



Universidad Nacional de Rosario  
Facultad de Ciencias Exactas, Ingeniería y Agrimensura  
T.U.I.A  
Procesamiento del Lenguaje Natural

**Trabajo Práctico 2**

2024

Autora:

Nombre y Apellido	Nº de Legajo
Sol Kidonakis	K-0624/6

Docentes: Juan Pablo Manson y Alan Geary.

Fecha de entrega: Miércoles 18 de noviembre de 2024.

# Índice

1. **Resumen**  
Página 2
2. **Introducción**  
Página 3
3. **Metodología**  
Página 5
  - 3.1 Fuentes de Datos
  - 3.2 Herramientas Utilizadas
4. **Ejercicio 1 - RAG**  
Página 10
  - 4.1 Base de Datos de Grafos
  - 4.2 Base de Datos Tabular
  - 4.3 Base de Datos Vectorial
  - 4.4 Modelos de Clasificación
    - Clasificador basado en Aprendizaje Automático
    - Clasificador basado en LLM
5. **Querys Dinámicas a las Bases de Datos**  
Página 16
  - 5.1 Consultas en Grafos
  - 5.2 Consultas en Base de Datos Tabular
  - 5.3 Consultas en Base de Datos Vectorial
6. **ChatBot**  
Página 20
7. **Ejercicio 2 - Agente**  
Página 22
  - 7.1 Configuración del Agente
  - 7.2 Prompt del Sistema
  - 7.3 Ejemplo de Respuestas del Agente
8. **Conclusión**  
Página 26
  - 8.1 Desafíos en la Preparación de Datos
  - 8.2 Funcionamiento del Chatbot
  - 8.3 Desempeño del agente
9. **Bibliografía**  
Página 28

# 1. Resumen

En este trabajo se desarrolló un chatbot experto en el juego de mesa *Viticulture*, diseñado para ofrecer respuestas precisas y mantener conversaciones fluidas tanto en español como en inglés. Los objetivos principales incluyeron la creación de un sistema capaz de gestionar información compleja sobre el juego y facilitar interacciones eficientes con los usuarios. Para ello, se emplearon métodos avanzados de recopilación y organización de datos, comenzando con web scraping para obtener información clave. La información fue estructurada en tres tipos de bases de datos: una vectorial construida con Chroma, una de grafos para representar relaciones conceptuales y una tabular para datos estructurados.

Se implementaron dos clasificadores: uno basado en ejemplos y embeddings, y otro apoyado en modelos de lenguaje de gran escala (LLM), lo que permitió mejorar la clasificación y recuperación de información. Además, se desarrollaron funciones para búsquedas dinámicas en las bases de datos y herramientas de reranking para optimizar la relevancia de los resultados. Estos elementos fueron integrados en un chatbot que combina todas estas capacidades.

En una segunda fase, se construyó un agente con Ollama utilizando el enfoque ReAct (Reasoning + Acting), capaz de buscar y emplear información de las bases de datos vectorial, tabular o de grafos según las necesidades del contexto.

# 1. Introducción

Entre los títulos más destacados del género de los eurogames se encuentra *Viticulture*, un juego que simula la gestión de un viñedo, combinando mecánicas de colocación de trabajadores y gestión de recursos para ofrecer una experiencia estratégica y temática. Sin embargo, el acceso a información relevante sobre este juego se encuentra fragmentado en múltiples formatos y plataformas: manuales impresos, reseñas en sitios web especializados, debates en foros, y métricas recopiladas en bases de datos de rankings y popularidad.

Con esta problemática en mente, se desarrolló un sistema integral que combina técnicas de PLN, modelado de datos y herramientas de inteligencia artificial para estructurar, analizar y proporcionar acceso eficiente a información sobre *Viticulture*.

## 2.1 Objetivos Específicos

Para lograr este trabajo, se plantearon los siguientes objetivos específicos:

- Extraer información relevante de diversas fuentes.
- Estructurar los datos extraídos en formatos que permitan su manipulación eficiente, creando una base tabular para estadísticas y métricas, una base vectorial basada en embeddings para búsqueda semántica, y un grafo RDF que modele las

relaciones entre diseñadores, mecánicas, categorías y otras entidades clave relacionadas con *Viticulture*.

- Diseñar y entrenar clasificadores eficientes, capaces de categorizar consultas según su contexto y determinar la base de datos más adecuada para obtener la respuesta correcta, comparando un modelo basado en embeddings con un modelo LLM para seleccionar la mejor solución.
- Implementar un chatbot dinámico que permita responder consultas de los usuarios en lenguaje natural, conectando las preguntas con las bases de datos estructuradas y proporcionando respuestas precisas, claras y relevantes.
- Desarrollar un agente inteligente basado en ReAct (Reasoning and Acting), capaz de interactuar con las bases de datos vectorial, tabular y de grafos de manera autónoma, combinando información de múltiples fuentes para resolver consultas complejas y generar respuestas contextualizadas.

## 2. Metodología

### 3.1 Fuentes de datos.

Para este proyecto, se recopilaron datos relevantes sobre *Viticulture* desde diversas fuentes. Las fuentes utilizadas fueron las siguientes:

1. Manual del Juego: Se extrajo texto del manual oficial de *Viticulture* en formato PDF.
  - Contenido: Reglas, objetivos, mecánicas, configuración inicial y componentes.
2. Reseñas Escritas: Se realizó scraping de sitios web especializados en juegos de mesa, como *Misut Meeple*, para capturar reseñas y análisis detallados del juego.
  - Contenido: Opiniones sobre la jugabilidad, balance y estrategia.
3. Reseñas en Video: Se descargó el audio del video *Late to the Game: Viticulture Review* que contiene reseñas del juego y se transcribió.
4. Foros: Se utilizó scraping dinámico para capturar discusiones y opiniones publicadas en *BoardGameGeek*.
5. Datos Tabulares: Información estadística sobre el juego fue extraída de la sección *Stats* de *BoardGameGeek*.
  - Contenido: Clasificaciones, calificaciones, popularidad, número de jugadores y otros datos métricos.

## 3.2 Herramientas Utilizadas a lo largo del trabajo

### LangChain

LangChain es un marco especializado para integrar modelos de lenguaje natural con datos externos, facilitando el procesamiento y la gestión de datos textuales. Una de sus herramientas más útiles en este proyecto fue el Recursive Character Text Splitter, que divide grandes textos en fragmentos manejables manteniendo suficiente contexto entre ellos. Esto resulta esencial para la indexación y recuperación semántica de textos largos, como manuales y reseñas.

LangChain permitió crear fragmentos de texto de aproximadamente 500 caracteres con una superposición de 80 caracteres, asegurando que cada fragmento contuviera información contextual suficiente para búsquedas precisas.

### Selenium

Es una herramienta de automatización de navegadores. Selenium permitió navegar por sitios web dinámicos, como *BoardGameGeek*, donde el contenido es generado por JavaScript y no puede ser capturado con herramientas estáticas. A través de Selenium, se automatizó la interacción con los elementos de la página, como botones de navegación y enlaces, garantizando que se cargue completamente el contenido antes de extraerlo.

## **Universal Sentence Encoder (USE)**

El Universal Sentence Encoder (USE) es un modelo de aprendizaje profundo preentrenado, diseñado para transformar texto en representaciones vectoriales (embeddings). En este proyecto, se utilizó para generar embeddings tanto de los fragmentos de texto como de las consultas de los usuarios, permitiendo realizar búsquedas semánticas precisas en la base de datos vectorial. El modelo elegido en este trabajo fue el multilingüe, para que pueda manejar consultas tanto en español como en inglés.

## **ChromaDB**

ChromaDB es un motor de búsqueda vectorial que almacena y gestiona embeddings de texto, optimizando la búsqueda semántica. Funciona comparando la similitud entre embeddings para devolver los resultados más relevantes a una consulta.

## **RDFlib y NetworkX**

RDFlib es una biblioteca especializada en modelado de datos en formato RDF, permitiendo representar relaciones entre entidades mediante nodos y aristas. Fue utilizada para modelar las relaciones entre diseñadores, mecánicas, categorías y el juego *Viticulture*. Por su parte, NetworkX facilitó la visualización y análisis de estos grafos,



mostrando conexiones clave como "Diseñador → Juego" o "Juego → Categoría".

## **Scikit-Learn**

La biblioteca Scikit-Learn proporcionó las herramientas para implementar el modelo de clasificación basado en aprendizaje automático. En particular, se utilizó un clasificador de regresión logística, una técnica sencilla pero poderosa para problemas de clasificación multiclase.

## **Qwen 2.5**

El modelo Qwen 2.5 se usó para implementar un clasificador avanzado basado en aprendizaje profundo. Este modelo es capaz de interpretar consultas complejas y ambiguas mediante el uso de prompts bien diseñados.

## **Ollama**

Ollama es un servidor de modelos de lenguaje que permite cargar y ejecutar modelos avanzados como "phi3:medium" localmente. Se instala y configura Ollama para servir como backend del modelo LLM.

## **LlamaIndex**

Es una biblioteca poderosa para integrar modelos de lenguaje en flujos de trabajo con diversas fuentes de datos. En este trabajo se usa

para conectar el modelo Ollama con herramientas de búsqueda en bases de datos (tabular, vectorial y de grafos). Se emplea la clase `ReActAgent` que implementa el paradigma de razonamiento y acción (Reasoning and Action), permitiendo que el modelo tome decisiones sobre qué herramienta usar para responder una consulta.

## **ReActAgent**

Este componente de `LlamaIndex` permite modelar un flujo de interacción en el que el modelo primero razona sobre la consulta, selecciona las herramientas adecuadas (como búsqueda vectorial, grafos o tablas) y luego combina los resultados para dar una respuesta coherente.

## 4. Ejercicio 1 - RAG

### Desarrollo e Implementación

El proyecto requirió modelar la información de *Viticulture* en tres tipos de bases de datos: vectorial, tabular y de grafos. Cada una fue diseñada para abordar diferentes necesidades de consulta y análisis, empleando herramientas específicas que permitieron estructurar, almacenar y recuperar datos de manera eficiente.

#### 4.1 Base de Datos de Grafos

La base de grafos modeló las entidades clave de *Viticulture* (juegos, diseñadores, mecánicas, categorías) y sus relaciones. Este enfoque estructurado permitió responder preguntas complejas sobre conexiones entre diferentes elementos del ecosistema de *Viticulture*.

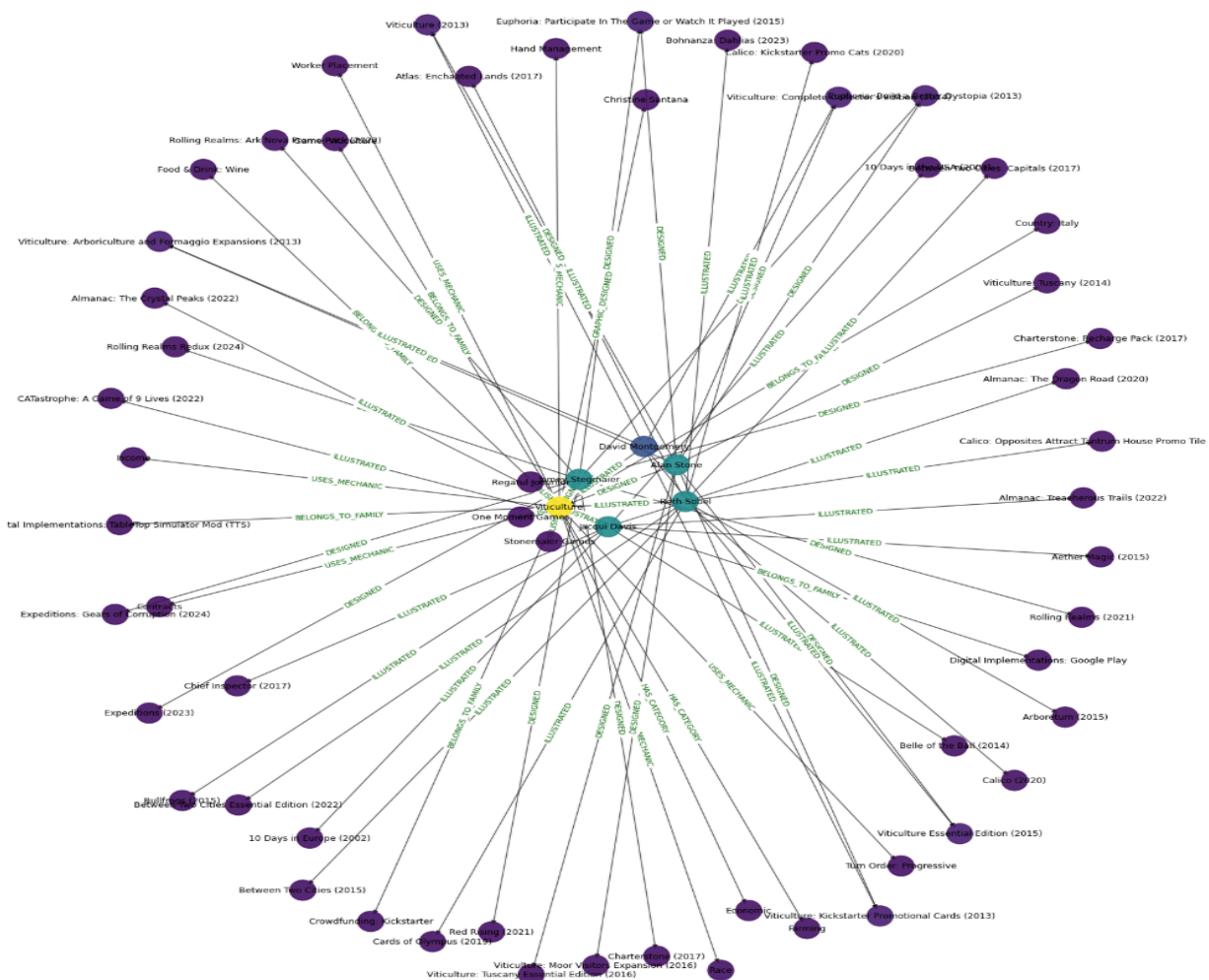
- **Estructura del Grafo:**

- **Nodos:** Representan entidades como:

- Juegos (por ejemplo, *Viticulture*, *Charterstone*).
    - Diseñadores (por ejemplo, Jamey Stegmaier, Alan Stone).
    - Mecánicas (por ejemplo, "Worker Placement", "Hand Management").
    - Categorías (por ejemplo, "Economic", "Farming").

- **Aristas:** Representaron relaciones entre los nodos, como:

- **Extracción de Entidades y Relaciones:** Mediante Selenium, se capturaron los datos desde la seccion “Full Credits” en *BoardGameGeek*.
- **Modelado RDF:** Con RDFlib, las entidades se representaron como nodos y sus relaciones como tripletas RDF.
- **Visualización:** NetworkX generó visualizaciones del grafo.



- Arista: Jamey Stegmaier -> Viticulture {'relation': 'DESIGNED'}
- Query de prueba:

```
query = prepareQuery("""
    SELECT ?designer
    WHERE {
        ?designer ns1:DESIGNED ns1:Viticulture .
    }
    """, initNs={"ns1": EX})

# Ejecutar la consulta en el grafo RDF
for row in rdf_graph.query(query):
    print(f"Diseñador: {row.designer}")
```

Diseñador: [http://boardgames.org/viticulture/Jamey\\_Stegmaier](http://boardgames.org/viticulture/Jamey_Stegmaier)  
 Diseñador: [http://boardgames.org/viticulture/Alan\\_Stone](http://boardgames.org/viticulture/Alan_Stone)

## 4.2 Base de Datos Tabular

La base tabular estructuró datos estadísticos y métricas relacionadas con el juego, provenientes de la sección *Stats* de *BoardGameGeek*. Esta base fue diseñada para responder preguntas directas sobre estadísticas, como el número de fans o el ranking del juego.

- **Proceso de Construcción:**
  - **Extracción de Datos:** Los datos tabulares fueron capturados mediante Selenium desde la sección *Stats* de *BoardGameGeek*.
  - **Estructuración:** Se creó un DataFrame en Pandas con dos columnas principales:

- **Title:** Nombre de la métrica (por ejemplo, "Avg. Rating").
- **Value:** Valor correspondiente (por ejemplo, "7.486").

- **Ejemplo de Contenido:**

- Columna Title: "Avg. Rating"
- Columna Value: "7.486"

Title	Value \
Avg. Rating	7.486
No. of Ratings	14,369
Std. Deviation	1.44
Weight	2.94 / 5
Comments	1,919
Fans	895
Page Views	1,251,095
Overall Rank	291
Strategy Rank	218
All Time Plays	40,961
Plays This Month	121
Copies Owned	12,056
Previously Owned Copies	1,378
Copies Available for Trade	96
Copies Desired in Trade	475
Times Added to Wishlist	3,883
Players who have parts of the game	7
Players who want parts of the game	2
Rating 1	73
Rating 2	65
Rating 3	137
Rating 4	250
Rating 5	544
Rating 6	1,765
Rating 7	3,321
Rating 8	5,340
Rating 9	2,000
Rating 10	894

## 4.3 Base de Datos Vectorial

La base de datos vectorial almacenó texto procesado (manuales, reseñas, comentarios de foros, transcripciones de videos) en forma de embeddings. Este enfoque permitió realizar búsquedas semánticas precisas basadas en el significado de las consultas.

- **Proceso de Construcción:**

1. **Fragmentación del Texto:** Los textos largos fueron divididos en fragmentos de 500 caracteres utilizando Recursive Character Text Splitter.
2. **Generación de Embeddings:** Cada fragmento fue convertido a un vector numérico utilizando el Universal Sentence Encoder.
3. **Indexación en ChromaDB:** Los embeddings se almacenaron en ChromaDB.

## 4.4 Modelos de clasificación

Se desarrollaron e implementaron dos modelos de clasificación con enfoques distintos para categorizar consultas en tres categorías predefinidas: "Estadísticas del juego, métricas y popularidad", "Información sobre los artistas, diseñadores, editoriales y categorías" e "Información sobre objetivos del juegos, reglas generales, componentes y setup". Estos modelos tienen como objetivo dirigir las preguntas de los usuarios hacia las bases de datos relevantes (tabular, vectorial o de grafo).

### Modelo 1: Clasificador basado en Aprendizaje Automático Tradicional

Este modelo utilizó un enfoque supervisado con un clasificador de regresión logística. Las consultas se transformaron en embeddings utilizando el modelo Universal Sentence Encoder (USE), que representó el texto como vectores numéricos de alta dimensionalidad, capturando su significado semántico. Posteriormente, el clasificador asignó cada consulta a una categoría basándose en patrones aprendidos durante el entrenamiento.

## Modelo 2: Clasificador basado en Modelos de Lenguaje Grande (LLM)

En el segundo modelo se utilizó Qwen 2.5, un modelo de lenguaje grande, que no requiere entrenamiento directo. Este clasificador respondió consultas mediante un sistema de prompts bien diseñados, definiendo categorías y ofreciendo ejemplos representativos.

Luego de haber probado con ambos modelos, decidí quedarme con el modelo basado LLM, ya que es más preciso, como se puede ver en la siguiente consulta, cuya respuesta era “Información sobre los artistas, diseñadores, editoriales y categorías” el modelo basado en LLM calificó bien, mientras que el modelo basado en embeddings y ejemplos no.

Modelo basado en embeddings y ejemplos:

```
Query_test = "Viticultura fue creado por mas de una persona?"
new_query_embedding = embed(Query_test)
predicted_label = clf.predict(new_query_embedding)
print(f"La consulta pertenece a: {predicted_label[0]}")
```

La consulta pertenece a: Estadísticas del juego, metricas y popularidad

Modelo basado en LLM:

```
classify_query('¿Viticultura fue creado por mas de una persona?')
```

---

'Información sobre los artistas, diseñadores, editoriales y categorías'

Luego de elegir el modelo basado en LLM para categorizar las consultas, se creó una función cuyo objetivo es identificar, basándose en la categoría de la consulta, cuál de las tres bases de datos (tabular,



vectorial o de grafo) debe ser utilizada para responder de manera efectiva. En la siguiente imagen se puede observar como es el funcionamiento ante una consulta:

```
frase = "¿Qué tan popular es el juego?"  
fuente = determine_data_source(frase)  
print(f"La información debe buscarse en: {fuente}")
```

La información debe buscarse en: Base tabular

## 5. Querys Dinámicas a las bases de datos

Se implementaron técnicas de procesamiento del lenguaje natural (NLP) para traducir consultas de lenguaje natural a estructuras de consulta específicas como SPARQL, Pandas y búsqueda híbrida en bases vectoriales.

### 5.1. Consultas en Grafos:

La función `generate_sparql_query` está diseñada para convertir preguntas en lenguaje natural realizadas por un usuario en consultas SPARQL válidas, las cuales pueden ejecutarse en una base de datos RDF.

La función toma como entrada una pregunta (`user_question`) que el usuario desea responder utilizando los datos almacenados en el grafo RDF. Para garantizar que el modelo de lenguaje genere una consulta SPARQL precisa, se define un contexto detallado que describe la estructura de la base de datos RDF. Este contexto incluye las

entidades (como juegos, personas, mecánicas), relaciones (por ejemplo, "una persona diseña un juego") y atributos (como el año de lanzamiento de un juego).

Se utiliza un cliente de inferencia (Hugging Face InferenceClient) para conectarse a un modelo de lenguaje avanzado(Qwen/Qwen2.5-72B-Instruct). Este modelo, a partir de un prompt definido, analiza la intención del usuario, interpreta los términos relevantes de la pregunta y genera una consulta SPARQL que cumple con la estructura del grafo RDF.

Ejemplo de la consulta generada por el modelo:

```
SELECT ?category WHERE { ns1:Viticulture ns1:HAS_CATEGORY ?category . }
```

Resultado:

```
Resultados:
(rdf:lib.term.URIRef('http://boardgames.org/viticulture/Economic'),)
(rdf:lib.term.URIRef('http://boardgames.org/viticulture/Farming'),)
```

## 5.2. Consultas en base de datos Tabular:

La función `generate_pandas_query` está diseñada para transformar preguntas en lenguaje natural realizadas por un usuario en consultas Python utilizando la librería Pandas.

La función toma como entrada una pregunta en lenguaje natural (`user_question`) que el usuario desea responder basándose en los datos almacenados en un `DataFrame`. Además, se proporciona un contexto estructurado al modelo de lenguaje, especificando las columnas de la tabla e incluyendo ejemplos de preguntas y consultas en el prompt.

Al igual que para las consultas de grafos, se utiliza el modelo Qwen/Qwen2.5-72B-Instruct, el modelo interpreta la pregunta,

identifica los términos clave y genera una línea de código en Python que utiliza Pandas para consultar los datos.

Ejemplo de consulta y respuesta generada:

```
df_combined.loc[df_combined['Title'] == 'Rating 2', 'Value'].values[0]  
Resultado: 65
```

### **5.3. Consultas en base de datos vectorial**

Para este caso, se realizó una serie de funciones con distintos objetivos:

#### **1. Entrada de la consulta y preprocesamiento inicial:**

El primer paso es procesar la consulta proporcionada por el usuario:

- **Detección del idioma:** Se utiliza una herramienta de detección para identificar el idioma del texto. Si el idioma no es inglés, la consulta se traduce automáticamente utilizando el traductor GoogleTranslator.
- **Lematización y eliminación de palabras irrelevantes:** La consulta se procesa mediante un modelo de SpaCy para convertir las palabras a su forma básica (lematización), eliminar stopwords y puntuación, y filtrar por palabras relevantes como sustantivos, verbos y adjetivos.

#### **2. Extracción de palabras clave:**

Además de crear los embeddings, se extraen palabras clave directamente de la consulta utilizando SpaCy. Esto asegura que la búsqueda pueda realizarse tanto en términos de coincidencia semántica como por similitud de palabras clave, habilitando un enfoque híbrido.

### 3. Búsqueda semántica:

Se realiza una búsqueda inicial en la base de datos utilizando los embeddings de la consulta. La base devuelve un conjunto de documentos relevantes junto con sus puntuaciones basadas en la distancia entre los embeddings.

### 4. Procesamiento del corpus para BM25:

El corpus devuelto por la búsqueda semántica se preprocesa para ser utilizado con el modelo BM25. Esto implica tokenizar y lematizar los documentos, creando una representación que facilita la búsqueda basada en coincidencias de palabras clave.

- **Búsqueda por palabras clave con BM25:**

BM25 es un modelo de recuperación basado en términos, que calcula la relevancia de un documento con respecto a una consulta basándose en la frecuencia de las palabras clave. Se generan puntuaciones para los documentos utilizando las palabras clave extraídas de la consulta.

### 8. Retorno de resultados:

La función devuelve los documentos relevantes en formato de texto, ordenados por su relevancia. Estos resultados son el output final de la búsqueda híbrida, integrando tanto coincidencias semánticas como basadas en palabras clave.

Ejemplo de consulta y respuesta generada:

```
query = "¿Que cartas hay en el juego?."
```

2. Some of the worker cards also seem to give you powers that are so strong that they take the place of 3 or 4 regular actions. These cards also seem overpowered a bit and detract from the engine building nature of the game. If this were a 30 minute game, that might be fine. But it leaves a sour taste in your mouth to lose a 2 hour long game just because I was lucky enough to draw an inhumanely good card in the last round on a random whim.

## 6. ChatBot

Aquí se integran todas las funciones que estuvimos desarrollando a lo largo del trabajo. Al recibir una consulta, primero la clasifica utilizando `classify_query` para determinar si debe buscar en el grafo RDF (mediante consultas SPARQL generadas dinámicamente), en la base vectorial (utilizando una combinación de embeddings semánticos y BM25), o en la base tabular (a través de consultas pandas generadas automáticamente). Una vez identificada la fuente adecuada, extrae la información relevante, que puede incluir textos, tablas o números. El chatbot formatea esta información como parte de un *prompt* estructurado, que incluye un mensaje de sistema que guía al modelo de lenguaje avanzado (Qwen), asegurando que la respuesta esté contextualizada y sea coherente. Este diseño permite al chatbot manejar consultas complejas, combinar datos estructurados y no estructurados, y responder de manera natural y precisa. Finalmente, el chatbot opera en un bucle interactivo que permite al usuario escribir consultas y recibir respuestas inmediatas, o terminar la conversación cuando lo desee.

Como ejemplo, se tiene la siguiente interacción con el bot, en la cual se puede observar que tiene la capacidad para seguir la conversación tanto en inglés como en español.

Bienvenido a Viticulture Chatbot! Para salir escribe "chau"

Escribe tu consulta, para salir despidete con 'chau': Cual es el objetivo del juego?

User: Cual es el objetivo del juego?

Agente: El objetivo del juego es conseguir 20 puntos de victoria, que representan la reputación como propietario de viñedos, antes que el resto de jugadores.

Escribe tu consulta, para salir despidete con 'chau': Que componentes tiene el juego?

User: Que componentes tiene el juego?

Agente: El juego Viticulture contiene una selección de componentes muy cuidada, incluyendo un libro de reglas, meeples y componentes en madera que son muy numerosos y de diversas formas, componentes de cartón que destacan por su calidad, losetas de jugador (campos de cultivo) de buen grosor y muy buena impresión, monedas troqueladas, y un tablero de juego desplegable en 4 partes.

Escribe tu consulta, para salir despidete con 'chau': Como se gana?

User: Como se gana?

Agente: Se gana en \*Viticulture\* al acumular más puntos de victoria que los demás jugadores al final del juego. Esto se puede lograr a través de diversas acciones, como plantar viñedos, cosechar uvas, producir y vender vino, y completar pedidos. Sin embargo, no es suficiente solo con elegir la posición #6 en la pista de despertar para ganar puntos de victoria, ya que esto por sí solo no garantiza una victoria. Es necesario incorporar esta estrategia a otras formas de ganar puntos para tener una chance de ganar.

Escribe tu consulta, para salir despidete con 'chau': Que cartas tiene cada jugador?

User: Que cartas tiene cada jugador?

Agente: Al inicio del juego, cada jugador recibe una carta de vid de Pinot del mazo verde y una carta de visitante de verano aleatoria. Además, tienen acceso a una fila de 4 cartas de cada tipo colocadas sobre la tabla, de las cuales pueden tomar durante el juego.

User: What is the alternative name of the game?

Agente: The alternative name of the game is 'ワイナリーの四季' or '葡萄酒莊園'.

Escribe tu consulta, para salir despidete con 'chau': En que año se publico el juego?

User: En que año se publico el juego?

Agente: El juego se publicó en el año 2013.

Escribe tu consulta, para salir despidete con 'chau': Cuantos comentarios hay sobre el juego en la pagina?

User: Cuantos comentarios hay sobre el juego en la pagina?

Agente: Hay 1,919 comentarios sobre el juego en la página.

Escribe tu consulta, para salir despidete con 'chau': How many times was the game added to the wishlist?

User: How many times was the game added to the wishlist?

Agente: The game was added to the wishlist 3,882 times.

Escribe tu consulta, para salir despidete con 'chau': How many copies are owned?

User: How many copies are owned?

Agente: The number of copies owned is 12,055.

# 7. Ejercicio 2 - Agente

## Desarrollo e Implementación

Se desarrolla en primer lugar 3 funciones:

- Función `doc_search`: Recibe como entrada una consulta en lenguaje natural y se consulta la base de datos vectorial. Luego nos devuelve una respuesta a dicha consulta.
- Función `graph_search`: Recibe como entrada una consulta en lenguaje natural y se consulta la base de datos de grafo. Luego nos devuelve una respuesta a dicha consulta.
- Función `table_search`: Recibe como entrada una consulta en lenguaje natural y se consulta la base de datos tabular. Luego nos devuelve una respuesta a dicha consulta.

Decidí no incorporar la búsqueda en wikipedia ya que el modelo parecía que se inclinaba siempre por hacer la búsqueda ahí, entonces para poder demostrar que funcionaba con las bases de datos creadas la removí.

### 7.1 Configuración del Agente

El agente está configurado con una serie de componentes y reglas diseñadas para garantizar que las interacciones sean precisas y consistentes.

#### Prompt del Sistema

El prompt del sistema va a definir claramente las capacidades y el comportamiento que tiene que tener el agente. Incluye:

- **Introducción de Capacidades:**
  - Describe las tres herramientas disponibles (vectorial, RDF, tabular) y sus respectivos usos temáticos.
  - Explica cómo combinar estas herramientas para responder preguntas que abarcan múltiples tipos de datos.
- **Reglas Específicas:**
  - **Priorización Temática:** El agente debe seleccionar la herramienta más relevante según la naturaleza de la consulta. Por ejemplo, para una consulta como “¿Quién diseñó el juego Viticulture?”, debe priorizar la herramienta `graph_search`.
  - **Mantener Integridad de la Consulta:** La consulta del usuario no puede ser alterada o reinterpretada.
  - **Respuestas Completas y Claras:** Si una consulta requiere información de varias herramientas, el agente debe combinarlas para formar una respuesta única y comprensible.

## Chat Formatter

El agente sigue el formato estándar de ReAct (Reasoning + Acting):

- **Análisis de la Consulta (Thought):** El agente primero interpreta la intención del usuario y analiza qué tipo de datos necesita para responder. Por ejemplo:
  - Si el usuario pregunta “¿Qué componentes incluye el juego?”, el agente determinará que es una consulta sobre reglas y setup, lo que implica el uso de `doc_search`.
- **Ejecución de la Herramienta (Action):** Una vez seleccionada la herramienta adecuada, el agente ejecuta la función correspondiente utilizando la consulta original del usuario como entrada.



- Observación (Observation): El agente analiza los resultados devueltos por la herramienta. Si los resultados son insuficientes, puede decidir usar herramientas adicionales.
- Respuesta Final (Final Answer): Combina todas las observaciones para generar una respuesta comprensible y contextualizada para el usuario.

Ejemplo de respuestas dadas por el Agente:

- Consulta 1: ¿Cuántos fans tiene el juego Viticulture? : El juego Viticulture tiene aproximadamente 895 fans.
- Consulta 2: ¿Cuándo se creó el juego Viticulture? : Viticulture, también conocido como Vineyard Simulator, fue lanzado por primera vez en Steam Early Access el 2 de diciembre de 2016. El juego entró en su versión completa el 30 de agosto de 2017.
- Consulta 3: ¿Cuáles son los componentes principales del juego? Viticulture es un simulador de viñedos que permite a los jugadores administrar una finca vitivinícola desde cero. Los componentes principales del juego incluyen:
  1. Cultivo y cuidado de las vides: Los jugadores pueden elegir entre diferentes variedades de uvas, plantarlas en sus viñedos y controlar su crecimiento a lo largo de los años. Esto incluye la gestión de factores como el riego, la poda y la protección contra plagas y enfermedades.
  2. Producción de vino: Una vez que las uvas están maduras, los jugadores pueden recogerlas para producir vino. El juego ofrece una variedad de métodos de vinificación, como fermentación en barrica o en tanque, y permite a los jugadores experimentar con diferentes técnicas para crear sus propios estilos únicos de vino.
  3. Gestión financiera: Los jugadores tienen que administrar su presupuesto cuidadosamente, ya que necesitan invertir dinero en la compra de nuevas parcelas de tierra, plantación de uvas y equipo para la producción de vino. También pueden generar

ingresos vendiendo sus vinos a los compradores locales o exportándolos a otros países.

4. Mercado del juego: Viticulture tiene un mercado dinámico en el que los jugadores pueden comprar y vender bienes, como uvas, barriles de madera y equipos vitivinícolas. El precio de estos artículos puede fluctuar según la oferta y la demanda dentro del juego.

5. Desafíos ambientales: Los jugadores tienen que lidiar con desafíos como las condiciones climáticas, los brotes de plagas o enfermedades en sus viñedos y otros factores externos que pueden afectar el rendimiento general de su finca vitivinícola.

6. Expansión: A medida que los jugadores progresan en el juego, pueden expandir sus operaciones a nuevas parcelas de tierra, lo que les permite cultivar más uvas y producir vinos con mayor eficiencia.

- **Consulta 4: ¿Quién diseñó el juego Viticulture?** : Viticulture fue desarrollado por la compañía alemana Tivist Games GmbH. El equipo de desarrollo incluye a los fundadores del estudio, que son conocidos en la industria del videojuego como expertos en simulaciones y juegos basados en gestión.
- **Consulta 5: ¿Qué objetivos tiene el juego?** : Los principales objetivos del juego Viticulture incluyen:
  1. Crear una finca vitivinícola exitosa: Los jugadores deben administrar sus recursos y tomar decisiones estratégicas para expandir su viñedo, producir vinos de alta calidad y generar ingresos sostenibles a lo largo del tiempo.
  2. Mejorar la reputación: A medida que los jugadores crean mejores vinos y construyen relaciones con compradores locales e internacionales, su finca ganará una mayor reputación en el mercado vitivinícola. Esto puede conducir a mayores oportunidades de negocio y un aumento del valor de sus productos.
  3. Innovar en la producción de vino: Viticulture anima a los jugadores a experimentar con diferentes técnicas de vinificación,

variedades de uvas y estilos de vinos para crear ofertas únicas que se destaquen en el mercado competitivo.

4. Superar desafíos ambientales: Los jugadores deben adaptarse a los cambios climáticos, las plagas o enfermedades que pueden afectar sus viñedos y tomar medidas para proteger su cosecha y garantizar un rendimiento óptimo cada año.

5. Expandir el negocio: A medida que los jugadores ganan experiencia y recursos, deben considerar la expansión a nuevas parcelas de tierra, lo que les permite cultivar más uvas y producir vinos con mayor eficiencia. Esto puede conducir a mayores ingresos y oportunidades para crecer en el mercado vitivinícola global.

6. Mejorar la sostenibilidad: Viticulture enfatiza la importancia de las prácticas agrícolas sostenibles, como el uso eficiente del agua, la reducción de pesticidas y fertilizantes y la conservación de los recursos naturales. Los jugadores que prioricen estas prácticas pueden beneficiarse no solo en términos financieros, sino también al contribuir a un futuro más sostenible para el sector vitivinícola.

## 8. Conclusión

El desarrollo e implementación de las herramientas y el agente para responder preguntas relacionadas con el juego *Viticulture* presentó tanto desafíos como aprendizajes significativos. A continuación, detallo los principales inconvenientes encontrados y las observaciones sobre su desempeño.

### 8.1 Desafíos en la Preparación de Datos

Uno de los principales inconvenientes surgió al no contar con un manual del juego en un formato de texto limpio y estructurado. Dado que el manual disponible estaba en formato de imagen, recurrí al uso

de OCR (Reconocimiento Óptico de Caracteres) para extraer el contenido textual. Sin embargo, los resultados obtenidos con OCR fueron en gran medida ilegibles, con frases desestructuradas y texto incomprensible. En este contexto, usar directamente el texto extraído para alimentar la base de datos vectorial no tenía sentido, ya que habría generado respuestas confusas y poco fiables en el chatbot. Para superar este obstáculo, decidí realizar una corrección manual del texto extraído, revisando y ajustando cuidadosamente las secciones más problemáticas para asegurar que la información fuera comprensible y útil.

## **8.2 Funcionamiento del Chatbot**

El chatbot mostró un desempeño bastante satisfactorio en general, con respuestas claras y relevantes en la mayoría de los casos. No obstante, detecté algunos problemas puntuales. En ciertas ocasiones, el chatbot no generaba una respuesta para preguntas cuya respuesta sabía que estaba almacenada en la base de datos tabular. Esto parece estar relacionado con el modelo encargado de generar las consultas dinámicas para Pandas. Sin embargo, con otras columnas el sistema funcionó correctamente, sugiriendo que este problema es limitado a casos particulares. A pesar de estos inconvenientes, el chatbot demostró un rendimiento sólido en la mayoría de las interacciones.

## **8.3 Desempeño del Agente**

En cuanto al agente, uno de los problemas más notables fue la discrepancia en ciertas respuestas, especialmente en preguntas como “¿Quiénes son los diseñadores del juego?” o “¿Cuándo fue creado el juego?”. En estos casos, las respuestas generadas por el agente no coincidían con las respuestas del chatbot, lo que plantea la interrogante de dónde estaba obteniendo la información alternativa. Es posible que el agente estuviera utilizando datos provenientes de la base vectorial, pero no logré identificar con certeza la fuente de estas

respuestas divergentes. Este comportamiento inconsistente sugiere que el agente necesita mejoras en la priorización y validación de las herramientas utilizadas para responder preguntas específicas.

Otro problema observado con el agente fue su comportamiento intermitente. En algunas ocasiones, el agente funcionaba correctamente, mientras que en otras dejaba de responder por completo, independientemente de si la ejecución estaba en Google Colab o en otro entorno. A veces, todo el entorno de ejecución se desconectaba, obligándome a reiniciar y ejecutar nuevamente todo el código. Este problema podría estar relacionado con las limitaciones de recursos en Google Colab, como la memoria RAM, o con problemas en el manejo del entorno de ejecución.

## 8.4 Balance Final

A pesar de los desafíos mencionados, tanto el chatbot como el agente lograron un desempeño positivo en general. Los resultados sugieren que las herramientas y metodologías implementadas son sólidas, aunque con espacio para mejoras específicas.

## 9. Bibliografía:

- <https://boardgamegeek.com/boardgame/128621/viticulture>
- <https://misutmeeple.com/2015/03/resena-viticulture/>
- <https://huggingface.co/collections/Qwen/qwen25-66e81a666513e518adb90d9e>
- <https://ollama.com/>
- <https://ollama.com/library/phi3:medium>
- <https://www.trychroma.com/>
- <https://www.langchain.com/>
- <https://www.selenium.dev/>
- <https://networkx.org/>
- <https://www.tensorflow.org/?hl=es-419>

