Generazione procedurale di contenuti per videogiochi

Renato Mainetti, Jacopo Essenziale, Riccardo Cantoni, Manuel Pezzera, Alessandro Tironi, N. Alberto Borghese

Applied Intelligent Systems Laboratory - AISLab - Dip. di Informatica, Università degli Studi di Milano.
{renato.mainetti, jacopo.essenziale, manuel.pezzera, alessandro.tironi, alberto.borghese}@unimi.it

Abstract

Lo sviluppo di un videogioco richiede molti contenuti digitali (modelli 3D, texture, suoni, narrativa). La generazione procedurale di contenuti (PCG) può supportare e velocizzare gli artisti coinvolti nella produzione di questi contenuti. Introduciamo il dominio della PCG, identificandone gli aspetti salienti nel campo dell'intelligenza artificiale; presentiamo poi diverse tecniche implementate dal nostro laboratorio.

1 Introduzione

I dati pubblicati dall'Associazione Editori Sviluppatori Videogiochi Italiani *AESVI* relativi all'anno 2018 decretano l'importanza che il mercato dei videogiochi ricopre oggi in Italia, totalizzando un giro di affari di quasi 1,5 miliardi di euro e confermando l'andamento positivo di crescita. La produzione di titoli di successo è legata ad una corretta alchimia di più fattori in cui il lavoro manuale, necessario per la generazione di contenuti di alta qualità, risulta ricoprire oggigiorno un aspetto critico. L'industria video-ludica riesce a produrre ogni anno un limitato numero di titoli di gioco, colmando solo in parte la crescente richiesta. Un aiuto in questo ambito può però essere fornito da strumenti automatici, capaci di aiutare i professionisti coinvolti (game designer, sviluppatori, artisti 3D) riducendo i tempi necessari alla produzione di contenuti digitali. In [Hendrikx *et al.*, 2013] è illustrato come queste tecniche sono impiegate nella generazione di semplici asset audiovisivi (texture, suoni, mesh), ma anche nella costruzione di interi ambienti di gioco, obiettivi e storie. Nel corso degli anni, differenti strumenti sono stati impiegati per esplorare il dominio della generazione procedurale di contenuti (PCG); tra i più interessanti troviamo metodi basati su: **generazione di numeri pseudo-casuali** (PRNG), Grammatiche generative (L-system), Filtraggio di immagini (filtri morfologici e convoluzionali), algoritmi spaziali (frattali, diagrammi di Voronoi), **modellazione e simulazione di sistemi complessi** (Automi cellulari, sistemi multi-agente), **intelligenza artificiale** (Algoritmi genetici, Reti neurali).

2 Metodi

Presentiamo ora tre differenti approcci sviluppati per realizzare automaticamente contenuti in differenti domini: generazione di livelli, generazione di ambienti, generazione di behaviours ed infine generazione di narrativa.

2.1 Level design con Reti Neurali Ricorrenti

Il design dei livelli di gioco di un videogame rappresenta una sfida per qualsiasi level designer. Riuscire a catturare l'attenzione del giocatore e costruire ostacoli di difficoltà progressiva che permettano all'utente di imparare ed apprezzare il gioco può essere definita una forma d'arte. Abbiamo addestrato una rete neurale ricorrente (RNN) impiegando come dataset la matrice dei tile associata ad una semplificazione di 16 dei 32 livelli di SuperMario Bros©. I tile che vanno a comporre ogni livello sono stati codificati mediante caratteri testuali, generando così un testo. I caratteri ascii impiegati "# - = ?" codificano per: pavimento, cielo, blocchi distruttibili e blocchi di sorpresa (Figura 1). Scopo della RNN è inferire la grammatica su cui il testo è basato che, se riapplicata, permetta di generare nuovi testi e dunque nuovi livelli. Queste regole grammaticali, che in pratica codificano l'abilità del game designer, vengono ricavate dalla RNN durante l'addestramento. La fase di addestramento prevede che la rete riceva in input il testo un carattere alla volta, analizzando ed imparando progressivamente la frequenza e la disposizione dei tile all'interno dei livelli di gioco. Terminata la fase di addestramento risulta possibile inizializzare la RNN fornendo in input una sequenza di alcuni caratteri estratti dal testo iniziale e impiegare la rete addestrata per generare una nuova sequenza di caratteri rispettando la grammatica appresa. La fase successiva consiste nel riportare il testo generato nel dominio del gioco attraverso un'operazione di decodifica, trasformando cioè ogni carattere in un tile. Questo esperimento ha generato uno strumento capace di "inventare" interessanti livelli di gioco imparando dallo stile del game designer.

2.2 Generazione procedurale dell'ambiente

Tra i temi più trattati nella PCG trova spazio la generazione delle scene di gioco e in particolare della topologia e degli elementi d'ambiente (e.g., alberi, rocce, strade). Un mondo generato proceduralmente consente di ottenere scenari sempre diversi, permettendo dunque, se richiesto, di avere un'esperienza di gioco differente ad ogni partita. Abbiamo esplorato la creazione automatica del mondo di gioco generando una matrice bidimensionale tramite il Perlin noise [Lagae *et al.*, 2010]. I valori contenuti in questa matrice vengono impiegati per modificare l'altezza dei vertici di cui è composta

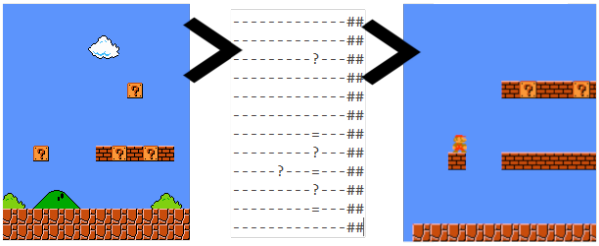

Figura 1: Operazioni di codifica, apprendimento e decodifica, effettuate per la generazione di un nuovo livello di gioco.

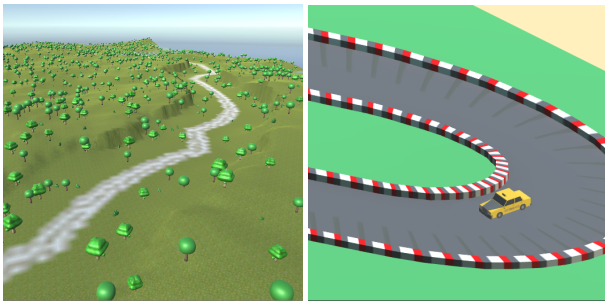

Figura 2: A sinistra la generazione di un terreno con vegetazione ed un sentiero. A destra uno scorcio del circuito automobilistico che l'agente ha imparato a percorrere in autonomia dopo la fase di addestramento basato su PPO.

la superficie planare che rappresenta il terreno e ottenere così degli avvallamenti e delle montagne. Successivamente su questa mappa viene applicato un insieme di texture (erba, roccia, neve). Infine vengono aggiunte delle strade sterrate sul terreno, generate anch'esse casualmente mediante B-spline. Per ogni strada viene generato inizialmente un percorso chiuso identificato da N punti di controllo equispaziati su una circonferenza. Questi vengono poi traslati in direzione radiale sommando alla loro posizione una quantità casuale. A partire dalla generazione della B-spline viene poi applicata la tessitura della strada che viene deformata seguendo la curvatura locale. Ultimo passo è la generazione di elementi ambientali e decorativi, come alberi, rocce e arricchimenti. (Figura 2).

2.3 Behaviours con Deep Learning

La possibilità di introdurre agenti intelligenti in videogiochi permette di sviluppare meccaniche sempre più complesse e coinvolgenti per il giocatore. La democratizzazione del Machine Learning, avvenuta negli ultimi anni, ha consentito lo sviluppo di tecniche di IA in grado di addestrare agenti software a implementare comportamenti, trasformando lo sforzo umano necessario a codificare un comportamento credibile in tempo macchina. Abbiamo sperimentato l'utilizzo di una classe di algoritmi di Apprendimento con Rinforzo che sfruttano una tecnica di ottimizzazione della policy denominata "Proximal Policy Optimization (PPO)" [Schulman *et al.*, 2017]. Tali algoritmi alternano il campionamento di informazioni basato su osservazioni dell'ambiente circostante all'ottimizzazione di una funzione obiettivo approssimata mediante discesa del gradiente stocastico. Utilizzando l'implementazione fornita all'interno del package ML-Agents, abbiamo addestrato un agente a guidare un'automobile all'interno di un circuito. Osservando la velocità dell'agente, la sua distanza dalle pareti e assegnando un reward positivo quando si sposta in avanti e uno negativo per ogni collisione con i bordi della pista, è stato possibile addestrare un agente alla guida autonoma seguendo la traiettoria ideale (Figura 2) ed evitare collisioni, sperimentando così la possibilità di inserire agenti in grado di esibire autonomamente comportamenti intelligenti all'interno di videogiochi.

2.4 Generazione procedurale di Narrativa

L'introduzione di una narrativa all'interno di videogiochi aiuta a migliorare il coinvolgimento e l'immersività dell'esperienza videoludica per l'utente ma comporta un aumento dei costi di produzione. Per minimizzare questo effetto abbiamo esplorato il dominio della "automatic narration" [Kybartas e Bidarra, 2017] sviluppando un tool gerarchico a due livelli che, alimentato da una base di conoscenza contenente materiale narrativo, fosse in grado di produrre strutture narrative sempre nuove. Lo strumento utilizza elementi ereditati dai modelli di Propp e della Fabula per formalizzare i componenti della storia e introdurli in strutture dati che rappresentano le proprietà di un determinato evento della storia. Tale struttura ricalca quella utilizzata nello standard STRIPS, elencando una serie di predicati logici veri nello stato attuale. Le storie vengono generate effettuando cammini aleatori su grafi probabilistici, rappresentanti i singoli eventi. Lo strumento permette di impostare dei vincoli nelle storie forzando la presenza di determinati nodi nel percorso generato. La fase di generazione della storia agisce in maniera agnostica dal punto di vista semantico; l'output è un semplice elenco di nodi visitati. Tali nodi vengono arricchiti di semantica in una fase successiva attingendo dalla base di conoscenza di materiale narrativo, garantendo così flessibilità sul modo di presentare la storia, che potrebbe essere narrata con del testo, con delle immagini o con animazioni generate proceduralmente.

Riferimenti bibliografici

[Hendrikx *et al.*, 2013] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, e Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1, 2013.

[Kybartas e Bidarra, 2017] Ben Kybartas e Rafael Bidarra. A survey on story generation techniques for authoring computational narratives. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):239–253, 2017.

[Lagae *et al.*, 2010] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, e Matthias Zwicker. A survey of procedural noise functions. In *Computer Graphics Forum*, volume 29, pages 2579–2600. Wiley Online Library, 2010.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, e Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.