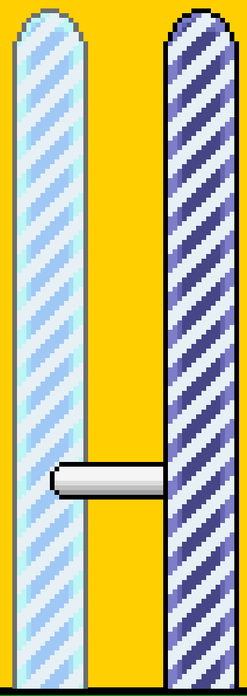
SUPER MARIO

GRAPH DB

DATA MANAGEMENT

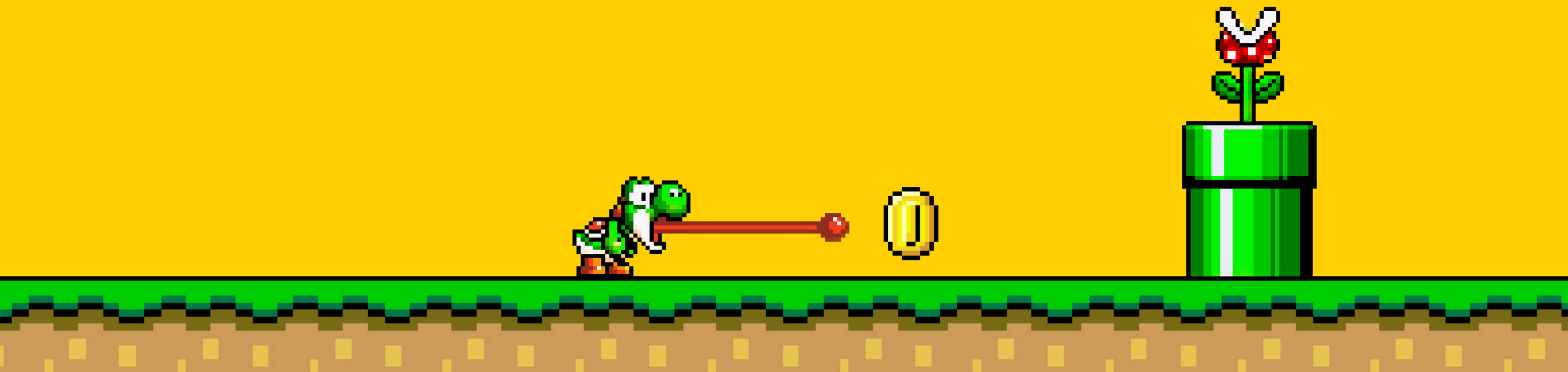
A.A.
2024/25

VINCENZO SIANO
FRANCESCO RENNA
SELIN DÖKMEN



INTRODUCTION

This project creates a visual network of relationships in the Mario Universe by building a graph database.
It links characters, bosses, enemies across many videogames.
Data was gathered through web scraping, API calls, and a Kaggle dataset, then cleaned, integrated, and stored in Neo4j for analysis.
The goal is to obtain a sort of "social net" of generic characters, enemies and bosses of the Super Mario universe, which are linked to the respective videogames in which they appear, or even present their respective interpersonal relationships.



III

WEB SCRAPING

Web scraping on Mario Wiki focused on main characters, enemies, and bosses.
Character pages provided names, roles, and game appearances.
Separate scrapers collected enemy and boss details. All data was saved in structured CSV formats using Python libraries for scalable extraction.

API

The Giant Bomb API was used to collect character data and relationships. After secure API key configuration, the Mario franchise and characters were queried.
Data about friends and enemies was extracted, processed in batches to respect rate limits, and compiled into DataFrames for incremental CSV storage.

KAGGLE

The Video_Games.csv dataset from Kaggle was integrated to enrich Mario game data with sales figures.
This dataset added an economic dimension to the project, complementing character and game details obtained from web and API scraping.
Sales data was matched and merged for comprehensive analysis.

DATA CLEANING DATASETS

This process focused on ensuring the **reliability** of the scraped and external data. We adopted several key strategies:

- Robust Scraping Logic: Adapted to inconsistent HTML structures across wiki pages.
- Filtering Irrelevant Entries: Removed non-informative or administrative links before scraping.
- Deduplication: Tracked URLs and entries to avoid duplicates.
- Table Restructuring: Standardized column formats and unified headers (e.g., merging “System,” “Console,” and “Format”).
- Text Normalization: Lowercased text, removed special characters, and standardized console names via a mapping dictionary.
- Handling Missing Values: Dropped rows missing critical fields (e.g., character name, game title).

DATA CLEANING RELATIONSHIPS

IV

This aspect focused on refining the **connections between entities** (character-to-game, enemy-to-boss):

- Link Validation: Ensured that only valid and relevant entity relationships were established.
- Structural Consistency: Normalized relationship formats to maintain uniformity across different entity types.
- Duplicate Relationship Removal: Identified and removed redundant connections between the same entities.
- Missing Link Handling: Flagged or excluded relationships where one or both entities lacked sufficient identifying information.
- Semantic Alignment: Used keyword-based matching to align entities across datasets when exact matches were unavailable, improving relationship accuracy.

DATA QUALITY ASSESSMENT

Ensuring high data quality was a critical step, we focused on:

1.COMPLETENESS

- Overall completeness is high at the table level (lowest: Bosses at 82%).
- Column-level analysis shows:
  - Species: 94% complete for characters, only 27% for bosses (due to missing info on Mario Wiki).
  - Year and Console: 100% complete.
- After integration:
  - Scraped data: 88%.
  - API data: 80%.
- Also assessed for sales and species across integrated datasets.

2.REDUNDANCY

- URL tracking and unique IDs were used to prevent duplication during scraping and integration.
- Duplicates were detected using key fields (e.g., name, title) and removed before loading into Neo4j.
- Redundant rows found:
  - Characters: 176
  - Enemies: 42
  - Bosses: 10
- Final merged data (scraped + API): 0 duplicates after integration.

3.CONSISTENCY

- Normalization (lowercasing, punctuation removal, trimming) for consistent naming.
- Standardization of console names using a mapping dictionary.
- Character names aligned before merging with API data.
- Console column in games_df had 73 unique names → Reduced to 27 after integration (63% reduction).

4.ACCURACY & CURRENCY

- Release years and sales were verified by cross-checking multiple sources.
- Acquisition scripts are up-to-date, but source data (Mario Wiki & Giant Bomb API) may change.
- Temporal accuracy is strong, enabling reliable time-based analysis.
- Currency is supported by the distribution of release years, highlighting key periods like the Wii & DS era, and the more recent Switch era.

VI

SCHEMA TRANSFORMATION

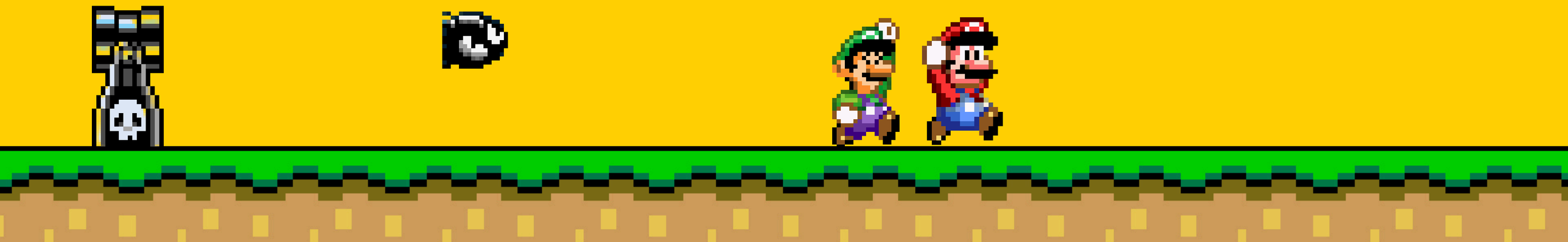
To prepare the data for integration into the graph database, all datasets—characters, enemies, bosses, and games—were transformed into a uniform tabular structure. This involved renaming columns to shared identifiers (e.g., using "Figure" for all entity types), normalizing console names, and merging sales and species data. The result was a single, clean dataset that could be easily mapped to the graph schema.

SCHEMA INTEGRATION

Once the data was cleaned and aligned, it was mapped to the Neo4j graph schema. Nodes were created for characters and games, and relationships were formed based on roles (e.g., CHARACTER_IN, ENEMY_WITH). A dual-schema approach was used: one for character-game appearances and another for inter-character relationships. This ensured clarity and referential integrity while preserving the richness of the Mario universe.

CORRISPONDENCES INVESTIGATION

Before merging datasets from different sources, we conducted a detailed investigation to identify matching entities across them. This required standardizing names by lowercasing, removing special characters, and applying manual mappings where necessary. This step was crucial to ensure that characters and games from the API aligned correctly with those from the scraped and Kaggle data.



NODES STRUCTURE

The DB contains two primary types of nodes, each with specific properties and constraints:

1. Character Nodes

- Represent all characters in the Mario universe, including:
  - Main characters (e.g., Mario, Luigi)
  - Enemies (e.g., Goomba, Koopa Troopa)
  - Bosses (e.g., Bowser, King Boo)
- Properties:
  - name: Character's name (e.g., "Mario")
  - species: Biological or fictional classification (e.g., "Human", "Koopa")

2. Game Nodes

- Represent individual games across various platforms.
- Properties:
  - name: Game title (e.g., "Super Mario Bros.")
  - console: Platform or system (e.g., "NES", "Switch")
  - year: Release year
- sales: Sales figures (numeric, defaulted to 0 if missing)

RELATIONSHIPS STRUCTURE

VII

The DB uses directed, semantically meaningful relationships to model interactions and appearances:

1. Appearance-Based Relationships based on their role:

- CHARACTER_IN: General appearance
- ENEMY_IN: Appears as an enemy
- BOSS_IN: Appears as a boss

2. Interpersonal Relationships from the GiantBomb API:

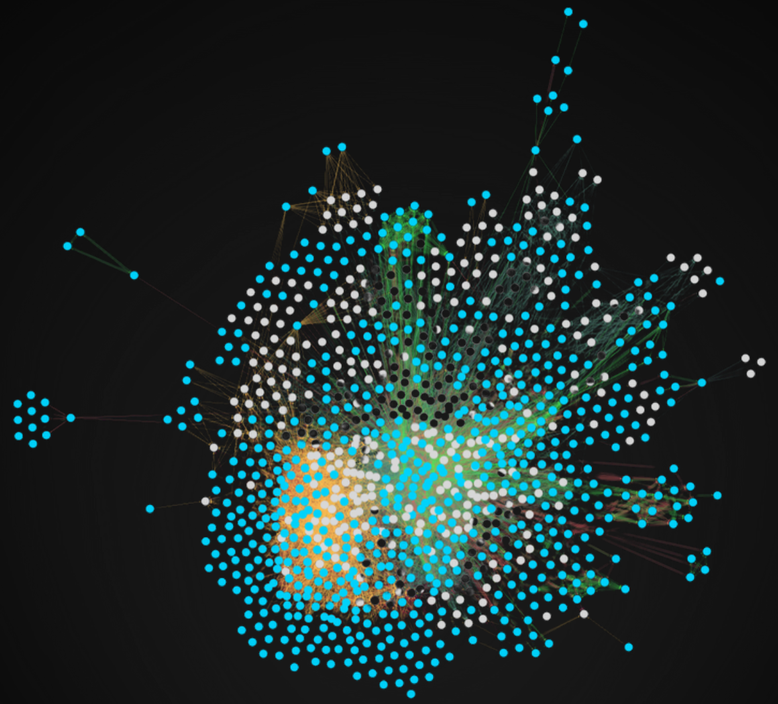
- FRIEND_WITH: Indicates a friendly relationship
- ENEMY_WITH: Indicates a rivalry relationship

Relationship Features

- All relationships are:
  - Directed (e.g., Mario → FRIEND_WITH → Luigi)
  - Typed (to distinguish role and semantics)
- Created dynamically using Cypher and the APOC library, based on the Relation column in the dataset.

DB STATISTICS

VIII



NODES
The final Neo4j graph database contains a total of 1055 nodes.

Divided in:

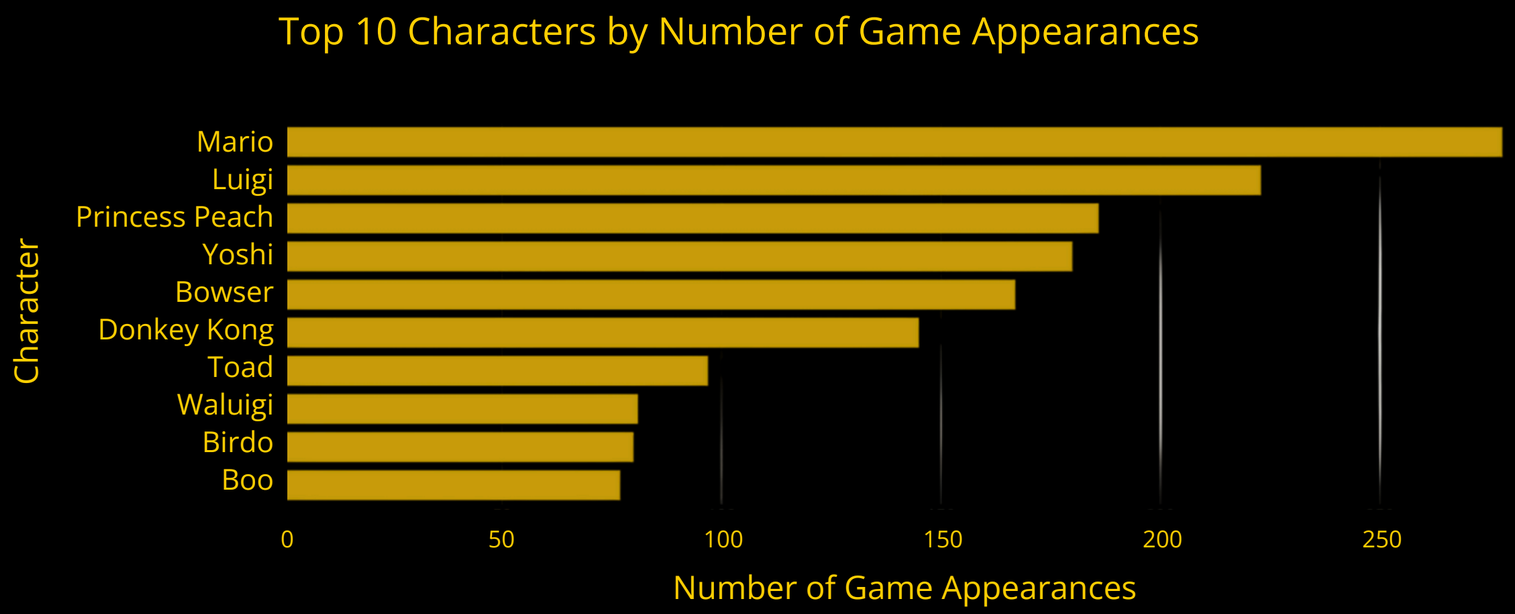
- CHARACTERS: 664
- GAMES: 391

RELATIONSHIPS
Total relationships: 9482

- CHARACTER_IN: 3324
- ENEMY_IN: 2438
- BOSS_IN: 1196
- FRIEND_WITH: 1456
- ENEMY_WITH: 1068

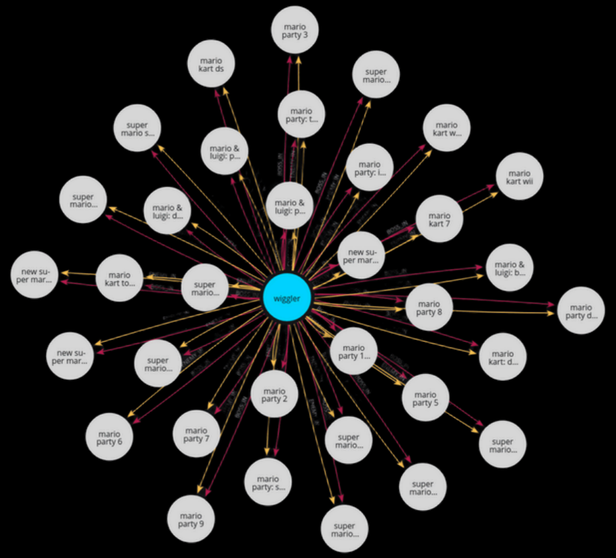
DB STATISTICS

IX



QUERY

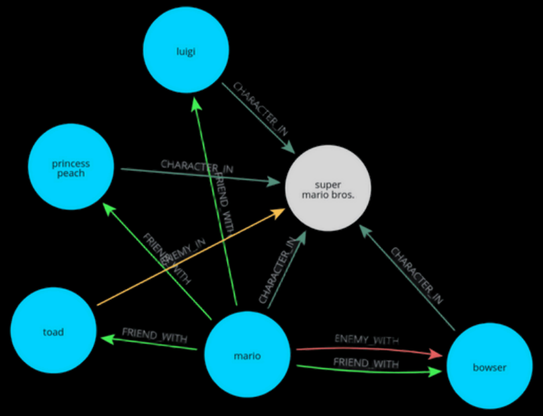
```
MATCH (c:Character)-[]->(g:Game)
RETURN c.name AS Character,
COUNT(DISTINCT g)
AS Appearances
ORDER BY Appearances DESC
LIMIT 10
```



QUERY 1

```
MATCH p = (c:Character {name: 'wiggler'})
-[r]->(g:Game)
RETURN p
```

Graph output of the query (i.e., Wiggler's Game Relationships).



QUERY 2

```
MATCH (c1:Character)-[r:FRIEND_WITH|
ENEMY_WITH]->(c2:Character)-[a1]->(g: Game)<-[a2]-
(c1)
WHERE c1.name = "mario" AND g.name = "super
mario bros." AND g.console = "NES"
RETURN c1,r,c2,a1,g,a2
```

Graph output of the query (i.e., friends or enemies of Mario that appear in a specific game and console).

THE END

VINCENZO SIANO 934168
FRANCESCO RENNA 925031
SELIN DÖKMEN 925034

XI