Documentazione GameDelogger

Aniello Pio La Pietra Antonio Iodice Cosimo Botticelli

https://github.com/shyimon/GameDelogger

2022

Contents

Chapter 1

Introduzione del Problema e Soluzioni Proposte

1.1 Il contesto

Questo progetto nasce dall'esistenza del cosiddetto **backlog** nel mondo dei videogiochi e dei videogiocatori, ossia una lista di giochi posseduti ma non ancora giocati, che si sono accumulati nel corso del tempo.

1.2 La nostra idea

La nostra proposta è quella di consigliare uno o più giochi al giocatore, prendendoli dal suo backlog, in base ai suoi gusti, ai giochi recentemente giocati e alla maniera in cui consuma tale prodotto ludico. Non è difficile immaginare un modulo siffatto implementato in sistemi come Backloggd, Steam o GOG.

1.2.1 I primi problemi e il cambio di direzione

Durante una prima sessione di brainstorming è subito spuntato un grosso problema: per identificare il prossimo gioco da giocare, sarebbe stata necessaria una misura di similarità fra i giochi, così da collegare i candidati ai giochi precedentemente graditi dall'utente. Questo, di per sé, è un problema non banale. Abbiamo quindi deciso di concentrarci sul definire un sotto-modulo di GameDelogger, in particolare quello che individua una misura di similarità fra coppie di giochi.

Dopo aver studiato approfonditamente l'ambiente e in particolare dopo aver ottenuto e analizzato il dataset, siamo arrivati alla conclusione che questo problema non richiede necessariamente algoritmi di intelligenza artificiale. Ciò non vuol dire che un algoritmo di intelligenza artificiale non possa adattarsi al sottomodulo, e non vuol dire che una soluzione orientata all'intelligenza artificiale non sia adatta al modulo completo. Tuttavia, le soluzioni desiderabili sono fuori dallo scope di questo progetto e dalle nostre competenze, lasciandoci in un limbo nel quale le soluzioni realizzabili sarebbero *overkill* per il problema, mentre quelle adatte sono fuori dalle nostre competenze.

In definitiva, questo documento riporta il nostro processo di lavoro, le soluzioni proposte con relative motivazioni e conclusioni.

Chapter 2

Descrizione del Problema e dei Dati

Avendo un certo grado di incertezza sull'algoritmo da impiegare per la risoluzione di questo problema, abbiamo deciso di lavorare innanzitutto sulla definizione dello stesso e sulla raccolta dei dati. Queste sono fasi fondamentali, e comprendere a fondo il problema con cui abbiamo a che fare è un passo indispensabile che prescinde dall'algoritmo usato.

2.1 Specifica PEAS

Performance

La misura di performance dell'agente è direttamente legata alla percezione che un utente avrebbe dei giochi considerati dal sottomodulo come simili, introducendo quindi un certo grado di soggettività. Possiamo pensare di implementare ciò tramite **reinforcement learning**. Nonostante ciò sia difficile all'interno del sotto-modulo, possiamo pensare all'interazione con il sistema in cui esso verrà implementato; se un utente recensisce positivamente un gioco consigliato da GameDelogger sulla piattaforma ospite, questo varrà come feedback positivo, e viceversa. Questa soluzione resta fuori dallo scope del progetto in quanto servirebbe un feedback costante da parte di un bacino d'utenza di cui non disponiamo. In maniera simile, nemmeno un dataset siffatto non esiste.

Environment

L'ambiente in cui l'agente opera è composto dai giochi disponibili e tutte le informazioni relative a essi che abbiamo a disposizione. L'ambiente è:

- **Completamente osservabile**, in quanto abbiamo accesso a tutte le informazioni relative a ogni singolo titolo in qualsiasi momento.
- **Deterministico**, in quanto i cluster da individuare dipendono da caratteristiche del gioco definite a priori e non soggette ad alcun tipo di casualità.
- **Sequenziale**, in quanto le azioni dell'agente hanno potenzialità di influenzare le scelte dell'utente, che a loro volta creeranno uno stato diverso per l'agente.
- **Statico**, in quanto l'agente considera il backlog come una lista di elementi non soggetta a mutazioni nel mentre delibera.
- **Discreto**, in quanto abbiamo un numero discreto di possibili giochi da consigliare.
- A **singolo agente**, in quanto la presenza di più agenti sarebbe superflua.

Actuators

Creazione e restituzione di uno o più giochi ritenuti rilevanti.

Sensors

I sensori dell'agente gli permettono di ottenere tutte le informazioni relative al backlog e ai giochi in esso contenuti.

2.2 Analisi del problema

Dopo aver analizzato l'ambiente, ci rendiamo conto che è molto importante definire con precisione l'entità del **gioco**, per metterlo in qualche tipo di relazione con gli altri giochi.

2.2.1 Il gioco

Un oggetto `gioco` identifica univocamente un prodotto videoludico, in questo specifico caso una entry del backlog, ed è composto dai seguenti parametri:

- **Nome**: è una stringa che identifica il gioco. Seppur raro, due giochi diversi potrebbero avere lo stesso nome, quindi non verrà utilizzata come identificativo.
- **Tempo di completamento**: il tempo di completamento medio, ottenibile tramite un semplice tool di web scraping da un sito come `www.howlongtobeat.com`.
- **Genere**: definito come stringhe separate da una virgola, il genere o generi del gioco è un fattore notoriamente importante per definirne la similarità rispetto ad altri giochi.
- **Sviluppatore**: è in buona sostanza l'autore o gli autori del gioco. Anche esso definito come stringa. Possiamo facilmente immaginare la rilevanza di questo dato; un autore tende ad avere un'impronta artistica rilevante su tutti i giochi che crea, aumentandone il grado di similarità anche se tutti gli altri parametri previamente descritti sono poco affini.
- **Anno di pubblicazione**: banalmente l'anno della prima pubblicazione del titolo che può essere utile per definire la similarità tra giochi. Quest'ultimi se usciti intorno allo stesso periodo possono avere molto di più in comune grazie alle tecnologie utilizzate o all'influenza di alcuni giochi più significativi dell'epoca, per questo il fattore temporale può essere rilevante.

Questi dati devono chiaramente essere estratti e organizzati, operazioni che andiamo di seguito a documentare.

2.3 Dataset

2.3.1 Scelta e ottenimento del dataset

L'idea è quella di raccogliere le informazioni di un certo numero di giochi dal sito `www.howlongtobeat.com`. Per fare ciò ci siamo avvalsi dell'utilizzo di un estensione browser

per effettuare il web scraping del sito web precedentemente citato, ricavando da esso gli attributi necessari per il nostro scopo, quali: il nome del gioco, il tempo di completamento, il/i genere/i, la casa di sviluppo e l'anno di pubblicazione. Da questa operazione abbiamo ricavato un dataset semi-strutturato di 200 titoli.

Esso sarà quindi in formato CVS e avrà una tale struttura:

Nome	**Tempo**	**Genere**	**Developer**	**Anno**
Bioshock: Infinite	$15\frac{1}{2}$ Hours	First Person, Shooter, Action	Irrational Games	2013
Doom	$6\frac{1}{2}$ Hours	First Person, Shooter	id Software	1993
Final Fantasy VII	51 Hours	Role-Playing	Square	1997

2.3.2 Data preparation

Pre-processing

L'unica operazione di pre-processing da fare è sul tempo: esso è espresso nel formato "numero di ore + $\frac{1}{2}$ (opzionale) + Hours", mentre a noi interessa un formato del tipo "numero di ore + .5(opzionale)". È Solo necessario, quindi, rimuovere la stringa "Hours" e trasformare $\frac{1}{2}$ in .5 quando presente.

Data Cleaning

Per una piccola quantità di giochi nel sito manca il genere o anche se c'è non è stato rilevato dallo scraping. Stesso discorso per il tempo, dato che alcuni titoli presenti sul sito avevano a disposizione solo una categoria di tempo, essa non veniva rilevata dallo scraper che prendeva solo una tipologia specifica di tempo. Vi sono varie possibili soluzioni per riempire le celle mancanti, quali: eseguire una ricerca incrociata con altri siti come per esempio lo stesso Wikipedia, il quale riporta il genere di molti giochi; intervenire umanamente ricontrollando `www.howlongtobeat.com`, o se si è a conoscenza del titolo aggiungerlo autonomamente.

Feature Scaling

l'anno di rilascio verrà normalizzato secondo la **min-max normalization**. Questo tipo di normalizzazione è stato scelto perché inquadra solo il periodo di tempo in cui un gioco è stato rilasciato, senza considerarne la distribuzione, dato che non ci interessa, come per esempio farebbe la *z-score normalization*. Alcuni esempi di questo tipo di normalizzazione sono riportati di seguito.

x	x_{norm}
1985	0.05
1997	0.36
2006	0.59
2012	0.74
2022	1.00

Se per l'anno di rilascio la distribuzione non ci interessa, per la durata è importante, in quanto descrive tendenze di mercato rilevanti. Del resto, un gioco è considerato "lungo" o "corto" anche in base alla durata media degli altri giochi. Per questo motivo abbiamo scelto la *z-score normalization*, di cui riportiamo alcuni valori di seguito. Sul dataset ottenuto tramite web scraping abbiamo calcolato un valore della media della distribuzione $\bar{x} = 36.32$ e un valore di distribuzione standard $\sigma = 51.43$.

x	x_{znorm}
10	-0.51176
15.5	-0.40482
22	-0.27844
51.5	0.28544
80	0.84931

Feature Selection

La selezione dei dati appropriati è stata già discussa in precedenza, ma andiamo ora a motivare ed elaborare il perché di queste scelte.

Ecco dunque le caratteristiche da noi scelte o costruite e il perché le riteniamo utili:

- **Nome:** Non ha rilevanza per quanto riguarda l'algoritmo, ma è utile per mostrare a video i nostri risultati.
- **Tempo di completamento:** Nonostante non sia una feature determinante della similarità fra giochi, molti utenti tendono a preferire giochi di durata simile, sia per gusto personale che per quantità di tempo a disposizione. Un'altra feature che potrebbe essere importante è per esempio la durata media di una sessione di gioco, a prescindere dalla durata dello stesso. Nonostante la questione sia interessante, resta aperta a causa di una mancanza totale di un tale dataset o API che permettano di reperire tali informazioni.
- **Genere:** È il metro di similarità più importante per noi, essendo quello che descrive meglio il contenuto del gioco. Un'analisi di mercato sarebbe utile a definire la similarità fra generi, ma per ora ci siamo limitati a definirla in base al numero di generi in comune fra i due giochi.
- **Developer:** Abbiamo preso in considerazione l'impronta artistica e stilistica che il team di sviluppo ha su un determinato gioco come dato importante.
- **Anno di pubblicazione:** Questo è un altro dato che non ci dice molto sul contenuto del gioco, ma che abbiamo ritenuto importante a causa dei grandi cambiamenti che il mercato videoludico ha subito dalla sua nascita. Gli avanzamenti tecnologici hanno portato a un importante cambiamento stilistico nello sviluppo di videogiochi, e le persone tendono a identificarsi con una determinata era videoludica che hanno particolarmente a cuore, motivo per cui abbiamo ritenuto questo dato rilevante.

Andiamo ora a vedere alcuni dei dati che non abbiamo preso in considerazione nonostante fossero presenti su `www.hoelongtobeat.com`, il perché e il se potrebbero essere usati in sviluppi futuri del progetto:

- **Piattaforme di pubblicazione:** In passato la piattaforma di appartenenza di un gioco era rilevante, in quanto diverse compagnie implementavano diverse politiche e addirittura diverse tecnologie per le loro console. Tuttavia oggi queste differenze si sono appiattite molto e tutte le console hanno tecnologie e prestazioni molto simili. Forse più importante, questo modulo è pensato per essere implementato su un servizio come Playstation Now, Microsoft GamePass, Steam, etc. Chiaramente in questi casi la piattaforma è implicata, e sarebbe informazione ridondante.

- **Publisher:** Questa è la casa di pubblicazione di un videogioco. Molto raramente i giochi pubblicati dalla stessa casa hanno similarità stilistiche, in quanto queste sono definite piuttosto dal team di sviluppo.

- **Altri tempi di completamento:** Il sito da noi preso in considerazione ha tempi di completamento per la campagna principale, la campagna + gli extra (quello da noi preso in considerazione) e il completamento del gioco al 100%. Le altre due categorie potrebbero essere molto utili per alcune piattaforme come Steam o Xbox, o qualsiasi altro servizio che fornisca la percentuale di completamento di un gioco per un determinato utente. Possiamo immaginare infatti di dare più peso al tempo di completamento del gioco al 100% se notiamo che l'utente tende a completare molti giochi al 100%.

Data Balancing

Siccome ogni gioco è unico, non abbiamo riscontrato necessità di eseguire un bilanciamento dei dati su questo particolare dataset. Inoltre c'è da dire che **ci aspettiamo** un certo sbilanciamento in un problema del genere, cosa notata durante lo studio di fattibilità.

Chapter 3

Prima Soluzione: Clustering

3.1 Motivazioni e risultati attesi

Una prima idea era quella di formare raggruppamenti di giochi simili tramite un algoritmo di clustering. Questa soluzione si adattava abbastanza bene inoltre al modulo principale del progetto, in quanto avremmo potuto restituire all'utente giochi casuali di un cluster più o meno grande. Difatti non ci interessa consigliare all'utente il gioco più simile in assoluto agli ultimi giocati, in quanto ciò potrebbe portare una certa noia e ridondanza, e quindi risultati poco graditi.

3.2 L'algoritmo da usare

L'idea è di usare un algoritmo *partizionale* ed *esclusivo*.

Potrebbe inoltre essere sia *agglomerativo* che *divisivo*, senza troppo impatto sul risultato finale. Tuttavia dobbiamo considerare che questo modulo andrà implementato in un progetto più grande, che prevede di consigliare un gioco a un utente. Possiamo quindi immaginare l'utilità di un algoritmo di tipo **agglomerativo**: potremmo pensare di creare un circondario dell'elemento atomico che è l'ultimo gioco giocato e fermarci a una certa soglia, quindi non prendendo in considerazione giochi molto distanti. Per lo stesso motivo, sarebbe preferibile un algoritmo **seriale**; elaborare tutti i pattern contemporaneamente non serve se alcuni di loro non verranno nemmeno considerati. L'unione di queste due considerazioni, unite con il fatto che il dataset di giochi presi in considerazione potrebbe essere di dimensioni considerevoli, potrebbe migliorare di molto le performance del modulo.

Date queste informazioni, e usando come riferimento gli algoritmi messi a disposizione da `scikit-learn`, un buon candidato è l'algoritmo `DBSCAN`, in particolare per il fatto che si comporta bene con cluster poco omogenei e non considera gli *outliers*. Questo nel nostro caso è utile, in quanto è possibile nonché probabile che:

1. Ci siano giochi molto generici e quindi appartenenti a cluster più grandi, e giochi facenti parte di nicchie ristrette, appartenenti quindi a cluster minori.

2. Ci siano giochi molto particolari e in definitiva dissimili da qualsiasi altro gioco presente nel dataset.

3.2.1 Problemi

L'algoritmo richiede una matrice in cui le colonne rappresentano i parametri e le righe gli elementi. Nel nostro caso ciò ha creato difficoltà, in particolare per quanto riguarda i **generi**. Dobbiamo considerare che i generi sono collezioni di stringhe, e non valori numerici, da cui nasce la difficoltà nel compararli. Inoltre un singolo elemento non ha senso se preso singolarmente ma ne assume se relazionato ad altri, cosa che ci ha portato alla conclusione che la distanza fra giochi è una misura *semi-metrica*, in quanto la disuguaglianza triangolare non sussiste.

Conseguentemente ai due problemi appena esposti, siamo arrivati alla conclusione che il carico computazionale sarebbe stato piuttosto grande, in quanto si sarebbe dovuta calcolare la distanza fra tutte le coppie di elementi, e che la misura di distanza stessa fosse danneggiata dalla presenza di paragoni fra stringhe non quantificabili in maniera numerica.

3.2.2 Soluzioni proposte

Analizzando il problema del paragone fra stringhe nel contesto dei generi, abbiamo considerato vari tipi di distanza di edit fra stringhe, ma essendo la misura che interessa a noi semantica piuttosto che testuale, nessuna di queste misure hanno fatto al nostro caso. La soluzione sarebbe, molto più semplicemente, considerare quanti generi hanno in comune i due giochi, rafforzando la conclusione che un singolo gioco preso singolarmente perde di significato.

Abbiamo preso anche in considerazione la possibilità di definire una metrica custom fra due elementi nel contesto di `scikit-learn`, cosa che non solo si è rivelata difficile, ma ci ha fatto notare che avremmo dovuto semplicemente calcolare la distanza, tramite una metrica custom, per ogni coppia di elementi, problema che sarebbe meglio risolvibile tramite tecniche di regressione, che andremo a discutere nel capitolo successivo.

3.3 Conclusioni

Il nostro problema, in definitiva, non si adatta a essere risolto tramite clustering, in virtù del fatto che i parametri testuali dei nostri elementi assumono un significato e ci danno una misura di similarità solo quando rapportati a parametri testuali di altri elementi.

La definizione di una distanza custom, nonostante sembrasse fattibile all'inizio, introduce una causa di grave inefficienza in quanto dovrebbe essere calcolata fra ogni coppia di elementi. Diverrebbe inoltre un elemento troppo centrale dell'intero algoritmo, sorpassando addirittura i raggruppamenti e rendendoli superflui.

Chapter 4

Seconda Soluzione: Regressione

4.1 Utilizzo di regressione multipla

Come risoluzione al problema si è pensato, in secondo luogo, all'utilizzo del modello di regressione multipla, utilizzando le caratteristiche del gioco come **predittori**. In questo modo si andrebbe a restituire la misura di similarità fra coppie di giochi, dandoci dunque un valore che ci aiuta a scegliere un numero definito di giochi da suggerire all'utente, dato che lo si utilizzerebbe come metrica. Da qui l'idea della metrica custom correlata alla regressione e l'idea di restituire una lista ordinata di elementi simili. La regressione ci è dunque sembrata un'idea più adatta al nostro scopo rispetto al Clustering.

4.2 Problemi

Il problema principale, che abbiamo riscontrato nel momento in cui abbiamo concepito l'idea di utilizzare il modello di Regressione al posto del Clustering, è che non abbiamo i valori noti e quindi andremmo a definire una semplice equazione, eliminando l'elemento di apprendimento, rendendo di fatto inutile quindi l'utilizzo di un algoritmo di questo tipo

4.3 Soluzioni proposte

Come soluzione proposta si è pensato di rendere il problema con il reinforcement learning e quindi definire la variabile dipendente gradualmente in base al feedback utente. Questo feedback però, come delineato nella sezione Performance della specifica Peas, verrà dato eventualmente da un utente sulla piattaforma che potrebbe ospitare GameDelogger, quindi per il momento non è possibile testarla nel concreto.

4.4 Conclusioni

Arrivati a questo punto si è pensato quindi di abbandonare l'idea di utilizzare un algoritmo di apprendimento e di passare all'utilizzo di algoritmi di ricerca. In particolare si è pensato di utilizzare la ricerca locale e infatti nel capitolo successivo abbiamo delineato e ampliato questa proposta.

Chapter 5

Terza Soluzione: Algoritmo di Ricerca

5.1 Il passaggio da un problema di apprendimento ad uno di ricerca

Scartato in un primo momento il clustering, il modello di regressione sembrava essere quello più adatto alla risoluzione del problema. Il principale limite, come già menzionato, è l'assenza di valori noti a priori che possano quindi indirizzare il problema verso un corretto uso dell'algoritmo di apprendimento. Mancando la parte vera e propria di "apprendimento" il risultato è una semplice equazione andando fuori dallo scope del progetto, quello di risolvere il problema tramite un algoritmo di Intelligenza Artificiale.

Rimanendo dentro il tema farebbe molto comodo utilizzare la **ricerca locale**, in quanto: il nostro modulo deve principalmente restituire dei *giochi simili* all'ultimo gioco gradito dall'utente oppure contrariamente può restituire dei giochi completamente diversi nel caso in cui l'ultimo gioco non è stato particolarmente gradito e implicitamente quindi si potrebbe desiderare un'esperienza diversa.

Definizione di *gioco simile* o *gioco opposto* coincide a meraviglia con i concetti di **massimo globale** e **minimo globale** che a seconda di ciò che l'algoritmo desidera trovare corrisponderà al **ottimo globale**. Per il problema preso in analisi può essere addirittura vantaggioso accontentarsi di **ottimi locali**, così da restituire non un solo gioco ma un insieme di essi più simili o dissimili all'ultimo giocato.

5.2 L'algoritmo da usare

5.2.1 Algoritmo Genetico Multi Obiettivo

Il nostro problema può adattarsi ad un utilizzo di un **algoritmo genetico**: definire una meta-euristica per stabilire quali individui tra una *popolazione* di giochi (nel nostro caso il backlog) possa essere più vicina all'ultimo gioco giocato dall'utente.

I giochi hanno più di un parametro fondamentale come specificato nella nostra Analisi del problema, quindi la nostra meta-euristica dovrà tener conto di più di un obiettivo da massimizzare. Andiamo quindi incontro ad un **problema Multi Obiettivo**.

Approccio ideale e Fronte di Pareto Scartiamo immediatamente un *approccio classico* al problema in quanto non sarebbe fattibile definire una media pesata tra i nostri parametri, trattandosi di stringhe oltre che interi. E' un tipo di operazione molto affine alle soluzioni precedenti, che per gli stessi motivi sono state scartate.

Non rimane quindi che adottare un **approccio ideale**: si terrà conto dei trade-off di tutte le funzioni obiettivo. L'insieme delle soluzioni non dominate andrà a formare il cosiddetto **Fronte di Pareto**, una soluzione per noi alquanto comoda perché porta con sè più di una soluzione ottima localmente come inizialmente era previsto.

5.3 Problemi

Un problema non di poco conto per un'implementazione di un algoritmo genetico arriva nel momento in cui bisogna applicare le operazioni di *selezione* e *mutazione*. Gli individui del nostro problema sono **preesistenti**, non possono in alcun modo essere generati dei nuovi. Limitare l'algoritmo alla sola operazione di *selezione* sarebbe alquanto riduttivo e di conseguenza anche l'algoritmo genetico non è congruo alla soluzione cercata.

5.4 Soluzioni proposte

Una soluzione può essere sicuramente quella di regredire ad un "banale" **algoritmo di ricerca locale** affinchè trovi la soluzione ottima in uno spazio di soluzioni preesistente. Quest'ultimo, guardando attentamente il dataset, è estremamente irregolare. Torna quindi il problema della similarità tra generi e sviluppatori, che essendo stringhe rendono difficile qualsiasi tentativo di stabilire una metrica coerente.

Pur supponendo di riuscire a stabilire un certo grado di similarità, lo spazio delle soluzioni, per come è strutturato, rende molto difficile la possibilità dell'algoritmo di migliorare iterativamente. Non siamo riusciti a trovare un modo alternativo di formare lo spazio delle soluzioni: ad esempio la rimozione di parametri come genere e/o sviluppatore toglierebbe significato e accuratezza alla soluzione.

5.5 Conclusioni

In definitiva neanche passando alla ricerca locale non è stato possibile dare una soluzione coerente e soprattuto implementabile. Quello che viene da pensare è che sia un problema che per natura non può essere risolto con l'utilizzo dell'Intelligenza Artificiale o che banalmente lo complicherebbe soltanto.

Chapter 6

Conclusioni e Sviluppi Futuri

6.1 Il perché della non riuscita del progetto

Dopo un'attenta analisi del percorso di sviluppo, delle soluzioni proposte e dei dati che abbiamo a disposizione, abbiamo individuato alcune imperfezioni intrinseche del nostro problema, che lo rende poco adatto ad essere risolto con le tecniche a disposizione. In particolare:

- L'elemento di apprendimento del progetto si basa sul feedback di ogni singolo utente, allo scopo di personalizzare l'esperienza. Questo, oltre a rimandare al *reinforcement learning*, richiede una base di utenza di cui non disponiamo.
- In funzione del punto precedente abbiamo preso in considerazione la similarità fra giochi, parametro che non necessariamente richiede feedback utente, ma così facendo siamo stati vittime di un *bias* nei confronti dell'intelligenza artificiale: abbiamo infatti dato per scontato che questa fosse la soluzione migliore quando invece non fa altro che complicare il problema.
- Il nostro dataset contiene valori testuali che vanno paragonati sotto il punto di vista semantico (in particolare la similarità fra generi videoludici), cosa per cui sarebbe utile un'analisi di mercato o un dataset che al momento non esiste. La proposta di scartare questa colonna del dataset è infattibile, in quanto è la più importante per definire la similarità fra giochi.

Una volta delineati i difetti del nostro progetto, ci proponiamo di cercare possibili soluzioni o sviluppi futuri.

6.2 Soluzioni che non fanno uso di Intelligenza Artificiale

Come abbiamo già detto in precedenza, la similarità fra giochi non deve necessariamente essere definita tramite intelligenza artificiale e anzi, una semplice equazione sarebbe più che sufficiente. Possiamo immaginarla in una forma simile a quella descritta di seguito.

Descriviamo innanzitutto i quattro parametri di similarità, ossia in base alla **durata**, all'**anno di uscita**, ai **generi** e al **team di sviluppo**.

Similarità durata

Definita come $simD(gioco_1, gioco_2) = \frac{min(durataGioco_1, durataGioco_2)}{max(durataGioco_1, durataGioco_2)}$. Questo valore sarà al massimo 1, nel caso di durata identica.

Similarità anno di pubblicazione

Definita come $simA(gioco_1, gioco_2) = \frac{min(annoGioco_1, annoGioco_2) - minAnno}{max(annoGioco_1, annoGioco_2) - minAnno}$. Anche questo valore varia da 0 a 1. C'è da specificare che il parametro $minAnno$ corrisponde all'anno di pubblicazione più remoto registrato nel dataset al momento.

Similarità generi

Definita come $simG(gioco_1, gioco_2) = \frac{numGeneriInComune}{max(numGeneriGioco_1, numGeneriGioco_2)}$. Ancora una volta, questo valore sarà al massimo 1, nel caso in cui i due giochi siano descritti con lo stesso numero di generi, tutti uguali.

Similarità sviluppatori

Definita in maniera booleana come $simS$. Vale 1 se i due giochi hanno lo stesso sviluppatore o team di sviluppo, 0 altrimenti.

Similarità giochi

Usando le similarità descritte prima, definiamo infine la similarità fra due giochi come segue:

$$simGiochi = w(simD) + x(simA) + y(simG) + z(simS)$$

Se vogliamo un valore di similarità compreso fra 0 e 1 ci basta porre $w + x + y + z = 1$. Ciò che ci resta ora è una semplice somma pesata, dove i pesi possono essere visti come l'importanza che diamo a ognuna delle caratteristiche rispetto alle altre.

Questi pesi potrebbero essere decisi tramite un'analisi di mercato, dando una soluzione definitiva e immutabile al problema che ci siamo posti all'inizio del progetto: il valore di similarità verrebbe usato per restituire all'utente una selezione di giochi simili a quelli che ha maggiormente apprezzato.

6.3 Soluzioni che fanno uso di intelligenza artificiale

Abbiamo stabilito che la definizione di similarità fra giochi non necessita di una soluzione basata su Intelligenza Artificiale; tuttavia, possiamo pensare di convertire questo valore di similarità in un valore di *affinità* personalizzato per ogni utente, e in queste quattro variabili potrebbero essere dettate, piuttosto che da un'analisi di mercato, da un grado di interesse da parte dell'utente: se esso tende a mostrare poco interesse per la durata (quindi se $simD$ varia molto da un titolo gradito all'altro), il suo peso sarà minore, andando a favorire parametri per cui ha un interesse più specifico (per continuare l'esempio, se gradisce solo giochi usciti dopo il 2015, il valore $simA$ avrà un gran peso e influenzerà molto il risultato).