# Progetto ICON

DI-CA 2021/2022

Capotorto Lorena l.capotorto@studenti.uniba.it 696947

Dimatteo Roberto r.dimatteo6@studenti.uniba.it 707120

# SOMMARIO

# 1 INTRODUZIONE

Steam è una piattaforma digitale della Valve Corporation che si occupa della distribuzione di una grande varietà di videogiochi, sia sviluppati dalla Valve stessa che da altre case produttrici. È una delle più grandi piattaforme di distribuzione digitale e al 2021 offre servizi ad oltre 69milioni di utenti giornalieri.

Visto questo uso elevato, e in quanto noi stessi giocatori che utilizzano spesso la piattaforma, abbiamo deciso di dedicarci, in questo caso di studio, alla creazione di un sistema che possa dare supporto ai giocatori nelle loro scelte e nella ricerca di nuovi giochi.
A tale scopo, abbiamo realizzato:

- un **recommender system** capace di consigliare nuovi giochi sulla base delle preferenze personali dell'utente, poiché tra le tante scelte di giochi disponibili è difficile sapere di preciso quella migliore;
- una **knowledge base** che permette all'utente di interrogare il dominio di interesse tramite un'interazione basata sulla forma 'domanda-risposta', per determinare cosa sia vero e permettere all'utente di acquisire informazioni sui videogiochi;
- un'**ontologia**, che descrive il dominio d'interesse specificando il significato dei simboli nel sistema.

Una volta avviato il programma, l'utente interagisce con questo, potendo in qualsiasi momento scegliere l'opzione che più può accontentare i suoi bisogni del momento potendosi spostare velocemente tra il recommender system, la knowledge base e l'ontologia.

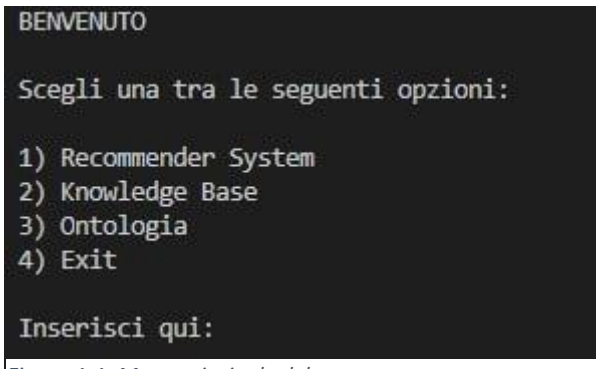


*Figura 1-1: Menu principale del programma*

## 1.1 DATASET UTILIZZATI E LIBRERIE

Il dataset utilizzato è: https://www.kaggle.com/datasets/nikdavis/steam-store-games, a cui sono già state applicate delle operazioni di data cleaning che consistono nella rimozione di elementi compromessi, ordinamento e un controllo di qualità.

Tutto il lavoro è stato scritto in linguaggio Python, utilizzando il code editor VsCode. Per l'intero progetto è stato fatto uso delle librerie **Pandas** e **NumPy**.

# 2 RECOMMENDER SYSTEM

Un sistema di raccomandazione o motore di raccomandazione è un software di filtraggio dei contenuti che crea delle raccomandazioni personalizzate specifiche per l'utente così da aiutarlo nelle sue scelte. Esistono tre tipi di approccio utilizzato per creare raccomandazioni: l'approccio collaborativo, l'approccio basato sul contenuto e l'approccio ibrido.

Abbiamo deciso di utilizzare l'approccio basato sul contenuto, anche detto **Content-based Filtering**, poiché grazie al dataset scelto abbiamo potuto accedere a una serie di informazioni dettagliate sui videogiochi, suddivise in categorie. Questa tipologia di approccio incrocia il contenuto di un elemento e il profilo di un utente. Il contenuto di un elemento è costituito dalla sua descrizione, attributi, parole chiave ed etichette. Questi dati vengono messi a confronto con il profilo utente che ne racchiude le preferenze.

Nel nostro caso, non avendo a disposizione un profilo utente precostruito, abbiamo optato per una interazione diretta con l'utente, il quale inserisce i dati del videogioco da lui scelto o preferito, per poi ricevere una raccomandazione basata su di esso.

```
GET RECOMMENDED

Benvenuto, digita le caratteristiche del gioco su cui vuoi che si avvii la raccomandazione

Inserisci il nome:
Counter-Strike

Inserisci lo sviluppatore:
Valve

Inserisci la casa pubblicatrice:
Valve

Inserisci le piattaforme:        (ricorda tra una parola e l'altra di mettere il simbolo ';' )
windows;mac;linux

Inserisci il genere:     (ricorda tra una parola e l'altra di mettere il simbolo ';' )
Action;Classic;Singleplayer
```

```
Questo è il videogioco che hai inserito:

            name developer publisher         platforms                       genres
0  Counter-Strike     Valve     Valve  windows;mac;linux  Action;Classic;Singleplayer

E' corretto?:   si
```

## 2.1 REALIZZAZIONE

Per la realizzazione del recommender system con Content-based Filtering, abbiamo utilizzato la libreria open source di Python **SKLearn**, la quale ci ha permesso di utilizzare una serie di algoritmi dedicati al Natural Language Processing.

```
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import linear_kernel
```

*Figura 2-1 Libreria sklearn*

Per poter calcolare la raccomandazione, abbiamo sfruttato 5 categorie prese dal dataset: “name”, “developer”, “publisher”, “platforms” e “genre”. Poiché però i dati prelevati da ciascuna categoria si presentano in un formato non adatto al calcolo della similarità tra videogiochi diversi, si è trovata la necessità di trasformare la concatenazione di queste 5 categorie in dei vettori di parole.

Questi vettori di parole sono una rappresentazione vettorizzata delle parole in un “documento”, delle quali il vettore porta con sé un significato semantico per ciascuna di esse.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

*Figura 2-2: Formula TF-IDF*

Tramite l’utilizzo della TF-IDF (term frequency-inverse document frequency) i vettori vengono computati elaborando così una matrice dove ogni colonna rappresenta una parola presa dalla concatenazione delle categorie scelte e allo stesso tempo rappresenterà il videogioco in sé.

Essenzialmente lo score del TF-IDF rappresenta la frequenza di una parola in un documento, pesata sul numero di documenti in cui compare. Questo vien fatto per ridurre l’importanza delle parole che possono apparire frequentemente e di conseguenza anche l’impatto che avrebbero nel calcolo finale.

La libreria SKLearn contiene una classe built-in chiamata “TfIdfVectorizer” che si occupa di produrre la matrice TF-IDF.

```
vectorizer = TfidfVectorizer(analyzer='word')
tfidf_matrix = vectorizer.fit_transform(steam_data['all_content'])
```

*Figura 2-3: Utilizzo della classe TfidfVectorizer*

Una volta prodotta, si può procedere nel calcolo della similarità. Esistono una serie di metriche che potrebbero essere utilizzate, come ad esempio la similarità del coseno, Manhattan, Euclidea e Pearson.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

*Figura 2-4: Formula della similarità del coseno*

Noi abbiamo scelto di utilizzare la similarità del coseno poiché è indipendente dalla magnitudo dei vettori e si presenta relativamente facile e veloce da calcolare, soprattutto in combinazione con l’utilizzo di una vettorizzazione TF-IDF; infatti, tramite l’utilizzo del prodotto scalare tra ogni vettore creato, si otterrà direttamente lo score della similarità. Nella precisione abbiamo optato per l’utilizzo della metrica del **linear kernel**, presa direttamente dalla libreria, la quale risulta equivalente alla **cosine similarity**, solo più veloce.

La linear kernel si presenta come una forma speciale del **polynomial_kernel**, dove se si hanno dei vettori-colonna il loro linear kernel sarà:

$$k(x, y) = x^\top y$$

Si otterrà così una matrice che conterrà gli score di similarità di ogni videogioco con ogni altro videogioco. Ovvero, nella matrice ogni videogioco è rappresentato da una colonna-vettore dove ogni sua posizione rappresenterà lo score di similarità che ha con ogni altro videogioco.

```
tfidf_matrix = vectorize_data(steam_data)
cosine_similarity = linear_kernel(tfidf_matrix, tfidf_matrix)
```

*Figura 2-5: Creazione della matrice TF-IDF e calcolo della similarità*

A questo punto per la raccomandazione vera e propria utilizzeremo un reverse mapping dei nomi dei videogiochi e degli indici presi dal dataframe, ovvero un meccanismo che possa identificare l'indice del videogioco dato il nome.

Verranno mostrati infine i primi 5 videogiochi più simili a quello dato in input dall'utente.

```
Ecco a te i 5 giochi più simili a quello proposto:

                             name                  genres                      developer  price
10           Counter-Strike: Source   Action;FPS;Multiplayer                          Valve   7.19
7     Counter-Strike: Condition Zero   Action;FPS;Multiplayer                          Valve   7.19
25  Counter-Strike: Global Offensive  FPS;Multiplayer;Shooter  Valve;Hidden Path Entertainment   0.00
5                           Ricochet   Action;FPS;Multiplayer                          Valve   3.99
3                 Deathmatch Classic   Action;FPS;Multiplayer                          Valve   3.99
```

*Figura 2-6: Risultato della raccomandazione*

Non sono state prodotte misurazioni e calcoli per quanto riguarda Precision e Recall, poiché nonostante il linear kernel sia una metrica utilizzata comunemente per gli SVM (support vector machines), ovvero in ambito di apprendimento supervisionato, in questo caso è stata utilizzata soltanto per il calcolo della similarità tra vettori normalizzati grazie alla tecnica del TF-IDF, e quindi non è presente alcun tipo di classificazione, pertanto non si avrebbero nemmeno le basi giuste per poter effettuare cosiddette misurazioni.

# 3 KNOWLEDGE BASE

Una knowledge base è una banca dati che riesce a fornire un supporto all'utente grazie alle informazioni presenti al suo interno, e rappresenta fatti sul mondo, un ragionamento su quei fatti e utilizza delle regole e altre forme di logica per dedurre nuovi fatti o evidenziare incongruenze. Quindi, la knowledge base (o KB) è definibile come un insieme di assiomi, cioè delle proposizioni che possono essere asserite essere vere che costituiscono dunque un ambiente volto a facilitare la raccolta, l'organizzazione e la distribuzione della conoscenza.

In merito al nostro caso di studio, la KB viene utilizzata per consentire all'utente di porre domande inerenti al dominio approfondito.

## 3.1 GESTIONE DELLE KNOWLEDGE BASE

Per la gestione della knowledge base, abbiamo utilizzato l'estensione **Pytholog** del linguaggio Python basata sul linguaggio di programmazione logica Prolog.

Sulla base del Prolog, quindi, abbiamo iniziato con il popolamento dei fatti della knowledge base, che avviene automaticamente prelevando i dati interessati alla costruzione del fatto direttamente dal dataset. Così facendo, nel caso in cui il dataset venisse aggiornato con nuovi dati, si aggiornerebbe direttamente anche la KB. Abbiamo poi aggiunto le regole, basandoci su come potessimo gestire i fatti che popolano la KB e su come si sarebbe potuta svolgere l'interazione da parte dell'utente.

### 3.1.1 Fatti

I fatti rappresentano gli assiomi sempre veri della knowledge base, e sono la base per il funzionamento delle regole. Abbiamo 6 tipologie di fatti, ognuno rappresentante una relazione tra i giochi e una delle possibili caratteristiche associate a questi.

Le tipologie di fatti presenti nella KB sono le seguenti, e per ognuno è riportato un esempio:

1) `"developer(name, developer)"`
   Rappresenta la relazione tra un gioco e chi lo ha sviluppato.
   es. `developer(counter-strike, valve)`

2) `"publisher(name, publisher)"`
   Rappresenta la relazione tra un gioco e chi lo ha rilasciato.
   es. `publisher(counter-strike, valve)`

3) `"prices(name, price)"`
   Rappresenta la relazione tra un gioco e il suo prezzo.
   es. `prices(counter-strike, 7.19)`

4) `"stars(name, star)"`
   Rappresenta la relazione tra un gioco e l'apprezzamento da parte degli utenti. 'star' è il valore in percentuale del rapporto tra i rating negativi e i rating positivi, successivamente convertiti in dei numeri interi compresi tra 1 e 5 in base al risultato del rapporto.

```
steam_data.loc[(steam_data['star'] >= 0) & (steam_data['star'] <= 12.5), ['star']] = 5
steam_data.loc[(steam_data['star'] > 12.5) & (steam_data['star'] <= 25), ['star']] = 4
steam_data.loc[(steam_data['star'] > 25) & (steam_data['star'] <= 37.5), ['star']] = 3
steam_data.loc[(steam_data['star'] > 37.5) & (steam_data['star'] <= 50), ['star']] = 2
steam_data.loc[(steam_data['star'] > 50), ['star']] = 1
```

   es. `stars(counter-strike, 5)`

5) `"genre(name, steamspy_tags)"`
   Rappresenta la relazione tra un gioco e il suo genere.
   es. `genre(counter-strike, action;fps;multiplayer)`

6) `"english(name, english)"`
   Rappresenta la relazione tra un gioco e la presenza o meno della lingua inglese. I valori "0" e "1" sono stati rispettivamente convertiti nelle stringhe "no" e "yes".

```
steam_data.loc[(steam_data['english'] == 0), ['english']] = 'no'
steam_data.loc[(steam_data['english'] == 1), ['english']] = 'yes'
```

   es. `english(counter-strike, yes)`

### 3.1.2 Regole

Le regole sono il fulcro dell'interazione con l'utente. Rappresentano delle domande che l'utente può rivolgere alla knowledge base senza il bisogno di conoscere la sintassi necessaria. Utilizzano i fatti che popolano la KB per poter mostrare all'utente i dati che questi ha richiesto.

Le regole presenti nella KB sono le seguenti, e per ognuna è riportato un esempio:

1) `"has_price(X, Y) :- prices(Y, X)"`
   Permette di ottenere informazioni sul prezzo di un gioco o su giochi di un determinato prezzo.
   es. `"has_price(X, counter-strike)" -> X = 7.19`

2) `"quality(X, Y) :- stars(Y, X)"`
   Permette di ottenere informazioni sulla qualità di un gioco, basandosi sulle stelle.
   es. `"quality(X, counter-strike)" -> X = 5`

3) `"developed_by(X, Y) :- developer(Y, X)"`
   Permette di ottenere informazioni su chi ha sviluppato un gioco.
   es. `"developed_by(X, counter-strike)" -> X = valve`

4) `"released_by(X, Y) :- publisher(Y, X)"`
   Permette di ottenere informazioni su chi ha rilasciato un gioco.
   es. `"released_by(X, counter-strike)" -> X = valve`

5) `"is_genre(X, Y) :- genre(Y, X)"`
   Permette di ottenere informazioni sul genere di un gioco.
   es. `"is_genre(X, counter-strike)" -> X = action;fps;multiplayer`

6) `"has_english(X, Y) :- english(Y, X)"`
   Permette di chiedere se il gioco presenta la lingua inglese o meno.
   es. `"has_english(X, counter-strike)" -> X = yes`

7) `"quality_check(X, Y, T, Z) :- stars(X, T), stars(Y, Z)"`
   Permette di ottenere informazioni sulla qualità di due giochi diversi, mettendoli a confronto basandosi sulle stelle.
   es. `"quality_check(team fortress classic, counter-strike, T, Z)" -> T = 4, Z = 5`

## 3.2 Interazione con l'Utente

Durante l'esecuzione del programma, la seconda opzione permetterà all'utente di interagire con la knowledge base e porre domande riferite ai dati dei giochi.

```
KNOWLEDGE BASE

Benvenuto, qui puoi eseguire ricerche sui giochi e sulle loro caratteristiche
Ecco le ricerche che puoi eseguire:
1) Ricerche sulle caratteristiche di un gioco
2) Confronti e ricerca di giochi in base ad una caratteristica
3) Verificare delle caratteristiche
4) Exit Knowledge Base

Quale scegli (inserisci il numero corrispondente alla tua scelta)?
```

*Figura 3-1: Schermata di benvenuto della KB*

Qui l'utente potrà scegliere tra 3 tipi di domande da porre alla KB:

1) Ricerche sulle caratteristiche di un gioco passato in input dall'utente, che, tramite le regole, permettono di ottenere la caratteristica che l'utente richiede:

```
Quale scegli (inserisci il numero corrispondente alla tua scelta)? 1
Dammi il nome di un gioco: counter-strike
Queste sono le caratteristiche che puoi cercare:
1) Chi lo ha sviluppato
2) Chi lo ha distribuito
3) Quanto costa
4) Quante stelle ha (su una scala da 1 a 5)
5) Di che genere è
6) Se è disponibile in lingua inlgese
Selezionane una:
```

*Figura 3-2: Menu mostrato alla scelta di ricerca 1*

Da qui, l'utente sceglie un tipo di informazione che vuole ricevere dalla KB selezionando un numero da 1 a 6 e al termine della richiesta ha la possibilità di fare una nuova ricerca nello stesso campo o tornare indietro al menu principale di interazione con la KB (figura 3-1).

```
Selezionane una: 3

 counter-strike costa: 7.19

Vuoi eseguire un'altra ricerca o vuoi tornare indietro? Indietro (sì), Continua (no)
```

*Figura 3-3: Risultato dell'operazione richiesta*

2) Confronti di due giochi scelti dall'utente in base alla qualità di questi e ricerca di giochi in base ad una caratteristica:

```
Quale scegli (inserisci il numero corrispondente alla tua scelta)? 2
Queste sono ricerche che puoi eseguire sulle caratteristiche:
1) Lista di giochi di un prezzo
2) Confronto di qualità tra 2 giochi
3) Indietro

Selezionane una (inserisci il numero corrispondente alla tua scelta):
```

*Figura 3-4: Menu mostrato alla scelta di ricerca 2*

Qui l'utente può scegliere quale tipo di ricerca approfondita esplorare selezionando un numero tra 1 e 2 per le ricerche e 3 per tornare indietro.

```
Selezionane una (inserisci il numero corrispondente alla tua scelta): 2
Dimmi il nome del primo gioco: counter-strike
DImmi il nome del secondo gioco: rune lord

I due giochi hanno la stessa qualità, perché la qualità di counter-strike e rune lord è uguale
```

*Figura 3-5-1: risultato dell'operazione di confronto richiesta*

```
Selezionane una (inserisci il numero corrispondente alla tua scelta): 1
Inserisci un prezzo: 3.99

Ecco la lista dei primi 100 giochi con prezzo  3.99 :

1 ) 140
2 ) 6120
3 ) a-10vr
4 ) a-escapevr
5 ) a-gents
6 ) a.i.spacecorps
7 ) a.r.e.s.:extinctionagenda
```

*Figura 3-5-2: risultato della ricerca di giochi con lo stesso prezzo*

3) Verifiche sulle informazioni dei giochi, interrogando direttamente i fatti:
All'utente viene chiesto di inserire manualmente il tipo di informazione da verificare, che corrisponde alle tipologie di fatti presenti nella KB.

```
Quale scegli (inserisci il numero corrispondente alla tua scelta)? 3
Questi sono le caratteristiche che puoi verificare:
1) developer
2) publisher
3) prices
4) stars
5) genre
6) english

Selezionane una (scrivi il nome dell'operazione da eseguire, tutto in minuscolo):
```

*Figura 3-6: Menu mostrato alla scelta di ricerca 3*

L'utente deve quindi inserire il nome del gioco e un'informazione corrispondente al tipo di verifica che vuole eseguire. Ha la possibilità poi di ripetere nuovamente l'interazione o di tornare indietro al menu principale di interazione con la KB (figura 3-1).

```
Selezionane una (scrivi il nome dell'operazione da eseguire, tutto in minuscolo): developer
Quale gioco vuoi controllare? counter-strike
Inserisci un dato corrispondente alla caratteristica scelta: valve
['Yes']
```

*Figura 3-7: inserimento dei dati da parte dell'utente e risultato dell'operazione richiesta*

# 4 ONTOLOGIA

L'ontologia, in informatica, è un modello di rappresentazione formale della realtà e della conoscenza. È una struttura di dati che consente di descrivere le entità (oggetti, concetti, ecc.) e le loro relazioni in un determinato dominio di conoscenza.

Abbiamo deciso di utilizzare e creare una ontologia che potesse rappresentare il dominio di un negozio di videogiochi, nel caso in cui l'utente, incuriosito da un apparecchio elettronico, utilizzi questo per farsi consigliare sui videogiochi presenti nel negozio o per esplorare le risorse e le informazioni su di essi, in maniera completamente automatizzata, senza la richiesta di una presenza "umana".

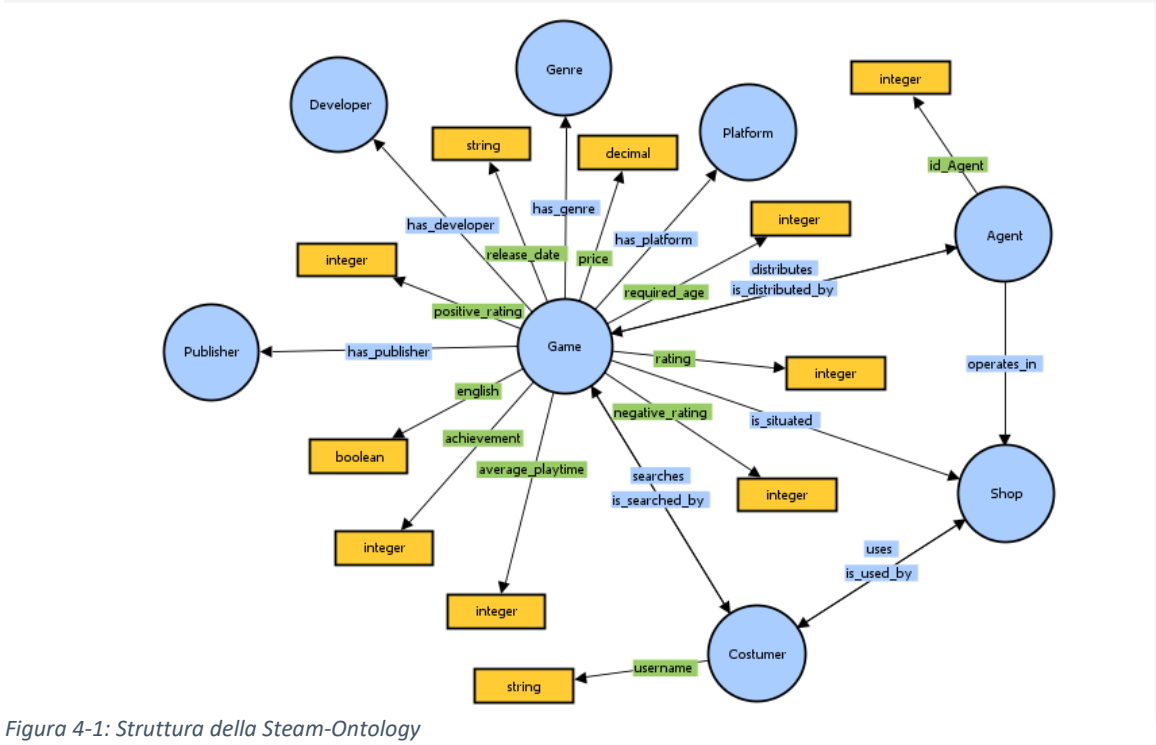


*Figura 4-1: Struttura della Steam-Ontology*

## 4.1 PROTÉGÉ

Per la realizzazione dell'Ontologia abbiamo deciso di utilizzare l'editor di ontologie open source **Protégé**. L'ontologia è organizzata in diverse classi, la cui maggior parte rappresenta una categoria legata al videogioco, che lo descrive nel dettaglio (developer, publisher, platform, genre); per il resto invece avremo le classi rappresentanti gli "attori" presenti nel negozio (agent, costumer) e lo stesso negozio in sé (shop).

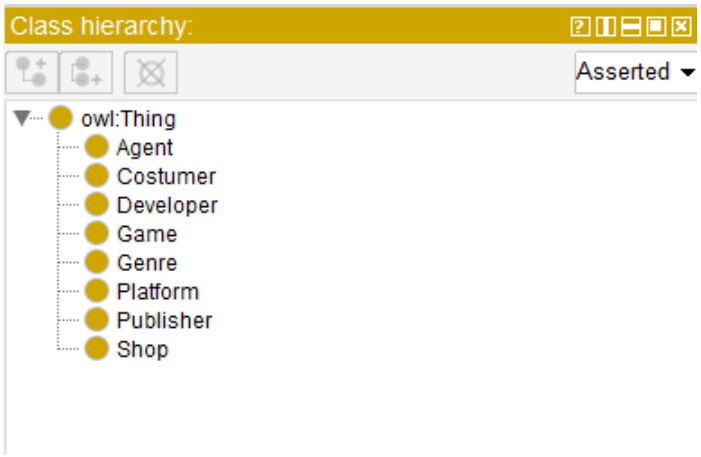


*Figura 4-2: Classi dell'Ontologia*

Oltre alla creazione delle classi, sono state create anche una serie di proprietà: object-properties e data-properties. Queste proprietà aiutano a relazionare due individui di classi uguali o diverse (object-properties) oppure permettono di relazionare un individuo al loro tipo di dato primitivo (data-properties).

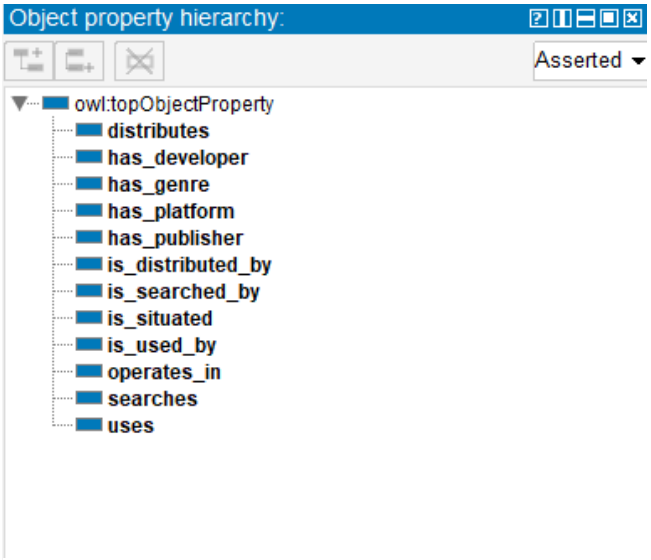


*Figura 4-3: Object properties*

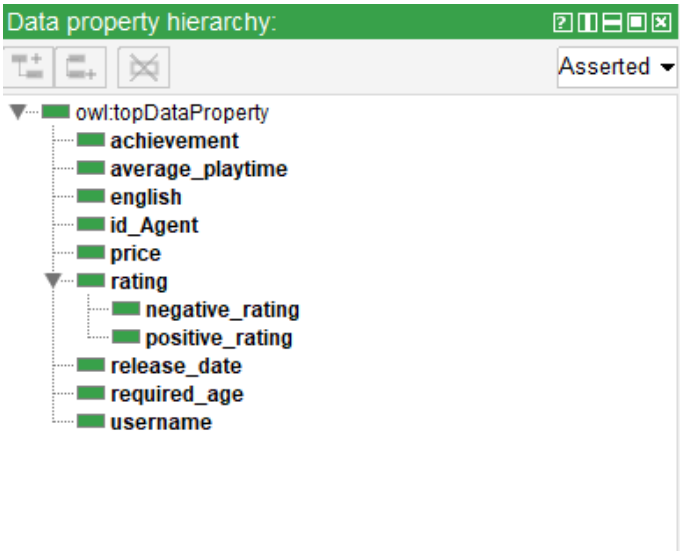


*Figura 4-4: Data properties*

Successivamente si è passati alla creazione degli individui stessi che rappresentano alcuni esempi di videogiochi presenti nel dataset.

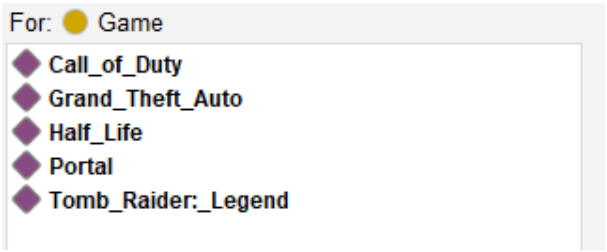


*Figura 4-5: Individui della classe Game*

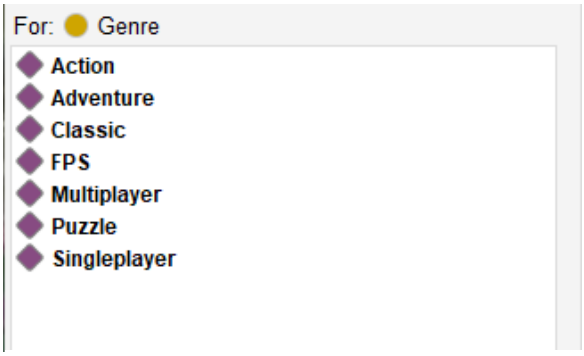


*Figura 4-6: Individui della classe Genre*

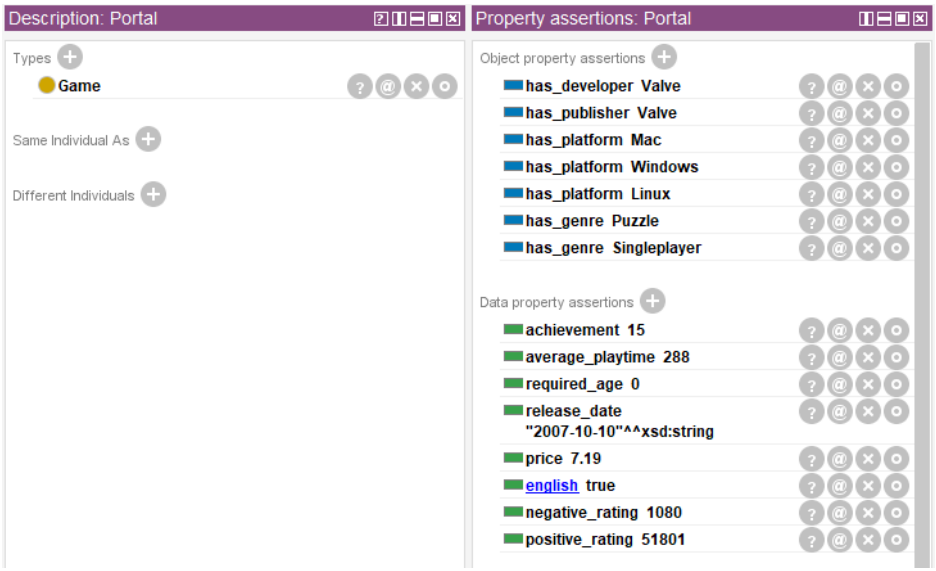


*Figura 4-7: Esempio di individuo con properties annesse*

È possibile anche interrogare l'ontologia tramite l'utilizzo di due tipologie di query, DL query e SPARQL.

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/loren/ontologies/2022/7/steam_ontology#>

SELECT ?Game ?achievement ?price
        WHERE{ ?Game table:achievement ?achievement .
         ?Game table:price ?price}
```

| Game | achievement | price |
|---|---|---|
| Call_of_Duty | "0"^^<http://www.w3.org/2001/XMLSchema#integer> | "14.99"^^<http://www.w3.org/2001/XMLSchema#decimal> |
| Tomb_Raider:_Legend | "0"^^<http://www.w3.org/2001/XMLSchema#integer> | "4.99"^^<http://www.w3.org/2001/XMLSchema#decimal> |
| Grand_Theft_Auto | "0"^^<http://www.w3.org/2001/XMLSchema#integer> | "0"^^<http://www.w3.org/2001/XMLSchema#decimal> |
| Half_Life | "0"^^<http://www.w3.org/2001/XMLSchema#integer> | "7.19"^^<http://www.w3.org/2001/XMLSchema#decimal> |
| Portal | "15"^^<http://www.w3.org/2001/XMLSchema#integer> | "7.19"^^<http://www.w3.org/2001/XMLSchema#decimal> |

Execute

*Figura 4-8: SPARQL query con visualizzazione dei Game, Achievement e Price per gioco*

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/loren/ontologies/2022/7/steam_ontology#>

SELECT ?Game ?has_platform
        WHERE{ ?Game table:has_platform ?has_platform .
        ?Game table:price 7.19}
```

| Game | has_platform |
|---|---|
| Half_Life | Mac |
| Half_Life | Linux |
| Half_Life | Windows |
| Portal | Mac |
| Portal | Windows |
| Portal | Linux |

Execute

*Figura 4-9: SPARQL query con visualizzazione dei giochi che hanno come object properties "has_platform" e un prezzo di 7.19*

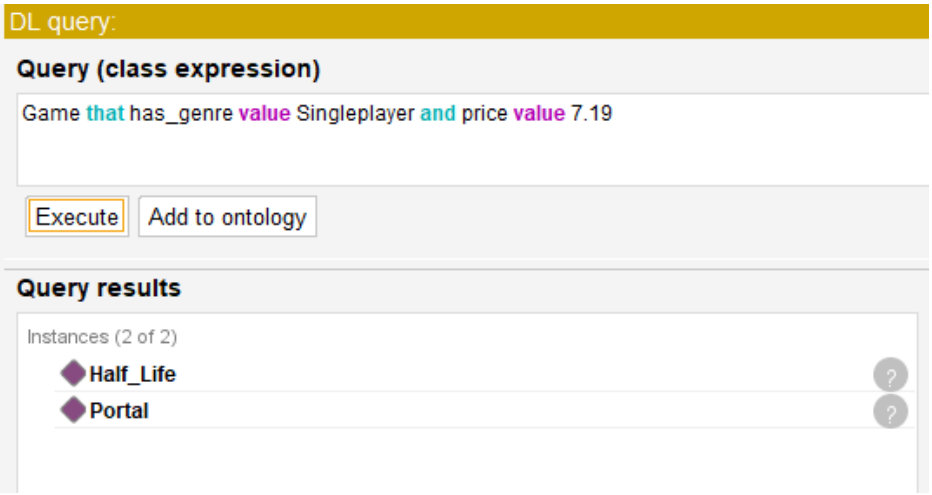


*Figura 4-10: DL query con ricerca di giochi che hanno come genere "Singleplayer" e come prezzo "7.19"*

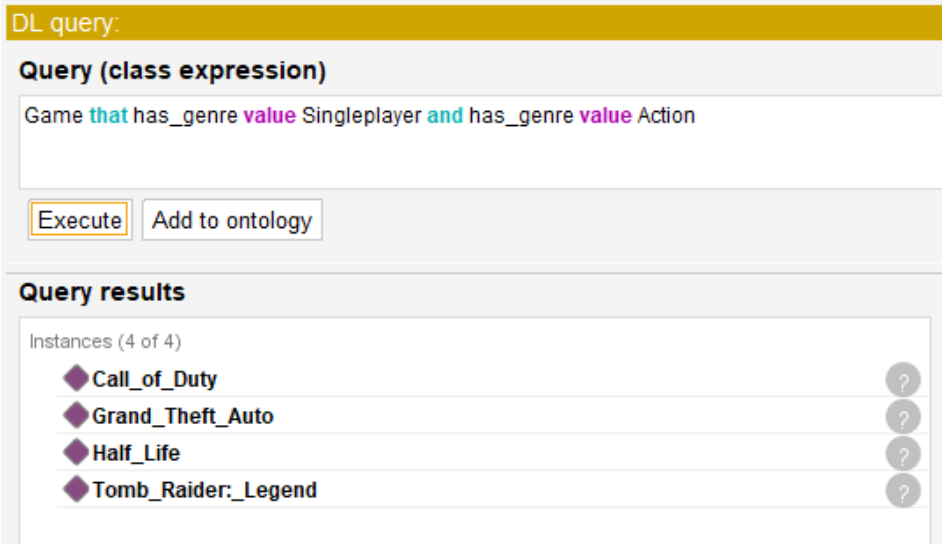


*Figura 4-11: DL query con ricerca di giochi che hanno come genere "Singleplayer" e anche "Action"*

## 4.2 OWLREADY2

Per la consultazione in Python dell'ontologia, è stata utilizzata la libreria **Owlready2**, attraverso la quale è stato possibile, in maniera semplice e comprensibile, interrogare l'ontologia precedentemente creata con il tool Protégé. Nell'esecuzione del programma si potrà scegliere tra la

visualizzazione delle classi, delle proprietà di oggetto, delle proprietà dei dati e della visualizzazione di alcune query d'esempio.

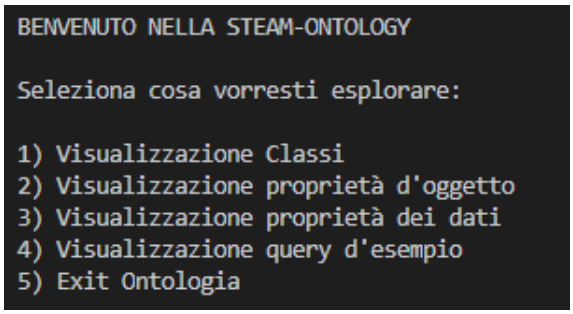


*Figura 4-12: Menù principale per l'esplorazione dell'Ontologia*

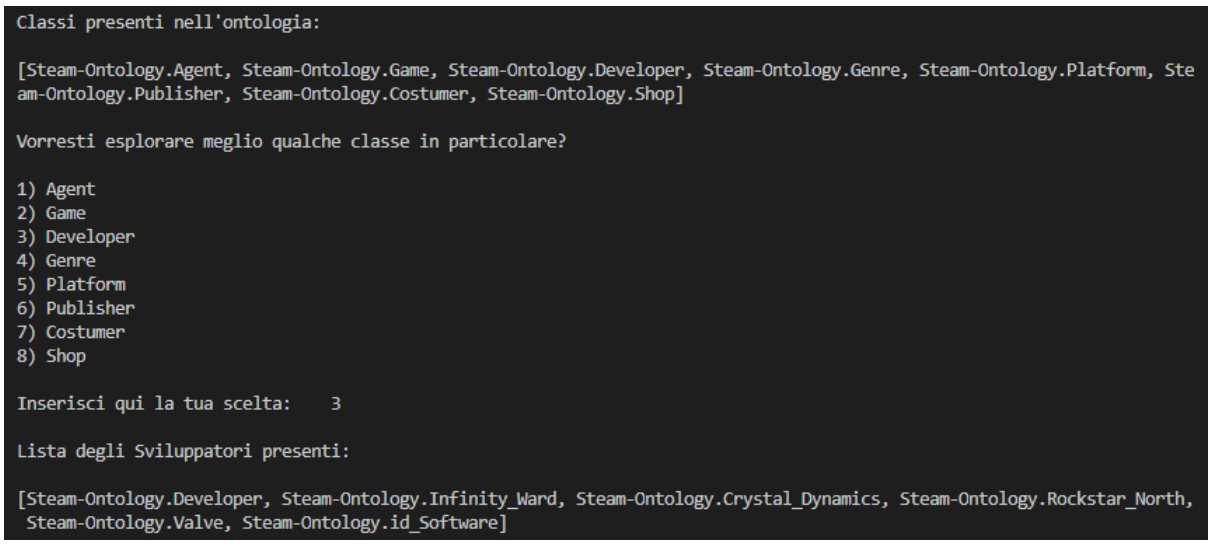


*Figura 4-13: Visualizzazione delle Classi presenti nell'Ontologia*

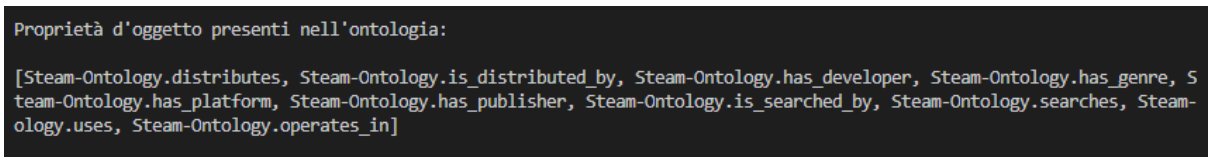


*Figura 4-14: Visualizzazione delle proprietà d'oggetto presenti nell'Ontologia*

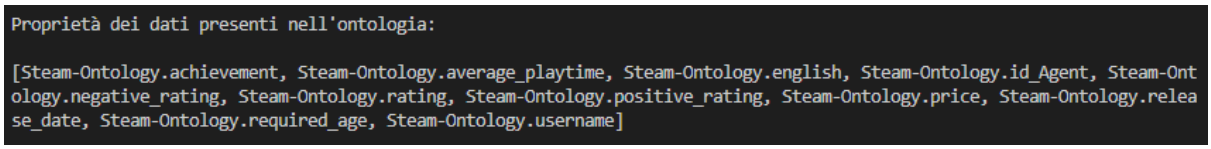


*Figura 4-15: Visualizzazione delle proprietà dei dati presenti nell'Ontologia*

```
Query d'esempio:

-Lista di Giochi che presentano la categoria 'Classic':

[Steam-Ontology.Grand_Theft_Auto, Steam-Ontology.Half_Life]


-Lista di Giochi che presentano lo sviluppatore 'Valve':

[Steam-Ontology.Half_Life, Steam-Ontology.Portal]


-Lista di Giochi che presentano la piattaforma 'Windows':

[Steam-Ontology.Call_of_Duty, Steam-Ontology.Grand_Theft_Auto, Steam-Ontology.Half_Life, Steam-Ontology.Portal, Steam-On
tology.Tomb_Raider:_Legend]
```

*Figura 4-16: Visualizzazione di qualche query d'esempio*

```
print("\nQuery d'esempio:")
print("\n-Lista di Giochi che presentano la categoria 'Classic':\n")
games = ontology.search(is_a = ontology.Game, has_genre = ontology.search(is_a = ontology.Classic))
print(games, "\n")
print("\n-Lista di Giochi che presentano lo sviluppatore 'Valve':\n")
games = ontology.search(is_a = ontology.Game, has_developer = ontology.search(is_a = ontology.Valve))
print(games, "\n")
print("\n-Lista di Giochi che presentano la piattaforma 'Windows':\n")
games = ontology.search(is_a = ontology.Game, has_platform = ontology.search(is_a = ontology.Windows))
print(games, "\n")
```

*Figura 4-17: Esempio costruzione del codice per la formulazione delle query*

# 5 CONCLUSIONI

Due possibili estensioni del progetto sono:

- L'ampliamento del dataset con l'aggiunta di un ulteriore dataset riferito ad un'altra grande piattaforma di distribuzioni digitale, Epic Games, e i dati possono essere combinati per ottenere risultati migliori e più precisi sia nella raccomandazione che nell'interrogazione della KB, implementando così l'interoperabilità semantica tra 2 domini simili.
- L'implementazione di una rete bayesiana per la previsione del successo di un gioco appena uscito sulla base di quello dei giochi attualmente presenti nel dataset.

La repository con tutti i file del progetto: https://github.com/Lunalorla/DI-CA_Icon-2021-2022

La lista delle dipendenze del progetto è presente nel file *requirements.txt* caricato sulla repository.

Il lavoro svolto è stato suddiviso in task assegnati ai due componenti del gruppo.