Politecnico di Milano
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Sviluppo di un framework per la progettazione e l'analisi di mappe per First Person Shooters

Relatore: Dott. Daniele LOIACONO

Elaborato finale di:
Luca ARNABOLDI Matr. 766343

Dedicata a S.

Sommario

Il *level design* è un componente essenziale nello sviluppo dei videogiochi, essendo la fase in cui vengono concretizzati gli aspetti di *gameplay* definiti durante la progettazione del gioco, e assume maggior peso per alcuni generi di gioco come quello degli *sparatutto in prima persona*. Tuttavia, nonostante la sua importanza, solo negli ultimi anni è emersa l'attenzione sul problema della mancanza di una *disciplina* del level design.

Abbiamo analizzato lo stato attuale del problema e come sta venendo affrontato dall'industria e nella ricerca accademica, e abbiamo realizzato un *framework* di design configurabile che genera automaticamente mappe che soddisfano determinati obiettivi di design imposti dall'utente, che può essere utilizzato come strumento di aiuto sia nella realizzazione di livelli che nella sperimentazione e analisi di nuovi *pattern* e metodi. Il nostro metodo sfrutta la *generazione procedurale del contenuto con algoritmi evolutivi* per creare tali mappe, le cui caratteristiche vengono valutate *simulando* partite nel gioco open-source *Cube 2: Sauerbraten*. Per rendere validi i dati raccolti durante le simulazioni abbiamo modificato l'intelligenza artificiale del gioco per ricreare un comportamento più realistico degli agenti artificiali.

Abbiamo realizzato alcuni esperimenti per verificare l'efficacia del framework usando diverse impostazioni dell'intelligenza artificiale e obiettivi di gameplay; abbiamo descritto la struttura delle mappe ottenute che ottimizzano gli obiettivi cercati, trovando e analizzando *pattern ricorrenti* e le meccaniche di gioco da essi prodotte, osservando anche quali connessioni intercorrono tra alcune delle statistiche raccolte durante le simulazioni.

Abstract

Level design is an essential task in the videogames development, being the step where the design of *gameplay* is mapped into an actual playing experience; accordingly, this step plays a key role in some kinds of games such as the *first person shooters*. However, despite its importance, the problem of the lack of a level design *discipline* has arisen only in the past few years.
We analyzed the current state of the problem and how the industry and the academic research deal with it, and we set up an adjustable design *framework* that automatically builds maps satisfying specific gameplay goals set by the user, which can be used as an assisting tool both for the creation of game levels and for the exploration and analysis of new design *patterns* and methods. Our approach exploits *search-based procedural content generation* through *evolutionary algorithms* to create such maps, whose features are evaluated simulating some matches in the open-source game *Cube 2: Sauerbraten*. To make sure the data gathered during the simulations are valid we have modified the game's artificial intelligence to make the artificial agents' behavior more realistic.
We carried out some experiments to test our framework's effectiveness using different AI setups and gameplay objectives; we described the structure of the resulting maps, finding and analyzing recurring *patterns* and the dynamics they produced, also observing the connections among the statistics gathered during the simulations.

Ringraziamenti

Vorrei ringraziare innanzitutto il Dott. Daniele Loiacono per avermi pazientemente assistito nella realizzazione di questa tesi, e per avermi dato la possibilità di unire agli studi la passione di una vita.

Ringrazio mio padre, i cui sacrifici in tutti questi anni mi hanno permesso di raggiungere questo importante traguardo.

Infine ringrazio Francesca, che crede e che ha sempre creduto in me, che mi ha supportato e sopportato soprattutto in questi mesi, dandomi coraggio anche da più di sedicimila chilometri di distanza.

Luca Arnaboldi

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Lo sviluppo di un videogioco è un processo complesso, e sono molteplici gli aspetti e le competenze richieste nella realizzazione di un prodotto di successo. È però il *game design* e soprattutto la ricerca di un innovativo **gameplay**[1] che fondano il successo di un gioco, i componenti che creano un'esperienza gratificante, avvincente e stimolante, che spingono il giocatore a rigiocarci più e più volte[2].

Fondamentale è per questo il **level design**, ossia la progettazione dei mondi virtuali in cui i giocatori prendono parte al gioco. Citando Jay Wilbur[2], *"il level design è dove la gomma incontra la strada"*[3], ovvero come tutti gli aspetti definiti in fase di progettazione vengono inseriti e concretizzati nel prodotto finale, e dunque come verrà visto e giocato dagli utenti; un *level design* non adeguato può comprometterne la qualità e dissuadere i giocatori dal giocarci.

Uno dei generi di videogiochi attualmente più popolari è quello dei **First Person Shooter**. Come per tutti i giochi che presentano una prospettiva in prima persona, che immerge completamente il giocatore nello spazio di gioco, la qualità dei livelli e dunque del *level design* per questo genere assumono particolare rilevanza. Inoltre l'enorme popolarità raggiunta dalla scena competitiva ha reso le modalità multigiocatore un fattore primario, motivo per cui nella realizzazione delle mappe è necessario considerare, oltre alle diverse caratteristiche e alle modalità del gioco, gli stili di diversi profili di giocatore e le loro possibili interazioni

[1]Con *game design* e *gameplay* si intendono il processo di creazione di contenuti, obiettivi che un giocatore si sente motivato a raggiungere, e regole che un giocatore deve seguire nell'effettuare decisioni significative nel raggiungimento di tali obiettivi[1].

[2]Manager e direttore dei progetti di *id Software* negli anni'90. Attualmente vice presidente del business development di *Epic Games*.

durante le partite per offrire un'esperienza di gioco piacevole e bilanciata, lavoro però difficile e potenzialmente dispendioso.

1.1 Motivazioni e scopo della tesi

La ricerca nel settore del *level design* dei *first person shooter* non è mai stata tra i principali interessi nell'industria videoludica, e di conseguenza risulta essere una disciplina piuttosto astratta, priva di un *vocabolario* e di standard condivisi, affidata solitamente alla sola esperienza di chi ne è responsabile; la letteratura specifica in materia è piuttosto scarsa e si concentra essenzialmente sull'insegnamento di specifici programmi di design, e di stili e convenzioni esistenti senza esaminarne però le meccaniche che li hanno portati a essere considerati efficaci. La mancanza di una *"teoria"* dietro ai *pattern* di design visti molte volte nei livelli dei giochi pubblicati rende più complicato sia l'approccio che la sperimentazione in questo campo.

Solo negli ultimi anni si sta cominciando a dare al *level design* una definizione concreta, così come alla figura del *level designer*. Gran parte dell'interesse all'argomento viene dall'ambiente accademico, i cui approcci si basano però principalmente sulla sola analisi dei comportamenti dei giocatori quando messi davanti ad alcuni *pattern* già ben conosciuti.

Basandoci su tecniche di **generazione procedurale dei contenuti con algoritmi evolutivi**, la cui efficacia nella creazione automatica di mappe per FPS che ottimizzano determinati problemi è gia stata esaminata in lavori precedenti [4, 5], abbiamo approcciato il problema da un'altra prospettiva, in modo più *esplorativo*: non più analizzare le conseguenze nel gameplay partendo da determinati pattern, bensì partire da precisi obiettivi di gameplay e cercare quali sono i pattern che li producono. Abbiamo quindi realizzato un *framework* per la realizzazione e l'analisi di mappe per FPS, che permette attraverso simulazioni e algoritmi evolutivi configurabili di ricercare mappe che soddisfano specifiche esigenze di design dei livelli, e che può essere utilizzato sia da un designer nella loro concreta realizzazione, sia nella ricerca e analisi di nuovi *pattern* e degli aspetti responsabili di determinate meccaniche di gioco.

La tesi è strutturata nel seguente modo:

Nel *secondo capitolo* si illustra lo stato dell'arte, sia dal punto di vista del *level design*, considerando esclusivamente i *first person shooter* e analizzandone lo stato nell'industria e nella ricerca accademica, che da quello della *generazione automatica del contenuto basata su algoritmi di ricerca*, tecnica usata per la realizzazione e valutazione automatica delle mappe.

Nel terzo capitolo viene presentato **Cube 2: Sauerbraten**[3], il gioco *open source* usato come base per il nostro lavoro. Vengono analizzati i punti di forza e i problemi presenti nel gioco, con particolare attenzione al modello dell'intelligenza artificiale; vengono poi illustrate le modifiche che sono state apportate per rendere tale modello più adatto ai nostri scopi.

Nel quarto capitolo viene illustrato il *framework* realizzato, descrivendo le possibili configurazioni utili a modellare le caratteristiche dei giocatori e delle mappe da evolvere nelle simulazioni: vengono descritti i parametri personalizzabili dei bot dell'intelligenza artificiale del gioco, le statistiche raccolte durante le partite e le metriche di analisi che da esse vengono derivate.

Nel quinto capitolo vengono mostrati e analizzati tre degli esperimenti che abbiamo effettuato per verificare l'efficacia del metodo; per ogni esperimento sono mostrate le mappe finali ottenute durante i processi evolutivi, rappresentazioni grafiche di alcuni dati realizzate per analizzare le dinamiche di gioco da esse prodotte e le statistiche di evoluzione.

Nelle conclusioni si riassume il lavoro svolto, le valutazioni dei risultati ottenuti, i problemi riscontrati nella realizzazione, le possibili critiche e i potenziali sviluppi futuri.

[3]Wouter van Oortmerssen, 2004.

Capitolo 2

Stato dell'arte

In questo capitolo illustriamo qual'è lo stato attuale per quanto riguarda il problema del *level design*; esaminiamo come il problema sta venendo affrontato sia relativamente agli sparatutto a singolo giocatore che a quelli multigiocatore dall'industria e nella ricerca accademica.

Viene poi introdotta la *generazione procedurale dei contenuti* (*Procedural Content Generation, o PCG*), con particolare riferimento alla generazione procedurale con *algoritmi di ricerca* (*Search Based Procedural Content Generation, o SBPCG*), osservando poi com'è stata usata in recenti lavori per risolvere problemi analoghi a quelli da noi affrontati.

Infine diamo una panoramica sulle caratteristiche, sulla storia e sull'evoluzione degli *sparatutto in prima persona* (*First Person Shooter, o FPS*), soffermandoci principalmente sui periodi e sui titoli che hanno influenzato in modo significativo il genere, e analizzando come hanno rivoluzionato i metodi utilizzati nella progettazione dei livelli.

2.1 Teoria del Level Design

Quello del **level designer**, definito come tale, è un ruolo relativamente giovane nei team di sviluppo dei videogiochi, e i suoi compiti sono tutt'ora non perfettamente definiti. Ovviamente sono sempre esistite nei team di sviluppo persone a cui veniva affidato il compito di creare i livelli dei giochi, ma in passato spesso queste non rappresentavano figure specializzate, ed era frequente che questo lavoro venisse portato avanti da responsabili di altri settori. Oggi si può dire

con certezza che il *level designer*, inteso come responsabile della creazione del mondo con cui il giocatore interagisce, ricopre un ruolo di estrema importanza nella creazione di un buon videogioco: ha infatti il compito di far diventare "giocabile" quella che altrimenti è solo l'*idea* del gioco, rendendo accessibili tutti gli aspetti (e i punti di forza) definiti in fase di *game design*, creando le mappe dove questo prende luogo. Ciò nonostante anche dopo decenni di storia dei First Person Shooter, e in generale dei videogiochi, non esistono ancora dei veri e propri *standard* di progettazione dei livelli; le conoscenze e le definizioni che si hanno degli elementi e del processo di design stesso sono dovute principalmente ad euristiche, all'osservazione e replicazione di modelli precedenti e alla propria esperienza. La convinzione generale che si ha è che il *level design* sia una specie di *forma d'arte*, un'abilità che dev'essere affinata con anni di esperienza e che non può essere imparata da un libro[6]. Infatti la maggior parte dei libri in commercio che trattano l'argomento si limitano essenzialmente ad insegnare come costruire mappe usando uno specifico programma, ciò che Kremers definisce *costruzione* di livelli più che design[7], senza entrare nel dettaglio dei motivi alla base di ciò che viene spiegato.

Spesso la figura del *level designer* viene affiancata a quella dell'architetto: così come l'architetto usa dei *pattern* funzionali alla vita delle persone reali il level designer ricerca delle strutture che creino meccaniche di gameplay interessanti per i giocatori, cercando di mantenere un aspetto coerente con il contesto ambientale. L'uso di un linguaggio architetturale che il giocatore può riconoscere ed associare a particolari ambientazioni è un modo per accrescere l'esperienza di gioco[8]; considerando la costante evoluzione della qualità (anche estetica) dei videogiochi è necessario che il *level designer* sia in grado di unire solide composizioni architettoniche ad aspetti tecnici tipici dei giochi (come le *regole* e dunque il *gameplay*) e del *software* in generale (come i limiti imposti dalla tecnologia per cui si sta sviluppando il prodotto). Considerando anche questo punto di vista si può considerare il *level design* tanto un'arte quanto una scienza[9].

Molte persone hanno provato a dare una descrizione della *disciplina* del *level design*, spesso basandosi solo sulla propria esperienza professionale[7], e solo ultimamente l'industria ha cominciato a considerarla un settore a se stante.

Sebbene molti designer siano d'accordo su alcuni concetti indispensabili ai fini di un buon design dei livelli non sempre l'efficacia dei metodi utilizzati è stata

dimostrata. Considerando giochi per singolo giocatore uno di questi aspetti, tra i più importanti da tenere in considerazione specialmente a causa della crescente complessità e quantità di dettagli dei livelli nei giochi moderni, è la gestione del **"level flow"**[6], ovvero la continuità degli spostamenti e delle azioni che il giocatore deve eseguire nello svolgimento del livello; il designer deve guidare il giocatore (che altrimenti si sentirebbe "perso" nel mondo virtuale), possibilmente in modo *trasparente*, catturando la sua attenzione verso i punti da seguire. L'importanza di questo aspetto è evidente in giochi come *Mirror's Edge*[1], in cui il giocatore deve spostarsi il più velocemente possibile nei livelli e viene quindi guidato grazie all'uso di un particolare codice cromatico che, seppur esplicito, è integrato perfettamente con lo stile grafico del gioco, e viene immediatamente assimilato dal giocatore; si ha invece un brillante esempio di integrazione "invisibile" negli stormi di uccelli in *Half-Life 2*, che potevano fungere sia da attrattore, sia come segnale di pericolo[10] in modo naturale e non invasivo, lasciando continuità ai comandi del giocatore[2]. In generale i designer usano elementi come luci, ombre, colori, suoni, geometrie e architetture particolari per guidare il giocatore. A livello accademico esistono diverse ricerche sull'efficacia di questi *dispositivi di controllo*: Alotto[11] esamina in particolar modo l'effetto delle differenze architettoniche, in strutture moderne e reali, sulle decisioni dei giocatori; concetto ripreso da Hoeg[6] e ampliato con un maggior uso di diverse illuminazioni, suoni e oggetti, il tutto contestualizzato come un vero gioco con una storia di fondo, distaccandosi dal contesto completamente realistico del lavoro di Alotto. Ancora, nel lavoro di Brady[12] si nota come strutture architettoniche riconoscibili e imponenti ispirate a famosi monumenti siano un fattore attrattivo per i giocatori. Sebbene i risultati ottenuti in questi lavori non abbiano sempre rispecchiato le aspettative iniziali si sono evidenziati alcuni aspetti che guidano il processo decisionale dei giocatori. Risultati più netti li ha avuti invece Brownmiller[13] concentrandosi sulla sola illuminazione come dispositivo di controllo.

Osservando il problema del *level design* non più dal punto di vista della navigazione dell'utente ma da quello dell'interazione con gli altri attori presenti nel gioco gli studi sono più limitati; nella progettazione di livelli *multiplayer* questo aspetto assume una rilevanza fondamentale, in quanto non essendoci più un per-

[1] Digital Illusions CE, 2008.

[2] Spesso in altri giochi, per ottenere lo stesso risultato, il giocatore veniva forzatamente spostato verso la direzione giusta, causando un distacco del coinvolgimento nel gioco.

corso da seguire sono i giocatori stessi a determinare in larga parte il gameplay dei livelli. Il designer può decidere quali dinamiche di gioco prediligere per un determinato livello, ma per farlo deve essere consapevole delle relazioni causa-effetto tra gli elementi di design e il gameplay, tenendo conto, anche in questo caso, di alcuni elementi indispensabili per garantire un'esperienza di gioco appagante come per esempio l'equilibrio tra i giocatori o il ritmo di gioco. In livelli progettati per il multiplayer è, per esempio, fondamentale prevedere quali siano quelli che Güttler et al. definiscono **punti di collisione**[14], ovvero i punti dove avvengono la maggior parte degli scontri, dove i giocatori scelgono quale tattica usare, decidendo come affrontare il nemico e quale arma usare. Sebbene questi siano spesso determinati direttamente dai giocatori, Güttler et al. affermano che il *design spaziale* del livello, che ne comprende l'architettura e il posizionamento dei *pick-up*[3] e dei punti di respawn[4], determinano la posizione e il conseguente gameplay nel punto di collisione stesso; dunque il designer mantiene comunque un grado di controllo su come il livello dovrebbe essere giocato. Osservando come nei suoi esperimenti aree create appositamente per attirare l'attenzione venissero ignorate, Güttler et al. rilevano anche come il design "estetico" non sia un fattore che incide in modo determinante per i giocatori paragonato agli elementi di gameplay, concetto ribadito anche da Brady [12]. Alcuni lavori sono stati effettuati nella rilevazione di **pattern** di design usati nella creazione dei livelli: Larsen[15] ha analizzato tre famosi videogiochi FPS multigiocatore con caratteristiche molto differenti, *Unreal Tournament 2004*[5], *Day of Defeat: Source*[6] e *Battlefield 1942*[7], identificando una serie di pattern con caratteristiche simili condivisi tra i tre giochi, e analizzandone gli effetti prodotti, le eventuali relazioni tra essi, e dando alcune linee guida su come utilizzarli. Lo scopo del lavoro di Larsen, tuttavia, si limitava principalmente alla definizione di un "vocabolario" che fosse condivisibile tra i designer; non vengono infatti misurati gli effetti dei pattern da lui identificati. In modo simile, Hullett e Whitehead identificano alcuni pattern per il design di livelli per singolo giocatore[16], anche se molte delle considerazioni da loro

[3]Oggetti che il giocatore può raccogliere a proprio beneficio, come armi, munizioni e kit medici.

[4]I punti in cui i giocatori ricominciano a giocare dopo essere stati uccisi in una partita multiplayer.

[5]Epic Games, 2004.

[6]Valve, 2005.

[7]DICE, 2002

fatte possano essere applicate anche ai livelli multigiocatore; inoltre per alcuni di questi pattern Hullett fornisce concrete dimostrazioni causa-effetto confrontando con buoni risultati il comportamento osservato su un campione di giocatori con quello ipotizzato[17]. Tuttavia, sebbene i risultati ottenuti da Hullett diano un sicuro contributo sulle conoscenze nel campo del level design e sulla comunicabilità dei suoi concetti, le sue analisi non coprono ancora in modo esaustivo la materia; inoltre, sebbene ne certifichino l'efficacia, si basano solamente su pattern già largamente adottati e conosciuti.

2.2 La Generazione Procedurale dei Contenuti

Con *generazione procedurale dei contenuti* si intendono metodi utilizzati per creare algoritmicamente in modo automatico dati e contenuti. Tipicamente, riferendosi al campo dei *videogiochi*[8], viene usata per generare armi, oggetti, mappe e livelli, ma trova applicazione anche nella generazione di *texture*, modelli geometrici, animazioni, musica e dialoghi.

Il primo gioco popolare ad usare questa tecnica, seppur non il primo in assoluto, fu nel 1980 *Rogue*[9], precursore di un genere di giochi definiti *roguelike*. La struttura e il contenuto dei livelli di *Rogue* venivano generati ogni volta in modo pseudo-casuale, rendendo l'esperienza di gioco diversa ad ogni partita. Tale modello è stato poi ripreso da altri giochi di successo come *Hack*[10] e la più moderna e nota serie di *Diablo*[11]. *Rogue* e i successivi giochi ad esso ispirati rappresentano un esempio di come la generazione procedurale potesse essere usata per generare un'esperienza di *gameplay* sempre nuova e garantire una rigiocabilità potenzialmente infinita. In modo simile veniva usata anche da altri giochi ma con scopi completamente diversi: le limitazioni tecnologiche presenti in passato non permettevano di produrre giochi con una quantità di contenuto elevato. Veniva quindi usata la generazione procedurale per generare il mondo virtuale e i livelli algoritmicamente, sfruttando alle volte "blocchi" di costruzione predefiniti, usando tuttavia parametri fissi; in questo modo i contenuti del gioco potevano

[8]Negli ultimi si sono cominciate ad usare tecniche di generazione procedurale anche nel campo degli effetti speciali nell'industria cinematografica, soprattutto per scene molto popolate.
[9]Michael Toy, Glenn Wichman, 1980.
[10]Jay Fenlason, Kenny Woodland, Mike Thome, Jonathan Payne, 1985.
[11]Blizzard Entertainment, 1996-2014.

essere generati *al volo* all'avvio dello stesso in modo sempre uguale, garantendo un notevole risparmio di risorse di memoria e una quantità di contenuto altissima che alle volte, non potendo essere controllato nella sua interezza, causava però situazioni di stallo e ingiocabilità. Alcuni esempi di come la generazione procedurale veniva usata per questo scopo (e dei suddetti problemi) sono *Elite*[12] e *The Elder Scrolls chapter II: Daggerfall*[13].

Sebbene oggi non esistano più problemi tecnologici legati ai dispositivi di memorizzazione la generazione procedurale viene tutt'ora regolarmente usata nella realizzazione di parti dei giochi moderni, e può avere un grande potenziale di innovazione nell'industria videoludica: le possibilità offerte dall'evoluzione dell'hardware hanno costantemente alzato le aspettative qualitative da parte degli utenti, che pretendono mondi più realistici e dettagliati. Dal punto di vista degli sviluppatori ciò si traduce in costi e tempi di produzioni sempre più alti, e di conseguenza rischi più alti nel caso di fallimento commerciale del prodotto e dunque una minore propensione alla sperimentazione e all'innovazione. La possibilità di generare e provare grandi quantità di contenuto in poco tempo può essere una soluzione a questo problema; *.kkrieger*[14] è un esempio, seppur "estremo"[15], di come sia possibile generare interamente un gioco di qualità usando solamente algoritmi di creazione automatica del contenuto.

Molti giochi contemporanei usano in qualche modo la generazione procedurale: in diversi giochi di strategia come quelli della serie *Civilization*[16] il terreno viene generato specificando alcuni parametri; in *Borderlands*[17] tutte le armi vengono generate automaticamente da un algoritmo che seleziona i componenti e le caratteristiche, assegnandogli anche un nome di conseguenza, rendendo ogni arma unica; in *Far Cry 2*[18] le condizioni atmosferiche modificano il terreno; in *Left 4 Dead*[19] la posizione dei nemici e la musica cambiano a seconda degli spostamenti

[12] David Braben, Ian Bell, 1984.
[13] Bethesda Softworks, 1996
[14] Farbrausch, 2004.
[15] Essendo un progetto dimostrativo lo scopo di .kkrieger non era quello di realizzare un gioco in linea con quelli del mercato, e presenta infatti diverse lacune soprattutto dal punto di vista della giocabilità. Nondimeno, occupando il gioco meno di 98kB, dimostra quali possano essere i benefici della PCG nella realizzazione di un gioco.
[16] Microprose, 1991 - 1996; Firaxis Games, 2001 - 2015.
[17] Gearbox Software, 2009 - 2014.
[18] Ubisoft Montreal, 2008.
[19] Valve Corporation, 2008.

del giocatore; in *Spore*[20] le animazioni delle creature si adattano in tempo reale alla loro fisionomia.

Le potenzialità della generazione procedurale hanno inoltre trovato terreno fertile negli sviluppatori *indipendenti*, che non potendo contare sulle risorse delle case di sviluppo dell'industria professionale puntano su metodi non convenzionali per creare giochi con un gameplay intrigante e inusuale, spesso con ottimi risultati. L'esempio più importante è probabilmente *Minecraft*[21], il cui intero mondo è generato automaticamente come una composizione di cubi tridimensionali, senza alcun intervento umano nel loro posizionamento iniziale e senza necessità di creare un modello tridimensionale che lo rappresenti. L'uso della generazione procedurale è stato anche un modo di ricreare in chiave moderna vecchi successi, come dimostrano *The Binding of Isaac*[22] e *Spelunky*[23] che incorporano diversi elementi del genere *roguelike*.

Oggi esistono anche strumenti *middleware* che sfruttano la generazione procedurale, soprattutto per quanto riguarda la creazione di terreni e di ambienti urbani o naturali. Questi strumenti vengono usati da molti sviluppatori professionisti per generare velocemente parti del gioco.

Sebbene per necessità gli algoritmi finora usati nei giochi pubblicati puntino alla rapidità, e siano relativamente semplici e poco controllabili l'ambiente accademico ha, negli ultimi anni, cominciato a mostrare interesse nel campo della generazione procedurale dei contenuti, ricercando tecniche più complesse, adattabili e controllabili che possono colmare le attuali lacune e dare un contributo importante all'industria videoludica introducendo nuove tipologie di giochi e soprattutto nuove metodologie di sviluppo[18].

Essendo il contenuto ottenuto sfruttando queste tecniche generato automaticamente da un algoritmo è molto probabile che questo non rispecchi alcuni canoni qualitativi minimi che ci si aspetterebbero dal contenuto creato a mano da un designer o da un artista. Per questo motivo nell'ambito accademico ci si sta concentrando in particolar modo non solo sulle tecniche di creazione automatica del contenuto ma anche su metodi di *valutazione* automatica del contenuto generato. A tal proposito è stata quindi definita da Togelius et al.[19] la **ge-**

[20]Maxis, 2008.
[21]Mojang, 2011.
[22]Edmund McMillen, 2011.
[23]Mossmouth, 2008.

nerazione procedurale con algoritmi di ricerca, una tipologia speciale di algoritmi *generate-and-test*[24] da cui differiscono per il fatto che il contenuto generato, piuttosto che venir semplicemente accettato o scartato, viene valutato assegnandogli un *punteggio* relativo all'idoneità misurata tramite un'opportuna *funzione di fitness*, che viene poi usato per selezionare i migliori candidati da usare come base per le generazioni successive. Le funzioni di fitness vengono definite ad-hoc considerando alcuni criteri che meglio rappresentano il problema in esame e la sua soluzione ideale.

2.2.1 Applicazioni nella generazione di mappe per First Person Shooter

Relativamente al genere degli sparatutto in prima persona la generazione procedurale attraverso algoritmi di ricerca si è già rivelato un utile strumento. In questo contesto il metodo è stato applicato per la prima volta da Cardamone et al.[4], che hanno affrontato il problema di ricercare quali tipologie di mappe creassero un *gameplay* il più interessante possibile; allo scopo gli autori hanno generato, attraverso algoritmi evolutivi, mappe per il videogioco *Cube 2: Sauerbraten* fondate sulla definizione di quattro tipi di rappresentazione strutturale[25], massimizzando una funzione di fitness basata su *simulazione*[26] determinata calcolando il *tempo di combattimento*, ovvero la durata dello scontro dal momento in cui un giocatore inizia a combattere un avversario al momento in cui il giocatore viene ucciso. La scelta di tale funzione di fitness si basa sull'ipotesi che, poichè le mappe degli FPS presentano diverse caratteristiche che possono essere strategicamente sfruttate per coinvolgere i giocatori in combattimenti più lunghi e interessanti, una durata elevata dei combattimenti corrisponde a una maggior offerta da parte della mappa di possibilità di fuga, nascondigli, bonus di salute

[24] Algoritmi che comprendono sia meccanismi di generazione che di valutazione che, a seconda di qualche criterio, determinano se accettare o meno il risultato. Differiscono in questo dagli algoritmi *costruttivi* che si basano solo sulla costruzione del contenuto. Per maggiori dettagli si rimanda a [19].

[25] Per maggiori dettagli si veda la sezione 4.4.1.

[26] Nell'ambito della generazione procedurale con algoritmi di ricerca le funzioni di fitness basate su *simulazione* ne calcolano il valore attraverso agenti artificiali che giocano una parte del gioco che dev'essere valutata. Differiscono dalle funzioni *dirette* e *interattive*, che valutano, rispettivamente, direttamente il contenuto generato e le interazioni con un giocatore reale. Per maggiori dettagli si rimanda a [19].

e migliore equipaggiamento[4]. Cardamone et al. mostrano che la generazione procedurale con algoritmi di ricerca può essere utilizzata per evolvere mappe giocabili, e che alcune rappresentazioni di mappa tra quelle da loro esaminate si adattano meglio all'obiettivo da loro cercato.

Sulla stessa linea è proseguito il lavoro di Stucchi[5], con l'obiettivo di generare mappe equilibrate tra due giocatori con livelli di abilità o armi diverse usando la generazione procedurale. Usando come base parte del lavoro svolto in [4], Stucchi ha generato, tramite algoritmi evolutivi, diverse mappe usando una funzione di fitness basata su simulazione che calcolava l'entropia delle uccisioni, ovvero la dispersività dei punteggi tra gli attori coinvolti nelle partite. Simulando partite tra due giocatori, e partendo da situazioni di forte svantaggio per una delle due parti, Stucchi mostra come variazioni nella struttura della mappa portino due attori con diversi livelli di abilità ma dotati di armi uguali a raggiungere una situazione più equilibrata per quanto riguarda il punteggio finale, tendenzialmente sfavorendo il giocatore più abile.

Ølsted et al.[20] notano tuttavia come l'approccio usato da Cardamone et al., da cui prendono diversi spunti per il loro lavoro, crei mappe inadatte a FPS multiplayer in stile *Counter-Strike*[27], cioè giochi di squadra con obiettivi specifici posizionati nella mappa e attorno ai quali si sviluppa la maggior parte del gioco. Inoltre affermano che le stesse non rispettano quelle che gli autori della ricerca definiscono *regole di buon combattimento*[28]. Nel lavoro di Ølsted et al. le mappe vengono realizzate selezionando casualmente e connettendo alcuni nodi da una griglia creando uno *scheletro* sul quale vengono generati i corridoi, successivamente ottimizzati per rispettare le *regole di buon combattimento* ed estesi con l'aggiunta di stanze, oggetti e punti di respawn. Il processo appena descritto, schematizzato in Figura 2.1, è stato derivato da uno studio delle caratteristiche delle mappe in *Counter Strike* e *Call of Duty*[29], considerandone in modo specifico la modalità *Search & Destroy*[30]. A differenza dei metodi utilizzati da Cardamone et al. e Stucchi per l'evoluzione delle mappe viene usato un approccio *interatti-*

[27]Minh Le e Jess Cliffe, 1999; Valve Software (attualmente Valve Corporation), 2000.

[28]Intese come strutture e meccaniche di gioco di base che dovrebbero essere garantite in ogni mappa. Per ulteriori dettagli si rimanda a [20].

[29]Infinity Ward, 2003

[30]Modalità il cui obiettivo è eliminare tutti gli avversari o far detonare una bomba posizionata vicino alla base nemica.

vo[31], non venendo ritenuto il comportamento di bot automatici sufficientemente simile a quello umano.

I risultati ottenuti dagli autori hanno mostrato che tramite un approccio evolutivo interattivo collettivo si possono ottenere progressivamente mappe che soddisfano maggiormente gli utenti coinvolti nel processo. La valutazione però si è basata essenzialmente su una considerazione binaria dell'apprezzamento della mappa da parte del giocatore, senza considerare aspetti specifici o dettagli del gameplay.

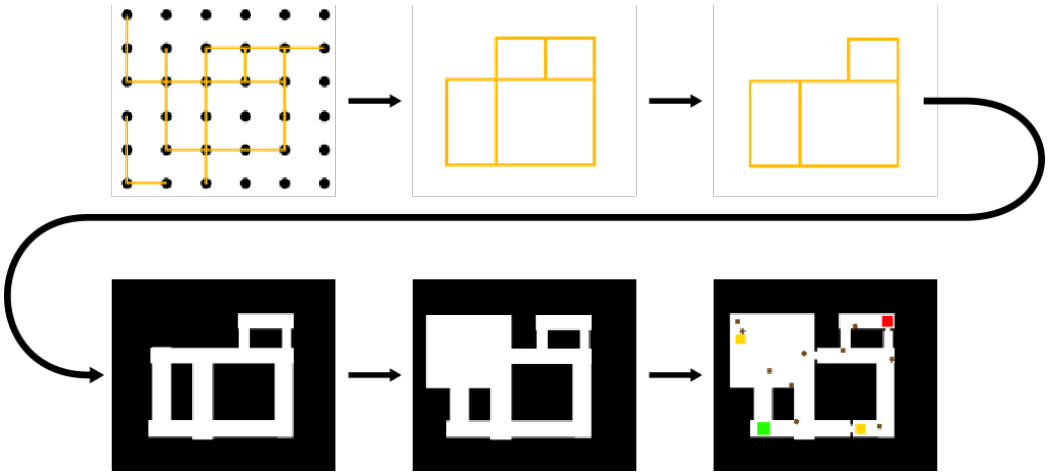


Figura 2.1: Rappresentazione del processo generativo di Ølsted et al. [20]

Altri obiettivi si sono posti invece Bhojan Anand e Wong Hong Wei[21], che hanno sfruttato la generazione procedurale con algoritmi di ricerca con l'obiettivo primario di generare *online*, automaticamente e in breve tempo mappe per sparatutto multigiocatore per la modalità di gioco *Capture and Hold*[32], e che allo stesso tempo fossero comunque giocabili, interessanti e bilanciate. Lo sviluppo delle mappe generate dal metodo di Anand e Wong è basato sull'interconnessione di diverse *tile*[33] predefinite che possono generare zone al chiuso, all'esterno o inac-

[31] Cioè basato sul feedback diretto dei giocatori[19]; nel caso specifico tramite selezione dei migliori candidati da un insieme casuale limitato della popolazione.

[32] Modalità di gioco tratta da Killzone 2 e 3 (Guerrilla Games, 2009, 2011) in cui due squadre competono per il controllo di alcuni punti strategici. Il punteggio di ogni squadra aumenta periodicamente in proporzione al numero di punti controllati, e vince la squadra che raggiunge per prima il limite prefissato.

[33] Letteralmente *piastrelle*, è un sistema basato su blocchi costruttivi fissi che possono essere accostati in modo ortogonale, creando la struttura desiderata. Nato per ovviare ai grossi limiti di memoria del passato, viene tutt'ora ampiamente sfruttato soprattutto per alcuni generi di

cessibili; le mappe vengono poi "pulite" dagli artefatti[34], ne vengono identificate le regioni e vengono create connessioni tra le stesse. Infine vengono posizionati i punti strategici come i punti di respawn, le bandiere e i punti di copertura. In Figura 2.2 viene raffigurato il metodo generativo appena descritto. L'evoluzione avviene tramite mutazione, eliminando e rigenerando una sezione di ogni mappa partendo da una popolazione iniziale di tre mappe casuali e scegliendo, ad ogni iterazione, le tre mappe migliori. Gli autori hanno scelto un approccio *diretto*[35] per la definizione della funzione di fitness, che viene calcolata come somma del punteggio assegnato a quattro fattori ritenuti importanti ricavabili direttamente dalla struttura della mappa. Questi comprendono la *connettività* tra le regioni, il numero dei *punti di collisione*, e il *bilanciamento del posizionamento delle bandiere*, calcolato in base alle loro distanze rispetto ai punti di respawn. Con questo metodo Anand e Wong hanno implementato un algoritmo per la generazione delle mappe che genera proceduralmente e online una mappa in tempi contenuti (nell'ordine di 10 secondi) senza sacrificare aspetti importanti di giocabilità di uno sparatutto multigiocatore. Tuttavia, sebbene il metodo in questione sia stato validato con discreti risultati attraverso prova diretta su un campione di giocatori, in generale l'uso di una funzione di fitness *diretta* limita l'uso degli algoritmi di ricerca: le variabili che definiscono la funzione di fitness devono essere ben definite e consolidate per ritenerle valide ma, come spiegato nella sezione 2.1 non è ancora del tutto chiaro quali e quanti siano gli elementi fondamentali di un buon livello, limitando il metodo alla sola valutazione di quegli aspetti che, essendo ampiamente condivisi, vengono ritenuti essenziali.

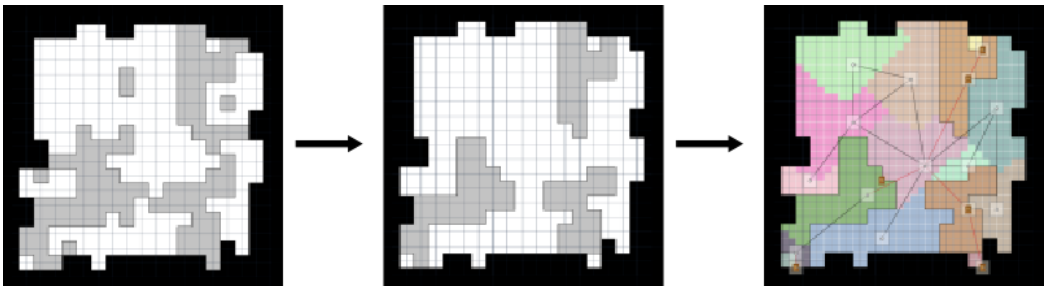


Figura 2.2: Rappresentazione del processo generativo di Anand e Wong. [21]

giochi.

[34]Edifici troppo piccoli o protuberanze.

[35]Cioè misurando direttamente aspetti statici della mappa[19].

2.3 Design degli FPS

I *First Person Shooters* sono un genere di videogiochi d'azione, le cui caratteristiche fondamentali sono una prospettiva in prima persona e un gameplay incentrato sull'uso di armi da fuoco, nei quali i giocatori navigano mondi virtuali sconfiggendo nemici di varia natura per raggiungere un obiettivo. Nel corso degli anni il design e il gameplay di questo genere è cambiato parecchio: se nei primi anni '90 giochi come *Doom*[36] puntavano principalmente su uno stile di gioco veloce e provocatorio, col passare del tempo gli sviluppatori hanno cominciato a sviluppare anche altri aspetti quali la strategia, la tattica e la trama. Si sono così sviluppate diverse scuole di pensiero e anche alcuni sottogeneri e ibridi: serie come quella di *Half-Life*[37] o *Bioshock*[38] mettono in primo piano la storia e l'ambientazione, *Strife: Quest for the Sigil*[39] e *Deus Ex*[40] introducono aspetti dei giochi di ruolo, *Quake III: Arena*[41] e la serie *Unreal Tournament*[42] puntano principalmente sul gioco competitivo, e i giochi cosiddetti *Modern Military Shooters* cercano di riprodurre, in modo realistico, gli scenari di guerra di conflitti storici contemporanei. È dunque indispensabile adeguare la creazione del mondo del gioco anche al suo stile e alle sue caratteristiche.

2.3.1 Evoluzione del Level Design negli FPS

A pari passo con l'evoluzione del genere anche lo stile, gli obiettivi e gli strumenti di design dei livelli si sono evoluti nel corso degli anni:

Prima del 1993: le basi di un nuovo genere

Sebbene non fosse il primo gioco a mostrare nè una prospettiva in prima persona nè le caratteristiche base degli FPS, *Wolfenstein 3D*[43] viene spesso considerato il precursore del genere. Il suo motore non era rivoluzionario rispetto a giochi simili del periodo, ma era progettato per essere leggero, veloce e accessibile a un

[36] id Software, 1993.
[37] Valve Corporation, 1998.
[38] Irrational Games, 2007-2013.
[39] Rogue Entertainment, 1996.
[40] Ion Storm, 2000-2013.
[41] id Software, 1999.
[42] Epic Games, 1999-2007.
[43] id Software, 1992.

pubblico il più vasto possibile, ed è stato questo il motivo del suo successo. *Wolfenstein 3D* era stato sviluppato tenendo conto della mentalità di gioco presente all'epoca: un approccio esplorativo, con molti elementi dei giochi d'avventura quali tesori sparsi per il livello utili ad aumentare il punteggio e passaggi segreti. I livelli erano appositamente progettati per incentivare il giocatore a esplorare ed esaminare quasi ogni muro del gioco, con l'obiettivo di mostrare il più possibile le caratteristiche del gioco e della tecnologia su cui era basato. Le limitazioni del motore però erano molteplici: i muri potevano essere posizionati solo ad angoli retti e avevano un'altezza sempre uguale, i livelli si sviluppavano su un solo piano e non era possibile mostrare un soffitto e un pavimento. Inoltre il design dei livelli era svolto tramite un editor basato su *tile*, e quindi su blocchi di dimensione fissa. Questi aspetti rendevano il gioco in qualche modo simile a molti altri giochi bidimensionali, in particolare ad alcuni tipi di sparatutto *top-down*[44]; l'unica differenza era la prospettiva del giocatore.

1993 - 1996: Doom e il Fattore Divertimento

Doom rappresenta una pietra miliare nella storia degli FPS: ebbe un successo tale da diventaree un vero e proprio fenomeno culturale, consolidando alcuni aspetti già visti in *Wolfenstein 3D* come caratteristiche fondamentali del genere. Il motore di Doom offriva diverse innovazioni: sezioni con pavimento e soffitto ad altezza variabile, ascensori, muri con angolature non ortogonali, pulsanti, leve ed elementi interattivi che modificavano parti del livello, illuminazione, anche dinamica, di parti dello spazio, possibilità di usare texture anche per superfici orizzontali e una rudimentale *skybox*[45], il tutto ereditando la rapidità del motore di *Wolfenstein 3D*. Gli sviluppatori erano consapevoli delle potenzialità del gioco e si posero tra gli obiettivi principali una progettazione dei livelli che ne esaltasse tutti gli aspetti. Anche per motivi commerciali[46], ciò è visibile sin dal primo livello: l'avventura inizia in una coppia di stanze piuttosto squadrate dove già si possono notare, tuttavia, diversi elementi che caratterizzano il gioco e lo stile del level design; soffitti ad altezza diversa, colonne, scale e gradini, e diversi

[44]Vengono definiti *top-down* giochi in cui si ha una prospettiva dall'alto del mondo di gioco.

[45]Una tecnica per creare sfondi che rappresentano spazi in lontananza, dando l'impressione di un estensione maggiore del livello.

[46]Il primo episodio veniva distribuito gratuitamente con la formula dello Shareware, e doveva avere lo scopo di aumentare l'interesse in potenziali acquirenti.

elementi estetici che segnano l'atmosfera cupa e violenta del gioco; sulla destra una finestra aperta permette di visualizzare la struttura a *"ferro di cavallo"*[47] dell'intero livello, dandogli una visione di insieme ma proibendogli di attraversarlo direttamente, con montagne e un cielo grigio a fare da sfondo all'intero edificio che contrariamente ai giochi precedenti, che rilegavano tutta l'azione in spazi chiusi senza una visuale sull'ambiente esterno, dava al giocatore l'impressione di essere immerso in un mondo che esisteva fuori dagli spazi navigabili; attraversata la seconda stanza un corridoio ricurvo, che contrasta con quelli ortogonali di *Wolfenstein 3D*, porta a un ponte a zig-zag che costringe il giocatore a rallentare esponendolo al fuoco nemico; la successiva ed ultima stanza mette il giocatore in una situazione di alta tensione, grazie all'esposizione da entrambi i lati al nemico e a un gioco di luci soffuse che cambiano continuamente di intensità.

Le strutture e il design appena descritto non aveva però solo scopo di "vetrina" del prodotto: si cominciano a vedere alcuni *pattern di design* complessi, ideati con l'obiettivo di incidere in modo ben preciso sul gameplay del giocatore.

Un'altra innovazione portata da *Doom* fu una modalità multigiocatore, che permetteva sia partite in cooperativa che in *Deathmatch*[48]. Le mappe utilizzate, tuttavia, erano le stesse della modalità a singolo giocatore; dati i diversi obiettivi delle due modalità la loro progettazione richiedeva diverse considerazioni, costringendo a compromessi che intaccavano la qualità dei livelli. Questo modello fu infatti presto scartato, in favore di una separazione delle mappe a seconda dello scopo.

Tutti i giochi del periodo erano accomunati da limiti tecnici: movimenti limitati e strumenti di design, primitivi, che costringevano a progettare livelli tridimensionali agendo su una planimetria bidimensionale dall'alto, rendendo difficile rappresentare realisticamente le strutture che aveva in mente il designer. Inoltre ambientazioni realistiche risultavano di scarso interesse e non adatti a un gioco, motivo per cui i giochi dell'epoca tendevano ad avere temi surreali o fantascienti-

[47] Struttura di una stanza o di un livello che permette al giocatore di visualizzarne la destinazione e lo spazio centrale, ma che lo costringe a girarci intorno o a prendere vie laterali più lunghe rendendo inutile o inaccessibile quella centrale diretta. Per ribadire il problema della mancanza di un linguaggio condiviso, questa è una definizione data da John Romero[22], autore del livello, che, nonostante descriva una struttura con uno scopo ben preciso ai fini del gameplay e dunque riutilizzabile in diversi contesti, non trova, definito come tale, altri riscontri nelle poche fonti esistenti sull'argomento.

[48] Termine, che si suppone sia stato coniato proprio dagli sviluppatori di *Doom*, che indica una modalità *tutti contro tutti*, per cui viene spesso preferita la denominazione *Free-For-All*.

fici (come si può vedere dal grafico in Figura 2.3), concentrandosi essenzialmente sul "fattore divertimento" e trascurando aspetti come storia e realismo.

Negli anni seguenti non vi furono grosse innovazioni nei metodi di level design. Anche *Quake*[49], nonostante l'importante impatto sul piano tecnologico, non portò sostanzialmente nulla di nuovo.

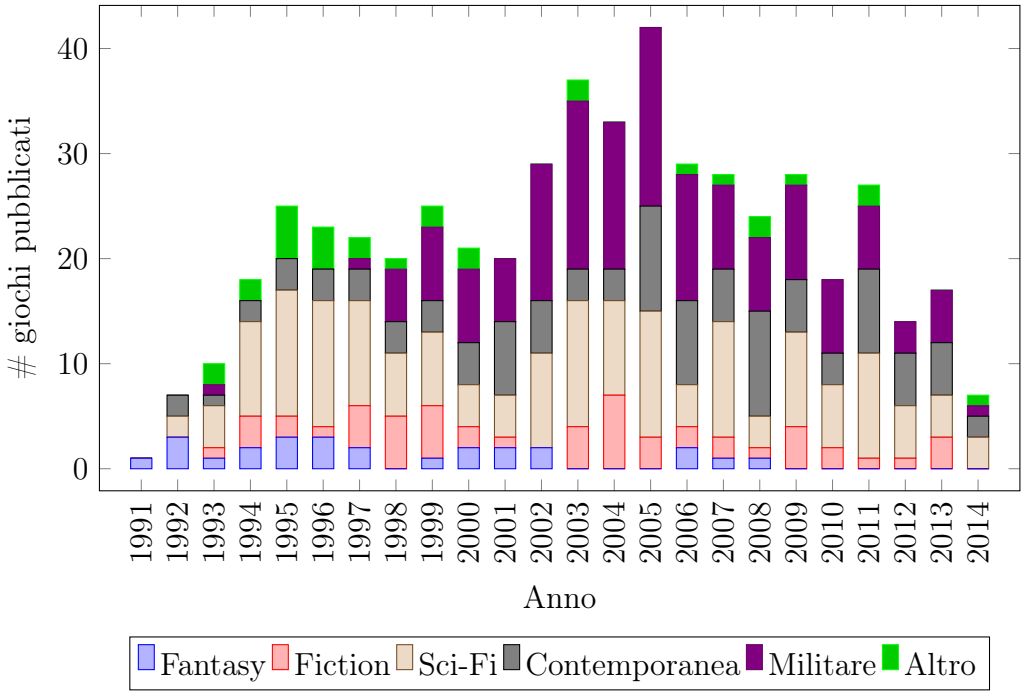


Figura 2.3: Numero di First Person Shooter pubblicati ogni anno, suddivisi per tipo di ambientazione.
Elaborato dalla lista consultabile all'indirizzo `https://en.wikipedia.org/wiki/List_of_first-person_shooters`. Con *Contemporanee* si intendono ambientazioni simili al mondo reale in un contesto storico simile a quello attuale, ma con uno scenario non militare (per cui viene usata la categoria *Militare*). Con *Fantasy*, *Sci-Fi* e *Fiction* si intendono ambientazioni di fantasia, con aspetti rispettivamente medievali, futuristici e di contesti storici remoti.

1996 - 2000: Un periodo di esplorazione

Alla fine degli anni '90 molti sviluppatori cominciarono a portare il genere in altre importanti direzioni. Un primo segno di cambiamento si vide con l'uscita di

[49] id Software, 1996

Duke Nukem 3D[50]. Basato sul motore BUILD[51], che metteva a disposizione un editor di livelli *What You See Is What You Get*[52], segnò un punto di rottura con la precedente tendenza al genere fantascientifico; a differenza di qualsiasi sparatutto precedente gli sviluppatori volevano replicare località reali, simili a quelle di Los Angeles. Grazie anche alla crescente fedeltà grafica si inizia dunque a prediligere ambientazioni più realistiche, *contemporanee* o basate sulla storia recente; si comincia a prestare attenzione ai dettagli, a costruire i livelli incentrandoli su un tema, a rendere gli ambienti più larghi e aperti, e a dare più importanza all'altezza come terzo grado di libertà di gioco. Il motore BUILD permetteva inoltre di applicare degli *script* preimpostati a *settori* di mappe che ne modificavano in modo dinamico la struttura aumentandone l'interattività: era possibile far crollare palazzi, distruggere muri, guidare carri armati o salire su un treno per scendere alla fermata successiva. Il motore però, come quello di *Doom*, non era ancora realmente tridimensionale, e molte di queste novità richiedevano alcuni accorgimenti e stratagemmi da parte del designer per integrarle senza soluzione di continuità con il resto del gioco.

Questo periodo vede anche la nascita e l'affermazione di alcuni sottogeneri e di *"ibridi"*, come *Quake III:Arena* e *Unreal Tournament*, incentrati sul gioco multiplayer, o come giochi che mischiano elementi dei *giochi di ruolo* come *Strife: Quest for the Sigil* e *Deus Ex*, o che introducono elementi di combattimento tattico come *Tom Clancy's Raibow Six*[53] o di gioco furtivo come *Thief: The Dark Project*[54]. Tutte queste nuove categorie richiedevano un cambiamento mirato nello stile del level design: un gioco incentrato sull'azione furtiva necessita di un'accurato studio dell'illuminazione e degli spazi percorribili, un gioco tattico richiede un'alta navigabilità, diverse soluzioni percorribili e coperture, ed elementi da gioco di ruolo significano necessità di alternare sezioni di "azione" ad altre di "riposo", di sviluppo della storia e di interazione con altri personaggi, spesso in un ambiente urbano, e dunque di una maggior capacità di far coesistere gli elementi *scenici* con quelli di gameplay. In questo senso *Deus Ex* spinse i designer al limite:

[50]3D Realms, 1996.

[51]Sviluppato da Ken Silverman nel 1995.

[52]L'editor forniva una modalità di progettazione planare 2D come editor precedenti, e una modalità in prima persona che permetteva di visualizzare immediatamente e modificare direttamente i *settori* della mappa.

[53]Red Storm Entertainment, 1998.

[54]Looking Glass Studios, 1998.

la possibilità di caratterizzare il personaggio sotto numerosi aspetti voleva dire poter avere approcci diversi alle sfide a cui veniva messo di fronte il giocatore, e ciò richiese un'enorme quantitativo di progettazione e pre-produzione dei livelli[23], che tuttavia ha contribuito all'enorme successo del gioco.

In questo periodo è nata un'altra delle serie più importanti nella storia degli FPS: *Half-Life*. Oltre ad avere un ottimo design dei livelli in generale il gioco introdusse diverse novità, divenute ormai uno standard: la transizione dei livelli passò dall'essere una netta separazione a un passaggio fluido da una zona all'altra senza interruzioni, con la possibilità di tornare nelle zone già visitate. Per rendere le transizioni immediate era necessaria un'attenta progettazione a priori dei livelli, al fine di garantire i tempi necessari per caricare la zona successiva in modo dinamico e trasparente durante il gioco. Ciò dava l'impressione al giocatore di essere in un mondo più realistico. Inoltre gli sviluppatori diedero grosso rilievo alla narrazione di una storia, che si sviluppa in un'unica sequenza di gioco senza interruzioni nè scene pre-renderizzate; il designer diventa dunque responsabile anche dei tempi e della "regia" del gioco.

Il passaggio alla completa tridimensionalità segna in questo periodo anche un incremento nella complessità del level design, poichè diventa necessario produrre esternamente praticamente ogni asset del gioco.

Dopo il 2001: L'esplosione sul mercato delle Console

Il genere degli sparatutto in prima persona veniva considerato poco adatto alle *console*, a causa delle loro periferiche di gioco che rendevano gli spostamenti decisamente lenti se paragonati alle velocità che si avevano su PC. La situazione cambiò nel 2001 con l'uscita di *Halo: Combat Evolved*[55], gioco che introdusse nel genere diverse caratteristiche visibili in molti titoli moderni: la limitata capacità di trasportare armi, la salute (o simili) che si rigenera automaticamente e una migliore intelligenza artificiale dei nemici, che cominciano a sfruttare l'ambiente a proprio vantaggio, segnano un passaggio verso approcci più tattici al genere; ciò meglio si sposava ai comandi offerti dalle console rispetto a giochi in cui la capacità e la velocità di puntamento erano gli aspetti principali. Cresce la complessità dei livelli, che necessitano una maggiore progettazione degli spazi, delle coperture e del posizionamento di punti di *check-point* e degli oggetti. Questo, unito alla

[55]Bungie, 2001.

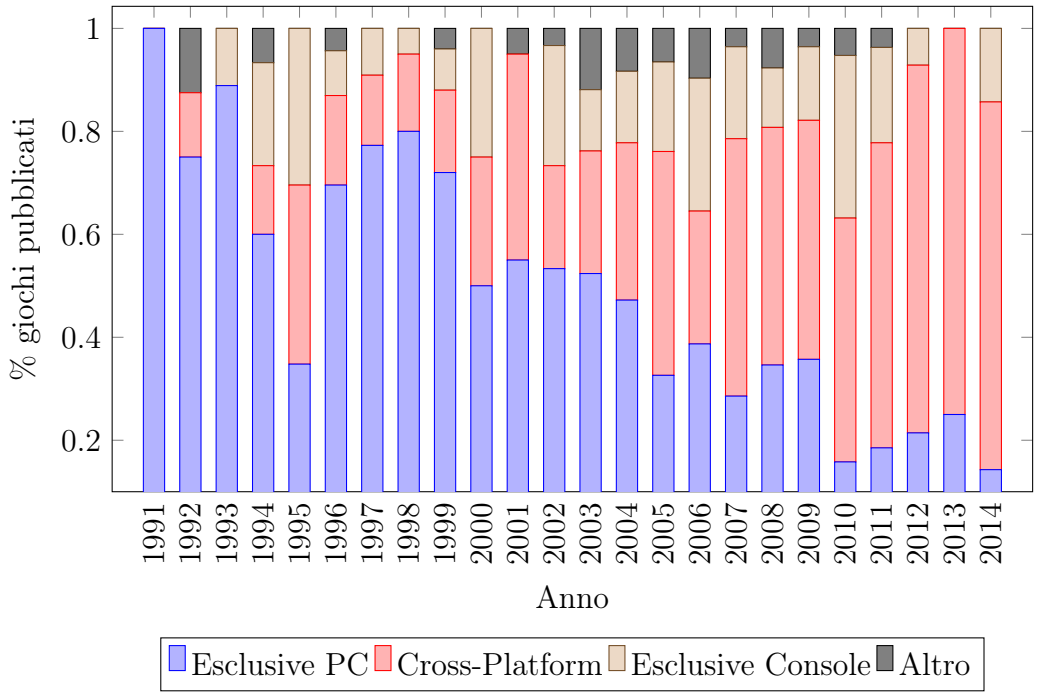


Figura 2.4: Percentuali di First Person Shooter pubblicati ogni anno suddivise per piattaforma.
Elaborato dalla lista consultabile all'indirizzo `https://en.wikipedia.org/wiki/List_of_first-person_shooters`.

sempre crescente qualità grafica, al costo di livelli sempre più lineari[24]: la conseguenza è ovvia, e dovuta al maggior lavoro necessario alla realizzazione di ogni spazio di gioco. La costante crescita della popolarità del genere sulle console, visibile in Figura 2.4 ha causato una spaccatura tra gli appassionati degli sparatutto, con una delle parti critica nei confronti dell'industria per essersi adattata troppo alle necessità di piattaforme e periferiche considerate non idonee, allontanandosi dal modello di gioco originale e rendendolo in qualche maniera troppo semplice.

Il *level design* si è evoluto molto dalla sua origine, ma negli ultimi anni non ha mostrato notevoli progressi; il problema potrebbe essere dovuto a una mancanza di strumenti adeguati, considerato quanto, rispetto ai motori di rendering in generale, l'evoluzione da questo punto di vista sia stata limitata[25].

2.4 Sommario

In questo capitolo abbiamo analizzato lo stato attuale del problema del *level design* per i videogiochi del genere *first person shooter*, identificando i motivi alla base della nostra e di analoghe ricerche sull'argomento. Abbiamo poi introdotto la tecnica della *generazione procedurale dei contenuti con algoritmi di ricerca* e analizzato alcuni lavori che l'hanno impiegata per generare automaticamente mappe per FPS con obiettivi specifici. Infine abbiamo fornito una panoramica storica sull'evoluzione nell'industria videoludica del *level design* per il genere.

Capitolo 3

Intelligenza artificiale in Cube 2

In questo capitolo descriviamo *Cube 2: Sauerbraten*, il videogioco usato per valutare tramite simulazioni le mappe genetate dai processi evolutivi, e le modifiche apportate all'intelligenza artificiale per renderlo più adatto alle nostre esigenze. Nella prima parte viene data una panoramica generale sulle caratteristiche del gioco; viene fornita un'analisi più approfondita sugli aspetti riguardanti l'intelligenza artificiale e alcuni dei problemi in essa riscontrati. Nella seconda parte vengono illustrate le modifiche apportatevi per renderla più realistica e adatta ai nostri scopi, con i relativi test per verificarne l'efficacia. Infine viene osservato quanto le modifiche effettuate incidano sugli equilibri di gioco rispetto al modello originale.

3.1 Cube 2: Sauerbraten

Cube 2: Sauerbraten è un videogioco *First Person Shooter* progettato da Wouter van Oortmerssen, rilasciato nella sua prima versione nel 2004, e aggiornato regolarmente fino al 2013, ed è il risultato dell'evoluzione del motore del suo predecessore, *Cube*. Presenta tutte le caratteristiche fondamentali del suo genere, e due modalità di gioco:

- *Campagna*, giocabile sia in cooperativa che in solitaria, che mette alla prova i giocatori in livelli a sviluppo sequenziale contro una serie di mostri comandati dall'intelligenza artificiale.

- *Bot Match*, una modalità di gioco che mette in competizione più giocatori (alcuni dei quali, o tutti, possono essere comandati a loro volta dall'intelligenza artificiale, rendendo anche questa modalità giocabile sia online multiplayer che offline in singolo giocatore, anche semplicemente in modalità spettatore), in diverse tipologie di sfide tutti contro tutti o a squadre, in mappe chiuse simili ad arene senza continuità di livelli.

Le diverse tipologie di gioco per quest'ultima modalità includono i classici *Deathmatch*, in cui si aumenta il proprio punteggio eliminando gli avversari, e *Capture The Flag*, in cui, per ottenere un punto, è necessario raccogliere una bandiera e riuscire a portarla alla propria base superando la resistenza avversaria. Vi sono anche variazioni meno comuni come *Capture*, in cui i giocatori devono mantenere il controllo di alcune "basi" per vincere, *Protect*, in cui si aumenta il punteggio della propria squadra toccando la bandiera della squadra avversaria, *Hold* in cui si ottiene un punto mantenendo la bandiera per venti secondi, e *Collect* nella quale vengono fatti cadere dei token alla morte di ogni avversario, da raccogliere e portare alla propria base. Infine, per alcune di queste modalità è possibile utilizzare alcuni modificatori come l'*instaGib*, con cui è presente solo l'arma *Rifle* e ogni colpo è letale, *Tactics* e *Efficiency*, con i quali i giocatori rinascono rispettivamente con equipaggiamento casuale o con tutto l'equipaggiamento possibile, e *Regen*, presente in modalità *Capture*, che rende le basi catturate punti di rigenerazione della propria salute.

Anche l'arsenale messo a disposizione dei giocatori è quello "standard", tipico di molti FPS: le armi in possesso alla rinascita di ogni giocatore sono una *motosega*, un arma corpo a corpo senza munizioni, e una *pistola*, l'arma da fuoco base, a danno ridotto e cadenza di fuoco media. Tra le armi più avanzate ci sono un *fucile a pompa*, che spara diversi proiettili in uno spazio a forma di cono, un *lanciarazzi* e un *lanciagranate*, che sparano proiettili esplosivi che seguono rispettivamente una traiettoria lineare e parabolica, un *mitragliatore* e un *fucile*[1], che sparano proiettili simili a quelli della pistola, ma con cadenza di fuoco e danni diversi: nel caso del mitragliatore i danni sono ancora limitati, ma la frequenza è molto maggiore; al contrario, il fucile spara proiettili che causano molti danni,

[1]Per distinguere meglio il *fucile* dal *fucile a pompa* verrà usata la loro nomenclatura inglese originale, rispettivamente *rifle* e *shotgun*.

ma necessita di tempi di ricarica piuttosto lunghi. Le armi disponibili ai giocatori sono le stesse sia in modalità *Campagna* che *Bot Match*.

Il punto di forza di *Cube 2* risiede nella struttura dei suoi livelli, e nel suo editor di mappe, unico nel genere, che permette anche una costruzione dei livelli in cooperativa con altri giocatori. Le mappe sono costituite da *cubi*, ognuno dei quali rappresenta un nodo di una struttura *octree*[2]: ogni cubo può essere diviso ricorsivamente in otto cubi, diventando così radice di un sottoalbero di altri otto nodi. Da ogni cubo è possibile creare o eliminare altri cubi per estrusione o intrusione di una delle facce, ed è possibile modificare la struttura di ognuno agendo sugli spigoli e sui vertici. Unito alla possibilità di alterare la "risoluzione" dei cubi, ciò permette di creare strutture dettagliate che formano scenari complessi in modo semplice, veloce ed economico, rispetto al normale procedimento usato nei moderni giochi. Inoltre la natura *open-source* permette di implementare qualsivoglia modifica al gioco.

Queste ultime caratteristiche, e la presenza di un'intelligenza artificiale di base, sono il motivo principale per cui *Cube 2* è stato scelto come base per le nostre ricerche.

3.2 Panoramica sull'intelligenza artificiale

L'intelligenza artificiale di *Cube 2: Sauerbraten* consiste in un unico modello di comportamento che viene applicato ciclicamente a tutte le istanze di bot che partecipano alla partita. Le azioni eseguite dagli attori vengono determinate da due elementi:

- una routine di **logica** di base comune a tutti bot ed eseguita una volta ogni ciclo di aggiornamento da ognuno, che gestisce tutte le azioni basilari e comuni a tutte le situazioni quali gli spostamenti, la scelta e l'impiego delle armi, il controllo e la raccolta dei pickup;

- una pila individuale di **stati obiettivo**, che determina il comportamento specifico di ogni bot relativamente alla situazione in cui si trova e all'evoluzione di questa, seguendo la routine specifica all'obiettivo di testa finchè

[2]L'*octree* è una struttura dati ad albero in cui ogni nodo ha esattamente otto figli, ideale per rappresentare e partizionare spazi tridimensionali come quelli cubici.

perseguibile; al verificarsi di particolari condizioni viene aggiunto un nuovo obiettivo alla pila o rimosso quello attuale, causando con alta probabilità una reazione a catena (essendo lo stato del gioco cambiato dal momento in cui ad ogni elemento della pila ne è stato impilato un altro) che riporta ad uno stato di base. Gli obiettivi si suddividono in:

- Wait: lo stato base, presente normalmente solamente come fondo della pila e come stato "minimo". Serve solamente a trovare il primo obiettivo utile possibile, nel peggiore dei casi semplicemente un nodo destinazine verso cui spostarsi.
- Defend: è uno stato specifico per partite ad obiettivo[3] per gestire la difesa di punti o artefatti di interesse[4], o dei compagni di squadra che li stanno trasportando.
- Pursue: gestisce la ricerca di un obiettivo, che in partite *Free For All* è semplicemente un nemico, mentre in altre modalità può essere il portatore nemico della bandiera o la bandiera stessa, o una propria base (se si è il portatore di token o della bandiera).
- Interest: valuta la possibilità di raggiungere un nodo di interesse, che tipicamente rappresenta un oggetto considerato utile.

In Figura 3.1 è schematizzata la routine di gestione dell'intelligenza artificale appena descritta.

3.2.1 Limiti del modello originale

Essendo la modalità a singolo giocatore secondaria, e data la natura e gli obiettivi principali dello sviluppo del gioco, l'intelligenza artificiale dei bot guidati dal computer è piuttosto semplice e genera comportamenti distanti da quelli avrebbero giocatori umani; questo porta a dinamiche di gioco poco realistiche e ne risentono in primo luogo l'esperienza di gioco offline, ma anche la qualità dei risultati degli esperimenti da noi effettuati. Per questo motivo si è prima di tutto reso necessario apportare alcune migliorie all'intelligenza artificiale del gioco

[3]Collect, Capture e Capture The Flag.
[4]Possono essere, a secondo della modalità, basi, token, la bandiera, ecc...

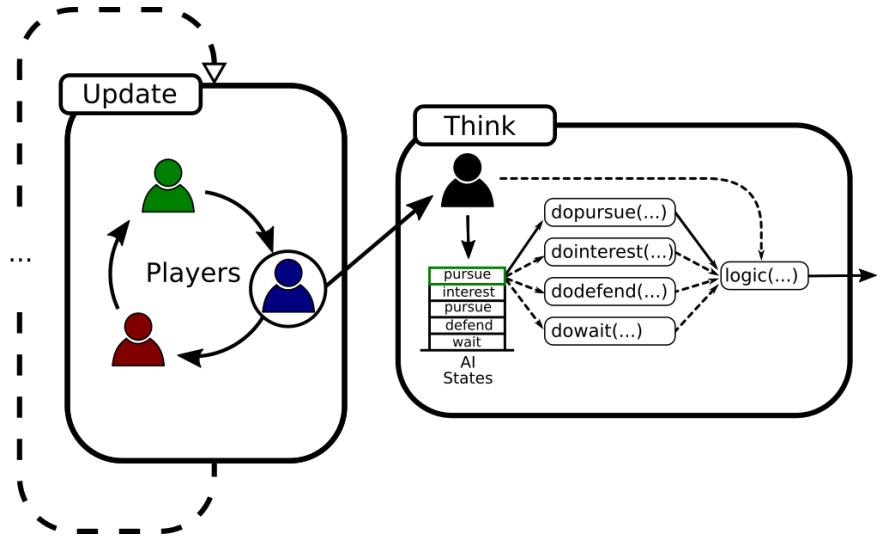


Figura 3.1: Routine dell'intelligenza artificiale.

cercando di sviluppare comportamenti che risultino più naturali e coerenti con lo stato della partita.

Uno dei problemi principali del modello di intelligenza artificiale originale è la presenza di un solo parametro distintivo *skill*, una rappresentazione dell'abilità di gioco generale del bot che copre e scala proporzionalmente tutte le caratteristiche del personaggio; tra gli aspetti direttamente influenzati vi sono:

- la probabilità di considerare un bersaglio entro la gittata massima della propria arma;
- la frequenza e l'errore di calcolo della posizione di mira;
- l'ampiezza e la profondità del campo visivo;
- la propensione a considerarsi in una situazione di "pericolo" in base alla salute rimasta;
- la propensione a considerare un oggetto vicino utile, e ad effettuare una ricerca di un oggetto anche a distanza elevata, se necessario;
- la propensione a saltare;
- la velocità di spostamento della visuale;
- la capacità di scegliere il nemico più vicino come obiettivo;

- la finestra di "memoria"[5] della posizione del nemico;
- la probabilità, a skill basse, di effettuare azioni perfette;

ne consegue un appiattimento della "personalità" dei bot, per cui bot con il medesimo valore di skill saranno identici in tutto e per tutto; a differenza di quanto avviene, per esempio, in giochi simili ma più sofisticati come *Unreal Tournament 2003* (e successivi), in cui ogni personaggio oltre ad avere una propria storia e personalità ha un insieme di attributi che ne caratterizzano lo stile e l'abilità di gioco, non ci potranno essere bot specializzati in una particolare abilità (per esempio esperti nella precisione di tiro ma non altrettanto negli spostamenti). Questo rappresenta un problema se si vogliono analizzare le dinamiche che derivano dal confronto di diversi profili di giocatore.

Un altro dei principali problemi del comportamento dei bot è la loro mancanza di adattamento alla situazione. Vi è in primo luogo una quasi totale assenza di adattamento alle caratteristiche dell'arma utilizzata: ad eccezione della motosega (che, essendo l'unica arma corpo a corpo, richiede alcuni comportamenti ad hoc), il lanciagranate e il lanciarazzi (i cui proiettili non presentano un andamento *hitscan*[6], ed essendo esplosivi hanno un alto fattore di rischio "suicidio"), per le quali vi sono alcuni accorgimenti, il modus-operandi dell'intelligenza artificiale per tutte le armi del gioco è lo stesso. Vengono dunque ignorate tutte le caratteristiche dell'arma in uso (come, per esempio, la cadenza di fuoco, lo spread[7], il volume spaziale coperto dagli spari ecc...) e annullati dunque quelli che potrebbero essere i vantaggi ottenuti usando un approccio accorto rispetto allo stato della partita; una volta entrati in una modalità "aggressiva" i bot tendono sempre a cercare lo scontro diretto, ravvicinato, attaccando senza molto criterio il nemico non appena l'arma è pronta a sparare. Si nota immediatamente qual'è il problema di tale comportamento mettendo a confronto un bot dotato di *rifle* (un arma lenta ma potente che garantisce un *frag*[8] al più dopo un paio di colpi a segno, con *spread* quasi nullo ma che richiede un'alta accuratezza da

[5] Per quanto tempo viene considerata conosciuta la posizione dell'obiettivo anche se fuori dal campo visivo.

[6] Vengono denominate *hitscan* le armi che calcolano immediatamente il punto di impatto del proiettile come la prima intersezione tra il raggio della sua traiettoria e un oggetto del gioco.

[7] L'angolazione della traiettoria del proiettile rispetto all'asse di mira.

[8] Termine sinonimo di uccisione, usato nei videogiochi e soprattutto negli sparatutto in prima persona.

parte del giocatore per essere efficace, utile dalla lunga distanza) con uno dotato di *shotgun* (un arma che spara un alta quantità di proiettili con alto spread in un volume conico, quasi inutile dalla lunga distanza ma molto efficace a distanza ravvicinata anche con una mira non perfetta): il bot dotato di rifle non sfrutterà le caratteristiche della propria arma ma andrà a rendere più efficace l'arma dell'avversario andando a combattere a corta distanza, garantedogli una situazione di vantaggio nella maggior parte delle situazioni; vantaggio che si assottiglia all'aumentare della precisione, direttamente proporzionale al parametro di skill, che per valori alti rende la mira quasi perfetta creando situazioni in cui quasi ogni colpo, sparato nell'istante in cui un avversario entra nel campo visivo, viene messo a segno. Qui sorge un altro dei problemi dell'IA: la gestione della mira, che viene descritta nel dettaglio nella sezione 3.4. Di seguito verranno illustrate le modifiche apportate al modello base per rendere l'intelligenza artificiale più realistica e plasmabile.

3.3 Parametrizzazione

La prima modifica che abbiamo apportato all'intelligenza artificiale è stata la scomposizione del parametro *skill* in parametri che influenzano alcuni degli aspetti che abbiamo considerato più significativi, mantenendo comunque quest'ultimo come un valore base generale dell'abilità del bot; i seguenti parametri influenzano direttamente aspetti già presenti nel modello originale:

- Aim (Mira): influenza l'abilità e la velocità con cui si ricava il punto esatto dell'obiettivo da mirare, e i tempi di reazione all'avvistamento di un avversario;
- Eyespeed (Velocità di Visualizzazione): influenza la velocità di spostamento della visuale;
- View (Visibilità): cambia l'estensione del campo visivo;

A questi si aggiungono due ulteriori parametri che influenzano nuovi aspetti introdotti tra le ottimizzazioni apportate che verranno spiegate nel dettaglio nelle sezioni successive di questo capitolo:

- Distancing (Distanziamento): influenza la capacità di tenersi ad una distanza ideale dall'avversario, a seconda dell'arma usata;

- Combat Skill (Abilità di Combattimento): determina quanto tempo il bot rimane fermo per prendere la mira;

3.4 Modello di mira

La routine di mira originale consiste semplicemente nel ricavare la posizione dell'obiettivo aggiungendovi, calcolandolo ad intervalli di periodo inversamente proporzionale all'abilità, un offset radiale anch'esso proporzionale che determina l'errore di puntamento. Tale errore rappresenta direttamente l'unico fattore distintivo tra la mira di un bot ad alta skill e quella di uno a bassa skill, e, ad esclusione di un fattore di scala costante dipendente dall'arma in uso, non tiene conto in alcun modo nè delle altre caratteristiche dell'arma stessa, nè degli spostamenti degli avversari, causando anche errori improbabili in situazioni nelle quali, realisticamente, difficilmente si verificherebbero (nella situazione più estrema, quando l'obiettivo è immobile tale casualità porta, alle volte, a ripetuti colpi a vuoto). Inoltre questo metodo annulla parzialmente gli effetti che particolari strutture del livello causerebbero alla mira e alla precisione dei giocatori.

3.4.1 Velocità di visualizzazione

Sebbene non sia un aspetto ad essa direttamente riconducibile, in pratica l'accuratezza della mira viene in gran parte influenzata dalla velocità di spostamento della visuale, anch'essa proporzionale all'abilità del bot: una volta visualizzato e scelto un avversario da aggredire i bot cominciano ad attaccare non appena l'arma è carica, indipendentemente dalla distanza del puntatore dall'effettiva posizione calcolata del bersaglio; la minore accuratezza di un bot a skill limitata è dovuto soprattutto a questo ritardo di puntamento. Questo aspetto, particolarmente svantaggioso per bot con abilità medio-bassa, è poco compatibile con la velocità di spostamento di un giocatore umano, anche neofita del genere, e genera un grosso difetto rispetto a bot con skill alta che raggiungono una mira quasi perfetta sia nei tempi che nel calcolo dell'errore di puntamento. È stata quindi drasticamente aumentata la velocità di spostamento della visuale, in modo da ridurre il divario tra i bot a bassa e quelli ad alta abilità, e resa regolabile tramite il parametro *Eyespeed* illustrato nella sezione 3.3. Questo sposta la gestione della mira principalmente sull'identificazione del punto obiettivo verso cui sparare.

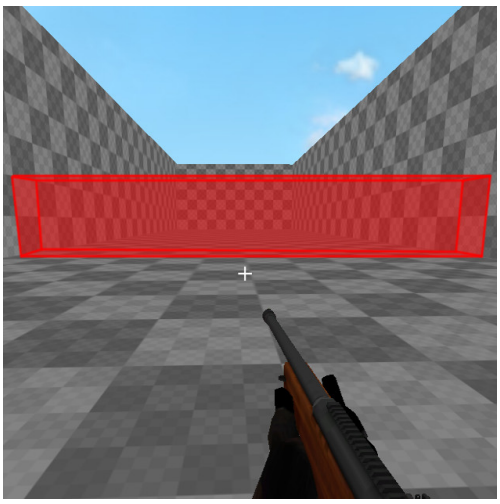
3.4.2 Metodo di puntamento

Le fondamenta del nuovo metodo di mira si basano su un concetto di "riflessi" e reattività agli spostamenti dell'avversario: ogni attore registra una scia degli ultimi spostamenti effettuati nell'arco dell'ultimo mezzo secondo, e la mira segue, con più o meno ritardo, tale scia. La posizione dei bot viene registrata ogni 16 ms (cioè una volta per fotogramma, ad una velocità di 60 fps[9]) in un buffer circolare che contiene lo storico degli ultimi spostamenti. L'errore di mira si traduce nella scelta di un offset rispetto all'indice di testa del buffer (indice della posizione più recente del bot). La scelta dell'indice avviene casualmente seguendo una distribuzione normale, con media e varianza dipendenti dagli attributi del bot: al diminuire della sua abilità cresceranno sia il valore medio, sia la varianza. Questo causa una ovvia concentrazione del ritardo su valori più alti quando la skill si abbassa, ma anche una maggiore casualità (inglobando così un aspetto presente nell'IA originale per cui bot con skill basse hanno una certa probabilità di "stupire" con azioni perfette). L'errore finale viene calcolato interpolando le coordinate delle posizioni registrate relativamente all'indice di ritardo ottenuto.

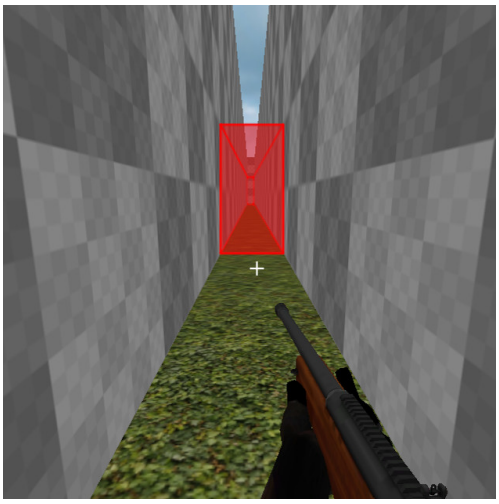
Come primo effetto di questa modifica si dovrebbe avere un incremento dell'accuratezza in spazi lunghi e stretti, poichè spostamenti tendenzialmente longitudinali si proiettano nello stesso punto sul piano di visualizzazione, come illustrato in Figura 3.2a: di conseguenza seguendo la scia dei movimenti anche un errore rilevante della mira, sull'asse longitudinale, porta il puntatore nei pressi della posizione reale dell'obiettivo, a differenza del modello originale per cui l'errore poteva casualmente verificarsi anche sull'asse trasversale e verticale.

Per verificare questa ipotesi e l'efficacia della modifica abbiamo eseguito un test mettendo a confronto due bot con uguale valore di *skill* ambedue dotati di *rifle* (l'arma che richiede la maggior precisione nel gioco) su due mappe di prova costruite appositamente in modo altamente stereotipato: la prima, denominata *Arena*, costituita da una stanza larga e aperta, senza ostacoli se non a protezione dei punti di respawn; la seconda, *Corridor*, da un unico corridoio lungo e stretto. Abbiamo dunque calcolato la differenza di prestazioni di una mappa rispetto all'altra (con caratteristiche diametralmente opposte) partendo da una situazione di equilibrio in una delle due, e confrontato i risultati ottenuti usando

[9]60 frames per second, è considerata la frequenza di aggiornamento dell'immagine standard a buone prestazioni.



(a) Spazio di mira in ambienti larghi



(b) Spazio di mira in ambienti stretti

Figura 3.2: Proiezione degli spostamenti sul piano di visualizzazione.

prima il modello di mira originale e poi quello ottimizzato. Sono state eseguite una serie di partite preliminari di prova nella mappa "Arena", al fine di regolare l'attributo di mira in modo tale che i bot, con e senza modifica, avessero un simile risultato di accuratezza mantenendo invariate tutte le altre abilità[10]; usando gli attributi di mira così ottenuti sono state simulate una serie di partite nella mappa "Corridoio" e confrontate le statistiche di accuratezza dei bot con e senza modifica, intese come la percentuale di colpi andati a segno rispetto a quelli sparati; successivamente è stato realizzato un secondo test, di controprova: sono stati equilibrati i livelli dei bot sulla mappa "Corridoio" e verificate le differenze sulla mappa "Arena". In Tabella 3.1 sono mostrati i parametri usati per i test, i risultati ottenuti come media dell'accuratezza ottenuta in 10 partite della durata di 10 minuti e le differenze percentuali tra i bot e le mappe. I risultati, mostrati nella tabella, mostrano che il nuovo modello di mira garantisce un miglioramento di precisione tra circa il 58% e il 190% [11] nella mappa "Corridoio" rispetto al modello di partenza, confermando l'ipotesi di partenza. Viceversa, partendo da una situazione di equilibrio nella mappa "Corridoio", le prestazioni del nuovo modello nella mappa "Arena" sono di conseguenza inferiori al modello base. La differenza

[10] Volendo verificare la sola capacità di ottenere un buon punto di mira i test sono stati eseguiti impostando la velocità di spostamento della visuale al massimo. In questo modo vengono filtrati fattori che influenzano indirettamente l'accuratezza.

[11] Nella gamma di valori confrontabili relativamente ai valori di accuratezza minimi e massimi ottenibili nei due modelli.

tende ad assottigliarsi all'aumentare dell'abilità, poichè la mira del modello originale tende alla "perfezione" per valori alti di *skill*, come si vede anche in Figura 3.3.

Inoltre si può notare come la differenza nella struttura del livello influenzi maggiormente il nuovo modello rispetto all'originale.

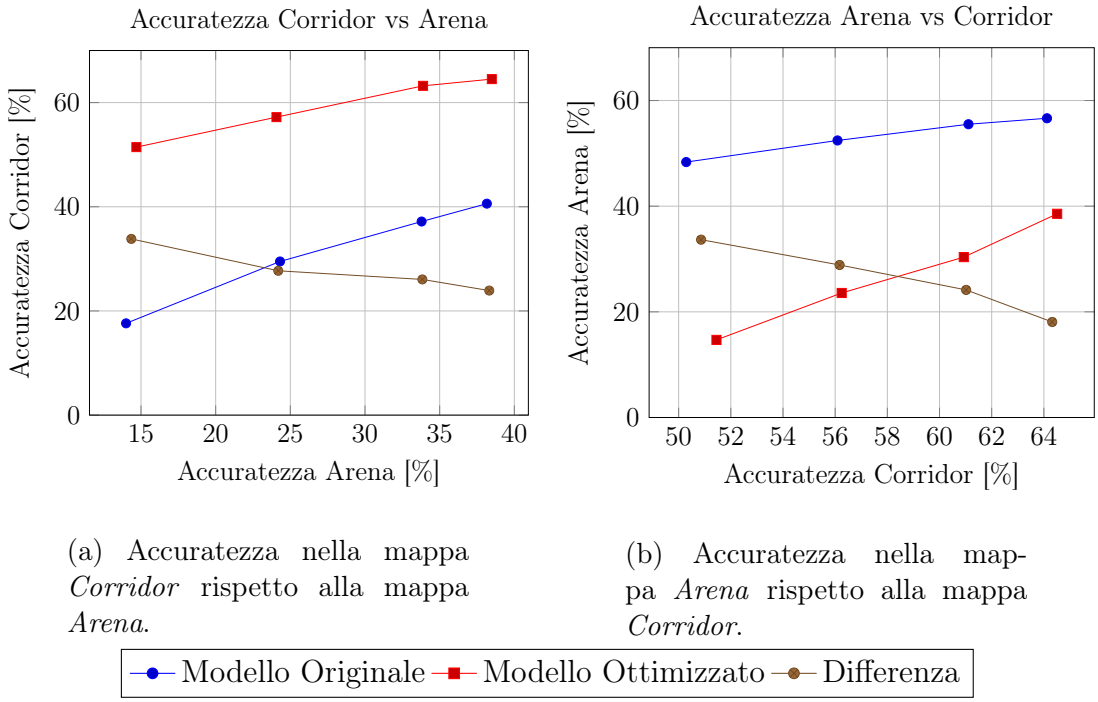
		Mappa				
		Arena		Corridor		
Ottimizzazione	Mira	Precisione %	$\Delta Acc._{\text{Mod}}(\%)$	Accuracy %	$\Delta Acc._{\text{Mod}}(\%)$	$\Delta Acc._{\text{Map}}(\%)$
No	6	14,00	+4,9	17,62	+192,0	+25,6
Sì	5	14,69		51,45		+250,2
No	8	24,31	-1,0	29,15	+96,3	+18,9
Sì	60	24,07		57,21		+137,7
No	10	33,79	+0,3	37,17	+70,1	+10,0
Sì	88	33,89		63,22		+86,5
No	12	38,16	+0,9	40,60	+58,9	+6,4
Sì	95	38,49		64,51		+67,6
No	15	48,35	-69,6	50,29	+2,3	-3,9*
Sì	5	14,69		51,45		-71,4*
No	18	52,44	-55,1	56,09	+0,3	-6,5*
Sì	55	23,57		56,25		-58,1*
No	20	55,51	-45,3	61,11	-0,3	-9,2*
Sì	80	30,36		60,93		-50,2*
No	22	56,64	-31,9	64,12	+0,6	-11,7*
Sì	95	38,56		64,51		-40,2*

*Differenza percentuale dell'accuratezza nella mappa Arena rispetto alla mappa Corridor, al contrario delle precedenti.
Risultati ottenuti impostando il valore di *skill* a 80 e la massima velocità della visuale.

Tabella 3.1: Confronto accuratezza con vecchio e nuovo modello di mira

3.4.3 Prevedibilità degli spostamenti

La precedente modifica non risolve un altro problema del modello di mira: spostamenti continui dell'avversario nella stessa direzione, soprattutto trasversali rispetto alla direzione di visualizzazione che a un occhio umano risulterebbero piuttosto prevedibili, rendono banale la schivata dei proiettili; la velocità di spostamento della visuale del modello originale non è sufficiente a colmare la distanza tra il reale obiettivo e il punto verso cui si sta mirando, mentre con le modifiche appena descritte gli spostamenti nella stessa direzione causano un ritardo medio di mira molto alto, e quindi una bassa precisione. È stato dunque necessario aggiungere fattore di correzione per diminuire, o aumentare, l'errore di mira in



(a) Accuratezza nella mappa *Corridor* rispetto alla mappa *Arena*.

(b) Accuratezza nella mappa *Arena* rispetto alla mappa *Corridor*.

Figura 3.3: Variazione precisione di tiro in ambienti differenti.

base all'imprevedibilità degli spostamenti dell'avversario, determinata da come vengono effetuati i cambi di direzione.

Per misurare il fattore di imprevedibilità viene calcolata la serie dei vettori degli ultimi spostamenti, misurando l'ampiezza dell'angolo tra ogni coppia di vettori contigui come raffigurato in Figura 3.4. Ogni spostamento genera un valore di imprevedibilità proporzionale alla misura dell'angolo rispetto allo spostamento successivo, ottenendo valori massimi per spostamenti opposti e nulli per spostamenti nella stessa direzione.

Ogni spostamento viene pesato relativamente al ritardo rispetto all'istante attuale, e la loro media costituisce il valore di imprevedibilità: spostamenti continui di un giocatore nella stessa direzione producono vettori di spostamento simili tra loro e un basso valore di imprevedibilità, riducendo l'errore di mira nei confronti di quest'ultimo; frequenti cambi di direzione producono, viceversa, un alto valore di imprevedibilità. Il calcolo di tale valore è riassunto dall'equazione 3.1, dove $\mathbf{v_i}$ è l'i-esimo vettore spostamento rispetto al più recente, e n è il numero di spostamenti considerati.

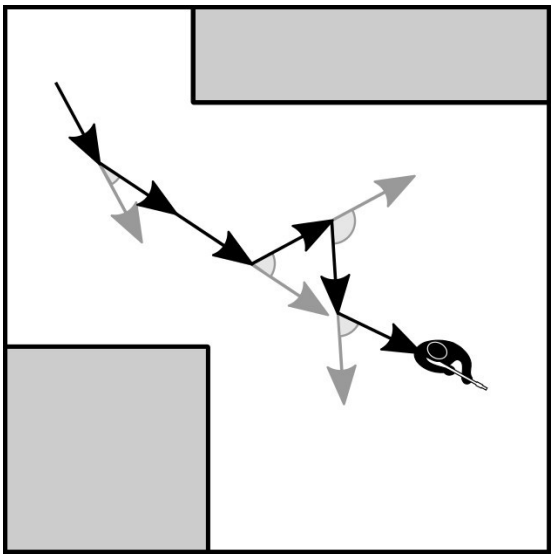


Figura 3.4: Raffigurazione del metodo di calcolo della prevedibilità degli spostamenti.

$$y = \sum_{i=0}^{n-1} \frac{(\mathbf{v_i} \cdot \mathbf{v_{i+1}} - |\mathbf{v_i}|\,|\mathbf{v_{i+1}}|)\,(1 - \frac{1}{n}i)}{2(n-1)}. \tag{3.1}$$

3.4.4 Penalizzazioni alla mira

Abbiamo aggiunto alcune penalizzazioni alla mira che possono essere causate dalla struttura della mappa, cosicchè influisca in modo maggiore sulle prestazioni dei bot durante il gioco. Tali modifiche comprendono

- l'aggiunta di un ritardo di reazione (tra 125 ms e 450 ms, casuale e dipendente dall'abilità del bot) rispetto all'ingresso dell'avversario nel campo visivo: evita che i bot reagiscano in modo troppo rapido all'entrata nel campo visivo di un avversario, dando così una maggiore importanza agli ostacoli presenti nel livello;

- l'aggiunta di un ritardo di reazione (tra 100 ms e 350 ms, casuale e dipendente dall'abilità del bot) rispetto all'uscita dell'avversario dal campo visivo: introduce un lasso di tempo entro il quale i bot possono ancora sparare nonostante l'uscita dell'avversario dal proprio campo visivo, aumentando anche in questo caso l'impatto degli ostacoli presenti nel livello sull'accuratezza dei bot;

- aggiunta di una penalità di mira dalla cortissima o lunga distanza: simula la maggior difficoltà a mirare soggetti in movimento dalla corta distanza, e si evita eccessiva precisione dalla lunga distanza;

3.5 Miglioramento Armi a Proiettile

Ulteriore attenzione è stata posta sulla mira delle cosiddette armi a *proiettile lento*, tenendo in particolare attenzione le caratteristiche del lanciarazzi. Avendo un tempo di viaggio piuttosto alto i proiettili di questa tipologia di armi sono facilmente schivabili; una strategia di mira come quella considerata finora (sia quella base, sia quella modificata appena descritta) produrrebbe quindi scarsi risultati, poichè basterebbe uno spostamento minimo per uscire dalla traiettoria del proiettile. Di conseguenza armi di questo tipo diventano poco utili per scontri dalla lunga distanza, e comunque inefficaci anche da una distanza media se non utilizzate adeguatamente.

A seconda della cadenza di fuoco, della velocità e degli eventuali effetti secondari dei proiettili la migliore strategia, in generale, è quella di coprire di proiettili una certa porzione di spazio intorno all'avversario, in modo da limitarne gli spostamenti sicuri, o di farne una previsione sulla posizione futura relativamente agli ultimi spostamenti. In aggiunta, se l'arma è a danno radiale[12], la strategia più efficace è quella di sfruttare l'area d'effetto intorno al punto d'impatto del proiettile piuttosto che cercare un colpo diretto; si tende, di conseguenza, a mirare verso un corpo immobile il più vicino possibile all'obiettivo, tipicamente verso il pavimento, ai piedi del bersaglio. È stato dunque aggiunto un comportamento ad-hoc per le armi di questo tipo presenti nel gioco, che aggiunge un'ulteriore correzione alla mira inizialmente calcolata: seguendo gli ultimi spostamenti dell'avversario ne viene fatta una previsione sulla posizione futura, tenendo conto anche del tempo di viaggio del proiettile e della distanza tra i due attori coinvolti; alla posizione finale ottenuta viene aggiunto un offset negativo sull'asse verticale. In Tabella 3.2 sono visibili i risultati di alcuni test effettuati mettendo a confronto due bot con stesso livello di abilità usando tutte le modifiche finora descritte

[12]Ovvero non danneggia solo l'oggetto impattato, ma tutto ciò che è presente entro un certo raggio.

salvo l'ultima, attivata solo per uno dei due, effettuando alcune partite di prova su mappe con diverse caratteristiche e a diversi livelli di abilità dei bot:

		Mappa					
		Arenas		Maze		Fragplaza	
Abilità Bot	Ottimizzazione	Score	$\Delta Score(\%)$	Score	$\Delta Score(\%)$	Score	$\Delta Score(\%)$
25	No	20,3	+42,36	18,6	-3,23	14,8	+17,57
	Sì	28,9		18,0		17,4	
50	No	19,8	+83,33	18,9	-2,12	17,7	+28,25
	Sì	36,3		18,5		22,7	
75	No	16,0	+157,50	25,7	+1,17	19,1	+51,83
	Sì	41,2		26,0		29,0	
100	No	19,3	+144,04	28,8	+23,26	21,2	+51,42
	Sì	47,1		35,5		32,1	

Tabella 3.2: Confronto prestazioni di utilizzo del lanciarazzi con e senza ottimizzazione

Nella mappa *Arenas*, formata da quattro stanze larghe connesse da strettoie nella quale è possibile sfruttare spazi ed effettuare spostamenti ampi e laterali, si può notare un drastico miglioramento delle prestazioni; la facilità di movimento rende facilmente schivabili i proiettili se indirizzati direttamente sul bersaglio e, di conseguenza, ci si aspettava un netto miglioramento dovuto alle ottimizzazioni.

Al contrario nella mappa *Maze*, formata da una serie di corridoi contorti, stanze ristrette e ridotta possibilità di movimento la possibilità di speculare sugli spostamenti dell'avversario è limitata; il vantaggio ottenuto è praticamente nullo, salvo per valori estremi del parametro skill per i quali rimane comunque limitato se confrontato con i risultati ottenuti negli altri casi.

I risultati ottenuti su una mappa presente nel gioco, *fragplaza*, seguono lo stesso trend e rientrano prevedibilmente nei valori ottenuti nelle due mappe precedenti, costruite appositamente per testare situazioni estreme.

3.6 Movimenti e gestione della distanza

Una volta visualizzato un bersaglio i bot entrano nello stato *"pursue"*, una modalità di inseguimento dell'avversario in cui viene cercato sempre lo scontro diretto a breve distanza. Di conseguenza, come accennato in precedenza, i bot tendono a non sfruttare alcuni dei punti di forza della propria arma, e, potenzialmente,

ad accrescere il vantaggio dell'avversario. I comportamenti degli attori risultano poco efficaci, e le prestazioni dipendono essenzialmente sulla sola capacità di mira.

Il modello navigazionale originale usato per gli spostamenti nella mappa consiste nella selezione ad intervalli regolari, o conseguentemente a un determinato evento, di un nodo contenente un obiettivo di interesse (che può essere un oggetto da raccogliere), di un nodo vicino al nemico o di un nodo casuale; ad ogni aggiornamento dello stato dei bot la posizione viene aggiornata per raggiungere tale nodo.

La gestione della fase di combattimento è stata ottimizzata aggiungendo un nuovo stato denominato *tactical* ai quattro già presenti[13], che sostituisce lo stato *pursue* quando il nemico è visibile ed entro una certa distanza. Nello stato *tactical* i nodi vengono classificati in base all'appartenenza ad aree considerate vantaggiose o meno, e selezionati, favorendo quelli considerati migliori, in modo analogo a quanto accade nello stato *pursue*. La classificazione delle aree avviene relativamente alla posizione e alla distanza dell'avversario:

- Viene considerata *svantaggiosa* l'area che contiene nodi che causano un avvicinamento significativo all'avversario, relativamente alle caratteristiche dell'arma; a questo scopo è stato necessario aggiungere un attributo alle caratteristiche delle armi che rappresenta la distanza ideale a cui tenersi dall'avversario. Per determinare la distanza ideale per ogni arma è stata eseguita una serie simulazioni nella mappa *Arena*, una mappa composta da un'unica e ampia stanza vuota già utilizzata per gli esperimenti illustrati nella sezione 3.4.2. I test sono stati effettuati con tutte le altre ottimizzazioni illustrate in questo capitolo, per ogni combinazione di armi e per diversi valori di distanza appositamente fissati a cui i bot devono tenersi rispetto all'avversario, compatibilmente con gli spazi disponibili. Per ogni coppia di armi si è calcolato il rapporto dei punteggi finali dei due bot, normalizzato in base al miglior valore ottenuto in ogni serie di distanze così da avere un punteggio compreso nell'intervallo $[0, 1]$ per ogni combinazione. Come distanza ideale di ogni arma si è quindi scelto un valore vicino a quello che ha prodotto, in media, il miglior risultato contro tutte le altre armi. In Figura 3.5 sono mostrati i punteggi medi ottenuti per alcune

[13] Si veda la sezione 3.2.

delle armi[14] in relazione alle distanze impostate, e in Tabella 3.3 le distanze alle quali si sono ottenuti i migliori punteggi e quelle impostate come parametri dell'IA, ridotte per permettere parzialmente spostamenti oltre i limiti ottimi misurati. L'area svantaggiosa viene quindi determinata come l'unione di un'area circolare intorno al nemico, con diametro calcolato in base alla distanza ideale impostata per l'arma in uso con l'aggiunta di un errore casuale e calcolato ad intervalli di un secondo basato sull'abilità del bot, e del semispazio posteriore al nemico rispetto all'attuale posizione, per evitare che vengano selezionati nodi che costringono il bot ad attraversare lo spazio vicino all'avversario, annullando l'effetto cercato.

- Viene considerata *vantaggiosa* un'area circolare intorno al nemico a una distanza maggiore rispetto a quella considerata svantaggiosa ed esterna a questa. Lo spazio vantaggioso assume così una forma ad anello che favorisce gli spostamenti laterali e lo *strafing*[15].

- L'area restante viene considerata *neutrale*.

Arma	Migliore distanza ottenuta nei test	Distanza ideale impostata
Shotgun	37.5	25
Rocket Launcher	150	100
Rifle	562.5	512.5

Tabella 3.3: Distanze ideali di combattimento.

In Figura 3.6 viene mostrata una rappresentazione delle aree che si ottengono con le suddette considerazioni.

In una prima fase la ricerca di un nodo valido avviene esclusivamente nell'area vantaggiosa, e in caso di fallimento questa viene estesa per include parte dell'area neutrale. Se anche nell'area neutrale non si trova un nodo valido la ricerca ripiega sul tipico stato *pursue* ed eventualmente sullo stato *wait* che considerano insiemi più ampi di nodi validi. Lo stato *pursue*, inoltre, mantiene il compito di ricer-

[14]Sono mostrati solo i grafici relativi alle armi usate negli esperimenti da noi effettuati nel capitolo 5.

[15]Lo *strafing* e il *circle strafing* sono tecniche di base fondamentali negli FPS che permettono ai giocatori di schivare i proiettili mantenendo la mira sul bersaglio.

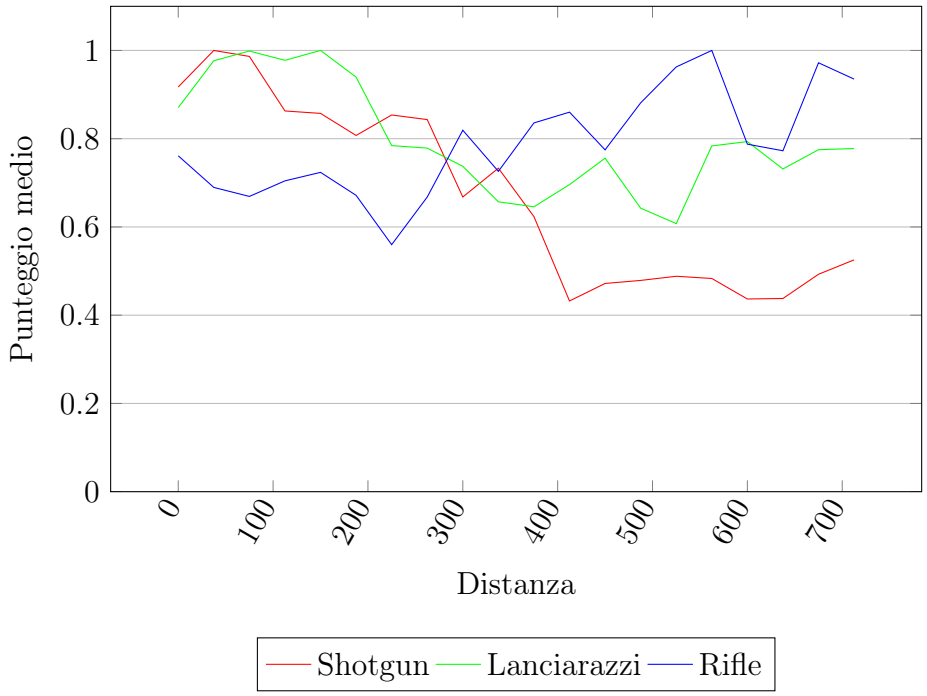


Figura 3.5: Punteggi medi delle armi a diverse distanze.

ca dell'obiettivo qualora questo dovesse uscire dal campo visivo o allontanarsi troppo.

3.7 Conoscenza della posizione degli avversari

Generalmente i bot ottengono la posizione dell'avversario solo quando quest'ultimo entra nel loro campo visivo. Tale conoscenza viene mantenuta anche in caso di perdita della linea visiva con l'obiettivo, per una finestra temporale di circa 3 secondi minimi, che aumentano in proporzione all'abilità del bot skill. Vi sono tuttavia situazioni in cui un bot ottiene la posizione di un avversario anche senza avvistamento:

- in situazioni di prolungata inattività (si è nello stato *wait*) viene ottenuta la posizione di un avversario per entrare in uno stato differente e rendere il bot meno passivo;

- successivamente a un'uccisione, la finestra temporale della "memoria" e l'obiettivo, sia dell'artefice che della vittima, non vengono reimpostate; di

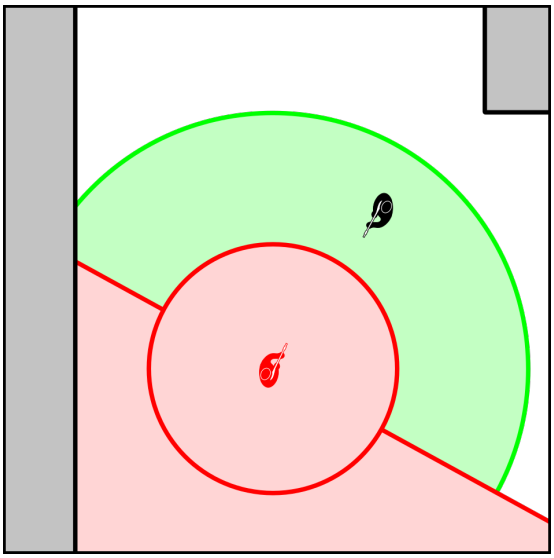


Figura 3.6: Raffigurazione del calcolo dell'area di spostamento nello stato *tactical*. L'area *vantaggiosa* è mostrata in verde mentre quella *svantaggiosa* in rosso.

conseguenza nell'istante del respawn entrambi conoscono la reciproca posizione, e si riportano in una situazione di conflitto con una frequenza molto alta, soprattutto in mappe piccole o con molta visibilità;

Sebbene da una parte non portino vantaggi o svantaggi ai bot nei combattimenti, evidentemente questi aspetti influenzano lo svolgimento delle partite. Ciò contrasta con i nostri obiettivi poichè alcune delle statistiche da noi raccolte, basate sui tempi di ricerca e combattimento e sul ritmo del gioco, verrebbero condizionate. È tuttavia vero che alcuni giocatori di alto livello potrebbero prevedere gli spostamenti di un avversario non più visibile o la posizione di *respawn* di un giocatore appena ucciso affidandosi alla propria esperienza. Tenendo conto di quanto appena detto sono state rimosse le cause che portano a una conoscenza ingiustificata (cioè senza diretta visualizzazione) della posizione di un avversario. È stato però aumentato il tempo necessario a perdere definitivamente l'obiettivo, e aggiunto un fattore casuale che, a livelli alti di abilità, permette di ottenere la posizione di un nemico appena ucciso.

3.8 Analisi degli equilibri

Per verificare come le modifiche effettuate abbiano influenzato l'equilibrio delle partite abbiamo simulato una serie di partite della durata di 10 minuti, ciascuna tra due bot, per ogni combinazione di livello di skill e per ogni arma sulla mappa *fragplaza*, una mappa semplice ma che contiene sezioni di diversa natura con strutture simili a quelle usate nelle mappe di prova finora usate. Abbiamo quindi confrontato e analizzato i risultati ottenuti usando l'intelligenza artificiale originale e quella ottimizzata.

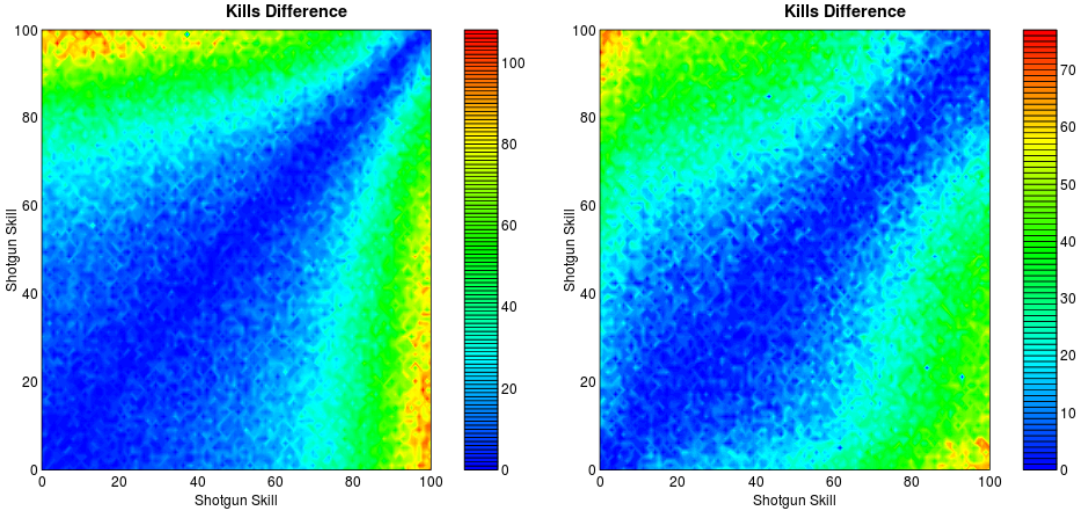
Nelle figure dalla 3.7 alla 3.15 si possono osservare i grafici dei risultati, relativi alla differenza e al rapporto[16] delle uccisioni dei due giocatori. Ogni punto nel grafico rappresenta la differenza o il rapporto di punteggio ottenuto mettendo a confronto due bot i cui valori di *skill* sono disposti sui due assi.

Dai grafici si può notare come, nel complesso, usando il nuovo modello sia diminuita la quantità totale delle uccisioni. Inoltre uno dei problemi con il modello originale era il drastico aumento di prestazioni di alcune armi al superamento di una certa soglia di abilità tendente al massimo valore, come si può notare per esempio nei grafici nelle figure 3.11a e 3.15a. In generale con il nuovo modello l'aumento prestazionale è maggiormente distribuito nell'intero intervallo di abilità, e le prestazioni tra armi diverse più bilanciate.

3.9 Sommario

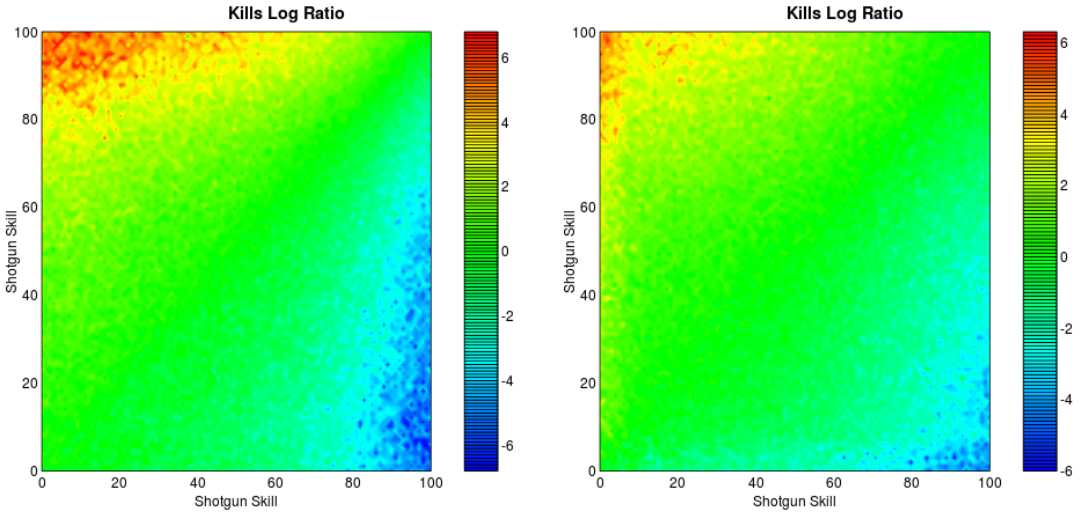
In questo capitolo abbiamo descritto *Cube 2: Sauerbraten*, il gioco da noi utilizzato per effettuare le simulazioni per valutare le caratteristiche delle mappe. Abbiamo descritto i problemi dell'intelligenza artificiale originale del gioco e le modifiche da noi apportate per renderla più realistica, confrontando le differenze delle prestazioni dei bot tra il modello originale e quello da noi proposto.

[16] Per rendere i risultati più evidenti i grafici del rapporto delle uccisioni usano una scala logaritmica in base 2.



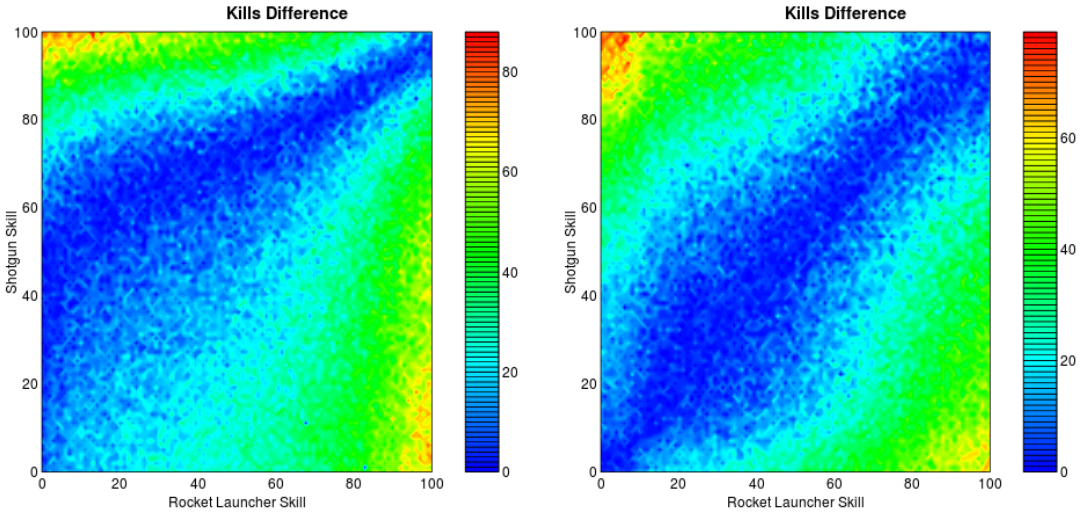
(a) Differenza di uccisioni modello vecchio (b) Differenza di uccisioni modello nuovo

Figura 3.7: Confronto differenza di uccisioni tra due bot dotati di *Shotgun*.



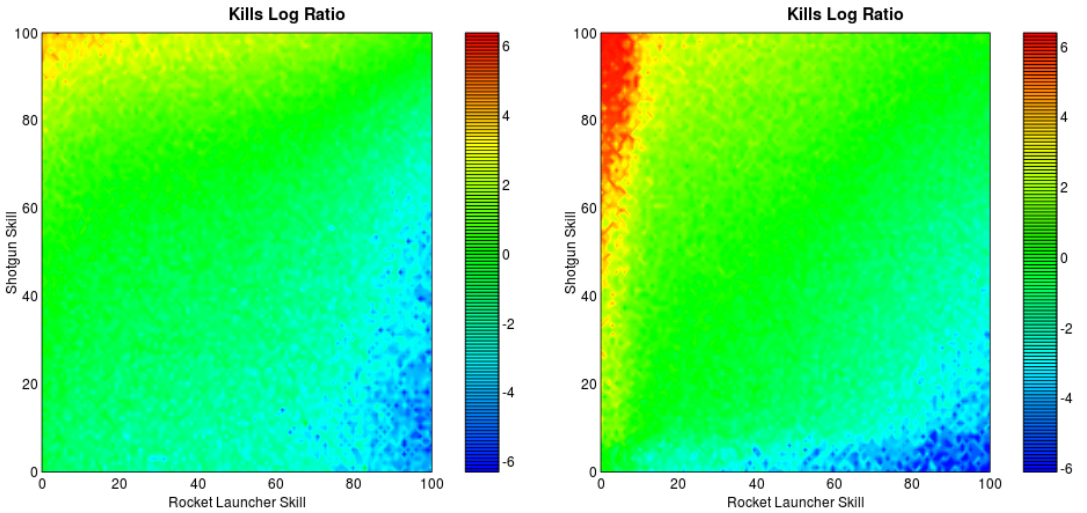
(a) Rapporto di uccisioni modello vecchio (b) Rapporto di uccisioni modello nuovo

Figura 3.8: Confronto rapporto di uccisioni tra due bot dotati di *Shotgun*.



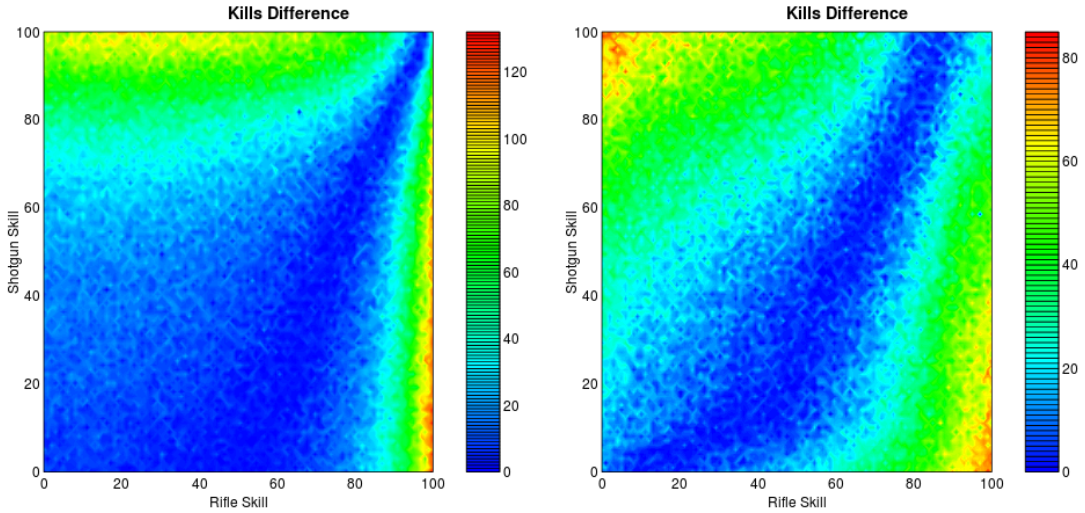
(a) Differenza di uccisioni modello vecchio (b) Differenza di uccisioni modello nuovo

Figura 3.9: Confronto differenza di uccisioni tra un bot dotato di *Shotgun* e uno dotato di lanciarazzi.



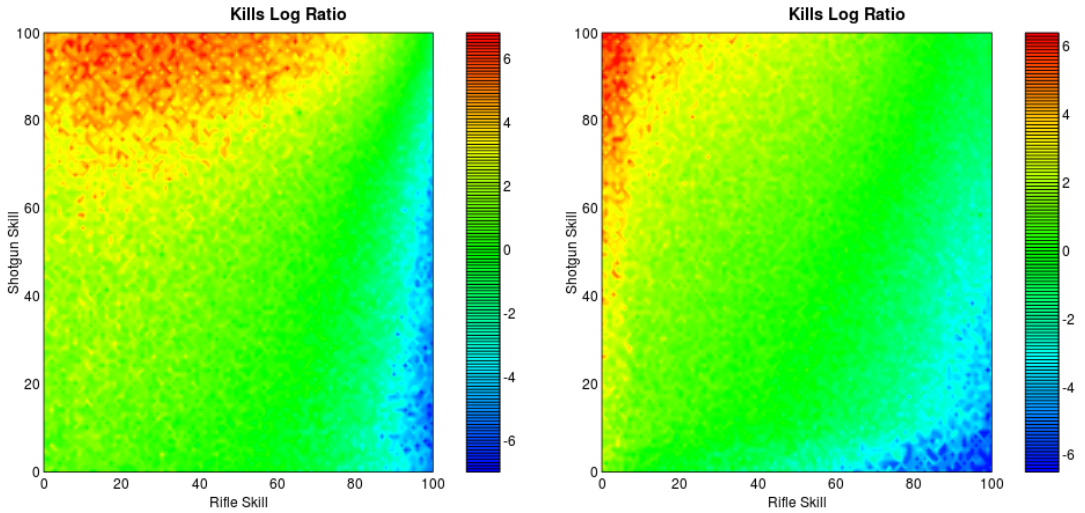
(a) Rapporto di uccisioni modello vecchio (b) Rapporto di uccisioni modello nuovo

Figura 3.10: Confronto rapporto di uccisioni tra un bot dotato di *Shotgun* e uno dotato di lanciarazzi.



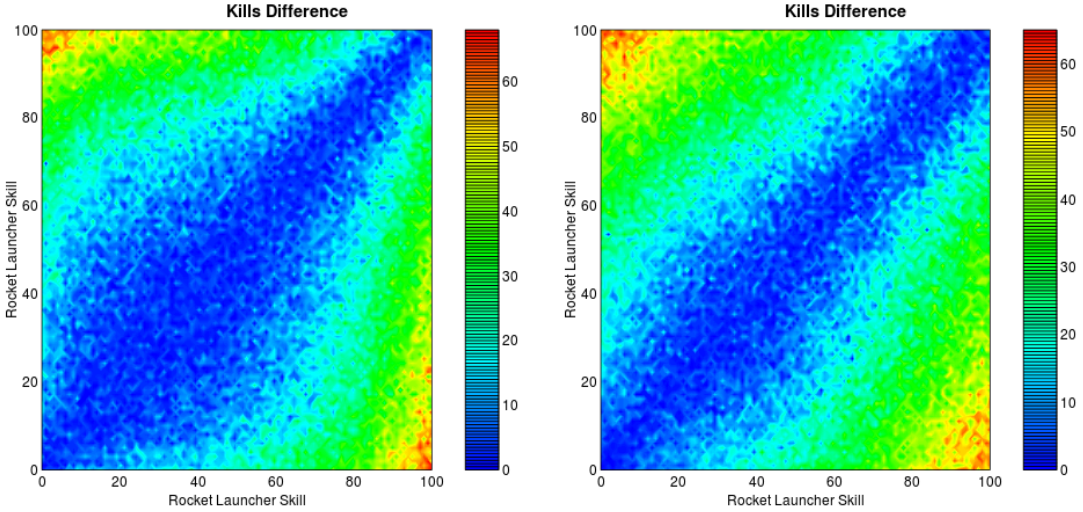
(a) Differenza di uccisioni modello vecchio (b) Differenza di uccisioni modello nuovo

Figura 3.11: Confronto differenza di uccisioni tra un bot dotato di *Shotgun* e uno dotato di *Rifle*.



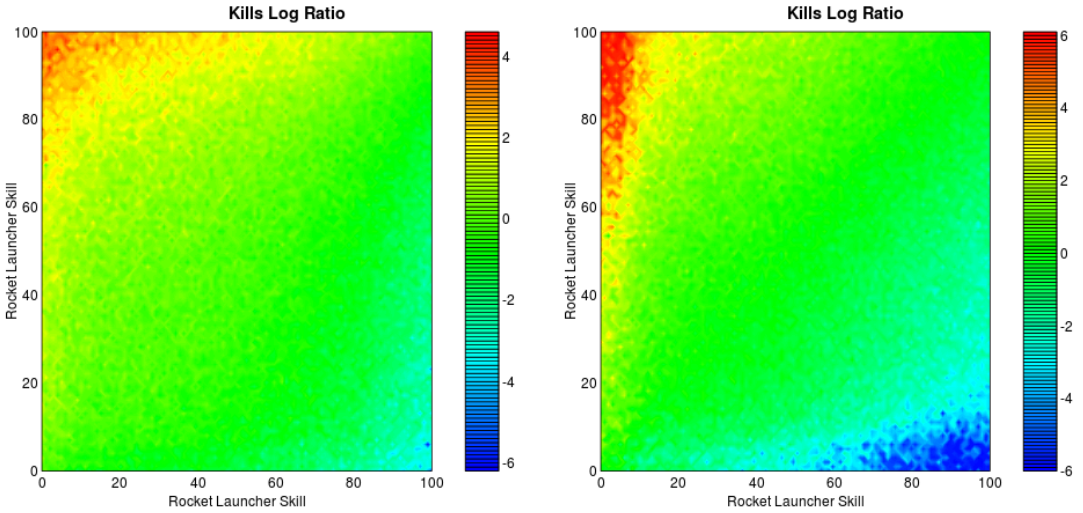
(a) Rapporto di uccisioni modello vecchio (b) Rapporto di uccisioni modello nuovo

Figura 3.12: Confronto rapporto di uccisioni tra un bot dotato di *Shotgun* e uno dotato di *Rifle*.



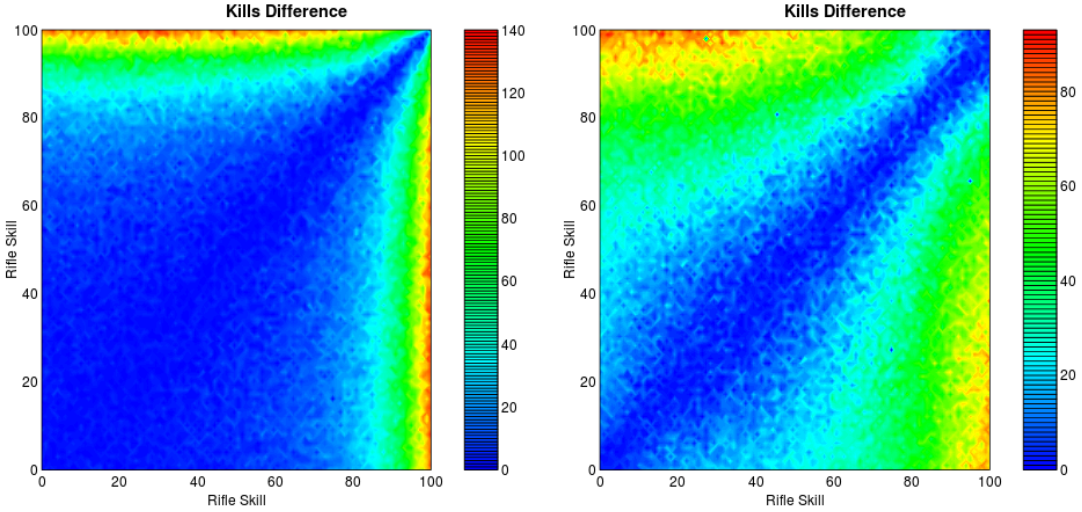
(a) Differenza di uccisioni modello vecchio (b) Differenza di uccisioni modello nuovo

Figura 3.13: Confronto differenza di uccisioni tra due bot dotati di lanciarazzi.



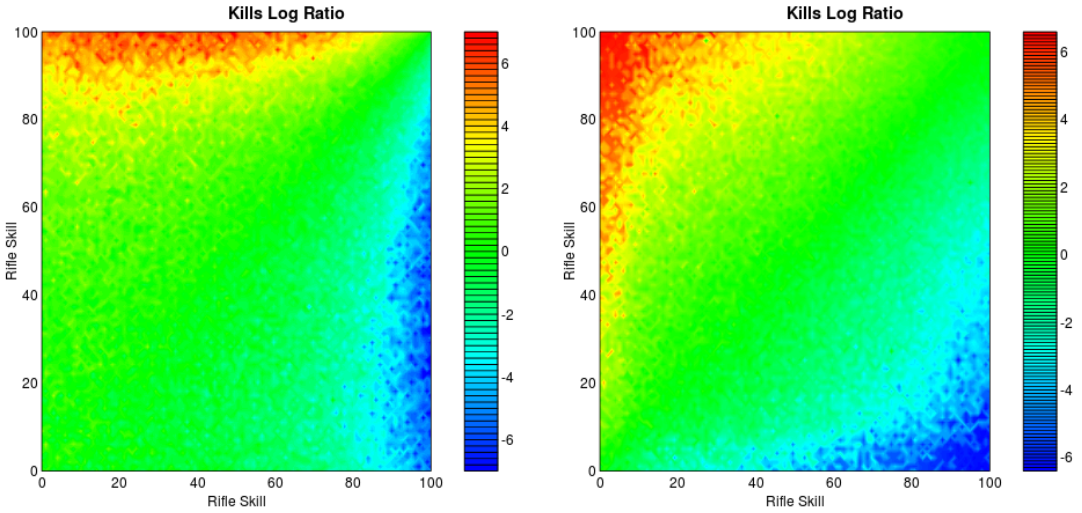
(a) Rapporto di uccisioni modello vecchio (b) Rapporto di uccisioni modello nuovo

Figura 3.14: Confronto rapporto di uccisioni tra due bot dotati di lanciarazzi.



(a) Differenza di uccisioni modello vecchio (b) Differenza di uccisioni modello nuovo

Figura 3.15: Confronto differenza di uccisioni tra due bot dotati di *Rifle*.



(a) Rapporto di uccisioni modello vecchio (b) Rapporto di uccisioni modello nuovo

Figura 3.16: Confronto rapporto di uccisioni tra due bot dotati di *Rifle*.

Capitolo 4

Caratteristiche del Framework

In questo capitolo descriviamo il *framework* realizzato come strumento d'aiuto nella progettazione dei livelli per First Person Shooter; il framework si basa su un'estensione del lavoro precedentemente svolto da Cardamone et al.[4] e ripreso da Stucchi[5], ed è composto da una parte di personalizzazione dei parametri dell'intelligenza artificiale che caratterizzano il comportamento dei bot e da una parte di impostazione degli algoritmi genetici per la generazione procedurale delle mappe. Nella prima parte del capitolo descriviamo nel dettaglio la parte relativa alla personalizzazione dell'intelligenza artificiale, elencando i parametri che è possibile modificare per modellare il comportamento di diversi profili di giocatore. Successivamente descriviamo la parte relativa alla generazione delle mappe, spiegando quali dati vengono raccolti durante le partite e le **metriche** che ne deriviamo, che possono essere impostate come obiettivo dell'algoritmo evolutivo, e descrivendo la rappresentazione delle mappe che viene utilizzata durante l'evoluzione.

4.1 Caratterizzazione dei bot

Le modifiche illustrate nel capitolo 3 sono state implementate in modo trasparente nell'IA di *Cube 2*: ogni ottimizzazione è attivabile tramite un'estensione dell'interfaccia del gioco, e ogni aspetto può essere attivato o disattivato singolarmente per ognuno dei bot che partecipano alla simulazione. Inoltre è possibile impostare, sempre per ognuno dei bot, un'eventuale costrizione sull'arma usata e i valori delle caratteristiche di *Abilità Generale*, *Mira*, *Visibilità*, *Velocità di visua-*

lizzazione, *Distanziamento* e *Abilità di Combattimento* già spiegati nel dettaglio nella sezione 3.3: questi ultimi rappresentano il livello di abilità in percentuale rispetto all'*Abilità Generale* di base specificata (per esempio se viene assegnato un livello di abilità base di 50 e un attributo di abilità di mira pari a 150 il livello dell'abilità di mira finale è uguale al 150% di 50, dunque 75). Sebbene non ci sia un limite al valore assegnabile a queste percentuali, il livello di abilità finale che si ottiene per ogni caratteristica è tuttavia troncato per rientrare, come originariamente il valore di *skill*, nell'intervallo [1,101], e viene calcolato come

$$a_i = \frac{s \cdot p_i}{100}, \tag{4.1}$$

dove $a_i \in [1, 101]$ è il valore finale dell'abilità i, $s \in [1, 101]$ è il valore generale di *skill* e $p_i \in [0, +\infty]$ è il fattore percentuale della caratteristica i.

4.2 Telemetria delle partite

Per permettere l'evoluzione e ottenere mappe che soddisfino gli obiettivi di gameplay fissati è necessario raccogliere diversi dati sullo svolgimento delle partite, che durante ogni simulazione vengono salvati in file di *log* per essere poi utilizzati per calcolare *metriche* di gioco. In questa sezione vengono elencati i dati che sono stati considerati importanti per lo scopo, e funge come riferimento per una migliore comprensione delle metriche che verranno spiegate nella sezione successiva.

Per ogni partita vengono registrati i seguenti dati generali, che misurano aspetti attribuibili direttamente alla configurazione della mappa:

- TimeToEngage (Tempo di Ingaggio): il tempo, espresso in millisecondi, trascorso tra la fine di ogni evento di combattimento e l'inizio del successivo, o tra l'inizio di ogni primo combattimento successivo all'istante di respawn;

- TimeInFight (Tempo in Combattimento): la durata di ogni evento di combattimento in millisecondi, intesa come l'intervallo di tempo in cui i bot seguono un nemico obiettivo, anche se non vi è un effettivo scontro diretto;

- NumberOfFights (Numero di Combattimenti): il numero totale dei combattimenti avvenuti durante la partita;

- TimeBetweenSights (Tempo tra gli Avvistamenti): il tempo, in millisecondi e per ogni evento, trascorso tra l'istante in cui viene persa linea visiva col nemico all'istante in cui viene riacquisita;
- NumberOfSights (Numero di Avvistamenti): quante volte un nemico esce dal campo visivo per poi rientrarvi;
- TimeToSurrender (Tempo alla Resa): il tempo trascorso, in millisecondi, dall'ultimo avvistamento dell'obiettivo al momento, se esiste, in cui un bot decide di interrompere l'inseguimento;
- NumberOfRetreats (Numero di Rese): il numero di volte in cui è trascorso un tempo sufficiente per un bot da fargli abbandonare l'inseguimento dell'obiettivo;
- DeadKills (Uccisioni da Morto): la quantità di uccisioni eseguite da un bot morto, dovute spesso a proiettili vaganti o a uccisioni reciproche contemporanee;
- StealthKills (Uccisioni a Sorpresa): la quantità di uccisioni eseguite da un bot su una vittima che non sta partecipando attivamente ad alcun combattimento;

Inoltre, per ogni bot che partecipa alla partita vengono raccolti anche i seguenti dati quantitativi:

- Frags: il numero di uccisioni eseguite dal bot;
- Shots: la quantità di colpi sparati;
- Hits: la quantità di colpi andati a segno;
- KillStreakAverage: la media aritmetica delle serie consecutive di uccisioni che si riescono ad eseguire consecutivamente prima di essere uccisi, tenendo conto delle sole serie di una o più uccisioni (non viene quindi tenuto conto delle serie di zero uccisioni che si verificano quando un bot muore consecutivamente due volte senza effetauare nessun *frag*);
- KillStreakMax: la serie massima di uccisioni consecutive;

4.3 Metriche di Analisi

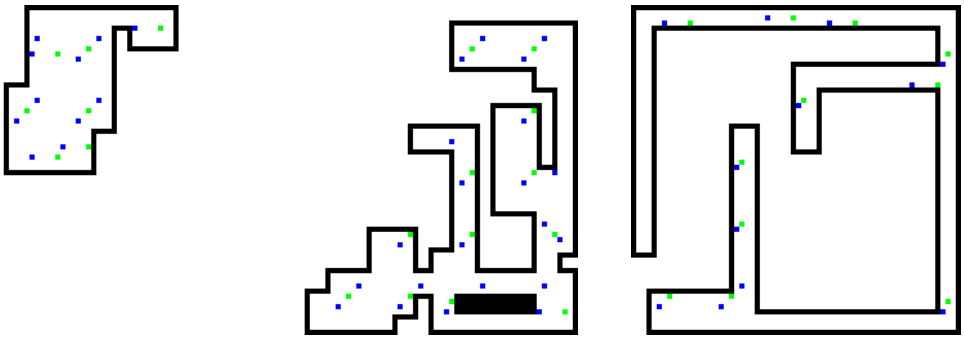
Prendendo spunto da alcuni dei fattori che, a detta di designer esperti del settore, sono necessari alla realizzazione di buone mappe per FPS[3, 26, 27][1], abbiamo selezionato alcune metriche che misurano aspetti potenzialmente interessanti durante la progettazione di un livello, e che abbiamo usato in alcuni esperimenti per evolvere alcune mappe che ne massimizzassero il valore al fine di poterne analizzare le caratteristiche, e per osservare quali relazioni intercorressero tra alcune di queste metriche.

4.3.1 Bilanciamento

Il bilanciamento di una mappa, intesa come la possibilità offerta dal livello di ottenere un risultato finale equilibrato tra i partecipanti, è probabilmente l'aspetto più importante da tenere in considerazione, soprattutto nel caso di giochi multigiocatore; sebbene siano importanti anche dimensione, *flusso* ed elementi visivi, se una mappa è sbilanciata nessuno ci vorrà giocare[26]. È importante notare che il bilanciamento non va necessariamente cercato tramite banale simmetria della mappa, ma tramite un'equilibrata contrapposizione di situazioni di vantaggio per una delle parti a opportunità di sfruttare tattiche di contrasto per l'altra.

Molte armi godono di un vantaggio intrinseco in particolari situazioni, e la creazione di livelli fortemente basati su un *concept* potrebbe portare a mappe strutturate in favore di un solo tipo di giocatore; in questa direzione può essere utile poter analizzare e trovare velocemente pattern che creino situazioni di gioco interessanti e bilanciate. Un modo efficace per calcolare l'efficienza di un giocatore, specialmente in uno sparatutto multigiocatore, è tramite il rapporto tra le uccisioni effettuate e quelle subite; evidentemente i migliori giocatori sono quelli che riescono ad effettuare un alto numero di uccisioni senza esporsi eccessivamente al fuoco nemico. Per valutare dunque l'equilibrio tra i giocatori abbiamo deciso di utilizzare il valore di **entropia** delle uccisioni di tutti i giocatori, utilizzato precedentemente anche da Stucchi in [5] per generare automaticamente mappe bilanciate:

[1] Noi ci concentriamo in particolare su mappe per FPS multiplayer, ma molte delle considerazioni che vengono fatte si possono applicare anche a FPS a singolo giocatore.



(a) Mappa con entropia 0.80 (b) Mappa con entropia 0.99 (c) Mappa con entropia 0.90

Figura 4.1: Esempi di mappe a diversi livelli di entropia.

$$f = H(K) = \sum_{i=1}^{n} - \left(\frac{k_i}{k_{tot}}\right) \log_2 \left(\frac{k_i}{k_{tot}}\right) \tag{4.2}$$

dove k_i sono le uccisioni dell'*i-esimo* giocatore, k_{tot} le uccisioni totali della partita e n il numero di bot. Il valore dell'entropia può essere compreso nell'intervallo [0,1]: in generale un valore tendente a 1 indica sostanziale equilibrio tra i giocatori, mentre un valore basso indica uno sbilanciamento in favore di una parte dei partecipanti[2].

In Figura 4.1 sono mostrati alcuni esempi di mappe ottenute cercando di bilanciare o sbilanciare il risultato tra due bot con uguali parametri di abilità, uno dotato di *Shotgun* e l'altro di *Rifle*; in Figura 4.1b è mostrata la mappa ottenuta cercando il perfetto bilanciamento, mentre in Figura 4.1a e 4.1c due mappe sbilanciate in modo diametralmente opposto, rispettivamente in favore del bot dotato di *Shotgun* e del bot dotato di *Rifle*. Osservando la mappa bilanciata, e confrontandola con le altre due, si nota come in questa esistano diverse parti le cui caratteristiche sono simili alle strutture visibili in entrambe le mappe sbilanciate.

4.3.2 Pace

Un altro aspetto da tenere in particolare considerazione in giochi multigiocatore è il ritmo di gioco e la frequenza degli incontri tra i giocatori, che nel caso degli sparatutto si traduce in frequenza dei combattimenti: in generale agli occhi di

[2]Essendo le nostre simulazioni effettuate tra due soli contendenti un valore basso di entropia rappresenta semplicemente un forte vantaggio per uno dei due giocatori.

un giocatore un mondo in cui intercorre un tempo elevato tra il contatto con un altro giocatore e il successivo può risultare noioso e poco coinvolgente. Può però essere vero anche il contrario: una frequenza di incontri troppo elevata rischia di far sentire il giocatore sopraffatto, impedendogli di attuare alcuna tattica e di prendere realmente parte alla partita, e può rendere l'esperienza di gioco frustrante. Per questo motivo nei giochi commerciali il numero massimo di giocatori viene solitamente limitato in base alle caratteristiche della mappa, ed è dunque necessario trovare un giusto compromesso.

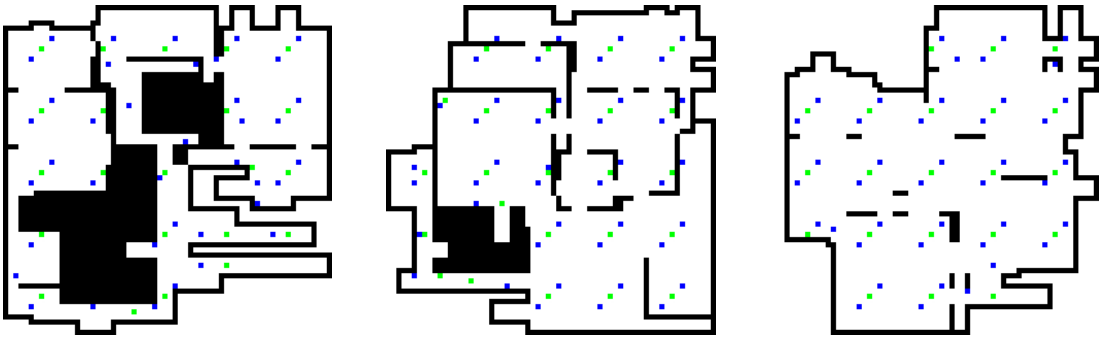
Per calcolare un valore del *pace* normalizzato nell'intervallo $[0, 1]$, analogo alla maggior parte delle altre metriche, abbiamo definito la seguente funzione sigmoidea basata sul rapporto tra il numero di combattimenti iniziati rispetto al tempo medio impiegato nella ricerca di uno scontro, trascorso dal momento di respawn o dal termine del combattimento precedente:

$$f = 2 \cdot \frac{1}{1 + \exp\left(-3000 \cdot \frac{NumberOfFights}{\sum TimeToEngage}\right)} - 1 \tag{4.3}$$

dove la sommatoria è ottenuta sommando tutti i tempi di ingaggio (*TimeToIngage*), in millisecondi, calcolati durante la partita. I parametri della funzione sono stati scelti per avere, con un tempo di ingaggio medio di un secondo, un valore di *pace* pari a 0.9. In Figura 4.2 sono mostrate tre mappe ottenute durante un processo evolutivo per massimizzare il valore di *pace*; è possibile notare come, in questo caso, la struttura della mappa sia divenuta sempre meno complessa e gli spazi più aperti, permettendo una maggiore visibilità e incentivando l'ingaggio in combattimento.

4.3.3 PursueTime e FightTime

Il *Pursue Time* (Tempo di Inseguimento) e il *Fight Time* (Tempo di Combattimento) sono misure globali che indicano la durata totale degli scontri, calcolata come percentuale di tempo impiegato nei combattimenti rispetto alla durata totale della partita. Nello specifico il *Pursue Time* indica la percentuale di tempo in cui i bot sono all'inseguimento di un obiettivo nemico, indipendentemente dal fatto che vi sia effettivamente uno scontro, ed è calcolato come



(a) Mappa con pace 0.25 (b) Mappa con pace 0.62 (c) Mappa con pace 0.95

Figura 4.2: Esempi di mappe ottenute nel corso un'evoluzione con massimizzazione del *pace*.

$$f = \frac{\sum TimeInFight}{NumberOfBots \cdot GameLength} \tag{4.4}$$

dove *GameLength* è la durata, in millisecondi, della partita. Il *Fight Time*, invece, misura in percentuale il tempo effettivamente impiegato negli scontri senza considerare gli intervalli nei quali gli obiettivi non sono visibili come

$$f = \frac{\sum TimeInFight - \sum TimeBetweenSights - \sum TimeToSurrender}{NumberOfBots \cdot GameLength} \tag{4.5}$$

Essendo valori percentuali possono assumere, evidentemente, valori compresi nell'intervallo $[0, 1]$.

4.3.4 SightLossRate e TargetLossRate

Il *Sight Loss Rate* (Tasso di Perdita di Vista) e il *Target Loss Rate* (Tasso di Perdita dell'Obiettivo) misurano quanto la struttura del livello ostacoli la visibilità e i combattimenti. In particolare il *Sight Loss Rate* misura, in percentuale, quanto tempo si perde per ogni combattimento a rintracciare un nemico perso di vista. Viene calcolato come

$$f = \frac{\sum TimeBetweenSights}{\sum TimeInFight} \tag{4.6}$$

Il **Target Loss Rate** misura la percentuale di scontri che terminano con una "fuga", senza che vi siano vittime. Viene calcolato come

$$f = \frac{NumberOfRetreats}{NumberOfFights} \tag{4.7}$$

Anche queste misure possono assumere valori nell'intervallo $[0, 1]$.

4.3.5 Metriche prestazionali

Oltre alle metriche che misurano caratteristiche delle mappe può essere utile misurare anche informazioni che riguardano le prestazioni dei singoli bot durante le partite:

- **NumberOfFrags** (Numero di Uccisioni): il numero totale delle uccisioni effettuate da ogni bot;
- **Accuracy** (Accuratezza): la precisione di tiro di ogni bot, calcolata come

  $$f = \frac{Hits_i}{Shots_i} \tag{4.8}$$

  dove $Hits_i$ e $Shots_i$ sono rispettivamente i colpi sparati e andati a segno dell'*i-esimo* bot. Sebbene rappresenti la percentuale di colpi andati a segno rispetto a quelli sparati, è possibile che assuma valori oltre una percentuale perfetta, dovuta alla presenza di armi che permettono di realizzare più colpi con un solo proiettile (nel caso di *Cube 2*, il Lanciarazzi e il Lancia Granate)[3].
- **MapBenefit** (Beneficio della Mappa): misura le prestazioni di ogni bot rispetto agli altri come il rapporto tra il suo punteggio e la somma dei punteggi degli avversari:

  $$f = \frac{Frags_i}{\sum_{j \neq i} Frags_j} \tag{4.9}$$

  dove $Frags_i$ è il numero di uccisioni dell'*i-esimo* bot.

[3]In generale, in altri giochi FPS, esistono molte armi di diversa natura che permettono uccisioni multiple. Spesso ciò viene addirittura premiato, ed è dunque possibile considerare questo possibile eccesso un aspetto normale piuttosto che un effetto collaterale.

- **StreakAverage** (Media Serie di Uccisioni): la media delle serie di uccisioni effettuate senza morire da ogni bot;

- **StreakMaximum** (Serie Massima di Uccisioni): la serie di uccisioni più lunga effettuata da ogni bot;

Di queste metriche viene inoltre calcolata una media sulla partita rispetto a tutti i bot che vi hanno preso parte.

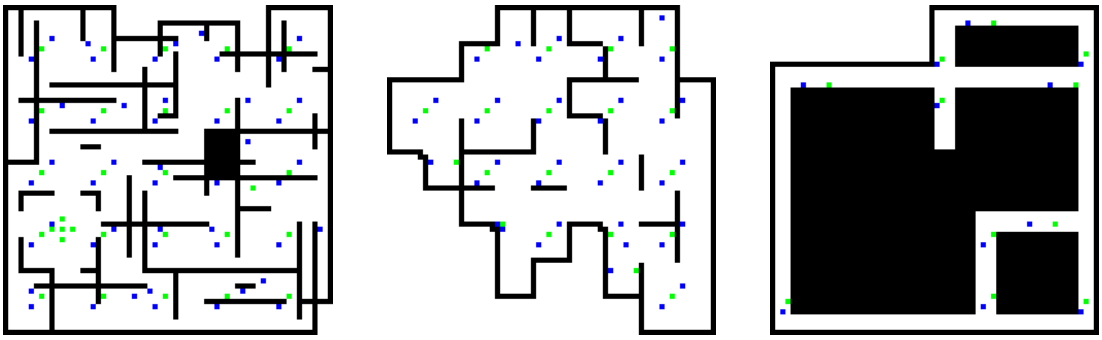
4.4 Evoluzione delle mappe

Grazie alla codifica relativamente semplice delle mappe di *Cube 2*, usata per generare dinamicamente i livelli di gioco, Cardamone et al. hanno sviluppato una codifica che permette di creare esternamente al gioco le mappe per poi essere importate e generate al volo dallo stesso; ciò ha permesso loro in [4] di realizzare uno strumento che, tramite **algoritmi genetici**, genera automaticamente mappe che massimizzano una funzione obiettivo nel processo evolutivo. L'algoritmo da loro usato impiegava operatori standard: evoluzione secondo un obiettivo di **fitness** singolo, un **crossover** a *singolo punto* con selezione dei geni tramite *Tournament Selection* a due candidati, un tasso di mutazione di $\frac{1}{n}$ (dove n è la dimensione del genoma), e permetteva di specificare l'*intervallo* e la *probabilità di mutazione*, la dimensione della popolazione e il numero di generazioni da effettuare.

L'algoritmo genetico è stato poi perfezionato da Stucchi in [5] con l'aggiunta di un algoritmo di calcolo di *fitness* **Multi-Obiettivo** e di un operatore di *crossover* matriciale.

L'algoritmo genera il **fenotipo** come un file di testo organizzato come una matrice di caratteri, isomorfo al livello che da esso viene generato nel gioco, in cui ogni carattere codifica una sezione della mappa, determinando la presenza o meno di un muro, di un punto di *respawn*, o di un oggetto in tale sezione. Il **genotipo** invece può avere diverse rappresentazioni a seconda delle diverse modalità possibili per la costruzione delle mappe, discusse nella sezione 4.4.1.

Abbiamo utilizzato questo strumento per realizzare esperimenti di evoluzione di mappe, usando come funzioni di fitness dell'algoritmo genetico le metriche esposte nella sezione 4.3.



(a) All-White (b) Grid (c) Random-Digger

Figura 4.3: Esempi di mappe ottenute con le rappresentazioni scartate.

4.4.1 Rappresentazione delle mappe

L'algoritmo sviluppato da Cardamone et al. permette di generare mappe casuali o come evoluzione genetica di mappe precedenti, mettendo a disposizione quattro tipi di rappresentazione:

- **All-White**: la costruzione della mappa avviene partendo da una stanza completamente vuota, ad eccezione dei soli muri esterni posti alle estremità dell'area disponibile, posizionando casualmente i muri;

- **All-Black**: complementare alla All-White, usa come base una mappa completamente occupata dalla quale vengono eliminate sezioni quadrate e corridoi creando spazi navigabili.

- **Grid**: la mappa viene sezionata come una griglia di muri, alcuni dei quali vengono eliminati creando spazi navigabili;

- **Random-Digger**: la mappa viene ottenuta scavando corridoi in modo sequenziale da una mappa inizialmente piena.

Per i nostri scopi abbiamo scelto di utilizzare la sola rappresentazione *all-black*, che genera mappe strutturalmente migliori, descritta nel dettaglio nella sezione 4.4.2. In Figura 4.3 sono mostrati alcuni esempi di mappe ottenute con le altre rappresentazioni, per i cui dettagli si rimanda a [4].

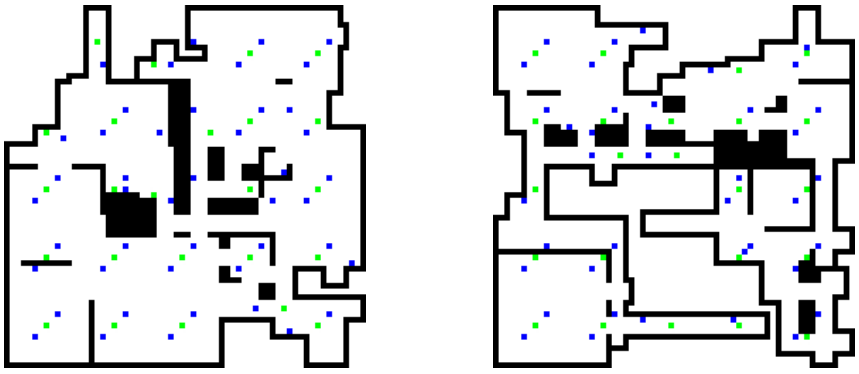


Figura 4.4: Esempi di mappe create con rappresentazione *All-Black*.

4.4.2 Rappresentazione All-Black

Nella rappresentazione *all-black* le aree di gioco vengono ottenute "scavando" stanze quadrate e corridoi da una mappa inizialmente completamente riempita di muri. Tali spazi sono codificati tramite un genoma di dimensione $n_g = 3 \cdot a + 3 \cdot b$ dove a e b sono rispettivamente il numero di stanze e corridoi. Le stanze, di forma quadrata, vengono definite da triplette $\langle x, y, s \rangle$ che definiscono le coordinate x e y del centro della stanza e la sua estensione s. I corridoi sono spazi rettangolari anch'essi definiti da triplette $\langle x, y, l \rangle$ che definiscono le coordinate x e y del punto di inizio e la lunghezza l del corridoio, con larghezza predefinita e costante e direzione definita dal segno del parametro l: se positivo il corridoio ha uno sviluppo lungo l'asse x, altrimenti lungo l'asse y. In Figura 4.4 vengono mostrati due esempi di mappe ottenute con questa rappresentazione. Presentando diverse arene di forma regolare e corridoi che collegano gli spazi abbiamo ritenuto le mappe create con questa rappresentazione più complete e paragonabili alle mappe create a mano rispetto a quelle ottenute con le altre rappresentazioni, motivo per cui l'abbiamo scelta per gli esperimenti realizzati scartando le altre.

4.5 Sommario

In questo capitolo abbiamo descritto le parti che compongono il *framework* da noi realizzato per assistere il design di livelli per *first person shooter*. Abbiamo inizialmente descritto i parametri che si possono impostare ai bot del gioco per modellare profili di giocatori da utilizzare nelle simulazioni; abbiamo poi elencato

quali dati abbiamo deciso di raccogliere durante le simulazioni e le metriche da essi derivati per misurare alcune caratteristiche e statistiche delle mappe e delle partite in base ai fattori che vengono ritenuti fondamentali da alcune figure importanti dell'industria videoludica. Infine abbiamo descritto gli aspetti fondamentali del procedimento di evoluzione delle mappe e le caratteristiche della loro rappresentazione.

Capitolo 5

Esperimenti di evoluzione

In questo capitolo mostriamo tre esperimenti di evoluzione, eseguiti scegliendo come obiettivi alcune delle metriche mostrate nella sezione 4.3.

Nel primo esperimento cerchiamo di generare mappe bilanciate che producano allo stesso tempo un ritmo di gioco elevato.

Nel secondo esperimento cerchiamo di generare mappe bilanciate che permettano di sviluppare lunghe serie di uccisioni.

Per il terzo esperimento invece ci siamo posti come obiettivo la generazione di mappe dispersive, che rendano difficili gli inseguimenti permettendo viceversa facili fughe.

Per ogni esperimento vengono mostrate le statistiche del processo evolutivo, analizzate alcune delle mappe ottenute sia a livello strutturale che dal punto di vista delle dinamiche create, e studiati i rapporti che esistono tra diversi aspetti della mappa.

5.1 Primo esperimento: Equilibrio e Ritmo

Per questo esperimento abbiamo cercato di evolvere alcune mappe che fossero contemporaneamente bilanciate e con un alto ritmo di gioco, quindi che massimizzassero i valori di *entropia* e *pace* illustrati nella sezione 4.3.

Abbiamo scelto di modellare due profili di giocatore con caratteristiche e strategie diametralmente opposte: un "tiratore scelto" (*Sniper*) efficace in scontri dalla distanza, e un "assaltatore" (*Assault*) che cerca lo scontro diretto con armi a corto raggio.

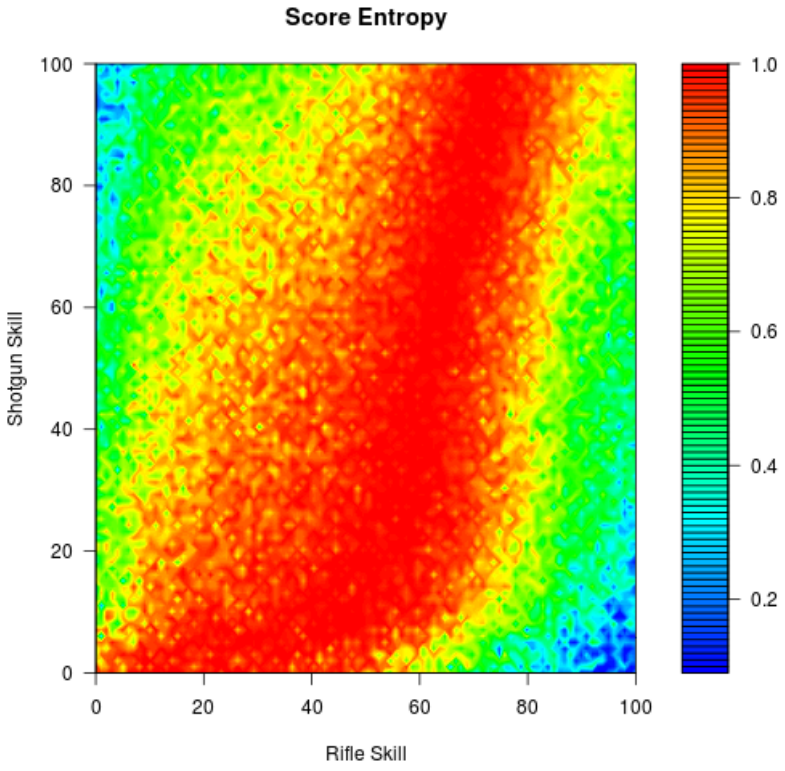


Figura 5.1: Livelli di entropia per i profili Assault e Sniper a diversi livelli di *skill*

5.1.1 Setup e statistiche di esecuzione

Per caratterizzare le due classi abbiamo inizialmente deciso i valori di abilità da assegnare ai bot tranne quello generale (*skill*), lasciato come parametro finale per regolare il divario tra i due bot, in modo da definire prima di tutto i loro punti di forza e debolezza. Per il profilo **Assault** abbiamo semplicemente ridotto il livello di mira rispetto a quello base: questo profilo rappresenta un giocatore aggressivo, che cerca sempre lo scontro ravvicinato, prediligendo dunque come arma lo *shotgun* che, usato opportunamente, mantiene un'alta efficacia anche con basse abilità di mira. Al profilo **Sniper**, invece, abbiamo assegnato il *rifle*; abbiamo aumentato il valore di base di mira e diminuito l'ampiezza del campo visivo per simulare parzialmente l'effetto di maggiore precisione e ridotta visibilità tipico dell'uso di un fucile da cecchino (non essendoci nell'arsenale di *Cube 2* un vero fucile da cecchino). Abbiamo ridotto l'abilità nel combattimento, cioè la capacità di sparare in movimento per rappresentare la staticità tipica di un simile

Profilo	*Assault*	*Sniper*
Arma	*Shotgun*	*Rifle*
Abilità Generale	*95*	*35*
Mira	*66*	*115*
Visibilità	*100*	*25*
Velocità di Visualizzazione	*100*	*100*
Distanziamento	*100*	*75*
Abilità di Combattimento	*100*	*25*

Tabella 5.1: Parametri di setup dei bot per l'esperimento 1.

profilo. Infine abbiamo ridotto la capacità di mantenere le distanze per esaminare gli effetti prodotti da un aspetto che contrasta con le considerazioni finora fatte.

Per rendere l'esperimento più interessante, soprattutto dal punto di vista del bilanciamento, le abilità dei due bot sono state sbilanciate in partenza, assegnando un valore di *skill* generale alto al profilo *Assault*, e basso a quello *Sniper*. In base a livelli di entropia calcolati con diverse configurazioni di abilità tramite simulazioni su una mappa di test[1], e visibili in Figura 5.1, è stata scelta una coppia di valori di *skill* con entropia approssimativamente uguale a 0.75. In Tabella 5.1 sono riassunti i valori assegnati agli attributi dei bot per la realizzazione di questo esperimento.

Abbiamo realizzato l'esperimento eseguendo 12 processi evolutivi[2] con algoritmo genetico multi-obiettivo sulla rappresentazione di mappa *All-Black*, con genotipo formato come rappresentazione di 15 arene e 50 corridoi, simulando partite della durata di 30 minuti ciascuna. Ogni processo evolutivo ha impiegato tra le 2 e le 3 ore, e ha usato una popolazione di 50 candidati per 30 generazioni, con *crossover* matriciale, selezione dei candidati tramite *tournament selection* a due candidati, intervallo di mutazione 0.3 e probabilità di crossover 0.3. Rispetto ai parametri usati negli esperimenti in [4] e [5] abbiamo dovuto ridurre la dimensione della popolazione, come compromesso derivato dalla maggiore durata delle partite scelta per i nostri esperimenti. Tale scelta è stata fatta in base ai risultati ottenuti da una serie di test preliminari effettuati per verificare la varianza delle

[1]Per i test è stata usata la mappa *fragplaza*, già utilizzata e descritta in precedenza per effettuare altri test.

[2]Il nostro obiettivo era di effettuare l'analisi su almeno 10 processi. Per ottimizzare i tempi e sfruttare meglio i processori abbiamo realizzato gli esperimenti avviando in parallelo tre simulazioni alla volta, ottenendone 12.

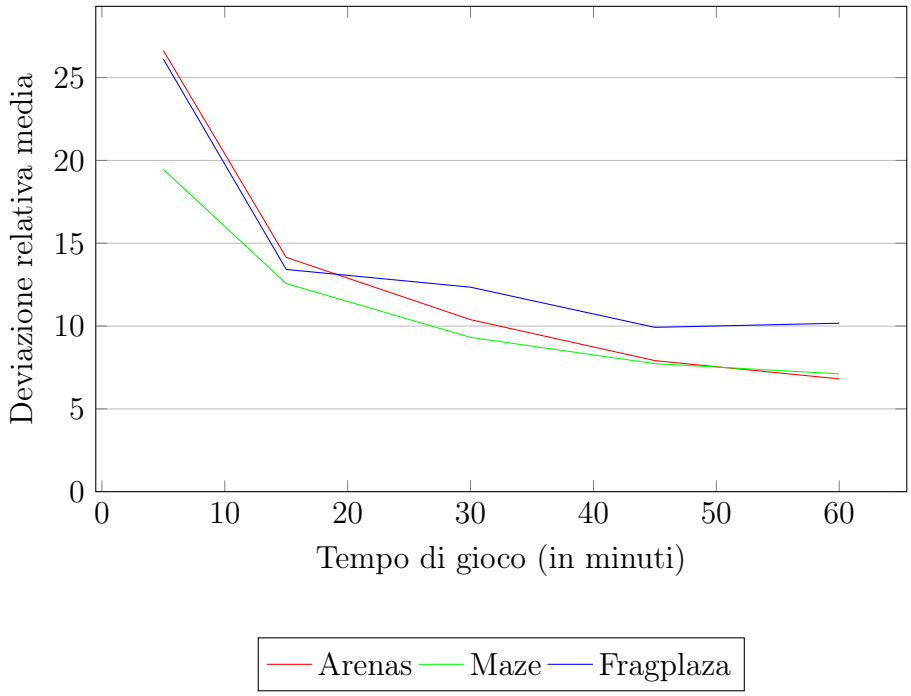


Figura 5.2: Deviazioni standard relative medie delle metriche su diverse mappe per diverse durate delle partite.

metriche in base alla durata delle partite: abbiamo identificato in 30 minuti un buon compromesso tra la varianza dei risultati, che per tale valore presentano una *deviazione standard relativa media* poco superiore al 10% come si vede dal grafico in figura 5.2, e la durata di ogni singola simulazione. I test sono stati effettuati per diverse durate e su diverse mappe (Arenas, Maze e Fragplaza, già usate e descritte per i test nel capitolo 3) simulando una serie di 10 partite tra due bot con abilità generale pari a 75 e tutte le altre abilità pari a 100, con diverse configurazioni di armi.

Abbiamo selezionato due dei processi evolutivi realizzati per mostrare le caratteristiche delle mappe ottenute. Per entrambi gli esempi vengono mostrate le mappe che massimizzano singolarmente i due valori di fitness, e una mappa scelta come "miglior compromesso", che massimizza un punteggio calcolato come somma pesata dei valori di fitness, usando 0.57 come peso per il valore dell'*entropia* e 0.43 per quello del *pace*.

Come primo esempio abbiamo scelto il processo evolutivo che ha prodotto la mappa più equilibrata tra tutte. In Figura 5.4 sono mostrate le mappe più significative ottenute.

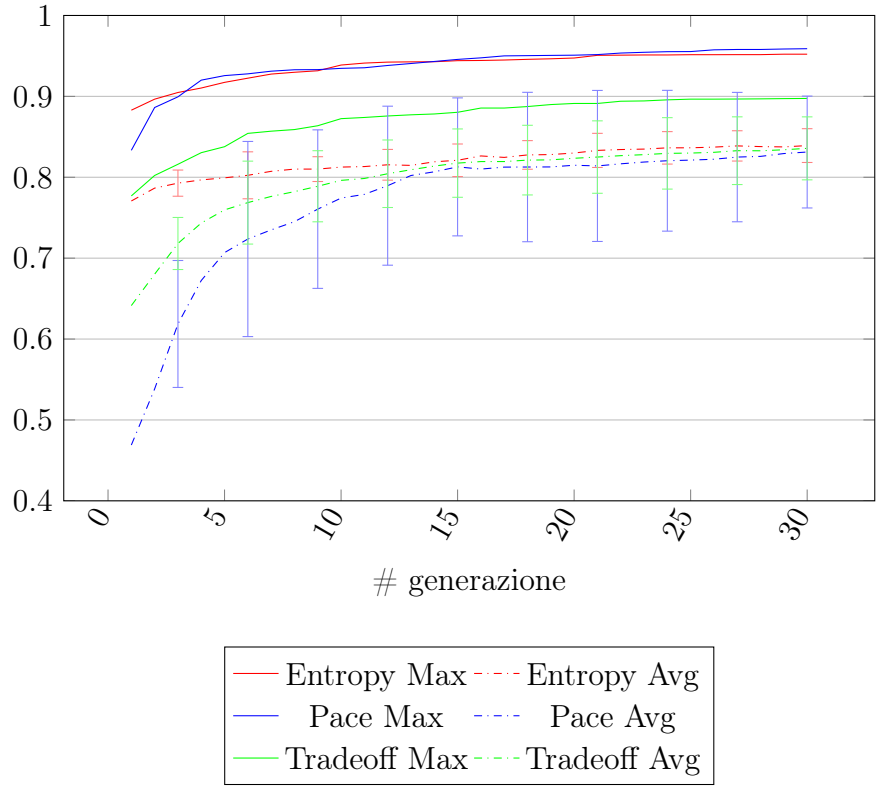
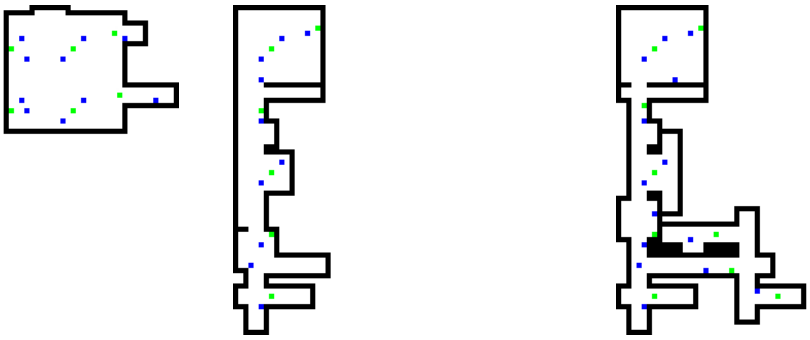


Figura 5.3: Statistiche dei processi evolutivi mostrati per il primo esperimento.

Per il secondo esempio abbiamo scelto un processo che ha generato, come più equilibrata, una mappa con una struttura più complessa e interessante rispetto al primo esempio. In Figura 5.5 sono mostrate le mappe più significative ottenute.

In Figura 5.3 sono mostrate le evoluzioni dei valori massimi e medi delle due funzioni di fitness, calcolati come la media dei relativi valori per ogni generazione su tutti i processi eseguiti. Si può osservare come sia l'entropia che il *pace* crescano in modo costante, seppur limitato una volta superata la metà del processo evolutivo. Il *pace* mostra un'evoluzione più netta, soprattutto nelle prime generazioni, probabilmente a causa del fatto che è più direttamente influenzato dalla struttura della mappa. L'evoluzione delle mappe *compromesso*, valutate usando 0.57 come peso per il valore dell'*entropia* e 0.43 per quello del *pace*, segue un andamento intermedio a quello dei due obiettivi.

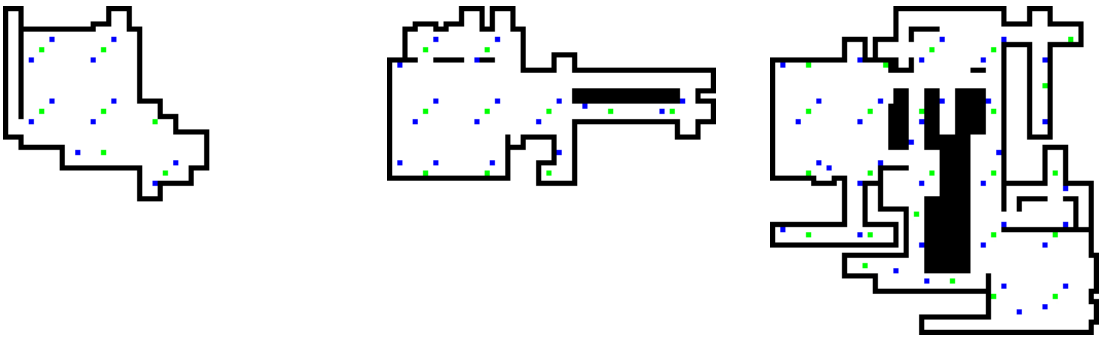


(a) Mappa con Entropia 0.724 e Pace 0.966

(b) Mappa con Entropia 0.965 e Pace 0.925

(c) Mappa con Entropia 0.988 e Pace 0.532

Figura 5.4: Mappe più significative ottenute nel primo esempio del primo esperimento.



(a) Mappa con Entropia 0.655 e Pace 0.971

(b) Mappa con Entropia 0.811 e Pace 0.892

(c) Mappa con Entropia 0.938 e Pace 0.335

Figura 5.5: Mappe più significative ottenute nel secondo esempio del primo esperimento.

5.1.2 Analisi delle mappe

La prima cosa che si nota guardando le mappe finali dei processi evolutivi è come la pura massimizzazione del *pace* tenda a creare mappe di dimensioni estremamente ridotte, solitamente composte da un'unica arena di dimensioni ridotte ed eventualmente qualche corridoio laterale: banalmente, una mappa in cui il contatto tra i giocatori è immediato e le distanze sempre ravvicinate portano a un ritmo di gioco sicuramente frenetico, ma probabilmente poco entusiasmante. Non sempre però alti valori di *pace* si traducono in mappe molto piccole: in Figura 5.6 è mostrato un esempio, l'unico ottenuto durante questo esperimento, di mappa che massimizzando il valore di *pace* ha prodotto una mappa piuttosto larga; l'alto ritmo di gioco è dovuto in questo caso all'elevata visibilità che si ha da un estremo all'altro della mappa (un altro esempio simile è la mappa mostrata nel capitolo precedente in Figura 4.2c). È evidente che, date le caratteristiche delle armi in gioco (e in particolare considerando come abbiamo parametrizzato i profili dei bot), per quanto appena detto il profilo *Assault* tende ad essere particolarmente avvantaggiato in mappe che presentano alti valori di *pace*. Esaminando le mappe con entropia massima si nota come queste siano composte specialmente da corridoi, e in generale da strutture più longilinee, spazi più lunghi e punti di accesso stretti, caratteristiche che favoriscono il profilo *Sniper*: l'arma *rifle* si adatta naturalmente a spazi lunghi, spazi che inoltre costringono a spostamenti longitudinali riducendo la possibilità di schivata, e che nel caso specifico aiutano lo *Sniper* rendendo meno limitante l'handicap derivante dal basso valore di visibilità assegnatogli. Queste mappe però tendono ad avere strutture più contorte, che abbassano la visibilità generale e di conseguenza la probabilità di trovare il nemico e quindi il ritmo di gioco. Le mappe che presentano buoni valori per entrambe le metriche sono tendenzialmente più semplici e mischiano caratteristiche di entrambi gli estremi: strutture più lineari e meno intricate che permettono una buona visuale sull'intero livello, mantenendo però una figura longilinea. La mappa mostrata in Figura 5.4b rappresenta un'ottima mappa considerando entrambi gli obiettivi posti per l'esperimento.

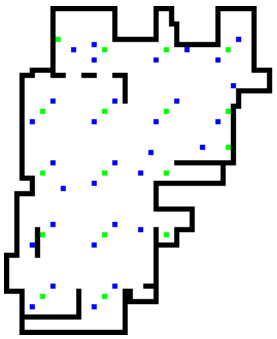
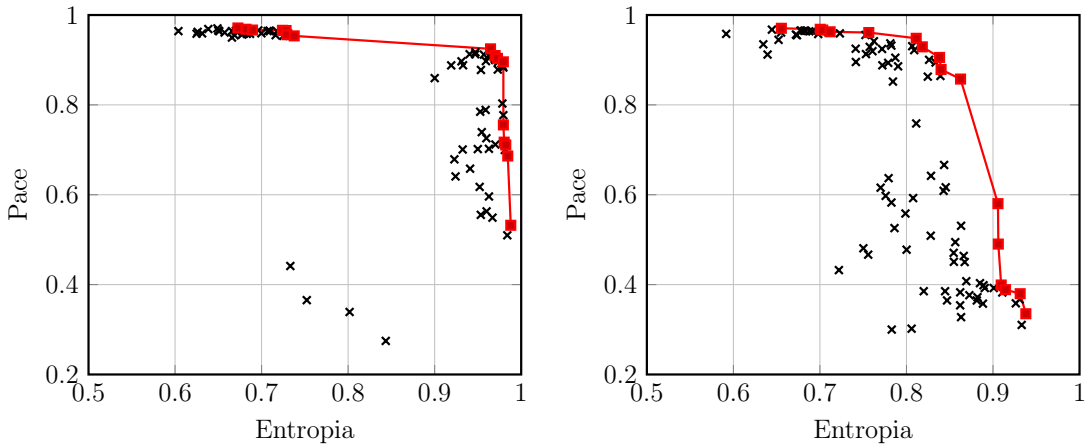


Figura 5.6: Mappa con Entropia 0.741 e Pace 0.928



(a) Primo esempio (b) Secondo esempio

Figura 5.7: Mappatura dei valori obiettivo delle mappe ottenute nei processi mostrati per il primo esperimento, con fronte di Pareto delle soluzioni ottime.

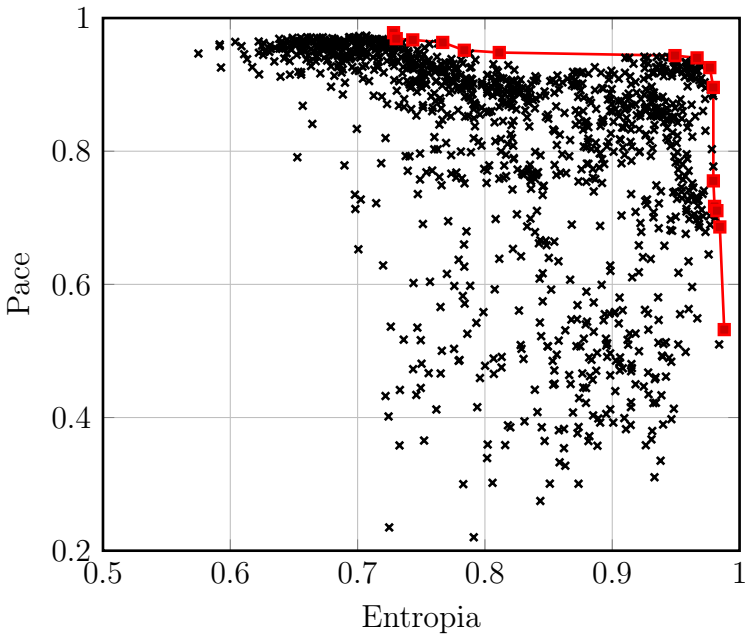


Figura 5.8: Mappatura congiunta dei valori obiettivo di tutte le mappe ottenute nel primo esperimento, con fronte di Pareto delle soluzioni ottime.

I grafici nelle figure 5.7 e 5.8 mostrano i due valori di fitness per ogni mappa ottenuta nell'ultima generazione, delimitati dal **fronte di Pareto** delle soluzioni ottime, rispettivamente per i due esempi analizzati e per l'intera popolazione dei 12 esperimenti effettuati. Essi rivelano che si sono riuscite a trovare mappe che costituiscono un buon compromesso tra i due obiettivi, con valori tendenti al limite massimo; evidenziano altresì il rapporto inverso che normalmente esiste tra i due: se si esclude il fitto gruppo di risultati in prossimità del "ginocchio" del fronte, e si considerano i punti di maggior concentrazione, si può notare un trend decrescente del *pace* rispetto all'*entropia*.

5.1.3 Analisi delle dinamiche

Dopo averle osservate a livello puramente strutturale abbiamo analizzato alcune dinamiche prodotte dalle mappe ottenute dalle evoluzioni; in particolare ci siamo concentrati sull'analisi delle uccisioni e delle morti, rilevandone le posizioni e in che modo sono avvenute tramite analisi visuale. In particolare ci siamo avvalsi di due tipi di analisi:

- **Heatmap** delle uccisioni (delle morti): una rappresentazione grafica della densità delle uccisioni (delle morti) avvenute nello stesso intorno; le zone rosse rappresentano le aree da cui sono state effettuate il maggior numero di uccisioni (dove si sono subite il maggior numero di uccisioni), mentre quelle blu le zone con minor concentrazione.

- Mappa delle **Kill Traces**: una rappresentazione, tramite segmenti, delle traiettorie dei colpi che hanno causato un'uccisione. I segmenti sono colorati in base alla distanza dell'uccisione, con il colore rosso a rappresentare uccisioni a breve raggio e il verde per uccisioni dalla lunga distanza. Ad un'estremità di ogni segmento un cerchio indica la posizione della vittima nell'istante di morte. Per ragioni di maggior visibilità è stato scelto di rappresentare solo un campione casuale delle traiettorie di dimensione massima pari al 10% del numero di blocchi navigabili nella mappa.

Sebbene le mappe "frenetiche" siano piuttosto banali e non rivelino particolari informazioni è comunque possibile notare alcune semplici dinamiche che contraddistinguono i due profili: specialmente riferendosi alle figure 5.16a e 5.19a

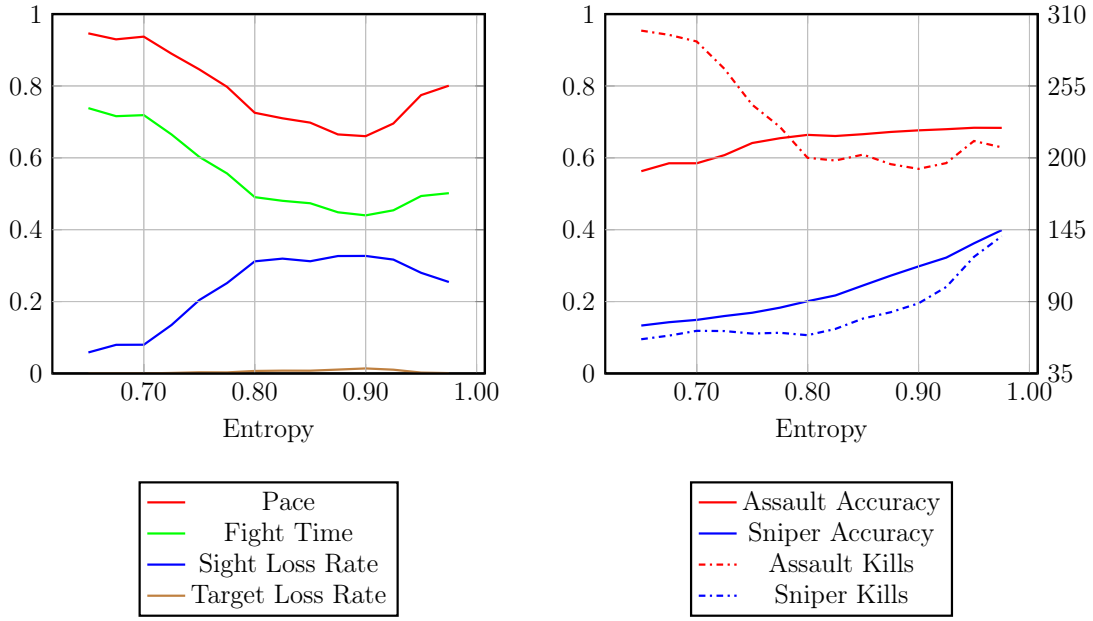
è possibile vedere come il profilo *assault* abbia massima efficacia al centro della stanza, dove raggiunge mediamente la distanza minima dal resto della mappa ed è in grado di effettuare il maggior numero di uccisioni. Viceversa, per il profilo *sniper* si nota una maggiore diffusione dei punti di uccisione, specialmente lungo le diagonali degli spazi, dove si ha una maggiore visibilità ed è possibile difendersi meglio dagli attacchi del nemico.

Una delle prime cose che si nota dall'analisi delle *heatmap* è come molti punti di alta concentrazione delle uccisioni siano condivisi tra i due profili. Ciò però non significa che le uccisioni avvengano nello stesso modo: confrontando le figure 5.12 con la 5.15 e la 5.18 con la 5.21 delle *kill traces* si nota come le uccisioni del bot *Assault* siano tendenzialmente a raggio medio-breve, mentre per il bot *Sniper* siano quasi esclusivamente dalla lunga distanza. Molti dei punti di uccisione condivisi sono angoli o incroci tra corridoi, almeno uno dei quali lungo; entrambi i profili si avvantaggiano di questa particolare costruzione: il bot *Assault* gode di un particolare vantaggio dagli angoli e in generale da ogni situazione in cui ci si può trovare improvvisamente davanti al nemico a breve distanza[3], e ciò è ben evidenziato in Figura 5.17c, nella quale si nota come quasi ad ogni intersezione tra corridoi o punti di accesso a diversi spazi ci sia una densità di uccisioni medio-alta; il profilo *sniper* invece, quando tali zone rappresentano punti terminali di lunghi corridoi e soprattutto se questi sono molteplici, si trova in una situazione di controllo dello spazio, e in generale in una posizione più sicura, grazie alla possibilità di spostarsi e sfruttare i muri come copertura. Osservando le mappe delle traiettorie si nota una predominanza di uccisioni dalla lunga distanza attraverso i corridoi più stretti, e la quasi assenza in altre situazioni, a confermare quanto detto nella sezione 5.1.2.

5.1.4 Relazioni con altre caratteristiche

In questa sezione mostriamo l'andamento di altre metriche in relazione a quelle esaminate nell'esperimento. Sono state selezionate solo alcune delle metriche viste nel capitolo 4, considerate più interessanti o che evidenziano maggiormente un collegamento. Il grafico in Figura 5.9a mostra l'andamento medio, considerando tutti i processi evolutivi ottenuti nell'esperimento, di statistiche relative alle ca-

[3]Tale situazione massimizza l'efficacia dell'arma *shotgun* e annulla parzialmente il difetto di mira assegnato al profilo.



(a) Statistiche delle mappe.

(b) Statistiche dei giocatori.

Figura 5.9: Relazioni tra le metriche misurate nel primo esperimento. I grafici delle uccisioni si basano sui valori di scala dell'asse destro.

ratteristiche delle mappe in funzione del valore, discretizzato a intervalli di 0.025, dell'*entropia*; il grafico in Figura 5.9b mostra invece l'andamento di statistiche relative ai bot.

A conferma di quanto detto nella sezione 5.1.2 si nota come il *pace* sia tendenzialmente inversamente proporzionale all'equilibrio; come già affermato questo è dovuto a una maggiore complessità e dimensione delle mappe più bilanciate, che causa anche un aumento del *Sight Loss Rate* (ovvero del tempo impiegato dai bot a ritrovare l'obiettivo una volta uscito dal proprio campo visivo) e di conseguenza un abbassamento del tempo di combattimento. Per valori di entropia maggiori di 0.90 si nota però un'interessante inversione di tendenza: ciò probabilmente è dovuto a un ulteriore allungamento e distensione delle mappe, che massimizza le distanze tra i due giocatori riducendo gli ostacoli (favorendo il bot *Sniper*), aumentando così allo stesso tempo la visibilità globale della mappa e dunque la facilità di localizzare l'avversario e iniziare uno scontro.

Per quanto riguarda le statistiche dei bot si nota un lieve aumento complessivo dell'accuratezza, dovuto sempre al restringimento e allungamento degli spazi nelle mappe più bilanciate. Inoltre si può osservare come l'equilibrio si sia ottenuto

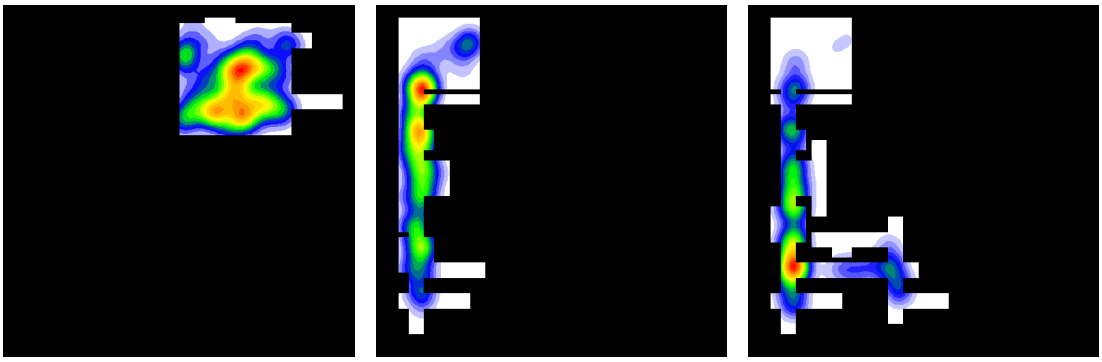
inizialmente grazie al peggioramento delle prestazioni del bot più abile, mentre per valori più alti di entropia sia dovuto a un importante aumento delle prestazioni del bot svantaggiato; anche questo si può attribuire a una maggior separazione degli spazi che, oltre una certa soglia, permettono inoltre più possibilità di fuga, altro fattore favorevole al bot *Sniper*. Anche i grafici del numero delle uccisioni evidenziano come un ritmo più basso renda quest'ultimo più performante, al contrario del profilo *Assault*.

5.1.5 Considerazioni finali

L'esperimento ha rivelato alcuni fatti interessanti. Innanzitutto è una prima dimostrazione dell'efficacia dell'applicazione di un algoritmo evolutivo con obiettivi multipli nella generazione di mappe che soddisfano diversi requisiti: c'è stata un'ampia esplorazione dello spazio delle soluzioni e l'evoluzione congiunta delle funzioni obiettivo è stata costante nei processi, e si sono riuscite a generare alcune mappe che rappresentano buoni risultati sia dal punto di vista dell'equilibrio che da quello del ritmo di gioco.

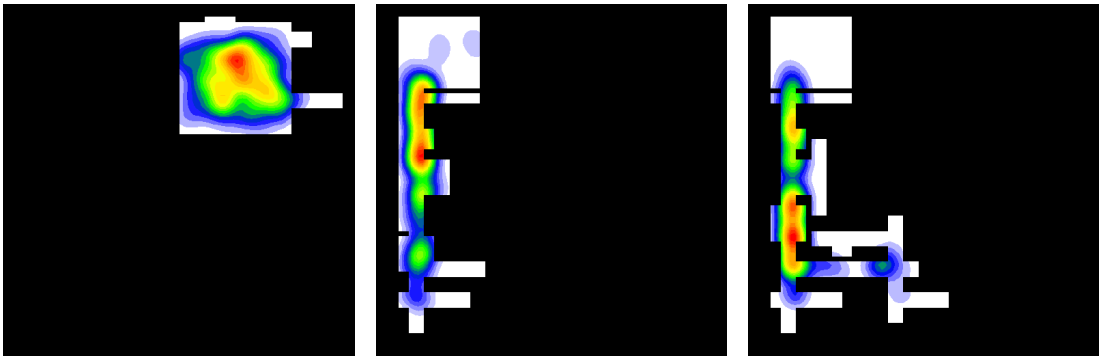
Nei due esempi mostrati il miglioramento dell'equilibrio dall'inizio alla fine del processo evolutivo è stato intorno al 10%; bisogna considerare che il valore dell'entropia ha una crescita molto limitata per valori tendenti a 1, e il nuovo modello dell'intelligenza artificiale produce un maggiore equilibrio generale per cui, nonostante l'ampio divario di abilità dei due bot, il valore dell'entropia ha assunto valori piuttosto alti sin dalle prime generazioni. Premettendo tali osservazioni una simile evoluzione si può considerare un successo, confermando dunque i risultati ottenuti da Stucchi[5] sull'efficacia del metodo nella generazione automatica di mappe bilanciate. Per quanto riguarda il *pace* l'evoluzione è stata più significativa e rapida, ma il problema risulta troppo semplice in quanto vengono trovate frequentemente strutture banali che ne massimizzano il valore. Ciò è causato dalla definizione che abbiamo dato del *pace*, che misura solamente la frequenza degli scontri; come accennato nella sezione 4.3.2 un ritmo troppo frenetico può diventare frustrante, e dal nostro punto di vista produce mappe poco interessanti. Tuttavia nell'ambito dell'esperimento in questione, con evoluzione a obiettivi multipli e simultanei, ha contribuito positivamente al raggiungimento dello scopo principale e alla creazione di mappe interessanti, non essendo stato l'unico aspetto che si è cercato di massimizzare durante i processi.

Si sono riuscite a rilevare anche diverse informazioni interessanti su come particolari strutture nella mappa abbiano influenzato i risultati e le prestazioni dei giocatori: come visto nella sezione 5.1.3 l'allungamento e restringimento degli spazi, le maggiori dimensioni generali e linearità della mappa hanno evidentemente portato vantaggio al profilo *Sniper*. Inaspettatamente queste variazioni hanno portato a cambiamenti non lineari nelle statistiche di gioco, con una progressione dei valori che ha mostrato un'inversione di tendenza, dimostrando che il problema è complesso e riconducibile a diversi fattori.



(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.10: Heat Maps delle uccisioni del bot Assault nel primo esempio del primo esperimento.

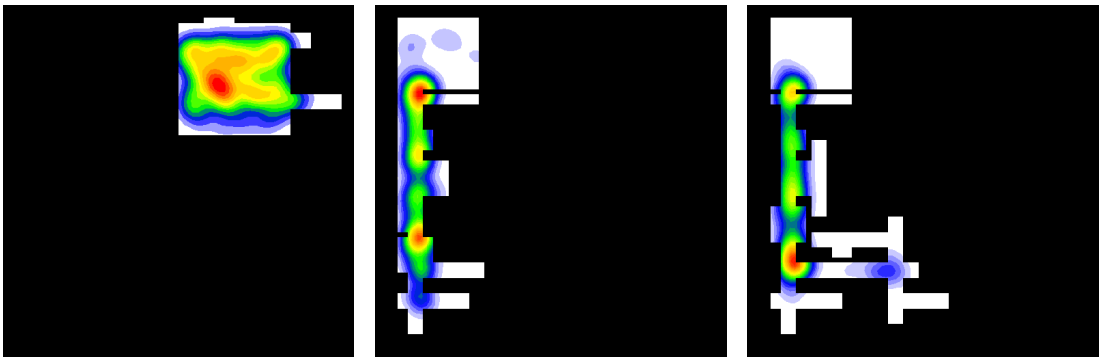


(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.11: Heat Maps delle morti del bot Assault nel primo esempio del primo esperimento.

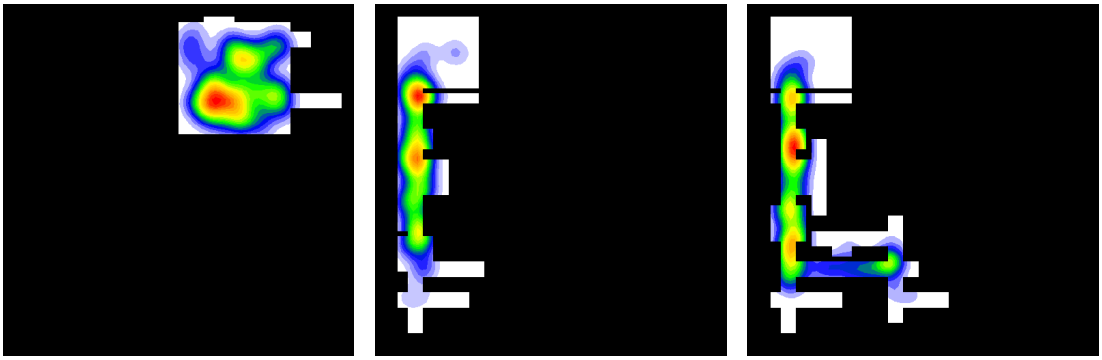
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.12: Mappe delle Kill Traces del bot Assault nel primo esempio del primo esperimento.



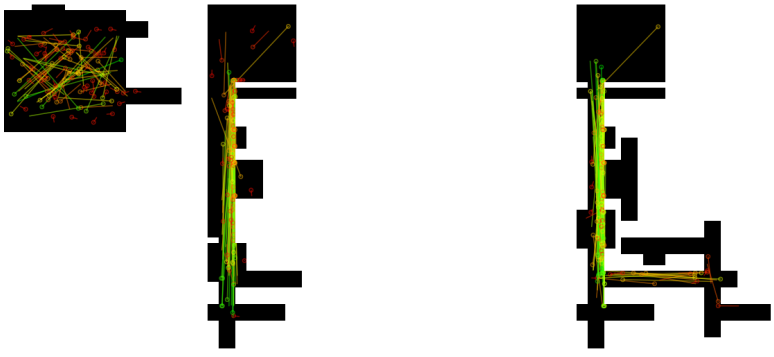
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.13: Heat Maps delle uccisioni del bot Sniper nel primo esempio del primo esperimento.



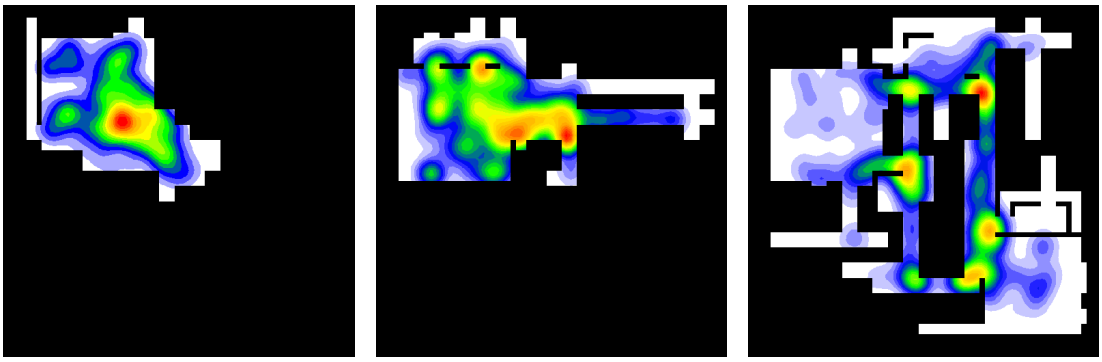
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.14: Heat Maps delle morti del bot Sniper nel primo esempio del primo esperimento.



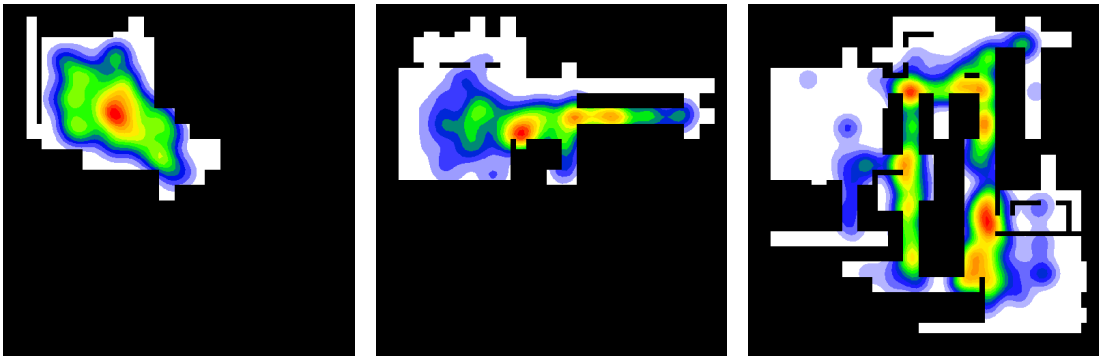
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.15: Mappe delle Kill Traces del bot Sniper nel primo esempio del primo esperimento.



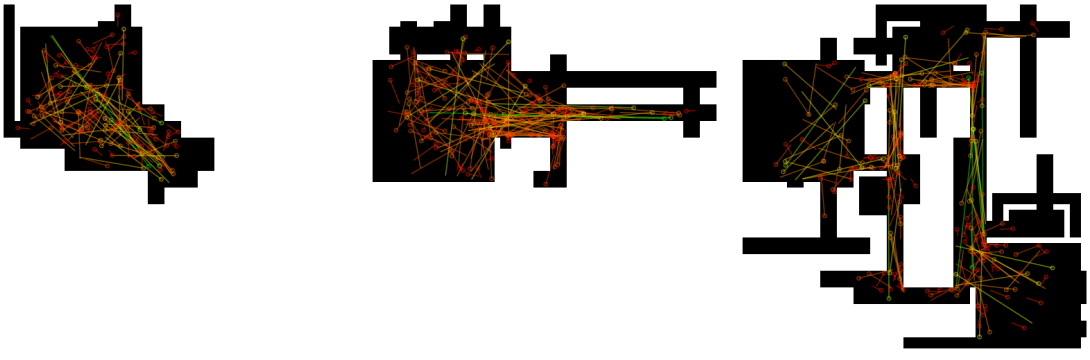
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.16: Heat Maps delle uccisioni del bot Assault nel secondo esempio del primo esperimento.



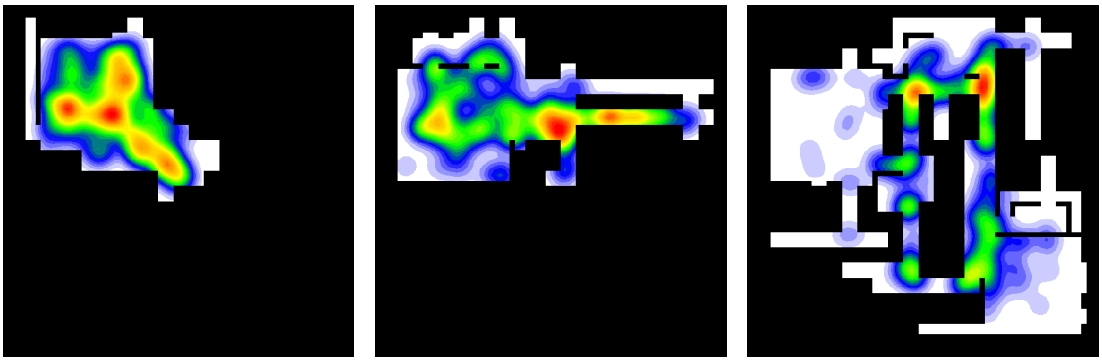
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.17: Heat Maps delle morti del bot Assault nel secondo esempio del primo esperimento.



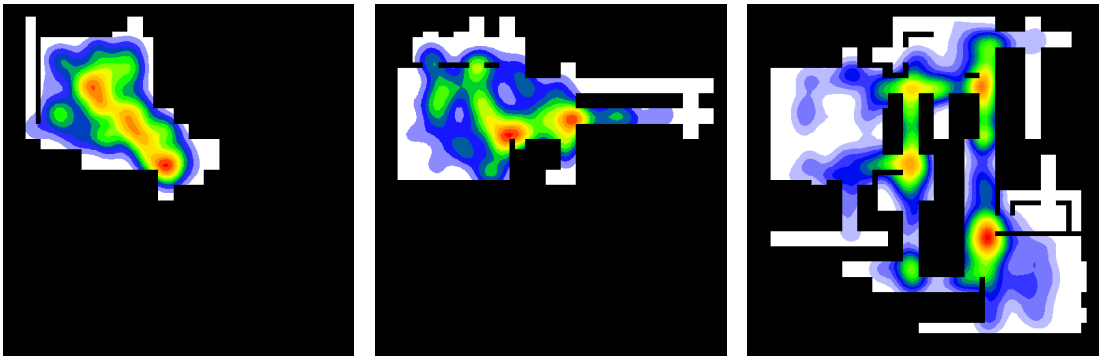
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.18: Mappe delle Kill Traces del bot Assault nel secondo esempio del primo esperimento.



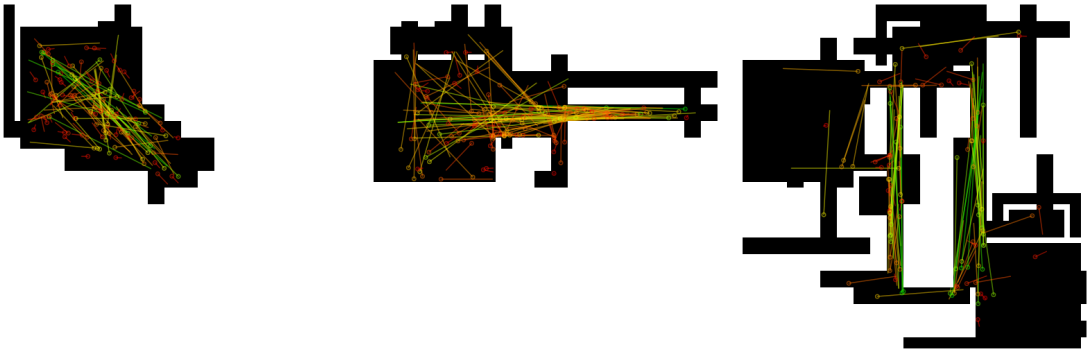
(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.19: Heat Maps delle uccisioni del bot Sniper nel secondo esempio del primo esperimento.



(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.20: Heat Maps delle morti del bot Sniper nel secondo esempio del primo esperimento.



(a) Mappa frenetica (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.21: Mappe delle Kill Traces del bot Sniper nel secondo esempio del primo esperimento.

Questa pagina è stata intenzionalmente lasciata vuota.

5.2 Secondo esperimento: Equilibrio e Serie di uccisioni

In questo esperimento ci siamo posti come primo obiettivo ancora la massimizzazione dell'equilibrio tra i giocatori, e come secondo obiettivo la massimizzazione delle serie di uccisioni medie, relativi ai valori di fitness *entropia* e *kill streak average* illustrati nella sezione 4.3.

Abbiamo scelto di utilizzare il profilo *assaltatore* (Assault), già utilizzato nel primo esperimento, e il profilo *"folle"* (Berserker), dotato di lanciarazzi, che cerca lo scontro ravvicinato incurante dei rischi che corre.

5.2.1 Setup e statistiche di esecuzione

Come per il primo esperimento abbiamo assegnato per prima cosa i valori di abilità tranne quello generale. Abbiamo mantenuto invariato il profilo **Assault**, le cui caratteristiche sono descritte nella sezione 5.1.1, lasciando invariati gli attributi assegnati nel primo esperimento. Per rappresentare il profilo **Berserker** gli abbiamo assegnato un arma che porta rischi anche per il suo possessore come il *lanciarazzi*; abbiamo ridotto notevolmente l'abilità di mira e l'abilità di rispettare le distanze, e aumentato l'abilità di combattimento (che diminuisce la staticità del personaggio nella fase di mira). In questo modo abbiamo modellato un giocatore frenetico, sempre in movimento non curante dei rischi che corre.

Anche in questo caso le abilità generali dei due bot sono state sbilanciate in partenza in base ai valori di entropia, visibili in Figura 5.22, calcolati simulando alcune partite usando gli attributi appena descritti sulla mappa *fragplaza*. Per

Profilo	*Assault*	*Berserker*
Arma	*Shotgun*	*Rocket Launcher*
Abilità Generale	*20*	*90*
Mira	*66*	*50*
Visibilità	*100*	*100*
Velocità di Visualizzazione	*100*	*100*
Distanziamento	*100*	*66*
Abilità di Combattimento	*100*	*120*

Tabella 5.2: Parametri di setup dei bot per l'esperimento 2.

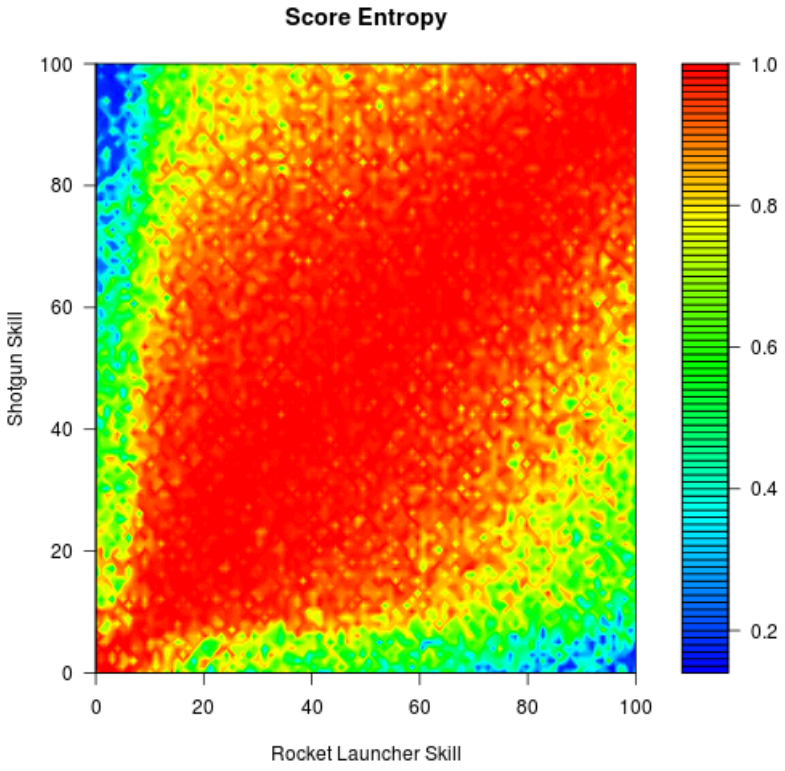


Figura 5.22: Livelli di entropia per i profili Assault e Berserker a diversi livelli di *skill*

questo esperimento però, al contrario del precedente, abbiamo assegnato un valore di *skill* generale basso al profilo *Assault*, e un valore alto al *Berserker*. In tabella 5.2 sono riassunti i valori utilizzati per la realizzazione di questo esperimento.

L'esperimento è stato realizzato eseguendo 12 processi evolutivi con algoritmo genetico multi-obiettivo sulla rappresentazione di mappa *All-Black*, con genotipo formato come rappresentazione di 15 arene e 50 corridoi, simulando partite della durata di 30 minuti ciascuna. Ogni processo evolutivo ha impiegato tra le 2 e le 3 ore, e ha usato una popolazione di 50 candidati per 30 generazioni, con *crossover* matriciale, selezione dei candidati tramite *tournament selection* a due candidati, intervallo di mutazione 0.3 e probabilità di crossover 0.3.

Abbiamo selezionato due dei processi evolutivi svolti per mostrare le caratteristiche e le dinamiche delle mappe ottenute. Per entrambi gli esempi vengono mostrate le mappe che massimizzano i due valori di fitness, e una mappa scelta come "miglior compromesso", che massimizza un punteggio calcolato come som-

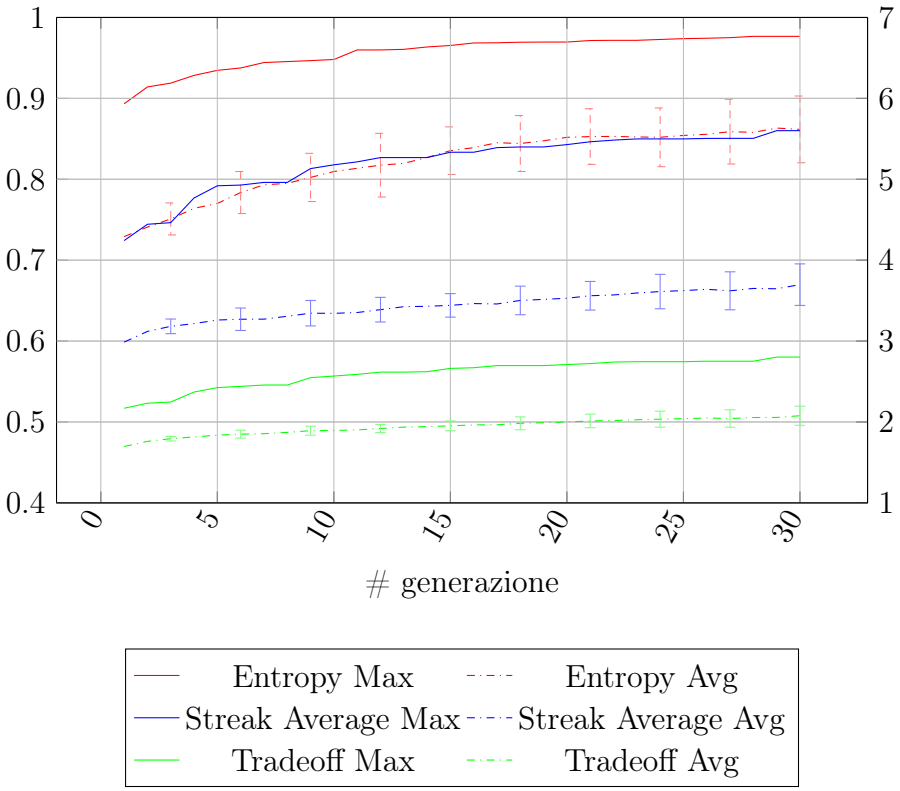


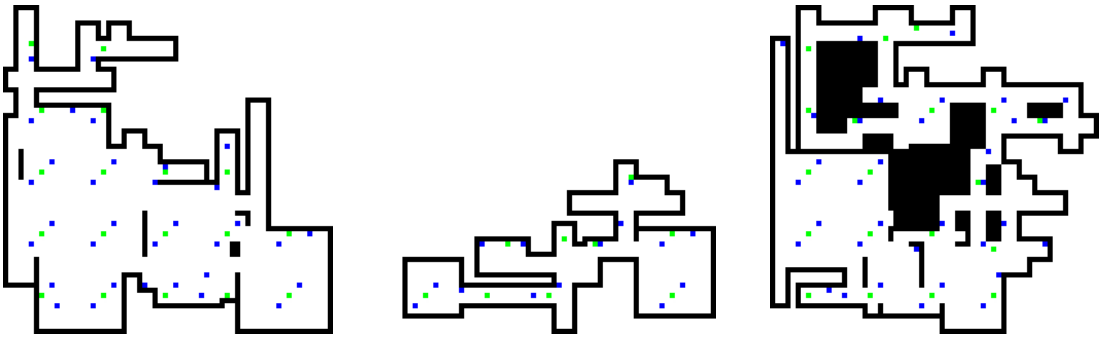
Figura 5.23: Statistiche dei processi evolutivi mostrati per il secondo esperimento. I grafici di *Streak* e *Tradeoff* si basano sui valori di scala dell'asse destro.

ma pesata delle fitness, usando 0.75 come peso per il valore dell'*entropia* e 0.25 per quello della media delle serie di uccisioni.

Per differenziare l'analisi rispetto al primo esperimento abbiamo scelto come primo esempio il processo evolutivo che ha prodotto la mappa con la media più alta di serie di uccisioni. Come secondo esempio abbiamo scelto un processo che ha mostrato buone statistiche di evoluzione e che ha generato mappe finali con una struttura interessante.

In Figura 5.24 sono mostrate le mappe più significative relative al primo esempio, e in figura 5.25 quelle relative al secondo.

In Figura 5.23 sono mostrate le evoluzioni delle medie dei valori massimi e medi delle due funzioni di fitness durante l'evoluzione tenendo conto di tutti i processi evolutivi effettuati. Rispetto a quanto osservato nella sezione 5.1.1, relativamente all'evoluzione dell'entropia nel primo esperimento, e tenendo conto delle stesse osservazioni, si ha una crescita più significativa dell'equilibrio. Per

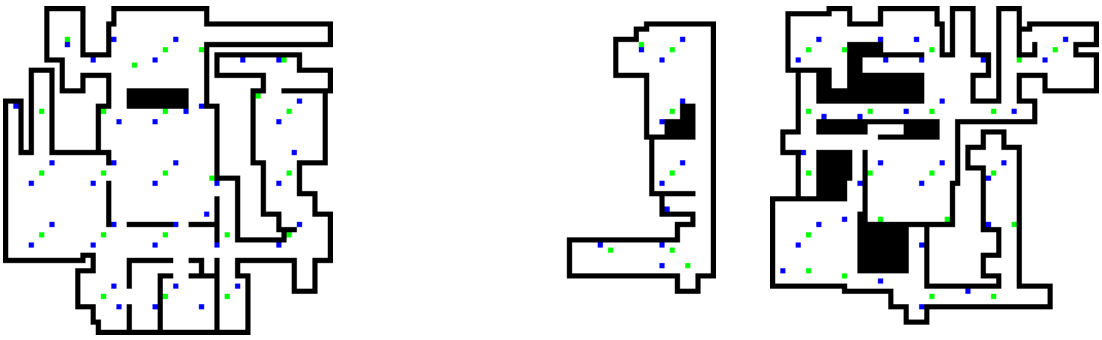


(a) Mappa con Entropia 0.467 e Kill Streak medio 6.91

(b) Mappa con Entropia 0.869 e Kill Streak medio 4.87

(c) Mappa con Entropia 0.971 e Kill Streak medio 2.00

Figura 5.24: Mappe più significative ottenute nel primo esempio del secondo esperimento.



(a) Mappa con Entropia 0.517 e Kill Streak medio 5.48

(b) Mappa con Entropia 0.872 e Kill Streak medio 4.86

(c) Mappa con Entropia 0.987 e Kill Streak medio 1.68

Figura 5.25: Mappe più significative ottenute nel secondo esempio del secondo esperimento.

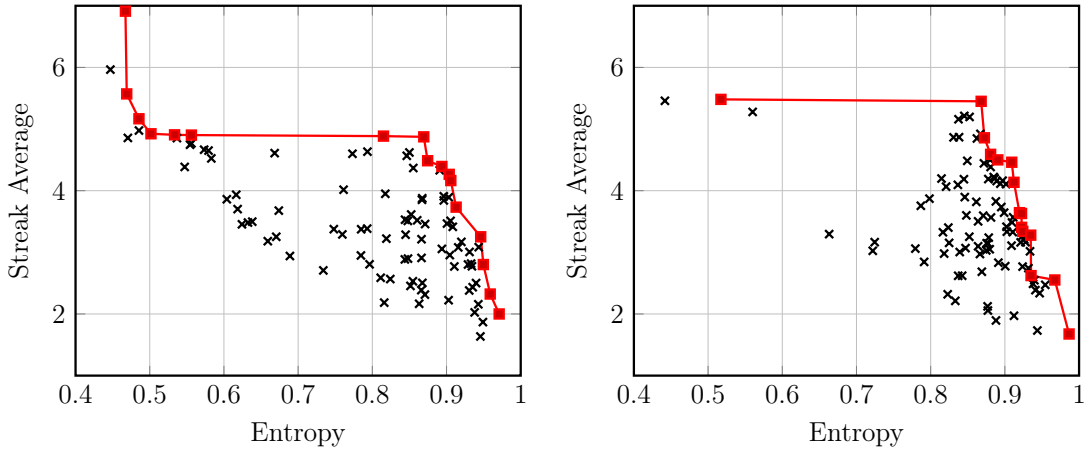
quanto riguarda le serie di uccisioni medie si ha un incremento del valore massimo dalla prima all'ultima generazione di circa 1.3 uccisioni, non molto significativo: ovviamente la difficoltà nell'aumentare le serie di uccisioni cresce in modo esponenziale per cui è difficile avere un incremento costante, soprattutto considerando che il secondo obiettivo dell'esperimento, la ricerca del bilanciamento, è probabilmente responsabile dell'effetto opposto. Entrambi comunque mostrano una progressione costante, e così anche l'evoluzione delle mappe compromesso.

5.2.2 Analisi delle mappe

Le mappe con le serie di uccisioni più alte, corrispondenti a bassa entropia, presentano una struttura molto aperta, con diverse stanze larghe e pochi corridoi. Queste caratteristiche (tenendo comunque conto del divario di abilità tra i due profili) favoriscono il profilo *Berserker*, soprattutto considerando le armi in gioco: la presenza di spazi ampi e la mancanza di spazi contorti da una parte sfavorisce lo *shotgun* e le armi a corto raggio in generale, dall'altra favorisce il lanciarazzi che trova il massimo vantaggio dalla media distanza. Al contrario le mappe più bilanciate presentano spazi molto più ristretti, diversi corridoi, angoli e cambi di direzione. Inoltre essendo più larghe riducono il ritmo di gioco e di conseguenza il numero delle uccisioni, altro fattore che probabilmente favorisce l'equilibrio.

Dai grafici nelle figure 5.26 e 5.27 si nota come un incremento nella media delle serie di uccisioni, anche non molto significativo, corrisponda solitamente a un drastico aumento dello squilibrio: evidentemente vi è una difficoltà esponenziale nel mantenere ed alimentare le serie di uccisioni, e dunque è necessario creare un divario sempre più ampio tra i due giocatori. Si sono avute alcune eccezioni, come nel primo esempio mostrato, in cui la media delle serie di uccisioni ha avuto un'evoluzione netta rispetto alla decrescita dell'entropia, comunque già a livelli che indicano un netto sbilanciamento tra i due bot.

Le mappe compromesso mostrate nei due esempi sono entrambe di dimensioni ridotte rispetto alle mappe ottime agli estremi, e presentano caratteristiche simili. Tuttavia non è evidente una struttura in generale più vataggiosa per il bot *assault*, meno abile, se non forse per una ridotta possibilità di fuga che costringe i giocatori a scontri più ravvicinati.



(a) Primo esempio (b) Secondo esempio

Figura 5.26: Mappatura dei valori obiettivo delle mappe ottenute nei processi mostrati per il secondo esperimento, con fronte di Pareto delle soluzioni ottime.

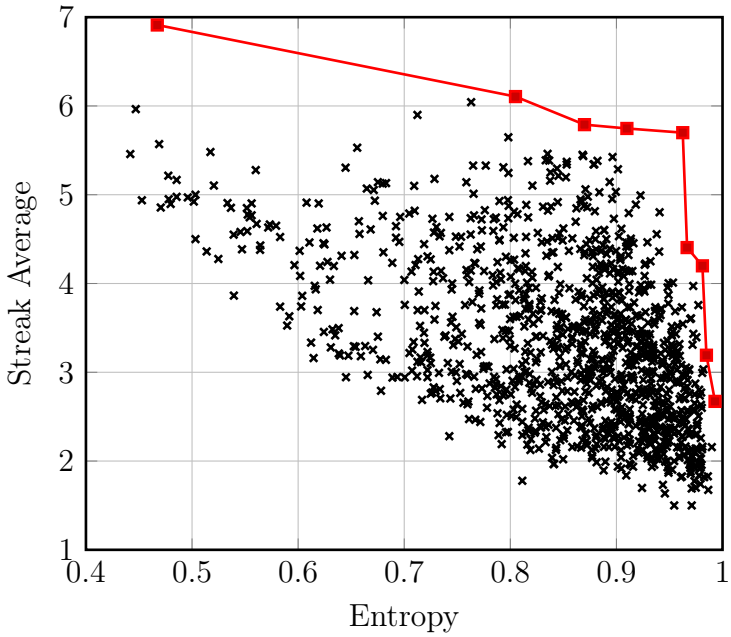


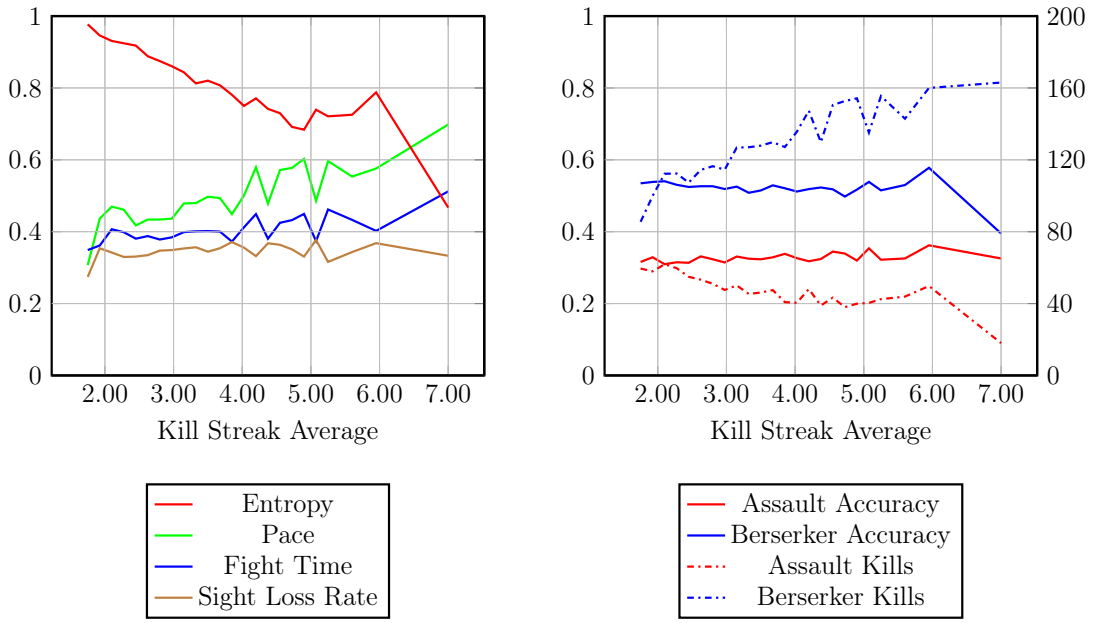
Figura 5.27: Mappatura congiunta dei valori obiettivo di tutte le mappe ottenute nel secondo esperimento, con fronte di Pareto delle soluzioni ottime.

5.2.3 Analisi delle dinamiche

Come nel primo esempio le uccisioni da parte del profilo *Assault* sono concentrate nelle strettoie, negli angoli e negli incroci. Le morti invece sono distribuite specialmente lungo i corridoi quando presenti: la vicinanza dell'avversario ai muri rende particolarmente efficace il lanciarazzi grazie all'area d'effetto dei suoi proiettili, e dunque i corrodoi rappresentano spazi ottimali per il profilo *Berserker*. In generale le morti causate da quest'ultimo sono più numerose in situazioni simili, e comunque a ridosso dei muri, come si vede dalle figure 5.30c e 5.36c, e anche in Figura 5.30a specialmente se confrontata con quelle causate dal suo avversario mostrate in Figura 5.33a; al contrario quando l'avversario è al centro della stanza l'efficacia del lanciarazzi diminuisce, così come la concentrazione delle morti del bot *Assault* al centro delle stanze. La presenza di muri non rappresenta invece in generale un vantaggio per lo *shotgun*, se non, per motivi diversi e già osservati, quando causano frequenti cambi di direzione. Le posizioni del bot *Berserk* durante le uccisioni sono molto più distribuite, poichè causate da un'arma efficace da una distanza più lunga che permette un maggiore spazio di manovra.

5.2.4 Relazioni con altre caratteristiche

Avendo effettuato nel primo esperimento un'analisi delle statistiche relativamente al valore di entropia per questo esperimento abbiamo optato per un'analisi rispetto alla media delle serie di uccisioni. Il grafico in Figura 5.28a evidenzia una proporzionalità diretta del *pace* rispetto alla serie di uccisioni media, e inversa dell'entropia. Anche in questo caso si nota una lieve inversione di tendenza per serie maggiori di 5. Queste inversioni possono essere semplicemente causate da come è stato posto il problema stesso: essendo l'obiettivo la massimizzazione contemporanea di più metriche si tende ad avere una progressione lineare per valori medio-bassi, mentre per valori maggiori si ha una concentrazione dei risultati ottimi tenendo conto di entrambi gli obiettivi. Questo ci da alcuni dati importanti: l'algoritmo evolutivo, dopo una prima fase di esplorazione in cui si assiste a un'evoluzione separata degli obiettivi, riesce a derivare delle mappe ottime da entrambi i punti di vista; allo stesso tempo viene reso evidente che esistono alcuni collegamenti tra le caratteristiche delle mappe.



(a) Statistiche delle mappe.

(b) Statistiche dei giocatori.

Figura 5.28: Relazioni tra le metriche misurate nel secondo esperimento. I grafici delle uccisioni si basano sui valori di scala dell'asse destro.

Le statistiche dei bot, mostrate in Figura 5.28b, mostrano prevedibilmente che all'aumentare della media delle serie di uccisioni, determinato principalmente dal bot più abile, cresce il divario di punteggio. Il bot meno abile evidentemente non contribuisce in modo significativo alle serie.

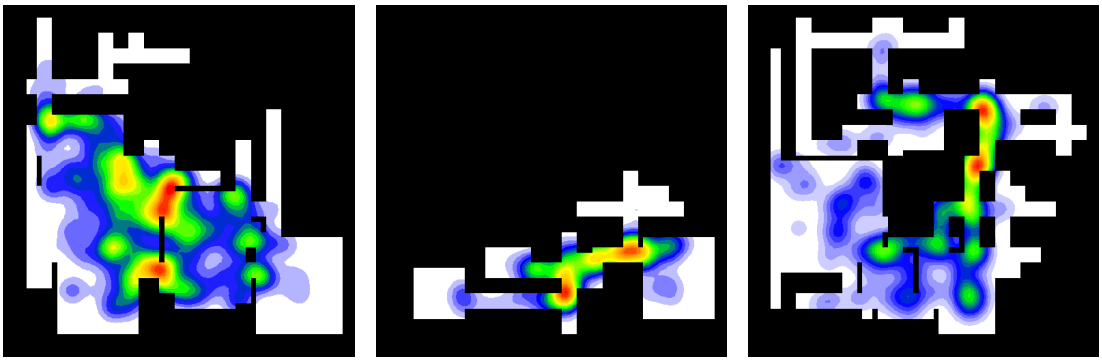
Lo stesso grafico evidenzia, sebbene non sia un fatto strettamente collegato all'obiettivo dell'esperimento, come nelle prime generazioni le prestazioni del bot *Berserker* crescano nonostante un lieve peggioramento nella precisione di mira, e viceversa come le prestazioni del bot *Assault* decrescano nonostante un lieve aumento dell'accuratezza, a dimostrazione della molteplicità degli aspetti che in una mappa possono essere sfruttati e che possono incidere sull'esito di una partita.

5.2.5 Considerazioni finali

Anche in questo esperimento viene riconfermata l'efficacia del metodo nella ricerca di mappe equilibrate, con risultati analoghi al primo esperimento.

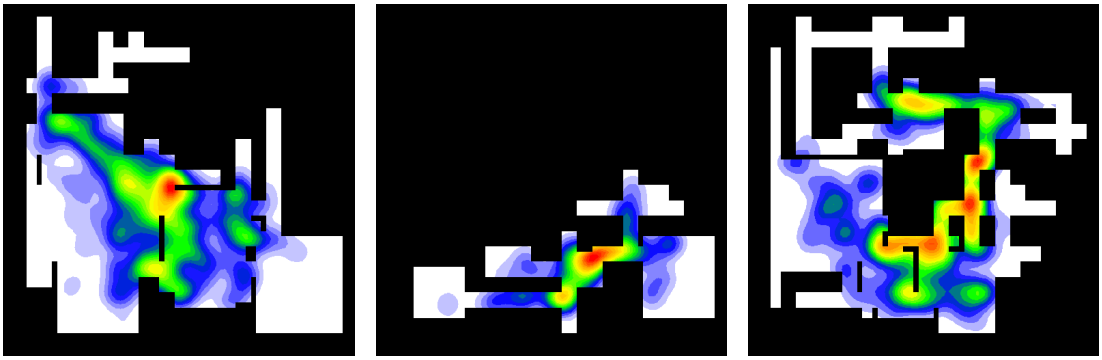
Prevedibilmente i risultati ottenuti per le serie di uccisioni sono fortemente correlati al divario prestazionale dei due bot. Nonostante si siano ottenute mappe che rappresentano un buon compromesso relativamente agli obiettivi cercati non è evidente quali siano, e se esistano o meno, le caratteristiche delle mappe che favoriscono l'incremento delle serie senza generare squilibrio tra i giocatori.

Anche in questo esperimento è stato comunque possibile ottenere qualche informazione sulle conseguenze che la struttura delle mappe ha sulle dinamiche di gioco con i profili utilizzati, soprattutto sull'effetto dei muri sulle armi ad area d'impatto, come osservato nella sezione 5.2.3.



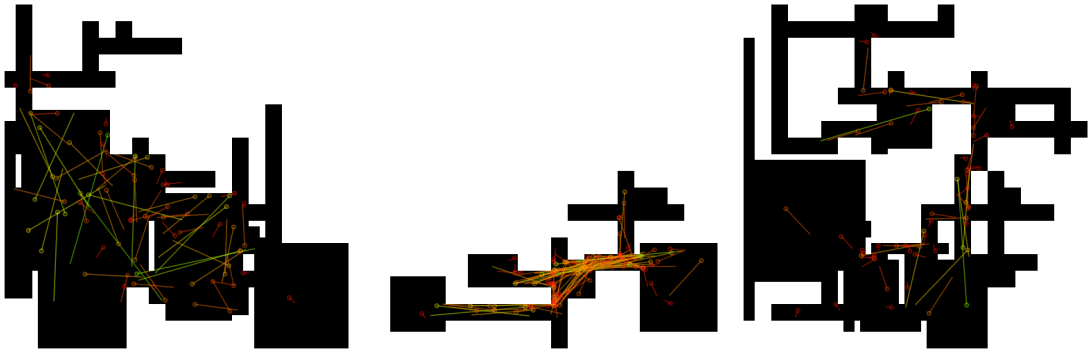
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.29: Heat Maps delle uccisioni del bot Assault nel primo esempio del secondo esperimento.



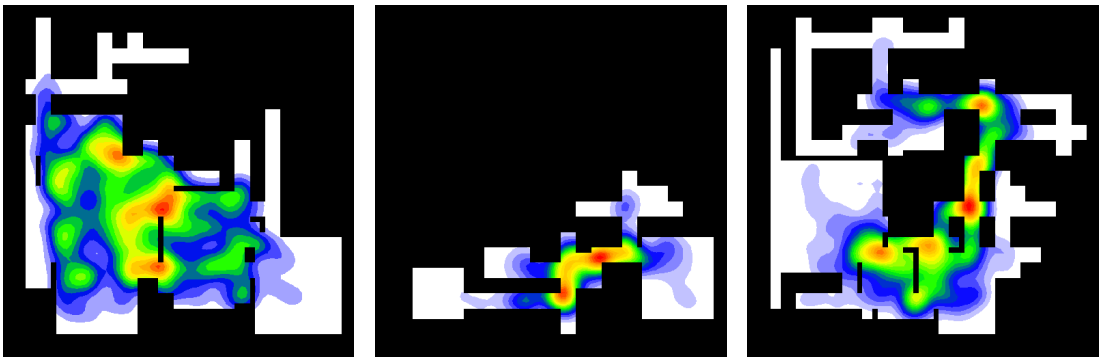
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.30: Heat Maps delle morti del bot Assault nel primo esempio del secondo esperimento.



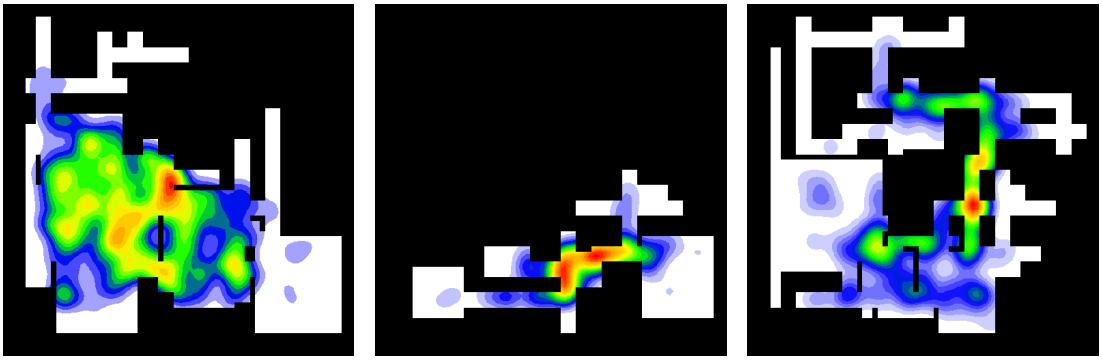
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.31: Mappe delle Kill Traces del bot Assault nel primo esempio del secondo esperimento.



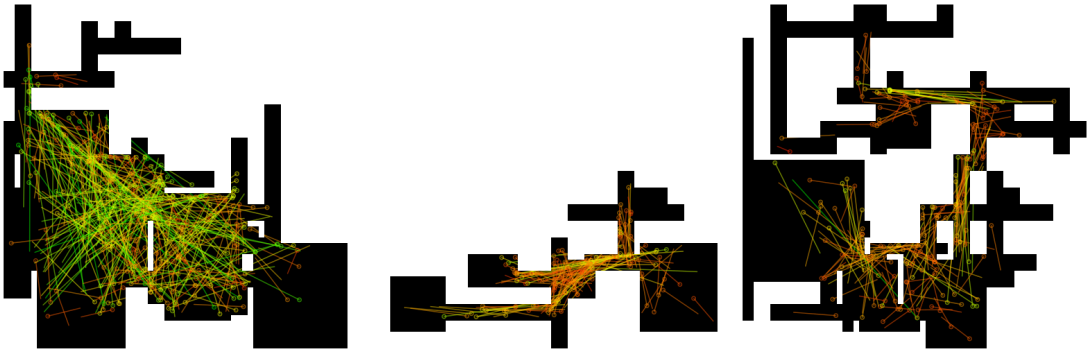
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.32: Heat Maps delle uccisioni del bot Berserker nel primo esempio del secondo esperimento.



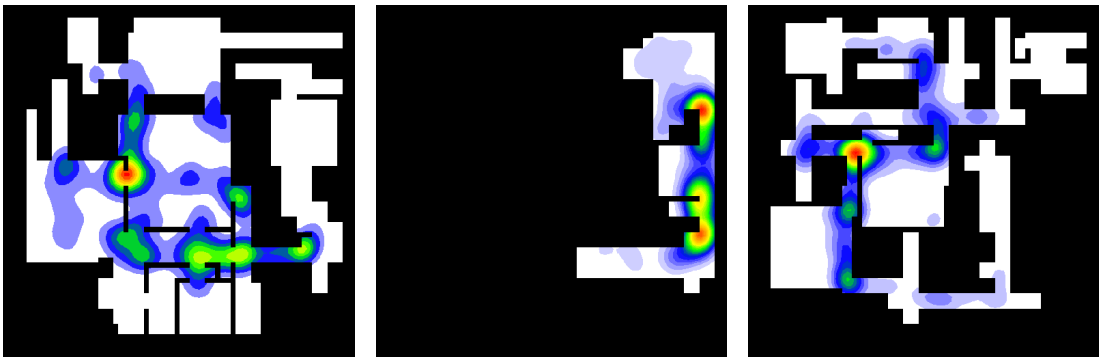
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.33: Heat Maps delle morti del bot Berserker nel primo esempio del secondo esperimento.



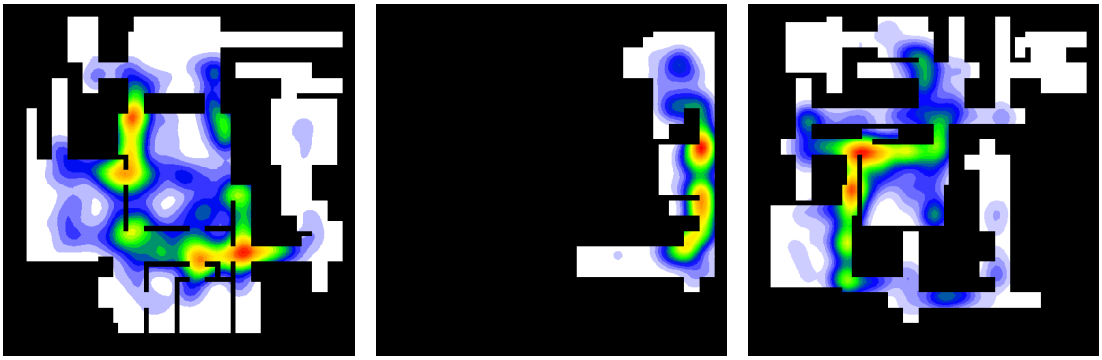
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.34: Mappe delle Kill Traces del bot Berserker nel primo esempio del secondo esperimento.



(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.35: Heat Maps delle uccisioni del bot Assault nel secondo esempio del secondo esperimento.

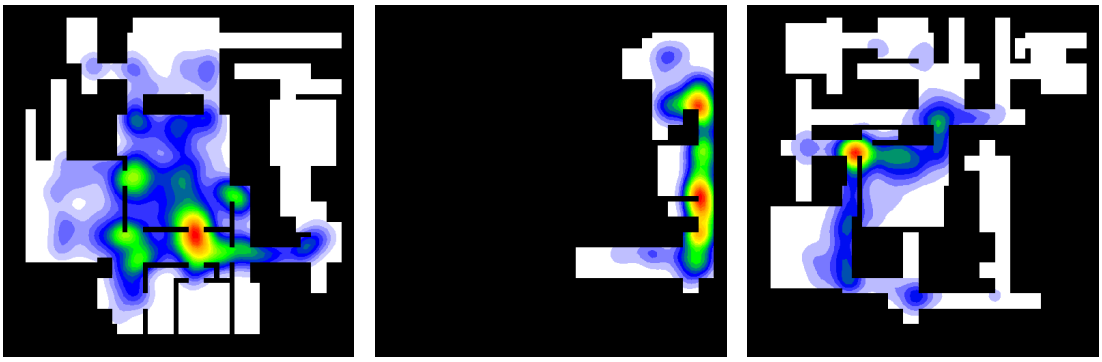


(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.36: Heat Maps delle morti del bot Assault nel secondo esempio del secondo esperimento.

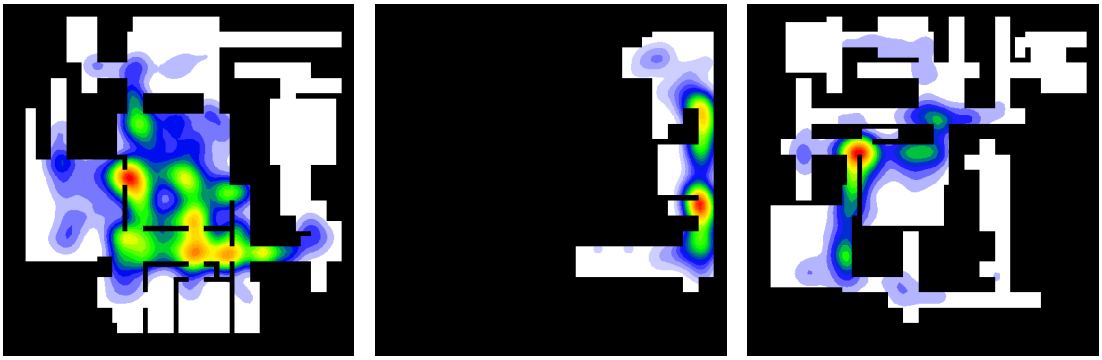
(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.37: Mappe delle Kill Traces del bot Assault nel secondo esempio del secondo esperimento.



(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.38: Heat Maps delle uccisioni del bot Berserker nel secondo esempio del secondo esperimento.



(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.39: Heat Maps delle morti del bot Berserker nel secondo esempio del secondo esperimento.

(a) Mappa serie uccisioni (b) Mappa compromesso (c) Mappa bilanciata

Figura 5.40: Mappe delle Kill Traces del bot Berserker nel secondo esempio del secondo esperimento.

Questa pagina è stata intenzionalmente lasciata vuota.

5.3 Terzo esperimento: Dispersività

Per l'ultimo esperimento mostrato, a singolo parametro, abbiamo ricercato la "dispersività" delle mappe, massimizzando il valore del *Target Loss Rate* che rappresenta la percentuale di combattimenti che si sono conclusi senza vittime.

Per questo esperimento, non dovendo cercare il bilanciamento, abbiamo usato due profili molto simili: il primo è il profilo *Berserker* già illustrato nella sezione 5.2; al secondo profilo, *Artilleryman*, è stato assegnato a sua volta il lanciarazzi, ma è stata data più importanza alla mira e alla distanza, penalizzando la rapidità d'azione.

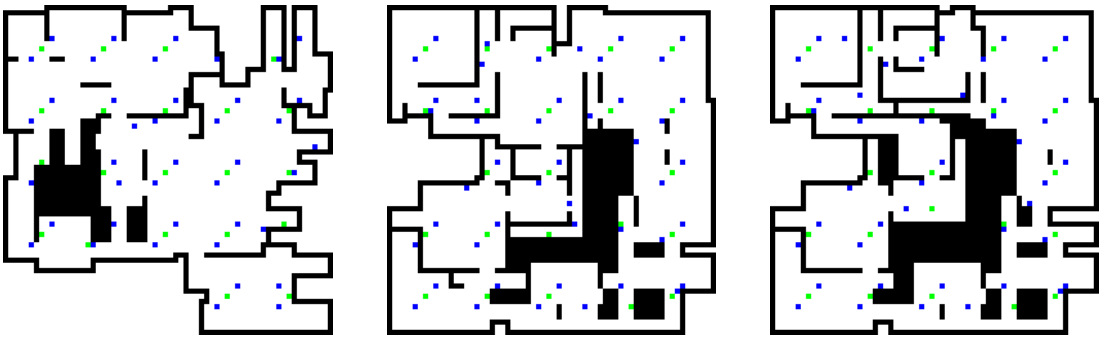
5.3.1 Setup e statistiche di esecuzione

A differenza dei precedenti esperimenti è stato assegnato lo stesso valore di *skill* a entrambi i bot. I parametri del profilo *Berserker* sono gli stessi usati per il secondo esperimento, e sono mostrati in tabella 5.2. Per il profilo **Artilleryman** invece, che modella un giocatore con un comportamento più cauto, sono stati assegnati valori più alti di mira e distanziamento, mantenendo gli tutti attributi neutri salvo una forte penalizzazione sull'immobilità durante la mira. I valori utilizzati per l'esperimento sono riassunti in Tabella 5.3.

Abbiamo realizzato l'esperimento eseguendo 12 processi evolutivi con algoritmo genetico a singolo obiettivo sulla rappresentazione di mappa *All-Black*, con genotipo formato come rappresentazione di 20 arene e 60 corridoi, lievemente maggiori rispetto agli esperimenti precedenti per favorire la creazione di mappe più complesse, considerando l'obiettivo dell'esperimento. Come nei casi precedenti sono state simulate partite della durata di 30 minuti ciascuna.

Profilo	*Berserker*	*Artilleryman*
Arma	*Rocket Launcher*	*Rocket Launcher*
Abilità Generale	*75*	*75*
Mira	*50*	*100*
Visibilità	*100*	*100*
Velocità di Visualizzazione	*100*	*100*
Distanziamento	*66*	*100*
Abilità di Combattimento	*120*	*25*

Tabella 5.3: Parametri di setup dei bot per l'esperimento 3.

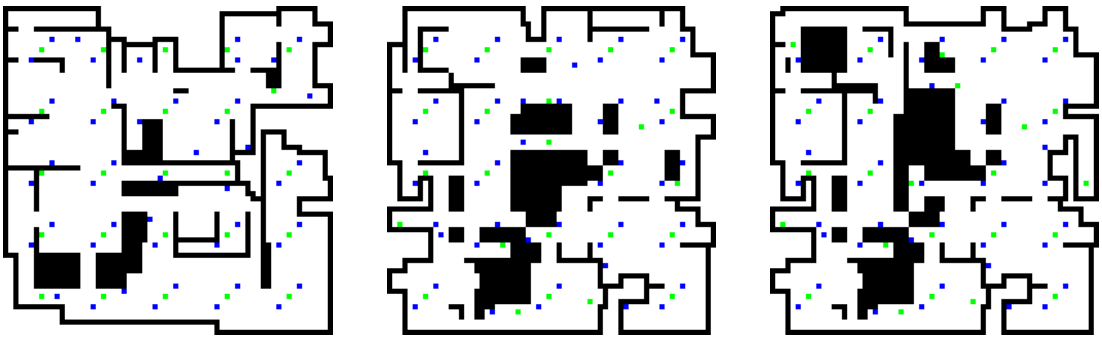


(a) Mappa con Target Loss Rate 0.002

(b) Mappa con Target Loss Rate 0.105

(c) Mappa con Target Loss Rate 0.185

Figura 5.41: Mappe più significative ottenute nel primo esempio del terzo esperimento.



(a) Mappa con Target Loss Rate 0.027

(b) Mappa con Target Loss Rate 0.080

(c) Mappa con Target Loss Rate 0.133

Figura 5.42: Mappe più significative ottenute nel secondo esempio del terzo esperimento.

Figura 5.43: Statistiche dei processi evolutivi mostrati per il terzo esperimento.

Ogni processo evolutivo ha impiegato tra le 3 e le 4 ore, e ha usato una popolazione di 75 candidati, maggiore rispetto agli esperimenti in quanto ci aspettavamo una crescita limitata e un problema più complesso, per 30 generazioni, con *crossover* matriciale, selezione dei candidati tramite *tournament selection* a due candidati, intervallo di mutazione 0.3 e probabilità di crossover 0.3.

Di seguito sono analizzati due dei processi evolutivi ottenuti e per ognuno sono mostrate, in ordine crescente del *Target Loss Rate*, tre mappe ottenute durante il processo.

In Figura 5.43 è mostrata l'evoluzione dei valori massimi e medi del *Target Loss Rate* ottenuti nelle generazioni dell'esperimento considerando la media calcolata su tutti i processi evolutivi realizzati. Dal grafico si osserva un'evoluzione costante e significativa della funzione di fitness, con una tendenza ad una potenziale ulteriore crescita.

5.3.2 Analisi delle mappe

Le mappe meno dispersive, mostrate nelle figure 5.41a e 5.42a presentano una struttura lineare; la navigazione di queste mappe costringe i giocatori ad attra-

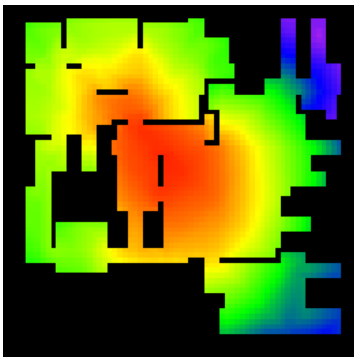
versare uno spazio centrale o a raggiungere vicoli ciechi, rendendo quasi inevitabile il ritrovamento del relativo obiettivo. La prima mappa citata, soprattutto, presenta uno spazio centrale molto ampio da cui è possibile raggiungere e osservare quasi tutte le estremità della mappa (e presenta infatti una media di combattimenti non terminati di 2 ogni 1000); la seconda, sebbene non abbia un simile spazio centrale e presenti una struttura apparentemente più complessa, ha uno "strozzamento" che taglia la mappa in due sezioni chiuse, costringendo i giocatori a passare e a incontrarsi in tale punto o a rimanere confinati in una delle due metà; evidentemente si riduce di conseguenza lo spazio di ricerca del nemico.

Le mappe più dispersive, invece, presentano una *struttura circolare*, con percorsi ciclici la cui lunghezza è direttamente proporzionale al valore di fitness ottenuto. Questi percorsi permettono fughe più lunghe, dando la possibilità ai bot di restare in costante movimento senza. Inoltre queste mappe non presentano evidenti punti centrali di collisione.

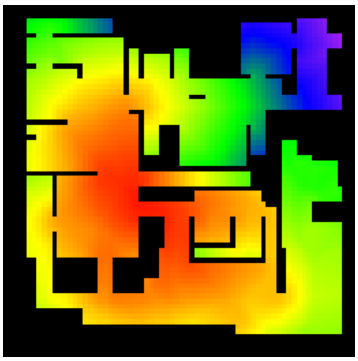
5.3.3 Analisi delle dinamiche

Nelle *heatmaps* delle uccisioni (e delle morti) delle mappe a bassa dispersività si notano poche zone ad alta concentrazione, relative alle *zone di collisione* dei giocatori: il gioco si concentra nell'intorno di questi punti essendo quelli con maggior probabilità di scontro. Le zone di collisione delle mappe in questione (quelle che nei due esperimenti hanno il *target loss rate* minore, mostrate nelle figure 5.41a e 5.42a) sono evidenti in Figura 5.44, nella quale vengono mostrate le mappe delle distanze medie di ogni punto verso tutti gli altri punti della mappa; le aree in rosso denotano le zone la cui distanza media da ogni altro punto della mappa è minore, dove è più probabile trovare punti di collisione, e confrontandole con le *heatmaps* delle morti e delle uccisioni si nota come queste siano concentrate negli stessi punti. Osservando le *heatmaps* delle uccisioni e delle morti dei livelli a dispersività maggiore si nota invece come le uccisioni (e soprattutto le morti) siano più distribuite, soprattutto lungo i percorsi circolari accennati nella sezione 5.3.2 che vengono così messi in evidenza.

Analizzando le dinamiche dei due profili si notano comportamenti simili, anche se per il profilo *Berserker* si può notare una maggiore distribuzione delle posizioni delle uccisioni e delle morti, probabilmente a causa della maggiore propensione all'avvicinamento verso l'avversario e a un maggior dinamismo rispetto al profilo



(a) Mappa delle distanze nel primo esperimento.



(b) Mappa delle distanze nel secondo esperimento.

Figura 5.44: Mappe delle distanze che evidenziano i punti di collisione nel terzo esperimento.

Artilleryman (soprattutto per il fatto che quest'ultimo si deve fermare per prendere la mira). Non sono evidenti però particolari strutture che avvantaggiano l'uno o l'altro profilo.

5.3.4 Relazioni con altre caratteristiche

Dai grafici in Figura 5.45 si nota come all'aumentare del tasso di perdita dell'obiettivo diminuiscono, prevedibilmente, tutte le metriche correlate al ritmo di gioco: il *pace* e il tempo di combattimento decrescono in modo costante, e di conseguenza si riduce anche il numero di uccisioni per entrambi i giocatori. La loro differenza, così come il loro valore di entropia, rimane però essenzialmente costante, e dunque non sembra essere un fattore che favorisce uno dei due profili.

Meno prevedibile, invece, è il lieve aumento che si nota nell'accuratezza di tiro, dovuta probabilmente al restringimento degli spazi che si ha progressivamente nelle mappe. Analogamente a quanto osservato per il secondo esperimento nella sezione 5.2.4 l'accuratezza ha una tendenza inversa rispetto al numero di uccisioni, sebbene le cause siano da attribuire a fattori separati, e probabilmente non c'è un collegamento diretto tra le due statistiche: la diminuzione del numero delle uccisioni si può infatti attribuire semplicemente al minor ritmo di gioco, e l'accuratezza alla struttura della mappa; è interessante comunque come quasi in tutti gli esperimenti l'accuratezza, che influenza direttamente l'abilità di uccidere dei bot, mostri un andamento opposto al numero delle uccisioni.

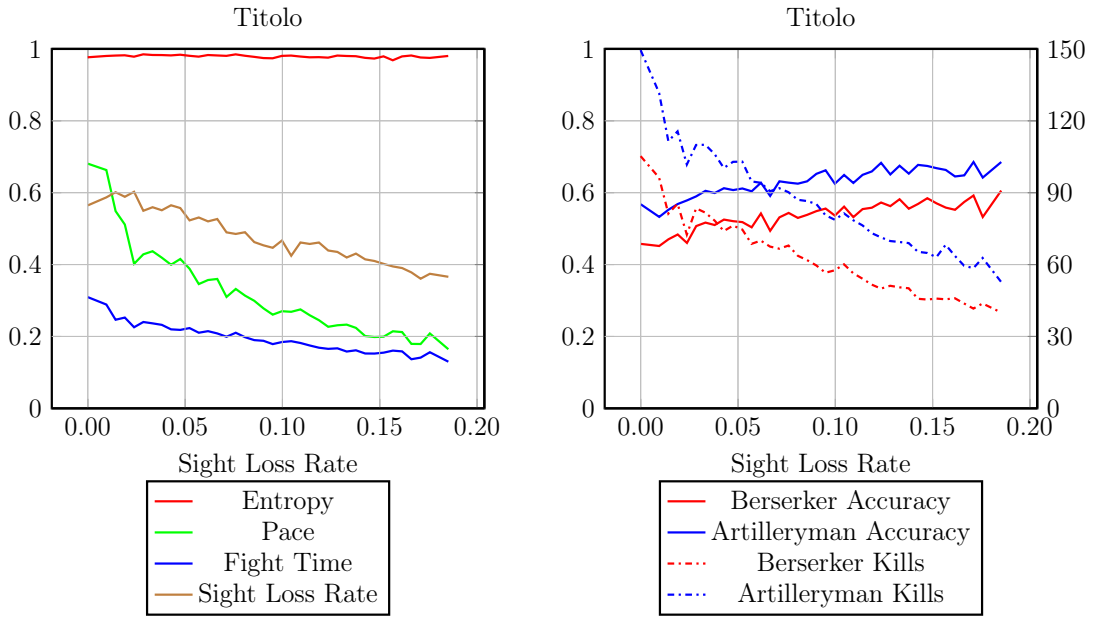


Figura 5.45: Relazioni tra le metriche misurate nel terzo esperimento. I grafici delle uccisioni si basano sui valori di scala dell'asse destro.

Inoltre inaspettatamente il *Sight Loss Rate*, che rappresenta la percentuale di tempo che nei combattimenti viene impiegata al ritrovamento dell'obiettivo quando non più visibile, ha una tendenza inversa al *Target Loss Rate* analizzato; era prevedibile pensare che una mappa con alta probabilità di perdere definitivamente l'obiettivo avesse un effetto contrario, ovvero un aumento del tempo in cui questo viene perso anche solo temporaneamente. Il comportamento osservato, invece, significa che le mappe ottenute causano un distaccamento immediato dei due giocatori e improbabilità di ritrovarsi, con un decremento generale della durata degli inseguimenti.

5.3.5 Considerazioni finali

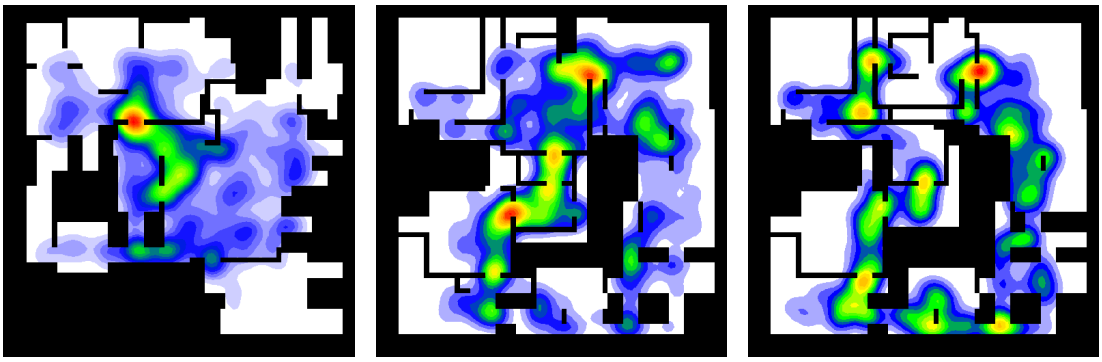
In questo esperimento abbiamo notato risultati significativi in quanto si sono ottenuti processi con un'evoluzione continua, costante e significativa. Inoltre si è evidenziata un'interessante struttura tipo, ricorrente nella maggior parte degli esperimenti, che produce come effetto una frequente perdita dell'obiettivo. Dal grafico in Figura 5.43 si può osservare come in media l'evoluzione non si fosse

ancora stabilizzata, ed è possibile che con un numero di generazioni più alto avremmo ottenuto un'evoluzione ancor più ampia.

Riguardo alle dinamiche di gioco dei due profili, più simili tra loro rispetto agli esperimenti precedenti, non si sono rilevate differenze significative; in questo caso una maggior distinzione dei loro attributi avrebbe fornito informazioni più interessanti sull'effetto delle mappe ottenute sulle loro prestazioni.

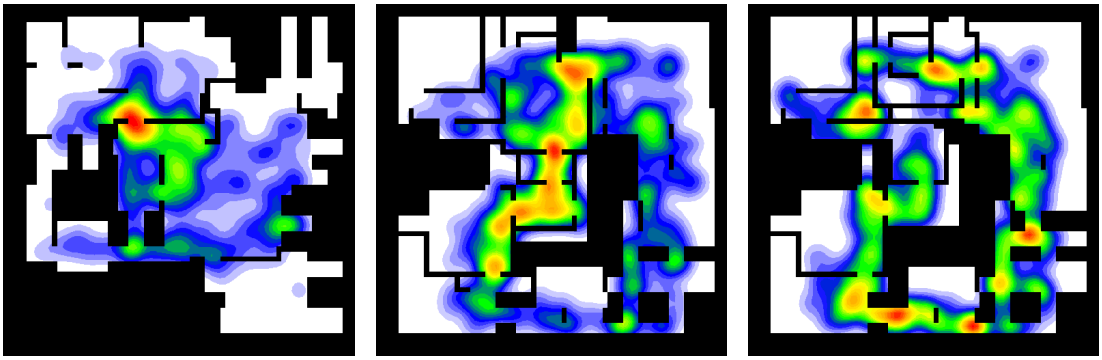
5.4 Sommario

In questo capitolo abbiamo descritto tre esperimenti che abbiamo effettuato per verificare l'efficacia del *framework*. Per ognuno degli esperimenti abbiamo descritto i parametri di setup dell'intelligenza artificiale e dell'algoritmo evolutivo; abbiamo poi analizzato le statistiche dei processi evolutivi, la struttura delle mappe ottenute, le meccaniche di gioco osservate tramite alcuni ausili visivi e le relazioni tra le caratteristiche misurate con le metriche definite nel capitolo 4, fornendo alcune considerazioni sui risultati ottenuti.



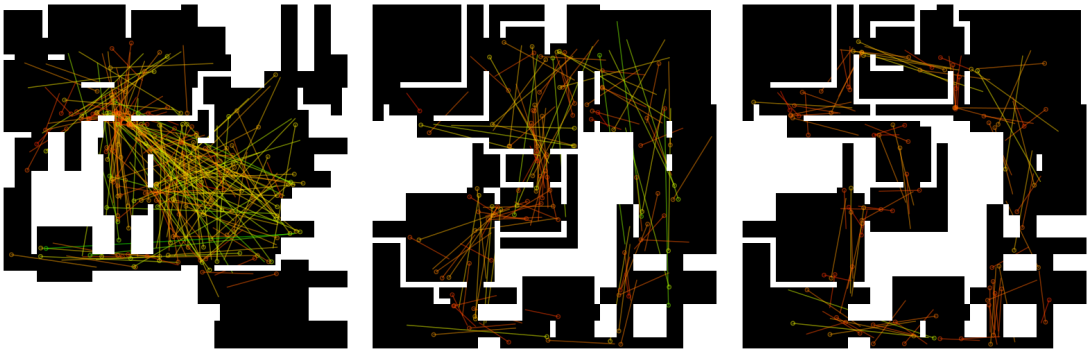
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.46: Heat Maps delle uccisioni del bot Berserker nel primo esempio del terzo esperimento.



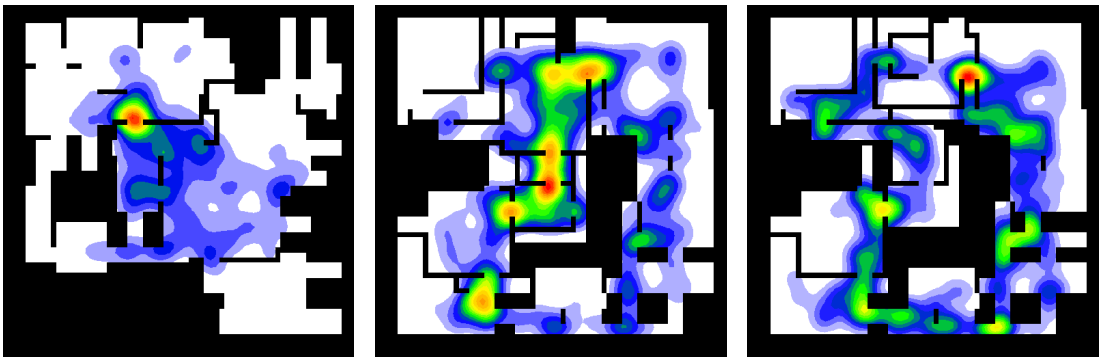
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.47: Heat Maps delle morti del bot Berserker nel primo esempio del terzo esperimento.



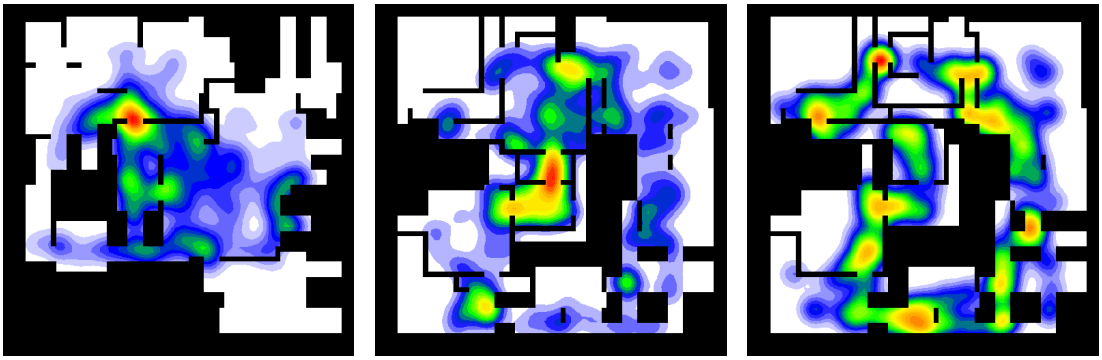
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.48: Mappe delle Kill Traces del bot Berserker nel primo esempio del terzo esperimento.



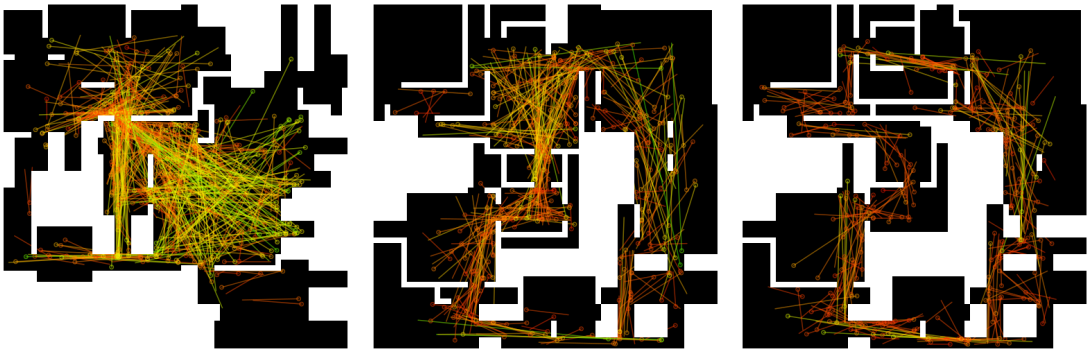
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.49: Heat Maps delle uccisioni del bot Artilleryman nel primo esempio del terzo esperimento.



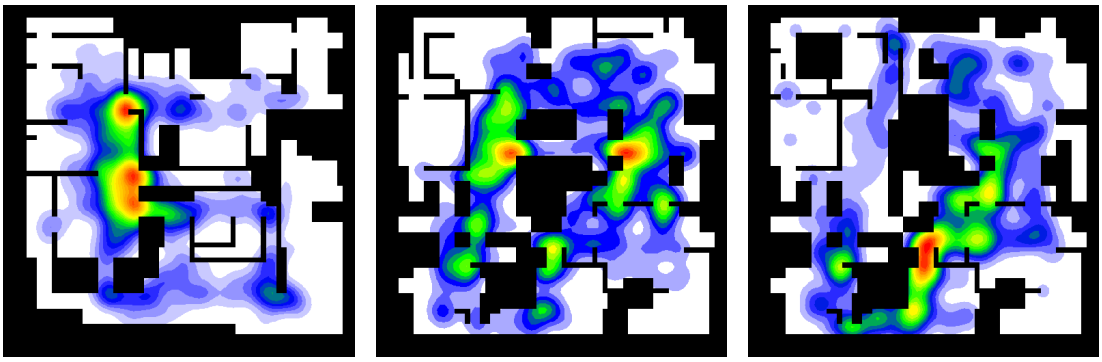
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.50: Heat Maps delle morti del bot Artilleryman nel primo esempio del terzo esperimento.



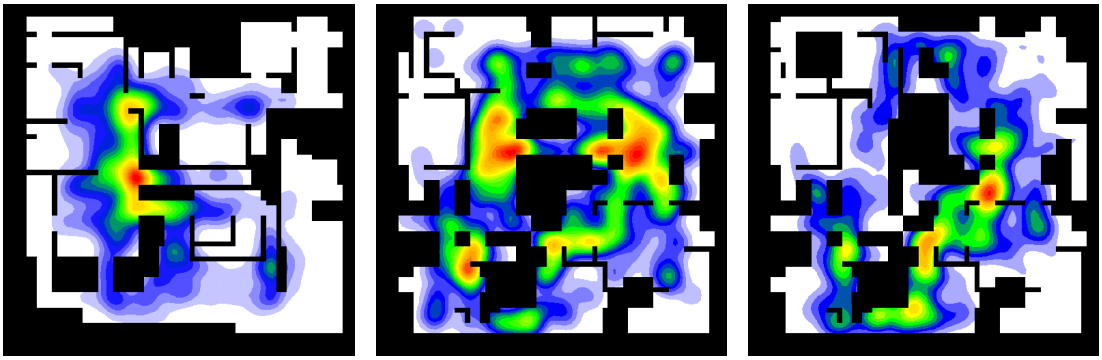
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.51: Mappe delle Kill Traces del bot Artilleryman nel primo esempio del terzo esperimento.



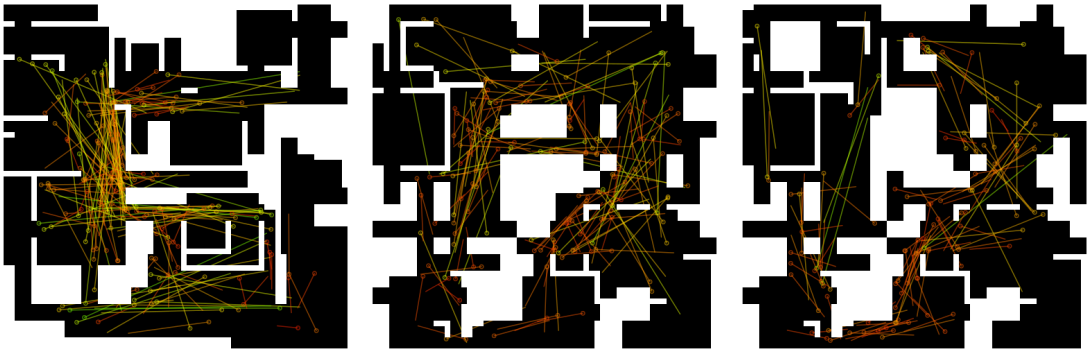
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.52: Heat Maps delle uccisioni del bot Berserker nel secondo esempio del terzo esperimento.



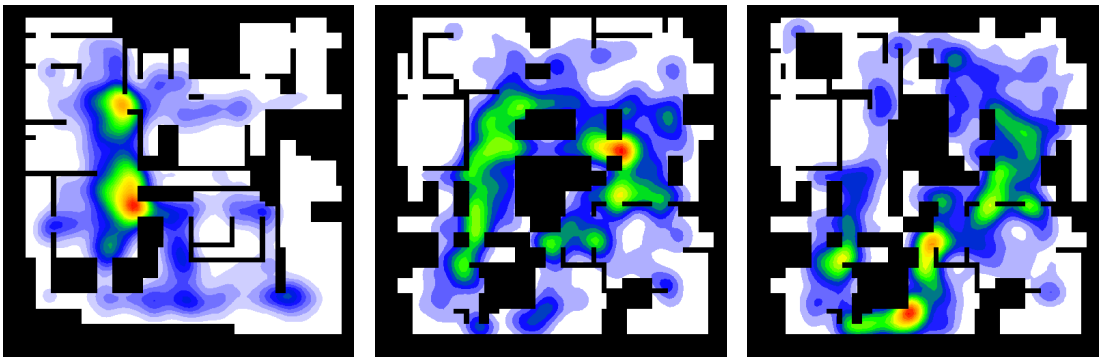
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.53: Heat Maps delle morti del bot Berserker nel secondo esempio del terzo esperimento.



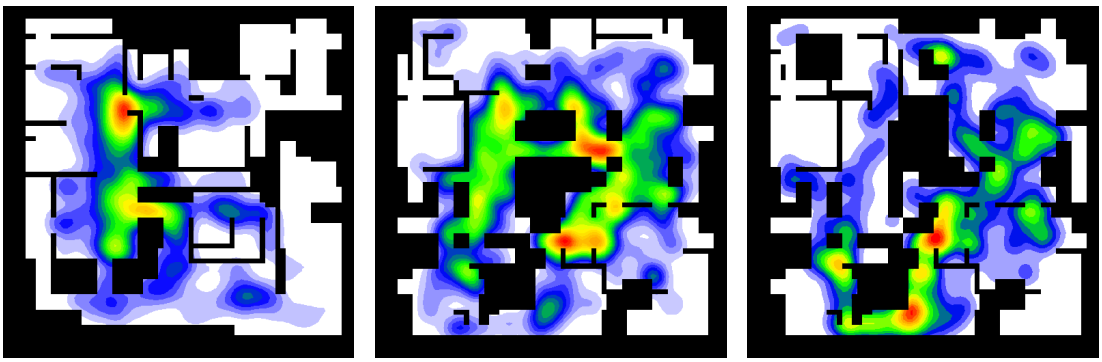
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.54: Mappe delle Kill Traces del bot Berserker nel secondo esempio del terzo esperimento.



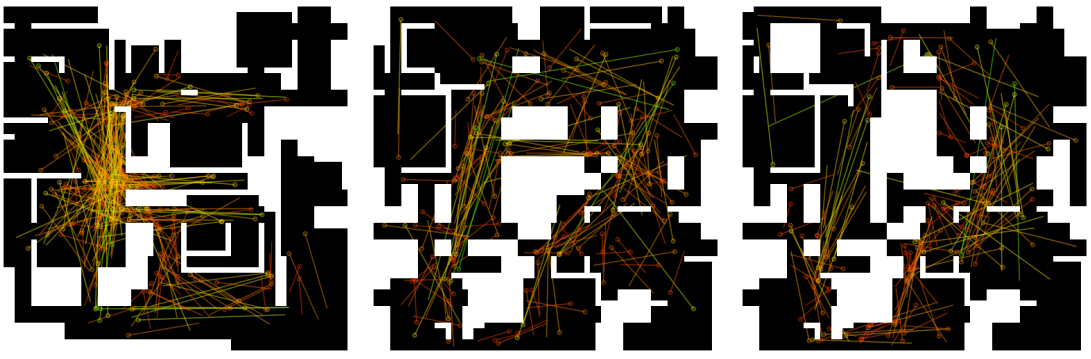
(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.55: Heat Maps delle uccisioni del bot Artilleryman nel secondo esempio del terzo esperimento.



(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.56: Heat Maps delle morti del bot Artilleryman nel secondo esempio del terzo esperimento.



(a) Mappa iniziale (b) Mappa intermedia (c) Mappa finale

Figura 5.57: Mappe delle Kill Traces del bot Artilleryman nel secondo esempio del terzo esperimento.

Capitolo 6

Conclusioni

L'obiettivo di questa tesi era quello di realizzare un *framework* che potesse assistere i *level designer* di videogiochi *first person shooter* nella realizzazione di livelli partendo da un'idea iniziale di gameplay: quello del *level design* è un campo ancora ampiamente inesplorato, come testimonia la bibliografia analizzata, sul quale però negli ultimi tempi si sta mostrando molto interesse.

Per raggiungere l'obiettivo ci siamo basati sul lavoro precedentemente svolto da Cardamone et al.[4] e Stucchi[5], utilizzando la *generazione procedurale dei contenuti con algoritmi di ricerca* per generare, attraverso *algoritmi evolutivi* basati su *simulazione*, mappe per il videogioco *Cube 2: Sauerbraten* che soddisfassero determinati requisiti. Nonostante siano state prese in considerazione diverse alternative la scelta di utilizzare *Cube 2* è stata dettata, oltre che dal vantaggio di avere alle spalle altri lavori che lo hanno utilizzato come base per ricerche analoghe, dall'accessibilità garantita dalla sua natura *open-source* e dalla presenza di un motore di intelligenza artificiale di base, necessario per un sistema evolutivo basato su *agenti autonomi*. È stato però necessario apportare alcune modifiche all'IA del gioco, poichè il comportamento dei bot creava meccaniche di gioco irrealistiche. Abbiamo quindi modificato il modello originale di IA per renderlo più adattivo allo stato della partita, tenendo particolarmente conto delle caratteristiche delle armi utilizzate e della mappa; con le modifiche apportate la struttura della mappa diventa un fattore più influente sulle prestazioni dei bot e sul risultato delle partite.

Prendendo spunto dagli aspetti che diversi designer considerano importanti nella realizzazione delle mappe per FPS, soprattutto per modalità multigiocatore,

abbiamo definito alcune statistiche e metriche da misurare e calcolare durante le partite, da usare come *funzioni di fitness* dell'algoritmo evolutivo al fine di ottenere le mappe desiderate.

Abbiamo quindi realizzato alcuni esperimenti di evoluzione di mappe basandoci sulla rappresentazione *All-Black* definita da Cardamone et al.[4], cercando di massimizzare alcune delle metriche definite, tra le quali l'equilibrio, il ritmo di gioco, la media delle serie di uccisioni e il tasso di perdita dell'obiettivo. Abbiamo esaminato due esperimenti di evoluzione *multi-obiettivo* e un esperimento a obiettivo singolo, impostando le caratteristiche degli agenti artificiali in modo da rappresentare diversi profili di giocatore e analizzando le mappe ottenute, le statistiche delle partite e alcune dinamiche di gioco tramite analisi visuale. L'uso di algoritmi evolutivi multi-obiettivo è una novità applicata in questo contesto.

Tutti i processi evolutivi hanno mostrato un'evoluzione costante relativamente agli obiettivi impostati. Nell'esperimento a obiettivo singolo, in cui abbiamo cercato mappe che massimizzassero la perdita dell'obiettivo (cioè la perdita di linea visiva con un nemico per un tempo prolungato), si sono ottenuti risultati interessanti, con un netto incremento del relativo tasso e lo sviluppo di mappe che presentavano una struttura ricorrente a forma di anello.

Anche negli esperimenti a obiettivi multipli, nei quali abbiamo ricercato la massimizzazione contemporanea del bilanciamento e del ritmo in uno, e del bilanciamento e delle serie di uccisioni medie nell'altro, le funzioni di fitness, considerate singolarmente, hanno mostrato progressi costanti anche se alle volte più limitati rispetto all'esperimento a singolo obiettivo. Questo però è dovuto anche intrinsicamente alla definizione del problema stesso: le due funzioni obiettivo, in entrambi i casi, erano in diretta contrapposizione, e l'aumento di una corrispondeva tipicamente a una decrescita dell'altra. Considerando però che l'obiettivo era la massimizzazione contemporanea di entrambe, l'uso di algoritmi evolutivi multiobiettivo ha dato buoni risultati, generando in entrambi gli esperimenti, con un'evoluzione costante, mappe che rappresentano un buon compromesso tra le due metriche.

Dagli esperimenti abbiamo notato alcune strutture responsabili di determinate meccaniche di gioco, come quella poc'anzi citata relativamente all'esperimento di massimizzazione del tasso di perdita dell'obiettivo, o favorevoli a particolari profili e armi, come i corridoi per i bot dotati di *rifle*, le strettoie e gli spazi contorti per

quelli dotati di *shotgun*, i muri per quelli dotati di lanciarazzi. Gli esperimenti hanno portato ad alcuni risultati prevedibili; nondimeno ciò dimostra che processi evolutivi basati su simulazioni possono portare a risultati significativi.

Abbiamo inoltre rilevato alcune relazioni interessanti tra le metriche calcolate dai dati raccolti durante le simulazioni: le prestazioni dei bot più abili sono favorite da mappe più frenetiche, mentre mappe più lente creano situazioni di maggior equilibrio. Anche le prestazioni delle armi sono influenzate dal ritmo di gioco, con armi a corto raggio che ottengono migliori risultati a ritmi elevati. Abbiamo osservato alcuni risultati inattesi nelle analisi delle metriche, con alcune di queste in apparenza ridondanti o strettamente legate che hanno invece mostato evoluzioni inaspettate: in corrispondenza di miglioramenti di prestazioni o incrementi del numero di uccisioni si è quasi sempre verificata una diminuzione della precisione di tiro dei bot in questione, e viceversa; era prevedibile invece uno sviluppo opposto a quello ottenuto per un simile fattore, che influisce in modo così diretto sulle prestazioni. L'unica eccezione è rappresentata dal primo esperimento, in cui i due profili avevano caratteristiche più distanti rispetto a quelli degli altri esperimenti; da questi risultati si può capire comunque come nei livelli siano diversi i fattori influenti sulle partite. Un altro esempio in questo senso, osservato nel terzo esperimento, è dato dalla proporzionalità *inversa*, al contrario di quanto ci saremmo aspettati, tra il *Sight Loss Rate* e il *Target Loss Rate*, entrambi aspetti che misurano la difficoltà di seguire l'obiettivo. Questi sono solo alcuni esempi osservati negli esperimenti mostrati, ma dimostrano la complessità delle scelte nella progettazione dei livelli.

Specialmente dopo avero notato questi particolari inaspettati, la possibilità di generare e analizzare in modo veloce mappe seguendo uno specifico obiettivo di *gameplay*, e che possono presentare diversi aspetti altrimenti difficilmente analizzabili, rendono il nostro lavoro un importante contributo al problema del *level design*, sia nell'esplorazione di nuovi *pattern* e delle loro caratteristiche che concretamente come strumento di aiuto alla progettazione dei livelli.

6.1 Problemi e possibili critiche

Nonostante i miglioramenti apportati, che risolvono alcuni dei problemi notati da Stucchi[5] nel comportamento dei bot, vi sono alcuni aspetti dell'intelligenza

artificiale che non sono stati ottimizzati, soprattutto a livello di navigazione della mappa. Ciò è apparso particolarmente evidente nella realizzazione del secondo esperimento: probabilmente si sarebbero ottenuti risultati più interessanti se i bot avessero avuto un concetto di "memoria" spaziale, grazie alla quale poter sfruttare al meglio le parti del livello a loro più vantaggiose rendendo potenzialemente maggiore il loro contributo nelle serie di uccisioni. Migliorare questo aspetto aumenterebbe in generale la qualità delle simulazioni.

Una delle difficoltà nell'usare un sistema evolutivo basato su *agenti autonomi* è la definizione delle metriche, che devono modellare efficacemente qualche aspetto del gioco da misurare. Le definizioni delle metriche da noi usate sono semplici, e misurano il più direttamente possibile, spesso come percentuali di tempo, gli aspetti che valutano, garantendo comunque la possibilità di svolgere a posteriori analisi approfondite sul significato dei risultati ottenuti. In questo modo siamo sicuri che le statistiche calcolate rappresentano effettivamente ciò per cui sono state definite.

Inoltre potrebbero esserci critiche sull'aspetto e la struttura delle mappe ottenute con la rappresentazione da noi usata, e sull'inadeguatezza per alcune tipologie di FPS come osservato da Ølsted[20]. Il nostro scopo però non è stato quello di realizzare un metodo per evolvere mappe immediatamente giocabili che si adattano online alle esigenze dei giocatori, bensì sviluppare uno strumento per assistere i designer che fornisce, offline, *meta-strutture* e pattern da utilizzare come base per la realizzazione di livelli che possano offrite determinate meccaniche di gioco.

6.2 Sviluppi futuri

Un aspetto di primaria importanza negli sviluppi futuri è l'implementazione di un sistema di navigazione migliore degli agenti artificiali del gioco, in modo da ovviare ad alcuni problemi, come quello analizzato nella sezione 6.1, rendendo così il *framework* più completo e in grado di rilevare ulteriori *pattern* di gameplay interessanti; se si trovasse un altro gioco che mettesse a disposizione un'IA più sofisticata e la possibilità di generarne dinamicamente le mappe un'alternativa

sarebbe di replicare ed estendere il lavoro fin qui svolto usando tale gioco come base.

Allo stesso modo un sicuro contributo verrebbe apportato dall'identificazione in generale di nuove metriche e funzioni di fitness. Bisogna tener conto anche del fatto che in questo lavoro non abbiamo sperimentato in modo esaustivo con tutte le metriche già definite, considerando anche la quantità di combinazioni di obiettivi che si possono ottimizzare usando algoritmi genetici multi-obiettivo. Un'estensione del lavoro finora svolto consisterà nell'analizzare le potenzialità della ricerca multi-obiettivo, utilizzando diverse e più numerose funzioni di fitness.

Inoltre nel nostro lavoro abbiamo utilizzato solo mappe sviluppate su un piano; uno sviluppo fondamentale, che sicuramente porterebbe a altri risultati interessanti e più complessi, consiste nell'implementare un algoritmo di evoluzione di mappe su più piani, in modo da poter anche analizzare strutture che sfruttano maggiormente le tre dimensioni.

Bibliografia

[1] Brenda Brathwaite e Ian Schreiber. *Challenges for Game Designers*. Course Technology, 2009.

[2] Kevin Oxland. *Gameplay and Design*. Pearson Addison Wesley, 2004. ISBN: 0321204670.

[3] Marc Saltzman. *Game design: secrets of the sages*. Macmillan Digital Pub., 1999. ISBN: 9781575952574.

[4] Luigi Cardamone et al. 'Evolving interesting maps for a first person shooter'. In: *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*. EvoApplications'11. Torino, Italy: Springer-Verlag, 2011, pp. 63–72. ISBN: 978-3-642-20524-8. URL: http://dl.acm.org/citation.cfm?id=2008402.2008411.

[5] Riccardo Stucchi. 'Evoluzioni di mappe bilanciate per FPS con algoritmi evolutivi'. Tesi di laurea mag. Politecnico di Milano, 2013.

[6] Thomas Hoeg. 'The Invisible Hand: Using Level Design Elements to Manipulate Player Choice'. Tesi di laurea mag. Southern Methodist University, dic. 2008.

[7] Rudolf Kremers. *Level Design: Concept, Theory, and Practice*. A K Peters/CRC Press, 2009. ISBN: 9781568813387.

[8] Christopher W. Totten. *An architectural approach to Level Design*. CRC Press, 2014. ISBN: 1466585412.

[9] Cliff Bleszinski. 'The Art and Science of Level Design'. In: *Session 4404 at GDC 2000* (mag. 2000).

[10] Matthew Gallant. *Guiding The Player's Eye*. http://gangles.ca/2009/05/26/guiding-the-eye/. Mag. 2009.

[11] Brandi Alotto. 'How Level Designers Affect Player Pathing Decisions: Player Manipulation through Level Design'. Tesi di laurea mag. Southern Methodist University, apr. 2007.

[12] John D. Brady. 'The Constraints and Affordances of Building a Single Player Level Based on Historic Reference'. Tesi di laurea mag. Southern Methodist University, dic. 2012.

[13] Joy Brownmiller. 'In-Game Lighting and its Effect on Player Behavior and Decision-Making'. Tesi di laurea mag. Southern Methodist University, dic. 2012.

[14] Christian Güttler e Troels Degn Johansson. 'Spatial Principles of Level-design in Multi-player First-person Shooters'. In: *Proceedings of the 2Nd Workshop on Network and System Support for Games*. NetGames '03. Redwood City, California: ACM, 2003, pp. 158–170. ISBN: 1-58113-734-6. DOI: 10.1145/963900.963915. URL: http://doi.acm.org/10.1145/963900.963915.

[15] Simon Larsen. *Level Design Patterns*. http://simonlundlarsen.com/wp-content/uploads/2015/06/Level-design-patterns.pdf. Apr. 2006.

[16] Kenneth Hullett e Jim Whitehead. 'Design Patterns in FPS Levels'. In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. FDG '10. Monterey, California: ACM, 2010, pp. 78–85. ISBN: 978-1-60558-937-4. DOI: 10.1145/1822348.1822359. URL: http://doi.acm.org/10.1145/1822348.1822359.

[17] Kenneth M. Hullett. 'The Science of Level Design: Design Patterns and Analysis of Player Behavior in First-Person Shooter Levels'. Tesi di dott. University of California, Santa Cruz, set. 2012.

[18] Julian Togelius et al. 'Procedural Content Generation: Goals, Challenges and Actionable Steps'. In: *Artificial and Computational Intelligence in Games*. A cura di Simon M. Lucas et al. Vol. 6. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 61–75. ISBN: 978-3-939897-62-0. DOI: http://dx.doi.org/10.4230/DFU.Vol6.12191.61. URL: http://drops.dagstuhl.de/opus/volltexte/2013/4336.

[19] Julian Togelius et al. 'Search-based Procedural Content Generation'. In: *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*. EvoApplicatons'10. Istanbul, Turkey: Springer-Verlag, 2010, pp. 141–150. ISBN: 3-642-12238-8, 978-3-642-12238-5. DOI: 10.1007/978-3-642-12239-2_15. URL: http://dx.doi.org/10.1007/978-3-642-12239-2_15.

[20] Peter Ølsted e Benjamin Ma. 'Interactive Evolution Levels for a Competitive Multiplayer FPS. Making players do the level designer's job'. Tesi di laurea mag. IT University of Copenhagen, dic. 2014.

[21] Bhojan Anand e Hong Wei Wong. 'ARENA - Dynamic Run-Time Map Generation for Multiplayer Shooters'. In: *Entertainment Computing - ICEC 2014 - 13th International Conference, ICEC 2014, Sydney, Australia, October 1-3, 2014. Proceedings*. 2014, pp. 149–158. DOI: 10.1007/978-3-662-45212-7_19. URL: http://dx.doi.org/10.1007/978-3-662-45212-7_19.

[22] Hamish, Todd. *Level design: Doom's "horseshoe"*. http://www.gamasutra.com/blogs/HamishTodd/20150217/236516/Level_design_Dooms_quothorseshoequot.php. Feb. 2015.

[23] Sam Shahrani. *Educational Feature: A History and Analysis of Level Design in 3D Computer Games - Pt. 1*. http://www.gamasutra.com/view/feature/2674/educational_feature_a_history_and_.php. Apr. 2006.

[24] Young, Shamus. *The Big Cost of Small Places*. http://www.escapistmagazine.com/articles/view/video-games/columns/experienced-points/9331-The-Big-Cost-of-Small-Places. Gen. 2012.

[25] Sam Shahrani. *Educational Feature: A History and Analysis of Level Design in 3D Computer Games - Pt. 2*. http://www.gamasutra.com/view/feature/131091/educational_feature_a_history_and_.php. Apr. 2006.

[26] Dennis Scimeca. *How To Build The Best Multiplayer FPS Maps: Part One*. http://www.g4tv.com/thefeed/blog/post/719216/how-to-build-the-best-multiplayer-fps-maps-part-one/. Dic. 2011.

[27] Dennis Scimeca. *How To Build The Best Multiplayer FPS Maps: Part Two*. `http://www.g4tv.com/thefeed/blog/post/719253/how-to-build-the-best-multiplayer-fps-maps-part-two/`. Dic. 2011.