_gameobject_script(uint32 entry, exp_create_gameobject_ai callback);
		void register_dummy_aura(uint32 entry, exp_handle_dummy_aura callback);
		void register_dummy_spell(uint32 entry, exp_handle_dummy_spell callback);
		void register_script_effect(uint32 entry, exp_handle_script_effect callback);
		void register_instance_script(uint32 pMapId, exp_create_instance_ai pCallback);
		void register_gossip_script(uint32 entry, GossipScript* gs);
		void register_go_gossip_script(uint32 entry, GossipScript* gs);
		void register_hook(ServerHookEvents event, void* function_pointer);
		void register_item_gossip_script(uint32 entry, GossipScript* gs);
		void register_quest_script(uint32 entry, QuestScript* qs);
		void register_creature_gossip(uint32, Arcemu::Gossip::Script*);
		void register_item_gossip(uint32, Arcemu::Gossip::Script*);
		void register_go_gossip(uint32, Arcemu::Gossip::Script*);
		void register_creature_script(uint32* entries, exp_create_creature_ai callback);
		void register_gameobject_script(uint32* entries, exp_create_gameobject_ai callback);
		void register_dummy_aura(uint32* entries, exp_handle_dummy_aura callback);
		void register_dummy_spell(uint32* entries, exp_handle_dummy_spell callback);
		void register_script_effect(uint32* entries, exp_handle_script_effect callback);
		void ReloadScriptEngines();
		void UnloadScriptEngines();
		bool has_creature_script(uint32) const;
		bool has_gameobject_script(uint32) const;
		bool has_dummy_aura_script(uint32) const;
		bool has_dummy_spell_script(uint32) const;
		bool has_script_effect(uint32) const;
		bool has_instance_script(uint32) const;
		bool has_hook(ServerHookEvents, void*) const;
		bool has_quest_script(uint32) const;
		bool has_creature_gossip(uint32) const;
		bool has_item_gossip(uint32) const;
		bool has_go_gossip(uint32) const;
		Arcemu::Gossip::Script* get_creature_gossip(uint32) const;
		Arcemu::Gossip::Script* get_go_gossip(uint32) const;
		Arcemu::Gossip::Script* get_item_gossip(uint32) const;
		Arcemu::Gossip::Trainer trainerScript_;
		Arcemu::Gossip::SpiritHealer spirithealerScript_;
		Arcemu::Gossip::Banker bankerScript_;
		Arcemu::Gossip::Vendor vendorScript_;
		Arcemu::Gossip::ClassTrainer classtrainerScript_;
		Arcemu::Gossip::PetTrainer pettrainerScript_;
		Arcemu::Gossip::FlightMaster flightmasterScript_;
		Arcemu::Gossip::Auctioneer auctioneerScript_;
		Arcemu::Gossip::InnKeeper innkeeperScript_;
		Arcemu::Gossip::BattleMaster battlemasterScript_;
		Arcemu::Gossip::CharterGiver chartergiverScript_;
		Arcemu::Gossip::TabardDesigner tabardScript_;
		Arcemu::Gossip::StableMaster stablemasterScript_;
		Arcemu::Gossip::Generic genericScript_;
	protected:
		InstanceCreateMap mInstances;
```

```
                CreatureCreateMap _creatures;
                GameObjectCreateMap _gameobjects;
                HandleDummyAuraMap _auras;
                HandleDummySpellMap _spells;
                HandleScriptEffectMap SpellScriptEffects;
                DynamicLibraryMap dynamiclibs;
                ServerHookList _hooks[NUM_SERVER_HOOKS];
                CustomGossipScripts _customgossipscripts;
                QuestScripts _questscripts;
                GossipMap creaturegossip_, gogossip_, itemgossip_;
};
```

# Appendice-B *ManGOS-ScriptDev2*

## B1 Struttura tabella `creature_template` (*MaNGOS*):

```
CREATE TABLE `creature_template` (
  `entry` mediumint(8) unsigned NOT NULL default '0',
  `difficulty_entry_1` mediumint(8) unsigned NOT NULL default '0',
  `difficulty_entry_2` mediumint(8) unsigned NOT NULL default '0',
  `difficulty_entry_3` mediumint(8) unsigned NOT NULL default '0',
  `KillCredit1` int(11) unsigned NOT NULL default '0',
  `KillCredit2` int(11) unsigned NOT NULL default '0',
  `modelid_1` mediumint(8) unsigned NOT NULL default '0',
  `modelid_2` mediumint(8) unsigned NOT NULL default '0',
  `modelid_3` mediumint(8) unsigned NOT NULL default '0',
  `modelid_4` mediumint(8) unsigned NOT NULL default '0',
  `name` char(100) NOT NULL default '0',
  `subname` char(100) default NULL,
  `IconName` char(100) default NULL,
  `gossip_menu_id` mediumint(8) unsigned NOT NULL default '0',
  `minlevel` tinyint(3) unsigned NOT NULL default '1',
  `maxlevel` tinyint(3) unsigned NOT NULL default '1',
  `minhealth` int(10) unsigned NOT NULL default '0',
  `maxhealth` int(10) unsigned NOT NULL default '0',
  `minmana` int(10) unsigned NOT NULL default '0',
  `maxmana` int(10) unsigned NOT NULL default '0',
  `armor` mediumint(8) unsigned NOT NULL default '0',
  `faction_A` smallint(5) unsigned NOT NULL default '0',
  `faction_H` smallint(5) unsigned NOT NULL default '0',
  `npcflag` int(10) unsigned NOT NULL default '0',
  `speed_walk` float NOT NULL default '1' COMMENT 'Result of 2.5/2.5, most common value',
  `speed_run` float NOT NULL default '1.14286' COMMENT 'Result of 8.0/7.0, most common value',
  `scale` float NOT NULL default '1',
  `rank` tinyint(3) unsigned NOT NULL default '0',
  `mindmg` float NOT NULL default '0',
  `maxdmg` float NOT NULL default '0',
  `dmgschool` tinyint(4) NOT NULL default '0',
  `attackpower` int(10) unsigned NOT NULL default '0',
  `dmg_multiplier` float NOT NULL default '1',
  `baseattacktime` int(10) unsigned NOT NULL default '0',
  `rangeattacktime` int(10) unsigned NOT NULL default '0',
  `unit_class` tinyint(3) unsigned NOT NULL default '0',
  `unit_flags` int(10) unsigned NOT NULL default '0',
  `dynamicflags` int(10) unsigned NOT NULL default '0',
  `family` tinyint(4) NOT NULL default '0',
  `trainer_type` tinyint(4) NOT NULL default '0',
  `trainer_spell` mediumint(8) unsigned NOT NULL default '0',
  `trainer_class` tinyint(3) unsigned NOT NULL default '0',
  `trainer_race` tinyint(3) unsigned NOT NULL default '0',
  `minrangedmg` float NOT NULL default '0',
  `maxrangedmg` float NOT NULL default '0',
  `rangedattackpower` smallint(5) unsigned NOT NULL default '0',
  `type` tinyint(3) unsigned NOT NULL default '0',
  `type_flags` int(10) unsigned NOT NULL default '0',
  `lootid` mediumint(8) unsigned NOT NULL default '0',
  `pickpocketloot` mediumint(8) unsigned NOT NULL default '0',
  `skinloot` mediumint(8) unsigned NOT NULL default '0',
  `resistance1` smallint(5) NOT NULL default '0',
  `resistance2` smallint(5) NOT NULL default '0',
  `resistance3` smallint(5) NOT NULL default '0',
  `resistance4` smallint(5) NOT NULL default '0',
  `resistance5` smallint(5) NOT NULL default '0',
  `resistance6` smallint(5) NOT NULL default '0',
  `spell1` mediumint(8) unsigned NOT NULL default '0',
  `spell2` mediumint(8) unsigned NOT NULL default '0',
  `spell3` mediumint(8) unsigned NOT NULL default '0',
  `spell4` mediumint(8) unsigned NOT NULL default '0',
  `PetSpellDataId` mediumint(8) unsigned NOT NULL default '0',
  `mingold` mediumint(8) unsigned NOT NULL default '0',
  `maxgold` mediumint(8) unsigned NOT NULL default '0',
  `AIName` char(64) NOT NULL default '',
  `MovementType` tinyint(3) unsigned NOT NULL default '0',
  `InhabitType` tinyint(3) unsigned NOT NULL default '3',
  `unk16` float NOT NULL default '1',
  `unk17` float NOT NULL default '1',
  `RacialLeader` tinyint(3) unsigned NOT NULL default '0',
```

```
  `questItem1` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `questItem2` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `questItem3` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `questItem4` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `questItem5` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `questItem6` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `movementId` int(11) UNSIGNED DEFAULT '0' NOT NULL,
  `RegenHealth` tinyint(3) unsigned NOT NULL default '1',
  `equipment_id` mediumint(8) unsigned NOT NULL default '0',
  `trainer_id` mediumint(8) unsigned NOT NULL default '0',
  `vendor_id` mediumint(8) unsigned NOT NULL default '0',
  `mechanic_immune_mask` int(10) unsigned NOT NULL default '0',
  `flags_extra` int(10) unsigned NOT NULL default '0',
  `ScriptName` char(64) NOT NULL default '',
  PRIMARY KEY  (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Creature System';
```

## B2 Struttura tabella `creature` (*MaNGOS*):

```
CREATE TABLE `creature` (
  `guid` int(10) unsigned NOT NULL auto_increment COMMENT 'Global Unique Identifier',
  `id` mediumint(8) unsigned NOT NULL default '0' COMMENT 'Creature Identifier',
  `map` smallint(5) unsigned NOT NULL default '0' COMMENT 'Map Identifier',
  `spawnMask` tinyint(3) unsigned NOT NULL default '1',
  `phaseMask` smallint(5) unsigned NOT NULL default '1',
  `modelid` mediumint(8) unsigned NOT NULL default '0',
  `equipment_id` mediumint(9) NOT NULL default '0',
  `position_x` float NOT NULL default '0',
  `position_y` float NOT NULL default '0',
  `position_z` float NOT NULL default '0',
  `orientation` float NOT NULL default '0',
  `spawntimesecs` int(10) unsigned NOT NULL default '120',
  `spawndist` float NOT NULL default '5',
  `currentwaypoint` mediumint(8) unsigned NOT NULL default '0',
  `curhealth` int(10) unsigned NOT NULL default '1',
  `curmana` int(10) unsigned NOT NULL default '0',
  `DeathState` tinyint(3) unsigned NOT NULL default '0',
  `MovementType` tinyint(3) unsigned NOT NULL default '0',
  PRIMARY KEY  (`guid`),
  KEY `idx_map` (`map`),
  KEY `index_id` (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC COMMENT='Creature System';
```

## B3 Struttura tabelle `creature_template_addon` e `creature_addon` (*MaNGOS*):

```
CREATE TABLE `creature_addon` (
  `guid` int(10) unsigned NOT NULL default '0',
  `mount` mediumint(8) unsigned NOT NULL default '0',
  `bytes1` int(10) unsigned NOT NULL default '0',
  `b2_0_sheath` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `b2_1_pvp_state` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `emote` int(10) unsigned NOT NULL default '0',
  `moveflags` int(10) unsigned NOT NULL default '0',
  `auras` text,
  PRIMARY KEY  (`guid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE `creature_template_addon` (
  `entry` mediumint(8) unsigned NOT NULL default '0',
  `mount` mediumint(8) unsigned NOT NULL default '0',
  `bytes1` int(10) unsigned NOT NULL default '0',
  `b2_0_sheath` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `b2_1_pvp_state` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `emote` mediumint(8) unsigned NOT NULL default '0',
  `moveflags` int(10) unsigned NOT NULL default '0',
  `auras` text,
  PRIMARY KEY  (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

## B4 Struttura tabelle `creature_movement` e `creature_movement_template` (*MaNGOS*):

```
CREATE TABLE `creature_movement` (
  `id` int(10) unsigned NOT NULL COMMENT 'Creature GUID',
  `point` mediumint(8) unsigned NOT NULL default '0',
  `position_x` float NOT NULL default '0',
  `position_y` float NOT NULL default '0',
  `position_z` float NOT NULL default '0',
  `waittime` int(10) unsigned NOT NULL default '0',
  `script_id` mediumint(8) unsigned NOT NULL default '0',
  `textid1` int(11) NOT NULL default '0',
```

```
  `textid2` int(11) NOT NULL default '0',
  `textid3` int(11) NOT NULL default '0',
  `textid4` int(11) NOT NULL default '0',
  `textid5` int(11) NOT NULL default '0',
  `emote` mediumint(8) unsigned NOT NULL default '0',
  `spell` mediumint(8) unsigned NOT NULL default '0',
  `wpguid` int(11) NOT NULL default '0',
  `orientation` float NOT NULL default '0',
  `model1` mediumint(9) NOT NULL default '0',
  `model2` mediumint(9) NOT NULL default '0',
  PRIMARY KEY  (`id`,`point`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Creature System';

CREATE TABLE `creature_movement_template` (
  `entry` mediumint(8) unsigned NOT NULL COMMENT 'Creature entry',
  `point` mediumint(8) unsigned NOT NULL default '0',
  `position_x` float NOT NULL default '0',
  `position_y` float NOT NULL default '0',
  `position_z` float NOT NULL default '0',
  `waittime` int(10) unsigned NOT NULL default '0',
  `script_id` mediumint(8) unsigned NOT NULL default '0',
  `textid1` int(11) NOT NULL default '0',
  `textid2` int(11) NOT NULL default '0',
  `textid3` int(11) NOT NULL default '0',
  `textid4` int(11) NOT NULL default '0',
  `textid5` int(11) NOT NULL default '0',
  `emote` mediumint(8) unsigned NOT NULL default '0',
  `spell` mediumint(8) unsigned NOT NULL default '0',
  `wpguid` int(11) NOT NULL default '0',
  `orientation` float NOT NULL default '0',
  `model1` mediumint(9) NOT NULL default '0',
  `model2` mediumint(9) NOT NULL default '0',
  PRIMARY KEY  (`entry`,`point`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Creature waypoint system';
```

## B5 Struttura tabella `creature_movement_scripts` (MaNGOS):

```
CREATE TABLE `creature_movement_scripts` (
  `id` mediumint(8) unsigned NOT NULL default '0',
  `delay` int(10) unsigned NOT NULL default '0',
  `command` mediumint(8) unsigned NOT NULL default '0',
  `datalong` mediumint(8) unsigned NOT NULL default '0',
  `datalong2` int(10) unsigned NOT NULL default '0',
  `datalong3` int(10) unsigned NOT NULL default '0',
  `datalong4` int(10) unsigned NOT NULL default '0',
  `data_flags` tinyint(3) unsigned NOT NULL default '0',
  `dataint` int(11) NOT NULL default '0',
  `dataint2` int(11) NOT NULL default '0',
  `dataint3` int(11) NOT NULL default '0',
  `dataint4` int(11) NOT NULL default '0',
  `x` float NOT NULL default '0',
  `y` float NOT NULL default '0',
  `z` float NOT NULL default '0',
  `o` float NOT NULL default '0',
  `comments` varchar(255) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

## B6 Struttura tabella `creature_ai_scripts` (*MaNGOS*):

```
CREATE TABLE `creature_ai_scripts` (
  `id` int(11) unsigned NOT NULL COMMENT 'Identifier' AUTO_INCREMENT,
  `creature_id` int(11) unsigned NOT NULL default '0' COMMENT 'Creature Template Identifier',
  `event_type` tinyint(5) unsigned NOT NULL default '0' COMMENT 'Event Type',
  `event_inverse_phase_mask` int(11) signed NOT NULL default '0' COMMENT 'Mask which phases this
event will not trigger in',
  `event_chance` int(3) unsigned NOT NULL default '100',
  `event_flags` int(3) unsigned NOT NULL default '0',
  `event_param1` int(11) signed NOT NULL default '0',
  `event_param2` int(11) signed NOT NULL default '0',
  `event_param3` int(11) signed NOT NULL default '0',
  `event_param4` int(11) signed NOT NULL default '0',
  `action1_type` tinyint(5) unsigned NOT NULL default '0' COMMENT 'Action Type',
  `action1_param1` int(11) signed NOT NULL default '0',
  `action1_param2` int(11) signed NOT NULL default '0',
  `action1_param3` int(11) signed NOT NULL default '0',
  `action2_type` tinyint(5) unsigned NOT NULL default '0' COMMENT 'Action Type',
  `action2_param1` int(11) signed NOT NULL default '0',
  `action2_param2` int(11) signed NOT NULL default '0',
  `action2_param3` int(11) signed NOT NULL default '0',
```

```
  `action3_type` tinyint(5) unsigned NOT NULL default '0' COMMENT 'Action Type',
  `action3_param1` int(11) signed NOT NULL default '0',
  `action3_param2` int(11) signed NOT NULL default '0',
  `action3_param3` int(11) signed NOT NULL default '0',
  `comment` varchar(255) NOT NULL default '' COMMENT 'Event Comment',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='EventAI Scripts';
```

## B7 Struttura tabelle `creature_ai_summons` e `creature_ai_texts` (*MaNGOS*):

```
CREATE TABLE `creature_ai_summons` (
  `id` int(11) unsigned NOT NULL COMMENT 'Location Identifier' AUTO_INCREMENT,
  `position_x` float NOT NULL default '0',
  `position_y` float NOT NULL default '0',
  `position_z` float NOT NULL default '0',
  `orientation` float NOT NULL default '0',
  `spawntimesecs` int(11) unsigned NOT NULL default '120',
  `comment` varchar(255) NOT NULL default '' COMMENT 'Summon Comment',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='EventAI Summoning Locations';

CREATE TABLE `creature_ai_texts` (
  `entry` mediumint(8) NOT NULL,
  `content_default` text NOT NULL,
  `content_loc1` text,
  `content_loc2` text,
  `content_loc3` text,
  `content_loc4` text,
  `content_loc5` text,
  `content_loc6` text,
  `content_loc7` text,
  `content_loc8` text,
  `sound` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `type` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `language` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `emote` smallint(5) unsigned NOT NULL DEFAULT '0',
  `comment` text,
  PRIMARY KEY (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Script Texts';
```

## B8 Classe `Script` (ScriptDev2):

```
struct Script
{
    Script() :
        pGossipHello(NULL), pGossipHelloGO(NULL), pGossipSelect(NULL), pGossipSelectGO(NULL),
        pGossipSelectWithCode(NULL), pGossipSelectGOWithCode(NULL),
        pDialogStatusNPC(NULL), pDialogStatusGO(NULL),
        pQuestAcceptNPC(NULL), pQuestAcceptGO(NULL), pQuestAcceptItem(NULL),
        pQuestRewardedNPC(NULL), pQuestRewardedGO(NULL),
        pGOUse(NULL), pItemUse(NULL), pAreaTrigger(NULL), pProcessEventId(NULL),
        pEffectDummyNPC(NULL), pEffectDummyGO(NULL), pEffectDummyItem(NULL), pEffectAuraDummy(NULL),
        GetAI(NULL), GetInstanceData(NULL)
    {}
    std::string Name;
    bool (*pGossipHello             )(Player*, Creature*);
    bool (*pGossipHelloGO           )(Player*, GameObject*);
    bool (*pGossipSelect            )(Player*, Creature*, uint32, uint32);
    bool (*pGossipSelectGO          )(Player*, GameObject*, uint32, uint32);
    bool (*pGossipSelectWithCode    )(Player*, Creature*, uint32, uint32, const char*);
    bool (*pGossipSelectGOWithCode  )(Player*, GameObject*, uint32, uint32, const char*);
    uint32 (*pDialogStatusNPC       )(Player*, Creature*);
    uint32 (*pDialogStatusGO        )(Player*, GameObject*);
    bool (*pQuestAcceptNPC          )(Player*, Creature*, Quest const*);
    bool (*pQuestAcceptGO           )(Player*, GameObject*, Quest const*);
    bool (*pQuestAcceptItem         )(Player*, Item*, Quest const*);
    bool (*pQuestRewardedNPC        )(Player*, Creature*, Quest const*);
    bool (*pQuestRewardedGO         )(Player*, GameObject*, Quest const*);
    bool (*pGOUse                   )(Player*, GameObject*);
    bool (*pItemUse                 )(Player*, Item*, SpellCastTargets const&);
    bool (*pAreaTrigger             )(Player*, AreaTriggerEntry const*);
    bool (*pProcessEventId          )(uint32, Object*, Object*, bool);
    bool (*pEffectDummyNPC          )(Unit*, uint32, SpellEffectIndex, Creature*);
    bool (*pEffectDummyGO           )(Unit*, uint32, SpellEffectIndex, GameObject*);
    bool (*pEffectDummyItem         )(Unit*, uint32, SpellEffectIndex, Item*);
    bool (*pEffectAuraDummy         )(const Aura*, bool);
    CreatureAI* (*GetAI             )(Creature*);
    InstanceData* (*GetInstanceData )(Map*);
```

```
    void RegisterSelf(bool bReportError = true);
};
```

## B9 Classe `MANGOS_DLL_SPEC CreatureAI` (*MaNGOS*):

```
class MANGOS_DLL_SPEC CreatureAI
{
    public:
        explicit CreatureAI(Creature* creature) : m_creature(creature) {}
        virtual ~CreatureAI();
        virtual void GetAIInformation(ChatHandler& reader) {}
        virtual void MoveInLineOfSight(Unit* pWho) {}
        virtual void EnterCombat(Unit* pEnemy) {}
        virtual void EnterEvadeMode() {}
        virtual void JustReachedHome() {}
        virtual void DamageDeal(Unit* pDoneTo, uint32& uiDamage) {}
        virtual void DamageTaken(Unit* pDealer, uint32& uiDamage) {}
        virtual void JustDied(Unit* pKiller) {}
        virtual void CorpseRemoved(uint32& uiRespawnDelay) {}
        virtual void SummonedCreatureJustDied(Creature* pSummoned) {}
        virtual void KilledUnit(Unit* pVictim) {}
        virtual void OwnerKilledUnit(Unit* pVictim) {}
        virtual void JustSummoned(Creature* pSummoned) {}
        virtual void JustSummoned(GameObject* pGo) {}
        virtual void SummonedCreatureDespawn(Creature* pSummoned) {}
        virtual void SpellHit(Unit* pCaster, const SpellEntry* pSpell) {}
        virtual void SpellHitTarget(Unit* pTarget, const SpellEntry* pSpell) {}
        virtual void AttackedBy(Unit* pAttacker);
        virtual void JustRespawned() {}
        virtual void MovementInform(uint32 uiMovementType, uint32 uiData) {}         */
        virtual void SummonedMovementInform(Creature* pSummoned, uint32 uiMotionType, uint32 uiData) {}
        virtual void ReceiveEmote(Player* pPlayer, uint32 uiEmote) {}
        virtual void PassengerBoarded(Unit * /*who*/, int8 /*seatId*/, bool /*apply*/) {}
        virtual void AttackStart(Unit* pWho) {}
        virtual void UpdateAI(const uint32 uiDiff) {}
        virtual bool IsVisible(Unit* pWho) const { return false; }
        virtual bool canReachByRangeAttack(Unit*) { return false; }
        bool AttackByType(WeaponAttackType attType = BASE_ATTACK);
        bool DoMeleeAttackIfReady();
        CanCastResult CanCastSpell(Unit* pTarget, const SpellEntry* pSpell, bool isTriggered);
        CanCastResult DoCastSpellIfCan(Unit* pTarget, uint32 uiSpell, uint32 uiCastFlags = 0, ObjectGuid
OriginalCasterGuid = ObjectGuid());
        Creature* const m_creature;
};
```

## B10 Classe `MANGOS_DLL_DECL ScriptedAI` (*ScriptDev2*):

```
struct MANGOS_DLL_DECL ScriptedAI : public CreatureAI
{
    public:
        explicit ScriptedAI(Creature* pCreature);
        ~ScriptedAI();
        void GetAIInformation(ChatHandler& reader) override;
        void MoveInLineOfSight(Unit* pWho) override;
        void EnterCombat(Unit* pEnemy) override;
        void EnterEvadeMode() override;
        void JustReachedHome() override {}
        void DamageDeal(Unit* pDoneTo, uint32& uiDamage) override {}
        void DamageTaken(Unit* pDealer, uint32& uiDamage) override {}
        void JustDied(Unit* pKiller) override {}
        void CorpseRemoved(uint32& uiRespawnDelay) override {}
        void SummonedCreatureJustDied(Creature* pSummoned) {}
        void KilledUnit(Unit* pVictim) override {}
        void OwnerKilledUnit(Unit* pVictim) override {}
        void JustSummoned(Creature* pSummoned) override {}
        void JustSummoned(GameObject* pGo) override {}
        void SummonedCreatureDespawn(Creature* pSummoned) override {}
        void SpellHit(Unit* pCaster, const SpellEntry* pSpell) override {}
        void SpellHitTarget(Unit* pTarget, const SpellEntry* pSpell) override {}
        void AttackedBy(Unit* pAttacker) override { CreatureAI::AttackedBy(pAttacker); }
        void JustRespawned() override;
        void MovementInform(uint32 uiMovementType, uint32 uiData) override {}
        void SummonedMovementInform(Creature* pSummoned, uint32 uiMotionType, uint32 uiData) override {}
        void ReceiveEmote(Player* pPlayer, uint32 uiEmote) override {}
        void AttackStart(Unit* pWho) override;
        void UpdateAI(const uint32) override;
```

```
        bool IsVisible(Unit* pWho) const override;
        bool canReachByRangeAttack(Unit* pWho) override { return CreatureAI::canReachByRangeAttack(pWho); }
        virtual void Reset() = 0;
        virtual void Aggro(Unit*) {}
        void DoStartMovement(Unit* pVictim, float fDistance = 0, float fAngle = 0);
        void DoStartNoMovement(Unit* pVictim);
        void DoStopAttack();
        void DoCast(Unit* pTarget, uint32 uiSpellId, bool bTriggered = false);
        void DoCastSpell(Unit* pTarget, SpellEntry const* pSpellInfo, bool bTriggered = false);
        void DoPlaySoundToSet(WorldObject* pSource, uint32 uiSoundId);
        void DoResetThreat();
        void DoTeleportPlayer(Unit* pUnit, float fX, float fY, float fZ, float fO);
        Unit* DoSelectLowestHpFriendly(float fRange, uint32 uiMinHPDiff = 1);
        std::list<Creature*> DoFindFriendlyCC(float fRange);
        std::list<Creature*> DoFindFriendlyMissingBuff(float fRange, uint32 uiSpellId);
        Player* GetPlayerAtMinimumRange(float fMinimumRange);
        Creature* DoSpawnCreature(uint32 uiId, float fX, float fY, float fZ, float fAngle, uint32 uiType,
uint32 uiDespawntime);
        SpellEntry const* SelectSpell(Unit* pTarget, int32 uiSchool, int32 iMechanic, SelectTarget
selectTargets, uint32 uiPowerCostMin, uint32 uiPowerCostMax, float fRangeMin, float fRangeMax, SelectEffect
selectEffect);
        bool CanCast(Unit* pTarget, SpellEntry const* pSpell, bool bTriggered = false);
        void SetEquipmentSlots(bool bLoadDefault, int32 iMainHand = EQUIP_NO_CHANGE, int32 iOffHand =
EQUIP_NO_CHANGE, int32 iRanged = EQUIP_NO_CHANGE);
        void SetCombatMovement(bool bCombatMove);
        bool IsCombatMovement() { return m_bCombatMovement; }
        bool EnterEvadeIfOutOfCombatArea(const uint32 uiDiff);
        EventManager& Events();
    private:
        bool   m_bCombatMovement;
        uint32 m_uiEvadeCheckCooldown;
        EventManager* m_events;
};
```

## B11 Struttura tabelle `script_texts` e `custom_texts` (*ScriptDev2*):

```
CREATE TABLE `script_texts` (
  `entry` mediumint(8) NOT NULL,
  `content_default` text NOT NULL,
  `content_loc1` text,
  `content_loc2` text,
  `content_loc3` text,
  `content_loc4` text,
  `content_loc5` text,
  `content_loc6` text,
  `content_loc7` text,
  `content_loc8` text,
  `sound` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `type` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `language` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `emote` smallint(5) unsigned NOT NULL DEFAULT '0',
  `comment` text,
  PRIMARY KEY  (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Script Texts';

CREATE TABLE `custom_texts` (
  `entry` mediumint(8) NOT NULL,
  `content_default` text NOT NULL,
  `content_loc1` text,
  `content_loc2` text,
  `content_loc3` text,
  `content_loc4` text,
  `content_loc5` text,
  `content_loc6` text,
  `content_loc7` text,
  `content_loc8` text,
  `sound` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `type` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `language` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `emote` smallint(5) unsigned NOT NULL DEFAULT '0',
  `comment` text,
  PRIMARY KEY (`entry`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Custom Texts';
```

## B12 Struttura tabella `script_waypoint` (*ScriptDev2*):

```
CREATE TABLE script_waypoint (
  entry mediumint(8) unsigned NOT NULL DEFAULT '0' COMMENT 'creature_template entry',
```

```
  pointid mediumint(8) unsigned NOT NULL DEFAULT '0',
  location_x float NOT NULL DEFAULT '0',
  location_y float NOT NULL DEFAULT '0',
  location_z float NOT NULL DEFAULT '0',
  waittime int(10) unsigned NOT NULL DEFAULT '0' COMMENT 'waittime in millisecs',
  point_comment text,
  PRIMARY KEY (entry, pointid)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Script Creature waypoints';
```

# Appendice-C *TrinityCore*

## C1 Struttura tabella `creature_template` (*TrinityCore*):

```
CREATE TABLE `creature_template` (
  `entry` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `difficulty_entry_1` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `difficulty_entry_2` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `difficulty_entry_3` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `KillCredit1` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `KillCredit2` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `modelid1` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `modelid2` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `modelid3` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `modelid4` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `name` CHAR(100) NOT NULL DEFAULT '0',
  `subname` CHAR(100) DEFAULT NULL,
  `IconName` CHAR(100) DEFAULT NULL,
  `gossip_menu_id` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `minlevel` TINYINT(3) UNSIGNED NOT NULL DEFAULT '1',
  `maxlevel` TINYINT(3) UNSIGNED NOT NULL DEFAULT '1',
  `exp` SMALLINT(2) NOT NULL DEFAULT '0',
  `faction_A` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0',
  `faction_H` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0',
  `npcflag` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `speed_walk` FLOAT NOT NULL DEFAULT '1' COMMENT 'Result of 2.5/2.5, most common value',
  `speed_run` FLOAT NOT NULL DEFAULT '1.14286' COMMENT 'Result of 8.0/7.0, most common value',
  `scale` FLOAT NOT NULL DEFAULT '1',
  `rank` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `mindmg` FLOAT NOT NULL DEFAULT '0',
  `maxdmg` FLOAT NOT NULL DEFAULT '0',
  `dmgschool` TINYINT(4) NOT NULL DEFAULT '0',
  `attackpower` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `dmg_multiplier` FLOAT NOT NULL DEFAULT '1',
  `baseattacktime` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `rangeattacktime` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `unit_class` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `unit_flags` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `dynamicflags` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `family` TINYINT(4) NOT NULL DEFAULT '0',
  `trainer_type` TINYINT(4) NOT NULL DEFAULT '0',
  `trainer_spell` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `trainer_class` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `trainer_race` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `minrangedmg` FLOAT NOT NULL DEFAULT '0',
  `maxrangedmg` FLOAT NOT NULL DEFAULT '0',
  `rangedattackpower` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0',
  `type` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `type_flags` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `lootid` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `pickpocketloot` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `skinloot` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `resistance1` SMALLINT(5) NOT NULL DEFAULT '0',
  `resistance2` SMALLINT(5) NOT NULL DEFAULT '0',
  `resistance3` SMALLINT(5) NOT NULL DEFAULT '0',
  `resistance4` SMALLINT(5) NOT NULL DEFAULT '0',
  `resistance5` SMALLINT(5) NOT NULL DEFAULT '0',
  `resistance6` SMALLINT(5) NOT NULL DEFAULT '0',
  `spell1` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell2` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell3` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell4` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell5` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell6` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell7` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `spell8` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `PetSpellDataId` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `VehicleId` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `mingold` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `maxgold` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `AIName` CHAR(64) NOT NULL DEFAULT '',
  `MovementType` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `InhabitType` TINYINT(3) UNSIGNED NOT NULL DEFAULT '3',
  `Health_mod` FLOAT NOT NULL DEFAULT '1',
  `Mana_mod` FLOAT NOT NULL DEFAULT '1',
```

```
  `Armor_mod` FLOAT NOT NULL DEFAULT '1',
  `RacialLeader` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `questItem1` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `questItem2` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `questItem3` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `questItem4` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `questItem5` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `questItem6` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `movementId` INT(11) UNSIGNED NOT NULL DEFAULT '0',
  `RegenHealth` TINYINT(3) UNSIGNED NOT NULL DEFAULT '1',
  `equipment_id` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `mechanic_immune_mask` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `flags_extra` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `ScriptName` CHAR(64) NOT NULL DEFAULT '',
  `WDBVerified` SMALLINT(5) DEFAULT '1',
  PRIMARY KEY (`entry`),
  KEY `idx_name` (`name`)
) ENGINE=MYISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED COMMENT='Creature System';
```

## C2 Struttura tabella `creature` (*TrinityCore*):

```
CREATE TABLE `creature` (
  `guid` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'Global Unique Identifier',
  `id` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0' COMMENT 'Creature Identifier',
  `map` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0' COMMENT 'Map Identifier',
  `spawnMask` TINYINT(3) UNSIGNED NOT NULL DEFAULT '1',
  `phaseMask` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '1',
  `modelid` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `equipment_id` MEDIUMINT(9) NOT NULL DEFAULT '0',
  `position_x` FLOAT NOT NULL DEFAULT '0',
  `position_y` FLOAT NOT NULL DEFAULT '0',
  `position_z` FLOAT NOT NULL DEFAULT '0',
  `orientation` FLOAT NOT NULL DEFAULT '0',
  `spawntimesecs` INT(10) UNSIGNED NOT NULL DEFAULT '120',
  `spawndist` FLOAT NOT NULL DEFAULT '0',
  `currentwaypoint` MEDIUMINT(8) UNSIGNED NOT NULL DEFAULT '0',
  `curhealth` INT(10) UNSIGNED NOT NULL DEFAULT '1',
  `curmana` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `MovementType` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `npcflag` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `unit_flags` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `dynamicflags` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  PRIMARY KEY (`guid`),
  KEY `idx_map` (`map`),
  KEY `idx_id` (`id`)
) ENGINE=MYISAM AUTO_INCREMENT=250007 DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC COMMENT='Creature
System';
```

## C3 Struttura tabella `smart_scripts` (*TrinityCore*):

```
CREATE TABLE `smart_scripts` (
  `entryorguid` INT(11) NOT NULL,
  `source_type` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `id` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0',
  `link` SMALLINT(5) UNSIGNED NOT NULL DEFAULT '0',
  `event_type` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `event_phase_mask` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `event_chance` TINYINT(3) UNSIGNED NOT NULL DEFAULT '100',
  `event_flags` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `event_param1` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `event_param2` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `event_param3` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `event_param4` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_type` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `action_param1` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_param2` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_param3` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_param4` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_param5` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `action_param6` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `target_type` TINYINT(3) UNSIGNED NOT NULL DEFAULT '0',
  `target_param1` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `target_param2` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `target_param3` INT(10) UNSIGNED NOT NULL DEFAULT '0',
  `target_x` FLOAT NOT NULL DEFAULT '0',
  `target_y` FLOAT NOT NULL DEFAULT '0',
  `target_z` FLOAT NOT NULL DEFAULT '0',
  `target_o` FLOAT NOT NULL DEFAULT '0',
```

```
  `comment` TEXT NOT NULL COMMENT 'Event Comment',
  PRIMARY KEY (`entryorguid`,`source_type`,`id`,`link`)
) ENGINE=MYISAM DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

## C4 Classe `CreatureScript` (*TrinityScript*):

```
class CreatureScript : public ScriptObject, public UpdatableScript<Creature>
{
    protected:
        CreatureScript(const char* name);
    public:
        bool IsDatabaseBound() const { return true; }
        virtual bool OnDummyEffect(Unit* /*caster*/, uint32 /*spellId*/, SpellEffIndex /*effIndex*/,
Creature* target) { return false; }
        virtual bool OnGossipHello(Player* /*player*/, Creature* /*creature*/) { return false; }
        virtual bool OnGossipSelect(Player* /*player*/, Creature* /*creature*/, uint32 /*sender*/, uint32
/*action*/) { return false; }
        virtual bool OnGossipSelectCode(Player* /*player*/, Creature* /*creature*/, uint32 /*sender*/, uint32
/*action*/, const char* /*code*/) { return false; }
        virtual bool OnQuestAccept(Player* /*player*/, Creature* /*creature*/, Quest const* /*quest*/) {
return false; }
        virtual bool OnQuestSelect(Player* /*player*/, Creature* /*creature*/, Quest const* /*quest*/) {
return false; }
        virtual bool OnQuestComplete(Player* /*player*/, Creature* /*creature*/, Quest const* /*quest*/) {
return false; }
        virtual bool OnQuestReward(Player* /*player*/, Creature* /*creature*/, Quest const* /*quest*/, uint32
/*opt*/) { return false; }
        virtual uint32 GetDialogStatus(Player* /*player*/, Creature* /*creature*/) { return 100; }
        virtual CreatureAI* GetAI(Creature* /*creature*/) const { return NULL; }
};
```

## C5 Classe `UnitAI` (*TrinityCore*):

```
class UnitAI
{
    protected:
        Unit* const me;
    public:
        explicit UnitAI(Unit* unit) : me(unit) {}
        virtual ~UnitAI() {}
        virtual bool CanAIAttack(Unit const* /*target*/) const { return true; }
        virtual void AttackStart(Unit* /*target*/);
        virtual void UpdateAI(uint32 const diff) = 0;
        virtual void InitializeAI() { if (!me->isDead()) Reset(); }
        virtual void Reset() {};
        virtual void OnCharmed(bool apply) = 0;
        virtual void DoAction(int32 const /*param*/) {}
        virtual uint32 GetData(uint32 /*id = 0*/) { return 0; }
        virtual void SetData(uint32 /*id*/, uint32 /*value*/) {}
        virtual void SetGUID(uint64 /*guid*/, int32 /*id*/ = 0) {}
        virtual uint64 GetGUID(int32 /*id*/ = 0) { return 0; }
        Unit* SelectTarget(SelectAggroTarget targetType, uint32 position = 0, float dist = 0.0f, bool
playerOnly = false, int32 aura = 0);
        template <class PREDICATE> Unit* SelectTarget(SelectAggroTarget targetType, uint32 position,
PREDICATE const& predicate)
        {
            const std::list<HostileReference*>& threatlist = me->getThreatManager().getThreatList();
            if (position >= threatlist.size())
                return NULL;
            std::list<Unit*> targetList;
            for (std::list<HostileReference*>::const_iterator itr = threatlist.begin(); itr !=
threatlist.end(); ++itr)
                if (predicate((*itr)->getTarget()))
                    targetList.push_back((*itr)->getTarget());
            if (position >= targetList.size())
                return NULL;
            if (targetType == SELECT_TARGET_NEAREST || targetType == SELECT_TARGET_FARTHEST)
                targetList.sort(Trinity::ObjectDistanceOrderPred(me));
            switch (targetType)
            {
                case SELECT_TARGET_NEAREST:
                case SELECT_TARGET_TOPAGGRO:
                {
                    std::list<Unit*>::iterator itr = targetList.begin();
                    std::advance(itr, position);
                    return *itr;
```

```
                }
                case SELECT_TARGET_FARTHEST:
                case SELECT_TARGET_BOTTOMAGGRO:
                {
                    std::list<Unit*>::reverse_iterator ritr = targetList.rbegin();
                    std::advance(ritr, position);
                    return *ritr;
                }
                case SELECT_TARGET_RANDOM:
                {
                    std::list<Unit*>::iterator itr = targetList.begin();
                    std::advance(itr, urand(position, targetList.size() - 1));
                    return *itr;
                }
                default:
                    break;
            }
            return NULL;
        }
        void SelectTargetList(std::list<Unit*>& targetList, uint32 num, SelectAggroTarget targetType, float
dist = 0.0f, bool playerOnly = false, int32 aura = 0);
        template <class PREDICATE> void SelectTargetList(std::list<Unit*>& targetList, PREDICATE const&
predicate, uint32 maxTargets, SelectAggroTarget targetType)
        {
            std::list<HostileReference*> const& threatlist = me->getThreatManager().getThreatList();
            if (threatlist.empty())
                return;
            for (std::list<HostileReference*>::const_iterator itr = threatlist.begin(); itr !=
threatlist.end(); ++itr)
                if (predicate((*itr)->getTarget()))
                    targetList.push_back((*itr)->getTarget());
            if (targetList.size() < maxTargets)
                return;
            if (targetType == SELECT_TARGET_NEAREST || targetType == SELECT_TARGET_FARTHEST)
                targetList.sort(Trinity::ObjectDistanceOrderPred(me));
            if (targetType == SELECT_TARGET_FARTHEST || targetType == SELECT_TARGET_BOTTOMAGGRO)
                targetList.reverse();
            if (targetType == SELECT_TARGET_RANDOM)
                Trinity::RandomResizeList(targetList, maxTargets);
            else
                targetList.resize(maxTargets);
        }
        virtual void DamageDealt(Unit* /*victim*/, uint32& /*damage*/, DamageEffectType /*damageType*/) { }
        virtual void DamageTaken(Unit* /*attacker*/, uint32& /*damage*/) {}
        virtual void HealReceived(Unit* /*done_by*/, uint32& /*addhealth*/) {}
        virtual void HealDone(Unit* /*done_to*/, uint32& /*addhealth*/) {}
        void AttackStartCaster(Unit* victim, float dist);
        void DoAddAuraToAllHostilePlayers(uint32 spellid);
        void DoCast(uint32 spellId);
        void DoCast(Unit* victim, uint32 spellId, bool triggered = false);
        void DoCastToAllHostilePlayers(uint32 spellid, bool triggered = false);
        void DoCastVictim(uint32 spellId, bool triggered = false);
        void DoCastAOE(uint32 spellId, bool triggered = false);
        float DoGetSpellMaxRange(uint32 spellId, bool positive = false);
        void DoMeleeAttackIfReady();
        bool DoSpellAttackIfReady(uint32 spell);
        static AISpellInfoType* AISpellInfo;
        static void FillAISpellInfo();
        virtual void sGossipHello(Player* /*player*/) {}
        virtual void sGossipSelect(Player* /*player*/, uint32 /*sender*/, uint32 /*action*/) {}
        virtual void sGossipSelectCode(Player* /*player*/, uint32 /*sender*/, uint32 /*action*/, char const*
/*code*/) {}
        virtual void sQuestAccept(Player* /*player*/, Quest const* /*quest*/) {}
        virtual void sQuestSelect(Player* /*player*/, Quest const* /*quest*/) {}
        virtual void sQuestComplete(Player* /*player*/, Quest const* /*quest*/) {}
        virtual void sQuestReward(Player* /*player*/, Quest const* /*quest*/, uint32 /*opt*/) {}
        virtual bool sOnDummyEffect(Unit* /*caster*/, uint32 /*spellId*/, SpellEffIndex /*effIndex*/) {
return false; }
};
```

## C6 Classe `CreatureAI` (*TrinityCore*):

```
class CreatureAI : public UnitAI
{
    protected:
```

```
        Creature* const me;
        bool UpdateVictim();
        bool UpdateVictimWithGaze();
        void SetGazeOn(Unit* target);
        Creature* DoSummon(uint32 entry, Position const& pos, uint32 despawnTime = 30000, TempSummonType
summonType = TEMPSUMMON_CORPSE_TIMED_DESPAWN);
        Creature* DoSummon(uint32 entry, WorldObject* obj, float radius = 5.0f, uint32 despawnTime = 30000,
TempSummonType summonType = TEMPSUMMON_CORPSE_TIMED_DESPAWN);
        Creature* DoSummonFlyer(uint32 entry, WorldObject* obj, float flightZ, float radius = 5.0f, uint32
despawnTime = 30000, TempSummonType summonType = TEMPSUMMON_CORPSE_TIMED_DESPAWN);
    public:
        void Talk(uint8 id, uint64 WhisperGuid = 0);
        explicit CreatureAI(Creature* creature) : UnitAI(creature), me(creature),
m_MoveInLineOfSight_locked(false) {}
        virtual ~CreatureAI() {}
        void MoveInLineOfSight_Safe(Unit* who);
        virtual bool CanRespawn() { return true; }
        virtual void EnterEvadeMode();
        virtual void EnterCombat(Unit* /*victim*/) {}
        virtual void JustDied(Unit* /*killer*/) {}
        virtual void KilledUnit(Unit* /*victim*/) {}
        virtual void JustSummoned(Creature* /*summon*/) {}
        virtual void IsSummonedBy(Unit* /*summoner*/) {}
        virtual void SummonedCreatureDespawn(Creature* /*summon*/) {}
        virtual void SummonedCreatureDies(Creature* /*summon*/, Unit* /*killer*/) {}
        virtual void SpellHit(Unit* /*caster*/, SpellInfo const* /*spell*/) {}
        virtual void SpellHitTarget(Unit* /*target*/, SpellInfo const* /*spell*/) {}
        virtual bool IsEscorted() { return false; }
        virtual void JustRespawned() { Reset(); }

        virtual void MovementInform(uint32 /*type*/, uint32 /*id*/) {}
        void OnCharmed(bool apply);
        virtual void JustReachedHome() {}
        void DoZoneInCombat(Creature* creature = NULL, float maxRangeToNearestTarget = 50.0f);
        virtual void ReceiveEmote(Player* /*player*/, uint32 /*emoteId*/) {}
        virtual void CorpseRemoved(uint32& /*respawnDelay*/) {}
        virtual void PassengerBoarded(Unit* /*passenger*/, int8 /*seatId*/, bool /*apply*/) {}
        virtual bool CanSeeAlways(WorldObject const* /*obj*/) { return false; }
    protected:
        virtual void MoveInLineOfSight(Unit* /*who*/);
        bool _EnterEvadeMode();
    private:
        bool m_MoveInLineOfSight_locked;
};
```

## C7 Classe `ScriptedAI` (*TrinityScript*):

```
struct ScriptedAI : public CreatureAI
{
    explicit ScriptedAI(Creature* creature);
    virtual ~ScriptedAI() {}
    void DamageTaken(Unit* /*attacker*/, uint32& /*damage*/) {}
    virtual void UpdateAI(uint32 const diff);
    void JustDied(Unit* /*killer*/) {}
    void KilledUnit(Unit* /*victim*/) {}
    void JustSummoned(Creature* /*summon*/) {}
    void SummonedCreatureDespawn(Creature* /*summon*/) {}
    void SpellHit(Unit* /*caster*/, SpellInfo const* /*spell*/) {}
    void SpellHitTarget(Unit* /*target*/, SpellInfo const* /*spell*/) {}
    void MovementInform(uint32 /*type*/, uint32 /*id*/) {}
    void OnPossess(bool /*apply*/) {}
    Creature* me;
    bool IsFleeing;
    void Reset() {}
    void EnterCombat(Unit* /*victim*/) {}
    void DoStartMovement(Unit* target, float distance = 0.0f, float angle = 0.0f);
    void DoStartNoMovement(Unit* target);
    void DoStopAttack();
    void DoCastSpell(Unit* target, SpellInfo const* spellInfo, bool triggered = false);
    void DoPlaySoundToSet(WorldObject* source, uint32 soundId);
    void DoResetThreat();
    float DoGetThreat(Unit* unit);
    void DoModifyThreatPercent(Unit* unit, int32 pct);
    void DoTeleportTo(float x, float y, float z, uint32 time = 0);
    void DoTeleportTo(float const pos[4]);
```

```
    void DoTeleportPlayer(Unit* unit, float x, float y, float z, float o);
    void DoTeleportAll(float x, float y, float z, float o);
    Unit* DoSelectLowestHpFriendly(float range, uint32 minHPDiff = 1);
    std::list<Creature*> DoFindFriendlyCC(float range);
    std::list<Creature*> DoFindFriendlyMissingBuff(float range, uint32 spellId);
    Player* GetPlayerAtMinimumRange(float minRange);
    Creature* DoSpawnCreature(uint32 entry, float offsetX, float offsetY, float offsetZ, float angle, uint32
type, uint32 despawntime);
    bool HealthBelowPct(uint32 pct) const { return me->HealthBelowPct(pct); }
    bool HealthAbovePct(uint32 pct) const { return me->HealthAbovePct(pct); }
    SpellInfo const* SelectSpell(Unit* target, uint32 school, uint32 mechanic, SelectTargetType targets,
uint32 powerCostMin, uint32 powerCostMax, float rangeMin, float rangeMax, SelectEffect effect);
    void SetEquipmentSlots(bool loadDefault, int32 mainHand = EQUIP_NO_CHANGE, int32 offHand =
EQUIP_NO_CHANGE, int32 ranged = EQUIP_NO_CHANGE);
    void SetCombatMovement(bool allowMovement);
    bool IsCombatMovementAllowed() { return _isCombatMovementAllowed; }
    bool EnterEvadeIfOutOfCombatArea(uint32 const diff);
    bool IsHeroic() { return _isHeroic; }
    Difficulty GetDifficulty() { return _difficulty; }
    bool Is25ManRaid() { return _difficulty & RAID_DIFFICULTY_MASK_25MAN; }
    template<class T> inline
    const T& DUNGEON_MODE(const T& normal5, const T& heroic10)
    {
        switch (_difficulty)
        {
            case DUNGEON_DIFFICULTY_NORMAL:
                return normal5;
            case DUNGEON_DIFFICULTY_HEROIC:
                return heroic10;
            default:
                break;
        }
        return heroic10;
    }
    template<class T> inline
    const T& RAID_MODE(const T& normal10, const T& normal25)
    {
        switch (_difficulty)
        {
            case RAID_DIFFICULTY_10MAN_NORMAL:
                return normal10;
            case RAID_DIFFICULTY_25MAN_NORMAL:
                return normal25;
            default:
                break;
        }
        return normal25;
    }
    template<class T> inline
    const T& RAID_MODE(const T& normal10, const T& normal25, const T& heroic10, const T& heroic25)
    {
        switch (_difficulty)
        {
            case RAID_DIFFICULTY_10MAN_NORMAL:
                return normal10;
            case RAID_DIFFICULTY_25MAN_NORMAL:
                return normal25;
            case RAID_DIFFICULTY_10MAN_HEROIC:
                return heroic10;
            case RAID_DIFFICULTY_25MAN_HEROIC:
                return heroic25;
            default:
                break;
        }
        return heroic25;
    }
    private:
        Difficulty _difficulty;
        uint32 _evadeCheckCooldown;
        bool _isCombatMovementAllowed;
        bool _isHeroic;
};
```