ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria con sede a Cesena
Corso di Laurea Triennale in Ingegneria Informatica

VIDEOGIOCHI E INTELLIGENZA ARTIFICIALE: GENERAZIONE AUTOMATICA DI CONTENUTO PERSONALIZZATO

Elaborato in

FONDAMENTI DI INFORMATICA L-B

Relatore:
Prof. ANDREA ROLI

Presentata da:
MIRKO MENNINO

Anno Accademico 2011/2012

Sessione III

Alla mia famiglia che ha sempre creduto in me

Indice

Introduzione

Un settore molto sviluppato nell'industria del divertimento riguarda i giochi, soprattutto quelli in forma digitale, utilizzati a scopo ricreativo ma anche a scopo educativo o riabilitativo. L'industria dei videogames è notevolmente cresciuta negli ultimi anni e l'attività videoludica occupa gran parte del tempo libero delle persone. Tuttavia esistono diversi elementi, tra i quali la standardizzazione delle regole e dei meccanismi dei titoli presenti sul mercato, l'alto costo di produzione e la crisi economica che tutt'oggi molte aziende devono affrontare, che hanno portato l'industria videoludica a non produrre più elementi innovativi se non nel reparto grafico. Pertanto c'è un maggiore interesse da parte di questo settore nella ricerca in ambito di Intelligenza Artificiale sui diversi meccanismi di produzione dei videogames.

Il processo di sviluppo di un gioco in generale può essere infatti rivoluzionato con l'introduzione di sistemi automatici che sostituiscono il metodo manuale per la creazione di qualsiasi contenuto che forma il gioco stesso. L'applicazione di algoritmi, generalmente *search-based* o comunque sia rientranti nella categoria della computazione evolutiva, permette di creare tramite la così definita generazione procedurale del contenuto, o PCG (*Procedural Content Generation)*, qualsiasi elemento di un gioco.

Inoltre è possibile affiancare a tale sistema un meccanismo di personalizzazione che porta alla creazione automatizzata di elementi di gioco che considerano anche aspetti cognitivi e/o emotivi del giocatore. L'unione della metodologia PCG e l'aspetto di personalizzazione forma una nuova metodologia, definita come generazione procedurale del contenuto guidata dall'esperienza (intesa come insieme di aspetti cognitivi ed emotivi del giocatore), chiamata anche EDPCG (*Experience-Driven Procedural Content Generation).*

Attraverso quest'ultima ci si pone l'obiettivo di una creazione automatizzata di giochi più immersivi, che personalizzano e modificano il proprio contenuto a seconda delle caratteristiche di ogni singolo giocatore, con costi meno onerosi rispetto all'attuale sviluppo manuale.

Oltre a questo aspetto di personalizzazione che sicuramente non è comune ai titoli in commercio al giorno d'oggi, il principio di casualità

presente negli algoritmi di computazione evolutiva potrebbe eventualmente anche portare alla creazione di nuove regole e/o meccanismi che aiuterebbero il mercato ad uscire dalla standardizzazione.

Il capitolo 1 tratta della generazione procedurale del contenuto, definendo un quadro generale introduttivo di questa metodologia e classificandola in base a diversi elementi. Inoltre approfondisce l'approccio PCG con algoritmi *search-based*, chiamato SBPCG.

Il capitolo 2 tratta degli algoritmi rientranti nell'area di computazione evolutiva che possono essere utilizzati per la SBPCG, focalizzandosi sulla programmazione genetica e la grammatica evolutiva.

Il capitolo 3 approfondisce il sistema EDPCG, visto come un'estensione del sistema precedente (SBPCG), nel quale viene incluso l'aspetto di personalizzazione. Si definiscono i diversi approcci possibili per la creazione del modello che rappresenti l'esperienza del giocatore sotto forma di insieme di aspetti cognitivi ed emotivi e come tale modello si inserisca all'interno del sistema precedente.

Il capitolo 4 espone l'attuale stato dell'arte dell'area di ricerca, fornendo una descrizione generale degli esperimenti fatti e delle applicazioni finora ritrovate nei titoli commerciali presenti sul mercato.

Capitolo 1

PCG: Procedural Content Generation

La generazione procedurale dei contenuti, o PCG (*Procedural Content Generation)*, è una metodologia per la generazione automatica del contenuto di un'entità, tipicamente un gioco (in qualunque forma si presenti, da quella digitale ai giochi da tavolo [1]) attraverso l'utilizzo di algoritmi o processi casuali che possono produrre, grazie alla loro natura aleatoria, una gamma imprevedibile di possibili contenuti relativi all'entità presa in considerazione. La chiave di volta di questa metodologia è il concetto di casualità: attraverso l'utilizzo di pochi parametri, l'applicazione del metodo PCG dovrebbe garantire una creazione di un numero elevato di possibili contenuti di un gioco, tutti differenti tra loro [2].

Nonostante la quasi totalità di riferimenti disponibili alla PCG riguardano il suo utilizzo in ambito ludico, nulla vieta di poter utilizzare questa metodologia anche in altri settori, come per esempio i servizi web, i siti internet o altri tipi di entità (come la musica [3]).

1.1 Il significato di *"Content"*

E' importante definire anche il significato che si attribuisce alla parola *Content* perché in base alla sua definizione possiamo distinguere se un contenuto rientra nel dominio della PCG oppure in altri domini, come per esempio quello relativo alla PG *(Procedural Generation).* I risultati prodotti dall'applicazione di algoritmi di PCG fanno parte di tutti gli elementi che in qualche modo influenzano il *gameplay* in modo significativo. Non sono inclusi invece in questo insieme, nonostante influenzino in maniera elevata il *gameplay*, tutti gli aspetti del gioco che riguardano il comportamento dei personaggi non giocanti, o NPC *(Non Player Character)* e il motore grafico del gioco. Questo perché si vuole escludere dalla PCG un'area di applicazione di algoritmi di ricerca e ottimizzazione che ha già un'ottima documentazione (a differenza della PCG) e viene definita come NPC AI. Esempi di contenuti prodotti dalla PCG sono: terreni, mappe, livelli, storie, dialoghi, missioni, personaggi, regole, dinamiche, armi [4].

1.2 PG: Procedural Generation

Non essendoci ancora nessun libro di testo sull'argomento e poiché l'interesse sulla PCG sta crescendo solo ultimamente, non si ha una distinzione chiara tra quest'ultima e la PG, e spesso i due termini vengono assimilati in uno unico, poiché possono utilizzare gli stessi algoritmi anche se per ottenere risultati diversi. La sostanziale differenza si basa proprio sui risultati generati: la PCG genera del contenuto che è formato da diversi componenti, mentre ogni singolo componente è generato (o meglio può essere generato) dalla PG.

In ambito ludico si differenziano perché la PG crea contenuti che non influiscono il *gameplay*. Per esempio, la *Procedural Textures* e la *Procedural Animation*, che riguardano la creazione automatica di texture e di animazioni rientrano in questa categoria [5]. Una texture creata tramite le tecniche di generazione procedurale rientra nell'area di ricerca della PG, l'insieme delle texture che formano l'ambientazione di un livello creato tramite tecniche di generazione procedurale rientra nell'area di ricerca della PCG.

1.3 Motivazioni per l'utilizzo di PG/PCG nei videogiochi

Yannakakis, Togelius & C. [4] [6], insieme al lavoro più recente di Iosup & C. [7], hanno individuato i principali motivi per i quali si dovrebbero utilizzare le tecniche di generazione procedurale del contenuto in ambiente videoludico:

I. Ultimamente la produzione di un gioco con contenuti di alto livello implica uno sforzo enorme in termini di costo e di tempo richiesto per il suo sviluppo. Normalmente ci si aspetterebbe che, con il progresso tecnologico sempre in costante aumento e con l'introduzione di dispositivi sempre più veloci e funzionali, questo processo di creazione si fosse quantomeno velocizzato oppure anche automatizzato, almeno parzialmente. Ma mentre la tecnologia dei videogames è avanzata dai semplici meccanismi di *Breakout* e *Pac-man* all'elaborazione realistica di ambienti in 3D, ricche dinamiche e dettagli grafici di alto livello come in *Halo* e in *Call of Duty*, la creazione del contenuto è ancora ampiamente manuale. Solitamente, un team di persone di differenti reparti della produzione del gioco (programmatori, ingegneri del suono, disegnatori, progettisti) è responsabile della creazione a mano di tutto il contenuto del gioco.

Questo incide notevolmente sul budget disponibile per lo sviluppo, soprattutto quando si vuole creare un titolo di ottima qualità (per esempio, il gioco *GTA 4* è il lavoro di 1000 persone su un periodo di 3 anni, ed è uno dei giochi più costosi di sempre in termini di budget, stimabile in 100 milioni di dollari [8]) e il trend misura che tale budget è costantemente in salita [9]. Di conseguenza, la creazione del contenuto viene vista come un "collo di bottiglia" per il budget totale di un gioco e per il time-to-market del prodotto. Anche se i budget di produzione rimangono sconosciuti per la maggior parte di giochi famosi, è possibile attestare che circa il 30-40% di esso viene speso per la creazione del contenuto. E' quindi evidente che una qualsiasi tecnologia che possa alleviare l'enorme onere della creazione del contenuto, come la PCG, potrebbe essere ben accolta dagli sviluppatori, dai critici e dal pubblico in generale;

II. Il collo di bottiglia creato dai costi elevati e dai lunghi tempi di produzione del gioco rappresenta una barriera per il progresso artistico e tecnologico dei giochi di qualsiasi piattaforma, poiché difficilmente uno sviluppatore rischia con idee nuove o meccanismi differenti per creare un prodotto diverso con questa altissima posta in gioco;

III. Un'altra ragione per volere utilizzare la PCG è che essa può potenzialmente creare giochi infiniti. Questo avviene quando il suo utilizzo riguarda la generazione di contenuti in tempo reale con un grado di varietà sufficientemente alto. La storia insegna che questo meccanismo può portare al successo: già nei primi anni '80 il videogames *Rogue* [10] apre la strada alla PCG attraverso la generazione automatica di *dungeon* sempre diversi da far esplorare al giocatore. L'ottica di poter creare giochi infiniti attrasse molti sviluppatori e questo metodo fu imitato numerose volte nel corso degli anni, per esempio anche dalla serie del gioco di successo *Diablo*;

IV. L'impiego della memoria di massa rappresenta un altro punto a favore dell'utilizzo della generazione procedurale del contenuto. Questo problema era rilevante maggiormente nei primi anni '80 quando le limitazioni di memoria delle piattaforme esistenti per utenti non permettevano la distribuzione di grandi quantità di contenuti pre-

progettati come i livelli di gioco. Il vantaggio di usare la PCG era che il contenuto rappresentato proceduralmente poteva essere lasciato "compresso" fino a quando non era necessario. Il gioco di avventura e commercio spaziale *Elite* è l'esempio per eccellenza. Esso riusciva a gestire centinaia di sistemi stellari in una decina di Kilobyte di memoria, rappresentando un pianeta in forma compressa solo con pochi numeri. Nella forma espansa, ogni pianeta aveva un nome, una popolazione, i prezzi delle materie prime e così via [11]. Anche oggi l'impiego di memoria di massa dovrebbe essere sempre tenuto in considerazione: nonostante il grande aumento di capacità degli hard disk moderni e il calo del prezzo di un singolo GB [12], se nessuno considerasse questo fattore ci ritroveremmo con pochi videogames da molti GB l'uno su un unico hard disk, e saremmo costretti o a disinstallare alcuni giochi o a comprare nuove memorie di massa per fare spazio ai nuovi. Un esempio recente che affronta questo problema di memoria è il gioco *.kkrieger* [12], sparatutto 3D in prima persona, simile nel genere a *Halo 3* (un gioco da più di 20 milioni di dollari di budget) che, a differenza di quest'ultimo, utilizza tecniche di generazione procedurale per creare texture, maglie, suoni che poi vengono a loro volta combinati tra loro con tecniche di generazione procedurale del contenuto per formare un'ambientazione completa. Il vantaggio è che *.kkrieger* richiede meno di 100 KB di memoria su disco rigido, circa 3-4 ordini di grandezza in meno rispetto ad un gioco similare come *Halo 3*;

Halo 3 Screenshot

.kkrieger Screenshot

V. Infine, la PCG può aumentare i limiti dell'immaginazione umana. Gli algoritmi possono creare delle nuove regole, livelli, storie, dalle quali il progettista può poi anche prendere ispirazione per formare le basi sulle quali creare nuovi giochi.

1.4 Il contenuto

Possiamo raggruppare, in base all'ultima classificazione di Iosup & C. [7], il tipo di contenuto che può essere prodotto dalle tecniche di generazione procedurale in cinque classi principali:

a) *Game Bits*
   Unità elementari del contenuto del gioco che non influenzano il *gameplay* del giocatore se considerate indipendentemente. Fanno parte di questa categoria: texture, suoni, vegetazione, strutture, comportamenti, fuoco, acqua, pietra, nuvole;
b) *Game Space*
   Ambiente nel quale si gioca, composto da diverse unità *Game Bits*. Fanno parte di questa categoria: mappe interne, mappe esterne, corpi d'acqua come mari, laghi, fiumi;
c) *Game System*
   Sistemi di gioco. Fanno parte di questa categoria: ecosistemi, reti di strade, urbanistica;
d) *Game Scenarios*
   In che modalità accadono gli eventi. Fanno parte di questa categoria: puzzle, storyboard, la storia, il concetto di livello;
e) *Game Design*
   Il design del gioco. Fanno parte di questa categoria: regole e obiettivi.

Il tipo di algoritmo che può essere scelto per la creazione di elemento di una classe dipende dal tipo di elemento stesso. Possono essere utilizzati diversi algoritmi, a partire dal più semplice generatore di numeri pseudo-casuali, alle grammatiche generative fino all'utilizzo delle più avanzate tecniche di programmazione evolutiva e reti neurali artificiali.

1.5 Tipologie di PCG

Al momento purtroppo non esistono libri di testo riguardanti la PCG, pertanto per classificare e differenziare le diverse tipologie di PCG che possono essere implementate ci si basa su articoli accademici che hanno

cercato di dare una base e una tassonomia relativamente a questo argomento [4].

A. Online – offline
   La prima distinzione riguarda se il contenuto generato avviene online (durante l'esecuzione del gioco) oppure offline (durante lo sviluppo del gioco). Un esempio di PCG online è quando il giocatore apre una porta di una struttura e il gioco istantaneamente genera l'interno di essa, cioè stanze, pareti, decorazioni che prima non c'erano. Un esempio di PCG offline è quando in fase di sviluppo del gioco un algoritmo crea il layout interno di una struttura che poi viene modificato e perfezionato dal progettista prima che il gioco sia completato. Chiaramente, la PCG online deve rispettare dei requisiti fondamentali perché crei contenuti validi: deve essere veloce e il contenuto prodotto deve essere di qualità (a seconda del contesto);

B. Contenuto necessario – opzionale
   La seconda distinzione è relativa all'importanza del contenuto, vale a dire se, rispetto al gioco stesso, esso è necessario oppure opzionale. Il contenuto necessario è richiesto dai giocatori per procedere nel gioco. Per esempio i *dungeon* che devono essere attraversati, i mostri che devono essere sconfitti, le regole cruciali del gioco e cosi via rientrano in questa categoria. Il contenuto opzionale è tutto quello che il giocatore può scegliere di non considerare, come armi o strutture che possono essere anche ignorate. Un'altra differenza tra i due contenuti è che quello necessario deve sempre essere di ottima qualità e funzionalmente corretto. Non è accettabile generare un *dungeon* impraticabile, regole non giocabili o mostri imbattibili se queste anomalie rendono impossibile il proseguire del gioco da parte del giocatore. Non è inoltre accettabile generare del contenuto che abbia una difficoltà non congrua con il resto del gioco. Dall'altra parte, per quanto riguarda il contenuto opzionale, è consentito che un algoritmo produca anche armi inutilizzabili o layout irragionevoli se il giocatore può scegliere di scartare l'arma e prenderne un'altra o uscire da una strana costruzione e andare da un'altra parte. Chiaramente l'importanza che viene data ad un contenuto dipende fondamentalmente dal design del gioco e dalla sua trama. Per esempio, lo sparatutto in prima persona *Borderlands* ha un meccanismo di

generazione casuale delle armi, molte delle quali non utili, ma analizzare queste armi è una parte del cuore del *gameplay* ed è costante nella trama del gioco. Dall'altra parte, una struttura con design semplici e di bassa qualità a livello di dettagli appare molto artificiale e può rendere non credibile un'ambientazione in un gioco che ha come obiettivo principale il realismo visuale come *Call of Duty 4: Modern Warfare*. E' interessante notare come alcuni tipi di contenuto possono essere opzionali in una categoria di giochi e necessario in altri (per esempio i *dungeon* opzionali). Quindi l'analisi di cosa si può definire come contenuto opzionale in un gioco va fatta caso per caso;

C. Gradi di controllo
   Un'altra distinzione dipende dal tipo di algoritmo di generazione che viene utilizzato e come esso può essere parametrizzato. Ad un estremo, l'algoritmo può semplicemente prendere soltanto un numero generato casualmente come input; all'altro estremo, l'algoritmo può prendere in input un vettore multidimensionale contenente parametri che specificano le proprietà del contenuto che deve essere generato. Nel caso della creazione di un *dungeon*, per esempio, l'algoritmo può essere eseguito avendo in input dei parametri riguardanti il numero di stanze, fattore di ramificazione dei corridoi, tesori e cosi via. Più gradi di controllo si hanno, più è possibile personalizzare e controllare il contenuto generato;

D. Generazione deterministica o stocastica
   Questa distinzione concerne il grado di casualità nel processo di generazione. E' possibile concepire algoritmi deterministici che generano sempre lo stesso contenuto con gli stessi parametri di ingresso, oppure contenuti sempre differenti anche con identici parametri di input come per esempio l'algoritmo di generazione del *dungeon* di *Rogue*;

E. Algoritmi costruttivi o generativi-con-test
   L'ultima distinzione riguarda l'output dell'algoritmo che viene eseguito. Gli algoritmi costruttivi generano il contenuto e terminano la loro esecuzione, producendo il risultato in uscita. Per esempio, l'algoritmo del processo Markoviano [14] è tipicamente un algoritmo di PCG costruttivo che produce contenuti stocastici. Comunque sia, è necessario che ci sia una forma di controllo in modo tale da evitare la produzione di contenuti

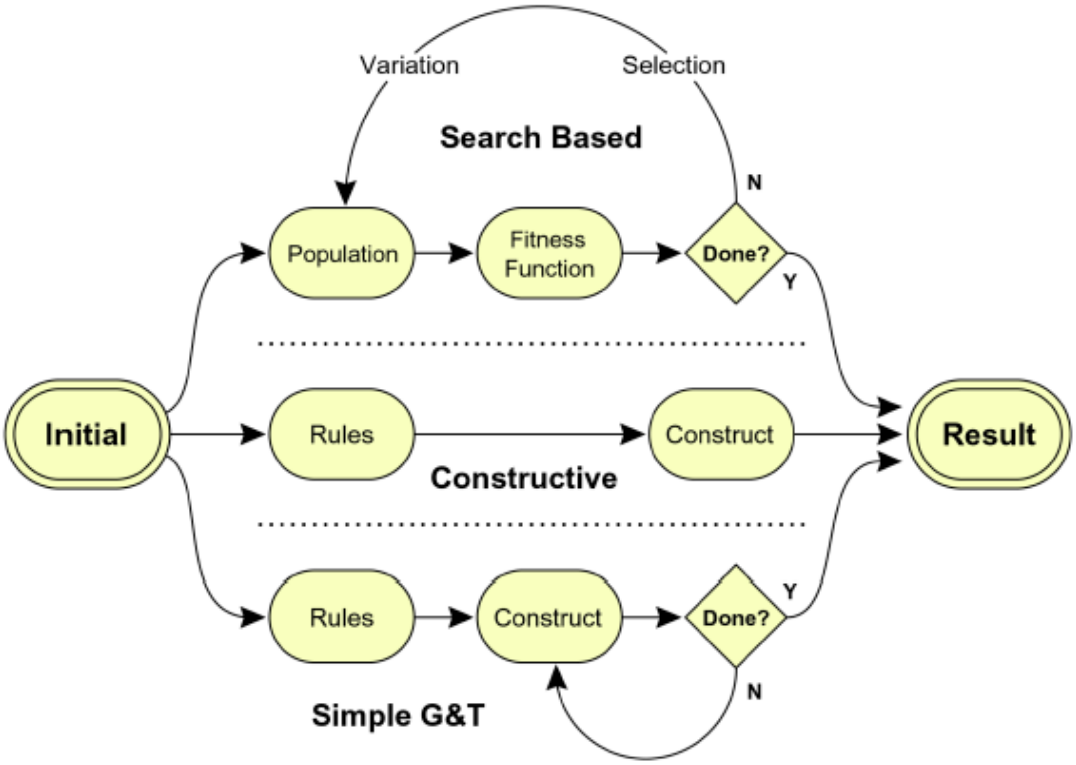
non idonei. Questo può essere fatto attraverso l'utilizzo di operazioni o sequenze di operazioni che garantiscono che non si possa produrre materiale non utilizzabile. Un esempio di questo approccio è l'uso di frattali per generare terreni. Un algoritmo generativo-con-test incorpora al suo interno un meccanismo di generazione e uno di test. Ciò vuol dire che durante l'esecuzione dell'algoritmo, ogni istanza di contenuto viene testato secondo alcuni criteri (che dipendono sempre dal design del gioco). Se il candidato non supera il test, esso viene totalmente o parzialmente scartato e rigenerato, e il processo continua fino a che il contenuto non ha una qualità sufficiente.

1.6 SBPCG: Search-Based Procedural Content Generation

E' possibile definire un particolare caso di PCG che utilizza in generale, per la generazione di contenuti, algoritmi evolutivi o comunque sia di ricerca e ottimizzazione. Viene chiamata SBPCG (Search-Based Procedural Content Generation) [4] e rappresenta un caso speciale dell'approccio PCG con algoritmi generativi-con-test, caratterizzato da queste proprietà:

- La funzione di test non accetta o rifiuta semplicemente il contenuto candidato sotto fase di esaminazione, ma lo valuta assegnandogli un numero o un vettore di numeri reali. Generalmente questa funzione di test può essere chiamata in diversi modi come per esempio funzione di idoneità (*fitness*), funzione di valutazione o di utilità;
- La generazione dei nuovi contenuti dipende dai valori di *fitness* assegnati ai contenuti precedenti che sono stati esaminati. L'obiettivo è quello di produrre nuovi contenuti con un valore di *fitness* sempre più alto.

Nel definire questo caso speciale di PCG si è utilizzato il termine *"search-based"* perché in questo modo esplicitamente si permette l'utilizzo di qualsiasi forma di algoritmo euristico e di ricerca/ottimizzazione. In realtà, molti esperimenti e esempi di applicazione di questa metodologia prevedono l'utilizzo di algoritmi evolutivi e la computazione evolutiva è stata da sempre il metodo più scelto per questa area di ricerca.



Tre approcci alla PCG. Bisogna evidenziare però il fatto che non tutti gli algoritmi di ricerca/ottimizzazione per la PCG hanno un meccanismo di creazione di una popolazione per il contenuto che deve essere prodotto, ma molti dei più usati (come per esempio gli algoritmi di computazione evolutiva) dispongono di questa funzionalità

A. Rappresentazione del contenuto e dominio di ricerca

Una questione centrale nell'ambito dei problemi di ottimizzazione e dei meta-euristici riguarda come rappresentare il contenuto che può evolvere. In altre parole, si tratta di definire come i genotipi (vale a dire la struttura dati che viene gestita dall'algoritmo evolutivo) vengono mappati ai fenotipi (la struttura dati o il processo che viene valutato dalla funzione di test). E' una terminologia presa dalla computazione evolutiva, ma delle considerazioni similari possono essere trovate in altre forme di ottimizzazione. Per esempio, nella creazione di un livello di un gioco, il genotipo potrebbe essere l'insieme di istruzioni per creare il livello, mentre il fenotipo è il livello. Possiamo sempre parlare di una distinzione tra genotipo e fenotipo quando utilizziamo algoritmi stocastici, anche in semplici casi come la ricerca delle soluzioni di un'equazione: in questo caso i valori delle variabili sono il genotipo, il risultato sostituendo questi

valori alle variabili e calcolando l'equazione è il *mapping* genotipo-fenotipo.

Un'altra importante distinzione sulla rappresentazione è tra la codifica diretta o indiretta. Nella prima il *mapping* genotipo-fenotipo implica una calcolo computazionale più semplice, per esempio quando la dimensione del genotipo è linearmente proporzionale alla dimensione del fenotipo e ogni parte del genoma mappa una specifica parte del fenoma. Nel secondo caso il *mapping* richiede una calcolo computazionale più complesso.

Un particolare caso oggetto di molti studi riguarda la rappresentazione dei candidati con vettori di numeri reali perché hanno delle caratteristiche più funzionali. Infatti essi possono essere facilmente analizzati e molti algoritmi standard si applicano più facilmente a questa rappresentazione in comparazione con altre rappresentazioni meno usuali. Chiaramente il vettore deve rispettare il requisito di avere una giusta dimensione. I vettori di dimensione troppo piccola sono incapaci di rappresentare propriamente il contenuto e quindi dovrebbero essere evitati, come bisogna evitare dimensioni troppo grandi che richiederebbero un calcolo computazionale più complesso. Una possibile soluzione è che l'algoritmo trovi la giusta dimensione per il vettore.

Un'altra caratteristica che la rappresentazione della SBPCG dovrebbe avere è una alta località, cioè che un piccolo cambiamento del genotipo dovrebbe in media portare piccoli cambiamenti nel fenotipo e piccoli cambiamenti al valore di *fitness*.

Infine, è importante che la rappresentazione scelta sia capace di rappresentare tutte le possibili soluzioni al problema.

- Esempio di rappresentazione diretta-indiretta a livello di gioco: un labirinto (per esempio utilizzato in giochi di avventura nei *dungeon* come *Rogue*). Esso può essere rappresentato:
  - Direttamente come una griglia dove i cambiamenti agiscono sul contenuto di ogni cella (che può contenere per esempio una porta, uno spazio libero, un mostro, un muro, il terreno);
  - Più indirettamente come una lista delle posizioni, dell'orientamento e della lunghezza dei muri;
  - Ancora più indirettamente come un insieme di pattern differenti riusabili di muri e spazi liberi, e una lista di come

essi sono distribuiti (con diverse variabili di trasformazione come rotazione e ordine di grandezza);
- Molto indirettamente come una lista di proprietà desiderabili (numero di porte, stanze, mostri, lunghezza dei percorsi e fattore di ramificazione);
- Indirettamente come un numero casuale.

Queste rappresentazioni producono molti domini di ricerca differenti. Nel primo caso ad ogni fenoma corrisponde un genoma, grazie al *mapping* 1-1. Anche la località è molto alta perché ciascuna mutazione può agire solo su una singola cella quindi i cambiamenti produrranno piccole variazioni anche nel valore di *fitness*. Comunque sia, dato che la lunghezza del genotipo deve essere uguale al numero di celle nella griglia, i grandi labirinti potrebbero incorrere nel problema della dimensione del vettore che rappresenta la mappa. Un labirinto 100x100 deve essere codificato come un vettore di lunghezza 10.000, che è più di quanto di solito molti algoritmi hanno a che fare. Invece, una soluzione estrema come la rappresentazione come un unico numero casuale non soffre di questo problema, ma dall'altro lato della medaglia corrisponde una bassa località.

B. Funzioni di valutazione

Una volta che il contenuto candidato è stato generato, deve essere valutato dalla funzione di *fitness* e gli viene assegnato un valore (o un vettore di valori) che riflette la sua idoneità per l'utilizzo nel gioco. La progettazione della funzione di *fitness* è molto importante: in base alle decisioni prese dal progettista, si può decidere cosa ottimizzare e formalizzare. Per esempio, se l'intenzione è creare un algoritmo che crei un contenuto "divertente" il progettista deve concepire una formula che rifletta quanto un componente può contribuire al senso del divertimento del giocatore.

Tre classi chiave di funzioni di *fitness* possono essere distinte per gli obiettivi della PCG:

- Funzioni di valutazione diretta

  Alcune proprietà vengono estratte dal contenuto generato e vengono mappate in un valore di *fitness* (numero di percorsi

possibili per uscire dal labirinto, grado di utilizzo di un arma e cosi via);

- Funzioni di valutazione basate su simulazioni
  Si basano sul provare direttamente il contenuto prodotto attraverso l'utilizzo di agenti artificiali che giocano direttamente il risultato generato. Poi vengono estratti dei dati dal *gameplay* dell'agente (tempo di gioco, vittoria, vite perse, stile di gioco, numeri di salti e cosi via) che vengono utilizzati per calcolare il valore di *fitness* del contenuto;
- Funzioni di valutazione interattive
  Valutano il contenuto basandosi sull'interazione diretta con il giocatore mentre sta giocando, quindi la *fitness* è valutata durante il *gameplay*. I dati possono essere ottenuti dai giocatori esplicitamente con questionari o implicitamente in base a come si comporta il giocatore (come nel caso di funzioni che si basano su simulazioni).

Per come è stata definita la natura della SBPCG, essa utilizza sempre algoritmi stocastici. Quindi non c'è modo di sapere esattamente cosa si ottiene con questi metodi, e generalmente non c'è modo di riprodurre esattamente lo stesso contenuto a meno che esso non venga salvato. Inoltre, non ci sono prove che un algoritmo meta-euristico converga al termine dell'esecuzione (tranne in pochi casi) e non c'è un tempo di completamento garantito, e nessuna garanzia che il risultato prodotto sia di buona qualità. Il tempo dipende solitamente dalla funzione di valutazione e dato che molte di esse prevendono anche una sorta di simulazione dell'ambiente di gioco creato, questo fattore incide molto sul tempo di generazione.

Capitolo 2

Computazione evolutiva

La Computazione Evolutiva è una metodo che fa parte dell'area di ricerca di Intelligenza Artificiale per la risoluzione di problemi e l'apprendimento automatico [15]. Raggruppa al suo interno diversi modelli di calcolo che si ispirano alla teoria di Darwin sull'evoluzione. Ogni sistema nel quale è valido il principio evoluzionistico di Darwin è caratterizzato da queste quattro proprietà [16]:

1. La presenza di una o più popolazioni di individui che competono tra di loro per le risorse che sono limitate;
2. La nozione di cambiamento dinamico della popolazione dovuta alle nascite e alle morti degli individui;
3. Un concetto di *fitness* che riflette l'abilità di un individuo di sopravvivere e di riprodursi;
4. Un concetto di eredità che riflette il fatto che i figli assomigliano ai genitori ma non sono identici a loro.

Ad ogni individuo è associato un genotipo o cromosoma che lo identifica. A sua volta, ogni cromosoma è composto da unità definite come geni. Il dominio dei possibili valori di un gene è composto da alleli.

The Metaphor

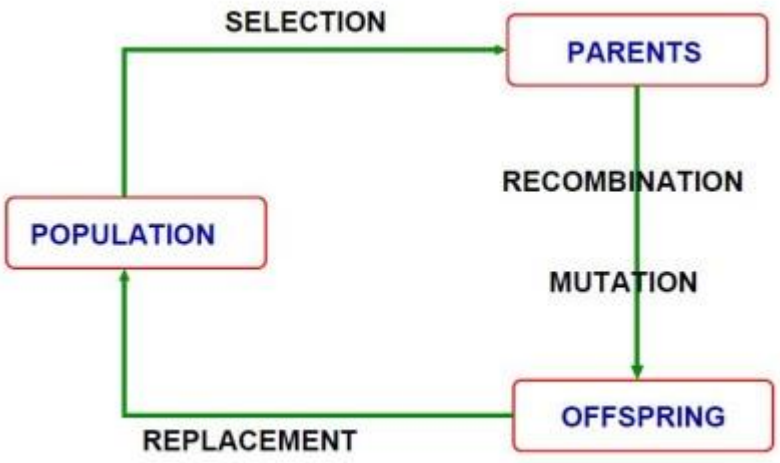
NATURAL EVOLUTION		ARTIFICIAL SYSTEMS
Individual	↔	A possible solution
Fitness	↔	Quality
Environment	↔	Problem

Analogie tra la natura e i sistemi artificiali

Gli algoritmi evolutivi, che fanno parte della Computazione Evolutiva, seguono in linea di massima lo stesso principio di esecuzione. Si parte da un

insieme di soluzioni possibili che formano la popolazione iniziale e iterativamente l'algoritmo cerca di individuare la soluzione desiderata (o un'approssimazione della stessa) tra la popolazione che cambia ad ogni iterazione [17]. Le diverse generazioni di popolazione si ottengono grazie ai meccanismi evolutivi e all'applicazione degli operatori genetici sugli individui, creando nuovi elementi con caratteristiche similari che rimpiazzano i precedenti. Le operazioni che fanno evolvere una popolazione sono:

- Selezione (*Selection*)
- Ricombinazione (*Recombination*)
- Mutazione (*Mutation*)
- Sostituzione (*Replacement*)



Ciclo evolutivo di una popolazione

2.1 Tecniche di Computazione Evolutiva

L'utilizzo dei principi di Darwin per la creazione di nuovi modelli di calcolo aventi per scopo la risoluzione automatica dei problemi di ottimizzazione combinatoria comincia già nei primi anni '50. Tuttavia, il termine Computazione Evolutiva nasce soltanto verso l'inizio degli anni '90, per unificare tre diverse rappresentazioni di una stessa tecnologia: la Programmazione Evolutiva di Fogel, la Strategia Evolutiva di Rechenberg e Schwefel e gli Algoritmi Genetici di Holland [15]. Per questo motivo le

tecniche che rientrano nella categoria di Computazione Evolutiva sono numerose:

- Algoritmi Evolutivi (*Evolutionary Algorithm*)
  - *Gene expression programming*
  - *Genetic algorithm*
  - *Genetic programming*
  - *Evolutionary programming*
  - *Evolution strategy*
  - *Differential evolution*
  - *Differential search algorithm*
  - *Eagle strategy*
- *Swarm intelligence*
  - *Ant colony optimization*
  - *Particle swarm optimization*
  - *Bees algorithm*
  - *Cuckoo search*
- In maniera minore comprende anche
  - *Teaching-learning-based optimization (TLBO)*
  - *Artificial life (also see digital organism)*
  - *Artificial immune systems*
  - *Cultural algorithms*
  - *Firefly algorithm*
  - *Harmony search*
  - *Learning classifier systems*
  - *Learnable Evolution Model*
  - *Parallel simulated annealing*
  - *Self-organization such as self-organizing maps, competitive learning*
  - *Self-Organizing Migrating Genetic Algorithm*
  - *Swarm-based computing*

2.2 Programmazione Genetica

La programmazione genetica, o GP (*Genetic Programming*), è una tecnica di Computazione Evolutiva sistematica ed indipendente dal dominio che rientra nell'area degli algoritmi evolutivi. Il vantaggio di questa tecnica è

che permette di risolvere i problemi senza che l'utente specifichi la forma o la struttura della soluzione [18].

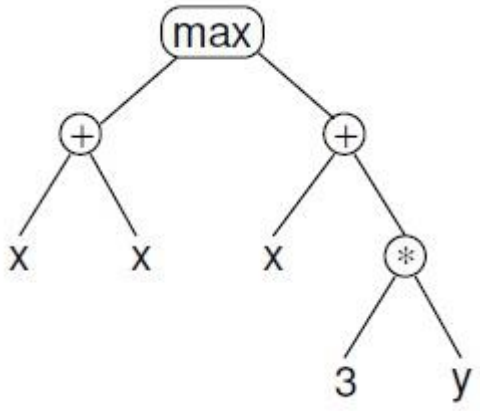
Nella programmazione genetica si evolve una popolazione di programmi. Si differenzia dagli algoritmi genetici (o GA, *Genetic algorithm*) per la rappresentazione della soluzione, in quanto la GP la rappresenta in linguaggio *LISP* o *Scheme*, mentre la GA la rappresenta come crea stringhe di numeri [19].

La programmazione genetica, essendo fortemente ispirata ai principi della natura, come quest'ultima è un processo casuale e quindi non garantisce mai dei risultati ottimali durante la sua esecuzione. Tuttavia, proprio questo principio di casualità può portare anche alla soluzione di problemi che rimanevano irrisolti tramite metodi deterministici.

La programmazione genetica inoltre è indipendente anche dal problema, nel senso che il diagramma che specifica la sequenza base dei passi esecutivi dell'algoritmo i GP non cambia mai dal tipo di problema o da ogni nuova esecuzione [20].

2.2.1 Rappresentazione

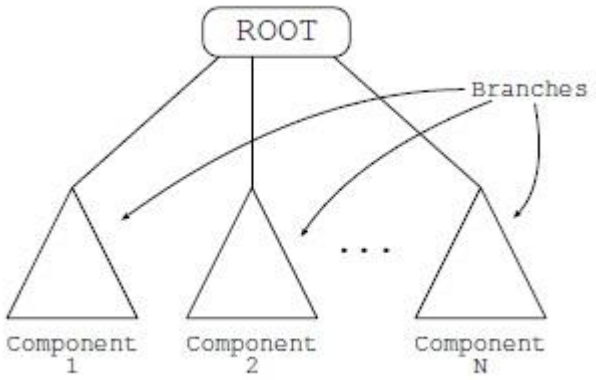
La più comune rappresentazione che viene utilizzata è la forma ad albero sintattico.



Rappresentazione ad albero del programma max(x+x, x+3*y)

Le variabili e le costanti del programma sono le foglie dell'albero e vengono chiamate terminali, mentre invece i nodi interni (nell'esempio in figura le operazioni matematiche) vengono chiamati funzioni. L'insieme dei terminali e delle funzioni formano l'insieme di primitive (*primitive set*) di un sistema di GP. Nel caso di programmi formati da più componenti, la rappresentazione usata è formata un

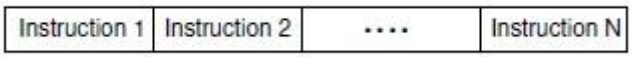
insieme di alberi (uno per ciascun componente, definiti come *branches*) raggruppati sotto uno speciale nodo radice [18].



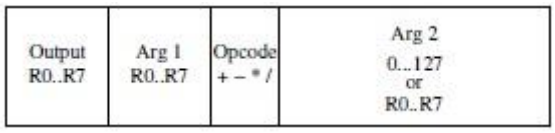
Rappresentazione di un programma con più componenti

Esistono anche altri metodi di rappresentazione e la scelta di quale usare dipende da diversi fattori di convenienza, efficienza, dagli operatori che devono essere utilizzati e cosi via. Alternative possibili di rappresentazione sono:

- Forma Lineare (*Linear GP*)
  I programmi vengono rappresentati come sequenze lineari di istruzioni.

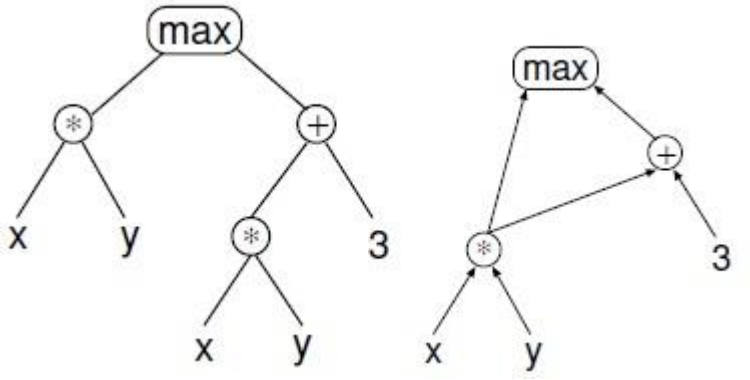


Tipica rappresentazione in forma lineare



Rappresentazione di una istruzione

  Dato che non tutti i computer possono eseguire programmi in rappresentazione ad albero, questo metodo evita l'utilizzo di interpreti o compilatori ed è generalmente più veloce a produrre risultati;
- Forma a grafo (*Graph-Based GP*)
  La rappresentazione ad albero per esempio è un tipo speciale di grafo e rientra in questa categoria, ma è possibile utilizzare anche grafi più complessi



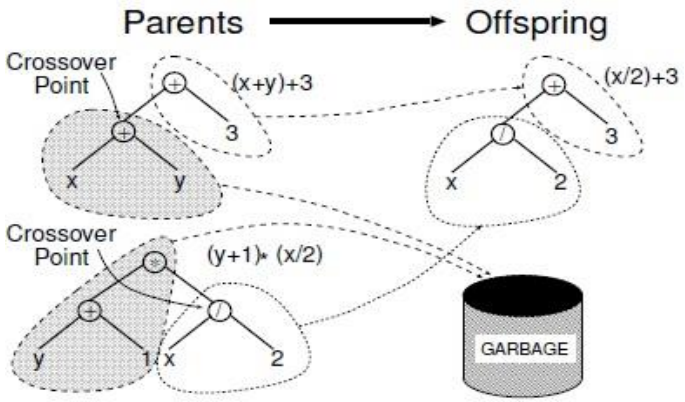
Confronto rappresentazione ad albero – rappresentazione a grafo. La seconda è più efficiente in quanto evita la ripetuta valutazione dello stesso sottoalbero (x*y)

- Forma Cartesiana (*Cartesian GP*)
  In questa rappresentazione i programmi sono cromosomi lineari contenenti interi, che vengono divisi in gruppi da tre o da quattro. Ogni gruppo corrisponde alla posizione in un array bidimensionale.

2.2.2 Operatori Genetici

Sono gli strumenti che permettono di modificare gli attuali componenti di una popolazione per creare nuovi individui con caratteristiche diverse [18].

- Ricombinazione (*Crossover*)
  Esistono diverse forme di ricombinazione. La più comune è la ricombinazione di sottoalberi. Vengono scelti casualmente e indipendentemente due nodi da ognuno dei due genitori (definiti come primo e secondo nodo radice di ricombinazione). Viene poi creato il figlio che è una copia del primo genitore tranne il sottoalbero che ha come nodo radice il primo di ricombinazione, che viene sostituito dal sottoalbero del secondo genitore che ha come nodo radice il secondo di ricombinazione. La scelta del primo e secondo nodo radice di solito non accade con probabilità uniformi, per evitare che lo scambio di sottoalberi si riduca ad un semplice scambio di foglie. Esistono diverse varianti di questo operatore, per esempio: *one-point crossover*, *uniform crossover*, *context-preserving crossover*, *size-fair crossover*;



Esempio di ricombinazione di sottoalberi. Vengono utilizzati delle copie dei genitori , quindi il materiale genetico di questi ultimi può essere riutilizzato per altre operazioni.

- Mutazione (*Mutation*)
  La forma più utilizzata è la mutazione di sottoalberi. Viene scelto un nodo di mutazione e il sottoalbero che ha tale nodo come nodo radice viene sostituito interamente da un nuovo sottoalbero generato casualmente. Esistono anche in questo caso diverse varianti, come la mutazione a punto, nel quale si sceglie causalmente un nodo che viene rimpiazzato da un altro proveniente dall'insieme di primitive che abbia la stessa arità;
- Riproduzione (*Reproduction*)
  Semplicemente copia l'individuo selezionato nella nuova generazione. Può servire nel caso si voglia mantenere comunque sia traccia della vecchia generazione nella nuova perché un individuo ha raggiunto un buon livello di *fitness* e non si vuole perderlo;
- Alteratori dell'architettura (*Architecture-Altering*)
  Nel caso in cui il programma contenga più componenti (*branches*) come ADF (*Automatic define function*), ADI (iterazioni), ADL (cicli) e cosi via esistono operatori che alterano l'architettura del programma, cancellando, aggiungendo o modificando tali componenti.

2.2.3 Passi Preparatori

Prima di procedere all'applicazione del sistema di programmazione genetica è necessario per l'utente svolgere una serie di passi dove si prendono decisioni chiave per il funzionamento della GP [18]. Tali scelte riguardano:

- L'insieme di terminali
  Formato dagli input esterni al programma, funzioni senza argomenti, costanti;
- L'insieme di funzioni
  Formato da tutte le funzioni che hanno argomenti. Insieme al primo passo forma l'insieme di primitive e indirettamente entrambi formano lo spazio di ricerca esplorato dalla GP;
- La funzione di fitness
  E' il meccanismo fondamentale di funzionamento della GP e possiamo definirla come la trasformazione sotto forma di funzione dell'obiettivo desiderato. Essa può essere misurata in diversi modi e la sua forma dipende fondamentalmente dal tipo di problema del quale si vuole trovare una soluzione. Può essere anche multi-obiettiva, nel caso in cui combina due o più elementi che possono anche essere in competizione l'uno con l'altro;
- I parametri per controllare l'esecuzione
  Questa sezione racchiude i parametri di controllo e di impostazione del problema, come la dimensione della popolazione, la probabilità di utilizzo degli operatori genetici (tipicamente si utilizza la ricombinazione nel 90% dei casi, il resto riguarda gli altri operatori), dimensione massima degli individui, metodo di inizializzazione della popolazione (i principali metodi sono *full*, *grow*, *ramped half-and-half*), metodo di selezione degli individui (il più utilizzato è chiamato *tournament selection*). Non è possibile determinare una impostazione ottimale di questi parametri perché dipendono troppo dalla casistica del problema e dalle risorse disponibili;
- Criterio di terminazione e designazione del risultato
  Specifica quando la GP deve terminare la sua esecuzione e il metodo per designare il risultato di quest'ultima (tipicamente è il miglior individuo incontrato nelle diverse generazioni).

2.2.4 Analisi Dell'Algoritmo Di GP

L'algoritmo di GP è sempre lo stesso indipendentemente dal problema del quale si vuole trovare la soluzione

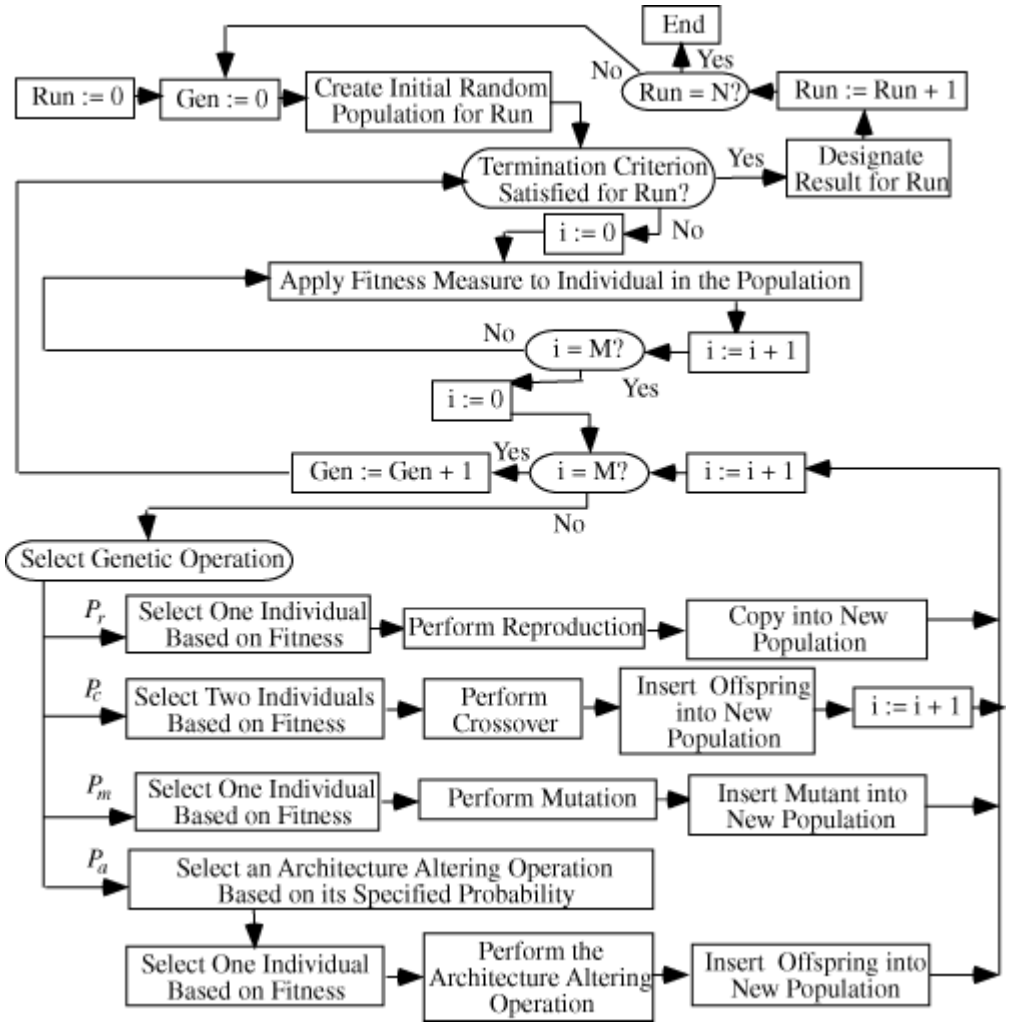


Diagramma di flusso della GP

La programmazione genetica parte con una popolazione iniziale di programmi composti da funzioni e terminali definiti nel primo e secondo passo preparatorio. Ogni individuo può essere di differente dimensione e forma oppure tutti uguali e questo dipende dalle scelte fatte nel quarto passo preparatorio. Poi si calcola il valore di idoneità alla risoluzione del problema di ogni individuo nella popolazione in base alla funzione di *fitness* definita nel terzo passo preparatorio. Successivamente si applicano gli operatori genetici per modificare la popolazione corrente creando i figli che apparterranno alla nuova generazione. Iterativamente si procede alternando le fasi di valutazione e modifica della popolazione fino a quando non viene soddisfatto il criterio di terminazione [20].

2.2.5 Requisiti

Per un funzionamento efficiente della GP, si richiede che l'insieme di funzioni definito nel secondo passo preparatorio abbia la proprietà definita come *closure*, che può essere suddivisa in due proprietà: *type consistency* ed *evaluation safety*. La prima è richiesta perché gli operatori genetici possono mischiare, unire, cancellare i nodi arbitrariamente senza considerare il tipo del nodo (numero intero, funzione, valore booleano e cosi via). La seconda è richiesta perché molte funzioni utilizzate possono "fallire" nella fase di esecuzione.

Inoltre l'insieme di primitive deve anche avere la proprietà di sufficienza (*sufficiency*). Significa che è possibile esprimere una soluzione al problema usando gli elementi dell'insieme di primitive. Nel caso il sistema GP fosse insufficiente, allora tale sistema sviluppa programmi che approssimano la soluzione desiderata.

Per garantire la proprietà *closure* esistono diversi metodi, per esempio introducendo dei vincoli nel sistema in modo tale che tutti gli individui vengano creati secondo qualche regola, cosicché il processo di creazione sia meno casuale. Un approccio è incorporare il concetto di tipo e il loro vincoli nel sistema GP (*strongly typed GP*). Questo implica per esempio, quando si scelgono nodi per la mutazione o per la ricombinazione, che il nodo sostituito deve essere dello stesso tipo del precedente (intero per intero, booleano per booleano e cosi via). Un altro approccio è l'esprimere gli obblighi tramite le grammatiche.

2.3 Grammatica evolutiva

Definita anche come GE (*Grammatical Evolution*) essa rappresenta uno speciale caso di programmazione genetica nel quale è presente il vincolo espresso tramite una grammatica per la struttura degli individui [21].

2.3.1 Forma di Backus-Naur

BNF (*Backus-Naur Form*) è una notazione per esprimere la grammatica di un linguaggio nella forma di regole di produzione. Essa contiene al suo interno terminali (da non confondere con i terminali definiti in GP) che sono gli oggetti che compaiono nel linguaggio (+, -, *, numeri, costanti e cosi via) e non-terminali, che possono essere espansi in uno o più terminali e non-terminali. Una grammatica può essere rappresentata dalla tupla {N, T, P, S}, dove N è l'insieme di

non-terminali, T l'insieme dei terminali, P l'insieme di regole di produzione che mappa gli elementi di N e T , S il simbolo iniziale che fa parte di N. Dove una regola di produzione permette una scelta multipla, le diverse scelte sono delimitate dal simbolo |.

$$N = \{\text{expr}, \text{op}, \text{pre\_op}\}$$
$$T = \{\text{Sin}, +, -, /, *, \text{X}, 1.0, (,)\}$$
$$S = \langle \text{expr} \rangle$$

(1) <expr> ::= <expr><op><expr> (0)
| (<expr><op><expr>) (1)
| <pre-op>(<expr>) (2)
| <var> (3)

(2) <op> ::= + (0)
| - (1)
| / (2)
| * (3)

(3) <pre-op> ::= Sin

(4) <var> ::= X (0)
| 1.0 (1).

Esempio di grammatica in notazione BNF

2.3.2 Rappresentazione

A differenza della GP, la GE non utilizza alberi per la rappresentazione dei programmi (cioè il genotipo) ma una sequenza di lunghezza variabile di numeri interi (in genere a 8 bit) che vengono interpretati grazie ad una speciale regola di *mapping* e alla grammatica specificata dall'utente. Ogni numero intero viene definito come codone (ispirazione sempre derivata dalla genetica).

220	240	220	203	101	53	202	203	102	55	220	202	241	130	37	202	203	140	39	202	203	102

Esempio di un individuo espresso come sequenza di interi

2.3.3 Processo di Mappatura

Il genotipo viene mappato nel programma attraverso la seguente funzione di *mapping* partendo dal simbolo iniziale**:**

$$regola = (valore\ del\ codone)\ MOD\ (numero\ di\ regole)$$

Si parte ovviamente dal simbolo iniziale. Per esempio, nella rappresentazione della grammatica e del genotipo soprastante si parte

da $< expr >$. Dal genotipo leggiamo 220, quindi la regola da scegliere per $< expr >$ è quella contrassegnata con lo 0 in quanto $regola = 220\ MOD\ 4 = 0$. Questo perché le regole di produzione di $< expr >$ sono 4. Quindi al posto di $< expr >$ otteniamo $< expr >$ $< op >< expr >$. Partendo sempre dal simbolo più a sinistra, si continua a leggere i successivi codoni per completare e trasformare la sequenza di numeri nel programma vero e proprio (nel caso precedente si otteneva $1 - 2sin^2(x)$).

Per quanto riguarda gli operatori, i passi preparatori e i passi dell'algoritmo, essi sono sostanzialmente gli stessi della GP, la differenza fondamentale è nella diversa rappresentazione.

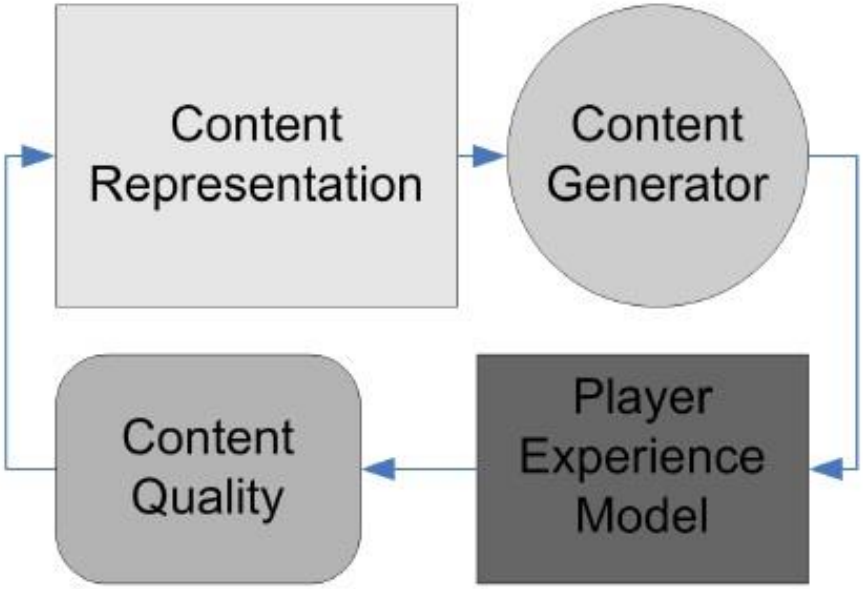
Capitolo 3

EDPCG: Experience-Driven PCG

Negli ultimi anni si è verificato un incremento nella dimensione e nella diversificazione del popolo "giocante". Circa venti anni fa la quasi totalità dei videogiocatori erano giovani ragazzi bianchi con degli interessi nella tecnologia, oggigiorno invece possiamo trovare dei giocatori in qualsiasi società e in qualsiasi fascia d'età. Questo significa che le abilità e le emozioni di ciascun giocatore di fronte allo stesso gioco possono essere anche molto differenti. Nasce quindi un interesse nel mondo videoludico per lo sviluppo di piattaforme con contenuti adattabili a seconda delle abilità e delle emozioni del giocatore, in modo tale da poter generare la stessa esperienza di *gameplay* per qualsiasi utente. Su questa base nasce un nuovo approccio alla PCG classica, che viene chiamato EDPCG (*Experience-Driven Procedural Content Generation*) nella quale viene introdotto l'aspetto di personalizzazione del contenuto [6]. Risulta fondamentale in questo nuovo metodo l'individuazione e il riconoscimento del particolare stato emotivo del giocatore, che è una delle aree di ricerca dell'*Affective Computing* [22]. Per questo il sistema EDPCG, a differenza del sistema SBPCG, necessita anche della creazione di un modello che permetta di conoscere l'esperienza del giocatore, ovvero ottenere la sintesi delle emozioni suscitate e dei processi cognitivi generati durante il *gameplay*.

I componenti della EDPCG sono:

- Modello di Esperienza del Giocatore
  Definito anche come PEM (*Player Experience Model*);
- Qualità del contenuto (*Content Quality*)
  Valutata anche in base alla PEM;
- Rappresentazione del contenuto (*Content Representation*)
  Rappresentato in modo tale da massimizzare l'efficacia, le prestazioni e la robustezza del generatore;
- Generatore di contenuti (*Content Generator*)
  Permette la creazione dei possibili contenuti per ottimizzare l'esperienza del giocatore in base alla PEM.



I principali componenti della EDPCG

Per quanto riguarda la rappresentazione e la generazione del contenuto valgono le stesse considerazioni che sono state fatte nella descrizione del sistema SBPCG, mentre viene rivisitato il concetto di valutazione del contenuto. Il modello di esperienza del giocatore invece è un concetto nuovo.

3.1 Modello di Esperienza del Giocatore

Il modello di Esperienza del giocatore può essere costruito attraverso l'utilizzo di una o più classi di approccio [6]:

- PEM soggettivo
- PEM oggettivo
- PEM basato sul *gameplay*

Le diverse modalità si differenziano in base al processo di ottenimento delle informazioni relative al giocatore. Inoltre, mentre i primi due enfatizzano sia l'aspetto emotivo che quello cognitivo dell'esperienza del giocatore, l'ultimo si focalizza soltanto sull'aspetto cognitivo e sull'interazione tra l'utente e il gioco. Nulla vieta di utilizzare degli approcci ibridi che combinano le tre diverse classi per ottenere dei PEM capaci di catturare l'esperienza del giocatore in maniera più precisa ed efficace.

3.1.1 PEM soggettivo

E' il modo più diretto per creare il modello di esperienza in quanto richiede che siano i giocatori stessi ad autovalutare la loro esperienza di gioco [6]. Esistono due diverse modalità per ottenere i dati: attraverso la risposta libera oppure attraverso la risposta forzata.

La modalità a risposta libera permette di ottenere molte informazioni sullo stato emotivo del giocatore ma è difficile da analizzare in quanto tali informazioni devono essere "estratte" dalla risposta. Un modo per ottenere le informazioni è individuare delle parole chiave nella risposta che possono corrispondere ad aspetti cognitivi e/o emotivi. Tuttavia questo metodo richiede che esistano delle valide ipotesi riguardo la relazione tra le parole chiave identificate e l'esperienza del giocatore. Questa modalità permette l'acquisizione di risposte in forma scritta (dopo la sessione di gioco) o in forma orale (sia durante che dopo la sessione di gioco).

La modalità forzata invece richiede che i giocatori rispondano a diversi questionari (a scelta singola o multipla). In questo caso i giocatori sono vincolati ad autovalutare l'esperienza solo attraverso le risposte predefinite, il che limita la possibilità di esprimere le emozioni ma è un processo che rende più facile l'analisi dei dati rispetto alla modalità a risposta libera. Sia le domande che le risposte nei questionari possono variare in lunghezza, anche se in generale è preferibile utilizzare domande e risposte brevi e chiare altrimenti si mette alla prova la memoria a breve termine e il carico cognitivo del giocatore. L'acquisizione delle risposte avviene sempre dopo la sessione di gioco. Le autovalutazioni forzate possono essere ulteriormente suddivise in:

- Classifiche
  Ai giocatori viene chiesto di rispondere attraverso una scala di valutazione (tipicamente da 1 a 5 dove 1 rappresenta la valutazione peggiore e 5 la valutazione maggiore);
- Preferenze
  Ai giocatori viene chiesto di comparare la loro esperienza in due o più sessioni del gioco (per esempio il livello X è più noioso rispetto al livello Y).

Nel caso in cui la valutazione attraverso metodi diretti avvenga dopo la sessione di gioco, non è stato ancora definito un tempo universale dopo il quale sottoporre il giocatore all'autovalutazione. Dipende quindi dal progettista stimare tale tempo considerando il fattore di discrezione e l'effetto post-esperienza. Le autovalutazioni infatti possono essere intrusive se richieste durante il *gameplay* (fattore di discrezione) e sono molto sensibili alla limitazione di memoria dei giocatori se vengono richieste dopo una lunga sessione di gioco (effetto post-esperienza). Il PEM soggettivo inoltre soffre del problema del rumore sperimentale, cioè non è garantita l'accuratezza dei dati a causa della possibile scarsa comprensione dei giocatori e dall'effetto di autosuggestione.

3.1.2 PEM oggettivo

L'esperienza del giocatore può essere collegata a un flusso di emozioni solitamente innescate da diversi eventi durante il *gameplay*. Tali emozioni possono a loro volta influenzare la fisiologia dello stesso giocatore, modificando in maniera più o meno percettibile l'espressione facciale, la postura, la parlata, la cognizione o altri fattori. Monitorare queste alterazioni può aiutare nel riconoscimento e nella sintesi delle emozioni del giocatore ed è il metodo proposto dall'approccio oggettivo [6]. I dati del giocatore di tipo fisiologico possono essere ottenuti per esempio attraverso:

- Elettrocardiografia (ECG);
- Fotopletismografia;
- Conduttanza cutanea (GSR);
- Respirazione;
- Elettroencefalogramma (EEG);
- Elettromiografia (EMG);
- Pupillometria

In aggiunta agli input di questo tipo, si può anche tracciare l'espressione del corpo del giocatore (definita come *motion tracking*) a differenti livelli di dettaglio (sguardo, espressione facciale, bocca e cosi via). Anche la parlata può essere utilizzata ed ha il vantaggio di essere discreta ed efficiente per l'elaborazione in tempo reale.

Una volta ottenuti i diversi dati, occorre stabilire la relazione tra questi e l'esperienza del giocatore, che può basarsi su modelli esistenti (approccio *model-based*) oppure no (approccio *model-free*). Nel primo caso esistono diversi modelli (per esempio la teoria della valutazione cognitiva) nei quali le reazioni del corpo umano sono mappate a specifiche risposte emotive (per esempio l'aumento del battito cardiaco può corrispondere al divertimento). Nel secondo caso si costruisce un nuovo modello che individui il *mapping* tra gli input e gli stati emotivi. Generalmente il PEM oggettivo che si crea è un ibrido tra i due diversi approcci. Il modello è tanto più accurato e tanto più complesso quante più informazioni fisiologiche si riescono ad ottenere riguardo il giocatore. L'approccio oggettivo ha lo svantaggio di essere molto intrusivo a causa delle modalità di ricezione degli input (ECG, EEG e cosi via) ed è difficilmente realizzabile. Inoltre tale metodo richiede che gli eventi presenti durante il *gameplay* (creati attraverso tecniche procedurali) forniscano stimoli sempre diversi, altrimenti le risposte fisiologiche dell'utente possono diminuire a causa dello stesso stimolo (effetto *habitation*).

3.1.3 PEM basato sul *gameplay*

L'assunzione principale di questo approccio è che tutti gli elementi che derivano dall'interazione tra l'utente e il gioco sono strettamente connessi all'esperienza del giocatore [6]. I giochi infatti possono coinvolgere i processi cognitivi che a loro volta possono influenzare le emozioni. Gli input di questo approccio sono proprietà statistiche spazio-temporali dell'interazione con il gioco. Per stabilire la relazione tra i dati e l'esperienza del giocatore si usa lo stesso processo definito per il PEM oggettivo (*model-based* o *model-free*), anche se l'insieme di modelli che possono essere utilizzati è più ampio (per esempio il modello BDI, la teoria di Scherer o altre teorie specifiche dei giochi, come la teoria del divertimento di Koster o teoria del flusso di Csikszentmihalyi). Nonostante sia probabilmente il metodo computazionalmente più efficiente e assolutamente meno intrusivo rispetto ai metodi precedenti, è un modello che spesso si basa fortemente su assunzioni relative alle relazioni tra dati ed emozioni che potrebbero rivelarsi non esatte.

3.1.4 Principi di modellazione

Esistono diverse modalità per la creazione del modello di esperienza e la scelta di quale metodo utilizzare dipende dal tipo dei dati ottenuti. Generalmente per creare il modello è possibile utilizzare qualsiasi algoritmo di apprendimento automatico (*machine learning*) come: reti neurali, reti Bayesiane, alberi decisionali, regressione lineare standard e cosi via. Altrimenti, per esempio nel caso in cui un'emozione è data in una forma a coppie (il livello X è più divertente del livello Y), il problema diventa di *preference learning* e si devono utilizzare altri tipi di algoritmi come per esempio la *neuro-evolutionary preference learning*.

3.2 Qualità del contenuto

Il PEM è uno strumento aggiuntivo durante la fase di valutazione della qualità del contenuto prodotto tramite generazione procedurale [6]. Attraverso il PEM infatti è possibile valutare quanto contribuisce un particolare contenuto allo stato emotivo del giocatore mentre sta giocando. La scelta comunque sia rimane sempre del progettista: il suo compito è stabilire che cosa ottimizzare (per esempio vuole creare del contenuto divertente, oppure frustrante) e formalizzare le sue scelte sotto forma di funzione di valutazione. Come nella SBPCG, esistono tre classi di funzione di valutazione:

- Diretta
  Proprietà estratte dal contenuto e mappate a dei valori qualitativi. La funzione di *mapping* può essere guidata da modelli teorici oppure dai dati. Nel primo caso il progettista è guidato dall'intuizione o teorie qualitative di emozioni o esperienza del giocatore per derivare il *mapping* tra il modello di esperienza e la qualità del contenuto. Nel secondo caso invece il progettista è guidato dai dati collezionati attraverso questionari o misure fisiologiche per creare la funzione di *mapping*;
- Basata su simulazione
  Un agente artificiale gioca il contenuto prodotto e vengono estratti dei dati relativi al suo *gameplay*. Il comportamento dell'agente può essere statico oppure dinamico, ovvero che cambia ed evolve nel tempo;

- Interattiva
  La valutazione avviene in tempo reale mentre il giocatore sta giocando. Essa può avvenire in modo esplicito (per esempio questionari) o implicito (comportamento del giocatore).

3.3 *Framework* EDPCG

In base alle considerazioni sulla EDPCG possiamo definire il *framework* standard di questa metodologia.

Modello di esperienza del giocatore		
Soggettivo	Oggettivo	Basato sul *gameplay*
Risposta Libera --- Forzata	Basato su modelli --- Senza basi	Basato su modelli --- Senza basi

Qualità del contenuto		
Diretta	Basata su simulazione	Interattiva
Guidata da teorie --- Guidata dai dati	Statica --- Dinamica	Implicita --- Esplicita

Rappresentazione del contenuto
Diretta --- Indiretta

Generazione del contenuto

Capitolo 4

Stato dell'Arte

Lo studio sulla generazione procedurale del contenuto e sulle sue diverse applicazioni è cominciato soltanto pochi anni fa. Fino ad allora infatti non esisteva neppure una comunità accademica che si dedicasse allo studio di quest'area di ricerca. La situazione è notevolmente cambiata recentemente grazie anche all'introduzione di diversi elementi quali:

- Una *mailing list* [23]
- Una gruppo di esperti della IEEE CIS che lavora su questi progetti [24]
- Diversi seminari annuali sull'argomento [25]
- Una *wiki* online [26]
- Una competizione internazionale [27]

Tuttavia al momento non esiste ancora un libro di testo che tratti di PCG e le uniche risorse disponibili sono i diversi articoli accademici che hanno cercato di fornire una tassonomia su questo argomento [4] [6] [7]. Peraltro la classificazione data non è universale ed è in continua evoluzione con la possibilità di variare o introdurre nuove caratteristiche ed elementi, essendo una nuova materia di studio. Nonostante l'interesse sulla PCG sia quindi recente, l'utilizzo di tecniche procedurali per la creazione del contenuto nasce già nei primi anni '80 con i videogames *Rogue* e *Elite*. Fino ad oggi tuttavia le tecniche PCG sono state poco utilizzate e in generale, almeno in riferimento ai titoli presenti in commercio, il loro utilizzo riguarda solo la creazione di piccole parti del contenuto di un gioco. Ricordando infatti le diverse categorie di contenuto che può essere prodotto (*Game Bits, Game Space, Game Systems, Game Scenarios, Game Design*) i titoli commerciali dal 1984 al 2010 che utilizzano tecniche PCG si concentrano soprattutto sulla creazione automatizzata dei *Game Bits, Game Space o Game Scenarios*. Lo scarso utilizzo di queste tecniche per giochi

commerciali può anche essere ricondotto alle poche informazioni sulla materia disponibili al momento. Dal punto di vista sperimentale invece sono numerosi i tentativi di applicazione di queste tecniche ed alcuni di essi includono anche l'aspetto di personalizzazione definito in EDPCG.

Games (with year of release)	Game Bits	Game space	Game Systems	Game Scenarios
Borderlands (2009)	x			
Diablo I (2000)		x		
Diablo II (2008)		x		x
Dwarf Fortress (2006)		x	x	x
Elder Scrolls IV: Oblivion (2007)	x			
Elite (1984)		x	x	x
EVE Online (2003)	x	x		x
Facade (2005)				x
FreeCiv and Civilization IV (2004)		x		
Fuel (2009)		x		
Gears of War 2 (2008)	x			
Left4Dead (2008)				x
.kkrieger (2004)	x			
Minecraft (2009)		x		
Noctis (2002)		x		
RoboBlitz (2006)	x			
Realm of the Mad God (2010)	x			
Rogue (1980)		x		x
Spelunky (2008)	x	x		x
Spore (2008)	x	x		
Torchlight (2009)		x		
X-Com: UFO Defense (1994)		x		

Utilizzo di tecniche di PCG in alcuni videogame

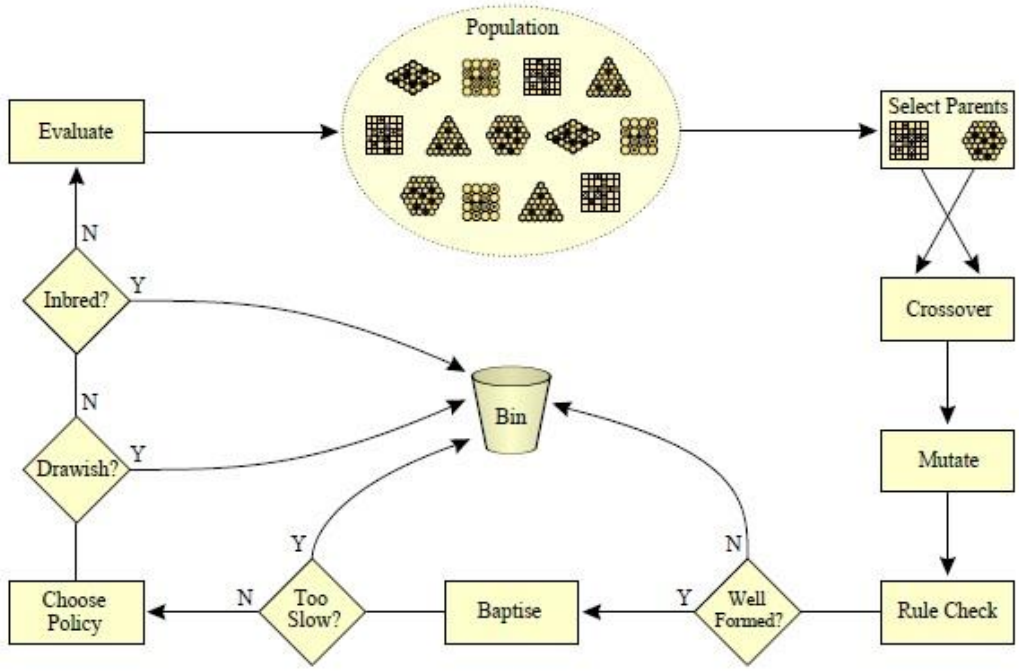
3.1 Esperimenti SBPCG

Per quanto riguarda la ricerca nell'ambito di utilizzo di tecniche procedurali per la generazione del contenuto attraverso algoritmi *search-based* sono stati compiuti diversi esperimenti in cui sono stati analizzati diversi aspetti della SBPCG. Riprendendo la classificazione precedente di tipologie di contenuti possibili da produrre, gli esperimenti finora condotti hanno riguardato:

- Generazione di elementi di *Game Bits*
  Per la creazione di armi e strutture;
- Generazione di elementi di *Game Space*
  Per la creazione di terreni e mappe per giochi strategici;
- Generazione di elementi di *Game Scenarios*

Per la creazione di puzzle, circuiti di corsa e livelli per giochi *platform*;

- Generazione di elementi di *Game Design*
  Per la creazione di regole e meccanismi di un gioco. L'applicazione di tale metodo non ha riguardato solo i videogiochi ma anche i giochi da tavolo [28]. E' stato anche progettato un sistema per la creazione di regole in modalità offline per giochi da tavolo definito *Ludi*, che utilizza una forma di programmazione genetica [1].



Il sistema *Ludi* per la generazione di regole di gioco

Gli esperimenti si differenziano oltre per il tipo di contenuto prodotto anche da altre scelte di progettazione come la rappresentazione, che varia dai vettori di numeri reali alla forma ad albero oppure alla forma matriciale.

Per quanto riguarda invece la funzione di fitness, è stata scelta maggiormente la forma diretta o la forma basata su simulazione. Principalmente le funzioni erano a una singola dimensione o combinazioni di diverse funzioni a singola dimensione.

3.2 Esperimenti EDPCG

La ricerca in questo ambito invece è ancora più recente e i pochi esperimenti possono essere suddivisi principalmente in base alle modalità di creazione del modello del giocatore e dal tipo di funzione di fitness utilizzata. Incrociando queste due caratteristiche possiamo stilare una tabella riassuntiva che quantifica il numero di esperimenti effettuati secondo le diverse modalità. Il modello PEM può essere Soggettivo (S), Oggettivo (O), basato sul *Gameplay* (G) o versioni ibride (SO, SG, OG, SOG). Esistono anche delle combinazioni impossibili, esse sono denotate con il simbolo "x".

		Modello del giocatore						
Funzione di fitness		S	O	G	SO	SG	OG	SOG
Diretta	Basata su teorie			4	2			
	Basata sui dati			3	4	2	1	2
Simulata	Statica	x	x	4	x	1		
	Dinamica	x	x	1	x			
Interattiva	Esplicita	1	x	x			x	1
	Implicita	x		1	x	x		x

In generale il metodo più utilizzato per la costruzione del PEM è quello basato sul *gameplay* o versioni ibride che includono tale metodo. L'obiettivo generalmente è sempre stato il mantenimento di un ottimale livello di difficoltà che manifestasse nel giocatore uno stato di divertimento costante. Per quanto riguarda le modalità con le quali sono stati ottenuti i dati, la scelta ha riguardato:

- Metodo Soggettivo
  Attraverso la modalità a questionario, sia in forma di classifica che in forma preferenziale;
- Metodo Oggettivo
  Attraverso l'utilizzo di *feedback* fisiologici o *motion traking*;
- Metodo basato sul *Gameplay*
  Attraverso l'estrazione di parametri caratteristici dell'interazione interfaccia-giocatore

Per quanto riguarda i videogiochi commerciali, al momento non esistono titoli che implementino funzionalità di personalizzazione del contenuto attraverso generazione procedurale.

Conclusioni

La tesi approfondisce l'utilizzo di algoritmi, tipicamente *search-based*, per la creazione automatizzata del contenuto di un gioco, con la possibilità di introdurre un aspetto di personalizzazione attraverso il modello di esperienza del giocatore, che permette di rendere personalizzati gli elementi generati, ovvero che la produzione di tali elementi consideri anche gli aspetti cognitivi e/o emotivi di qualsiasi giocatore.

La PCG vista come sola tecnica per la generazione algoritmica degli elementi è utilizzata, seppure raramente, anche in alcuni titoli commerciali e in tali casi il prodotto finale ha riscosso un notevole successo. Tuttavia spesso il suo utilizzo è relegato alla creazione di qualche elemento di gioco, nonostante sia possibile progettare interamente giochi basati solamente su PCG. La scelta di utilizzare le tecniche di generazione procedurale in maniera minore dipende da fattori di costo e di tempo, misurati anche a livello computazionale. In base a questi dati è possibile stabilire un confronto con il processo manuale e verificare la convenienza di utilizzo della PCG. Inoltre a svantaggio della generazione procedurale esiste il fattore casualità, molto presente negli algoritmi di computazione evolutiva, che non convince le aziende produttrici di videogiochi, dato che senza opportuni vincoli e controlli non è assolutamente detto che si ottengano i risultati sperati.

Per quanto riguarda l'aspetto di personalizzazione introdotto in EDPCG, essa è un'area di ricerca molto recente e attualmente non esistono titoli in commercio che implementino tale funzionalità. Questo è anche dovuto alla complessità nel campo di riconoscimento delle emozioni e/o degli aspetti cognitivi che rende difficile la creazione di modelli accurati. Il metodo che sicuramente produce i risultati migliori è un metodo ibrido che combina più approcci (soggettivo, oggettivo, basato sul *gameplay*), ma che si focalizza maggiormente su quello oggettivo.

Tuttavia quest'ultimo metodo allo stato attuale risulta essere molto invasivo e non è realizzabile facilmente. Fortunatamente le aziende produttrici di console stanno svolgendo ricerche e stanno mettendo (e metteranno) a disposizione sempre più dispositivi per una facile implementazione di tale sistema, attraverso l'introduzione di sensori

wearable o di *motion traking* (*PlayStation Move, Microsoft Xbox Kinect, Nintendo Wii Vitality Sensor*) .

Le altre realizzazioni per la creazione del PEM (soggettivo e basato sul *gameplay*) sono computazionalmente più efficienti, ma producono modelli fondamentalmente deboli, basati eccessivamente su assunzioni che potrebbero risultare non veritiere. Inoltre nel caso di PEM soggettivo, tale metodo richiede una gran mole di dati per poter produrre dei risultati accettabili.

Essendo aree di ricerca nuove, la tassonomia è in continua evoluzione ed è possibile ampliarla attraverso l'introduzione, per esempio, di altre modalità per l'acquisizione di dati nella creazione del modello del giocatore.

Vivendo nell'era dell'informazione viene logico pensare di creare un approccio PEM che crei il modello basandosi anche sul recupero dei nostri dati che noi stessi immettiamo in rete (preferenze, gusti e cosi via).

Il metodo EDPCG non deve essere considerato esclusivo per l'applicazione di videogiochi per divertimento: il sistema può essere adottato anche nei cosiddetti *"Serious Game"* e/o *"War Game"* che riguardano simulazioni virtuali per sviluppare all'utente delle abilità e competenze da applicare nel mondo reale, oppure nella cosi definita *"Gamification"*, che riguarda l'utilizzo delle meccaniche e dinamiche dei giochi come livelli, punti o premi, in contesti esterni al gioco per creare più interesse o risolvere problemi.

Ringraziamenti

Desidero ringraziare innanzitutto il Professor Andrea Roli per la sua disponibilità e per la pazienza, per avermi seguito passo a passo in ogni fase della stesura di questa tesi e per avermi guidato nella scelta di questo argomento.

Ringrazio i miei genitori Egisto e Celestina, anche se io preferisco sempre chiamarli mamma e papà, che mi hanno sempre supportato nelle mie scelte, mi hanno permesso di continuare il mio percorso e non hanno mai smesso di incoraggiarmi.

Ringrazio mio fratello Fabio, che ha sempre detto di sperare tanto e di riporre fiducia in me per il mio futuro, dandomi la carica per affrontare gli esami più difficili.

Ringrazio il mio migliore amico Daniele, l'Alice e Nicola, che sono le persone con le quali ho legato di più e alle quali tengo particolarmente. Insieme a loro e tutti gli altri veri membri dei Gianna Friends condivido la maggior parte delle mie giornate e nonostante tutto so che hanno sempre creduto in me.

Ringrazio Marzia, la mia personalissima e fidata "spalla", nonché compagna di corso, in questo lungo percorso. Grazie per tutto il tempo che abbiamo passato insieme a ridere e qualche volta pure a studiare, senza il tuo supporto morale non sarei riuscito ad uscire dai momenti più bui.

Ringrazio tutte quelle persone che, in modo sarcastico oppure no, dicevano che non mi sarei mai laureato. Devo dire che mi hanno spinto ancora di più nel voler completare prima possibile il mio percorso.

Infine un grazie a tutte le persone che mi sono scordato di citare ma che hanno contribuito, chi più e chi meno, alla mia crescita durante questo cammino universitario.

Grazie.

Bibliografia

[1] C. Browne, “Automatic generation and evaluation of recombination games”, Ph.D. dissertation, Queensland University of Technology, 2008.

[2] http://pcg.wikidot.com/ (Visitato il 31/01/2013).

[3] http://en.wikipedia.org/wiki/Generative_music (Visitato il 31/01/2013).

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation” in Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications), vol. 6024. Springer LNCS, 2010.

[5] http://pcg.wikidot.com/pcg-algorithm:procedural-generation (Visitato il 31/01/2013).

[6] G. N. Yannakakis and J. Togelius, “Experience-Driven Procedural Content Generation”. IEEE Transactions on Affective Computing, 2011.

[7] M. Hendrikx, S. Meijer, J. Van Der Velden, A. Iosup, “Procedural Content Generation for Games: A Survey” , 2012.

[8] http://digitalbattle.com/2010/02/20/top-10-most-expensive-video-game-budgets-ever/ (Visitato il 31/01/2013).

[9] http://en.wikipedia.org/wiki/Video_game_development (Visitato il 31/01/2013).

[10] http://en.wikipedia.org/wiki/Rogue_(video_game) (Visitato il 31/01/2013).

[11] http://en.wikipedia.org/wiki/Elite_(video_game) (Visitato il 31/01/2013).

[12] http://www.mkomo.com/cost-per-gigabyte (Visitato il 31/01/2013).

[13] http://en.wikipedia.org/wiki/.kkrieger (Visitato il 31/01/2013).

[14] http://en.wikipedia.org/wiki/Markov_chain (Visitato il 31/01/2013).

[15] http://en.wikipedia.org/wiki/Evolutionary_computation (Visitato il 11/02/2013).

[16] Kenneth A. De Jong, "Evolutionary Computation: A Unified Approach", The MIT Press, 2006

[17] http://en.wikipedia.org/wiki/Evolutionary_algorithm (Visitato il 11/02/2013)

[18] R. Poli, W. B. Langdon, and N. F. McPhee. A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

[19] http://www.geneticprogramming.com/Tutorial/ (Visitato il 11/02/2013).

[20] http://www.genetic-programming.com/gpflowchart.html (Visitato il 11/02/2013).

[21] M. O'Neill and C. Ryan, "Grammatical evolution", IEEE Transactions on Evolutionary Computation, vol. 5, no. 4, pp. 349–358, 2001.

[22] http://en.wikipedia.org/wiki/Affective_computing (Visitato il 12/02/2013).

[23] https://groups.google.com/forum/?fromgroups#!forum/proceduralcontent (Visitato il 20/02/2013).

[24] http://game.itu.dk/pcg/ (Visitato il 20/02/2013).

[25] http://fdg2012.org/workshops/pcg/cfp.html (Visitato il 20/02/2013).

[26] http://pcg.wikidot.com (Visitato il 20/02/2013).

[27] http://www.marioai.org (Visitato il 20/02/2013).

[28] V. Hom and J. Marks, "Automatic design of balanced board games," in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2007, pp. 25–30.