

# A MOBA Game Ontology and Knowledge-Enhanced Question Answering System for League of Legends

CMPE 58H – Social Semantic Web

Technical Report

Atahan Uz, Fatma Gizem Yılmaz

Department of Computer Engineering

Boğaziçi University

`{atahan.uz, gizem.yilmaz1}@std.bogazici.edu.tr`

December 20, 2025

## **Abstract**

MOBA games like League of Legends and DOTA 2 contain a lot of interconnected information: hero stats, abilities, items, counter-picks, synergies, and team compositions. The problem is that this data sits in different places, stored in different formats, with no shared vocabulary linking it together. There's no easy way to query it systematically or compare strategies across games.

This project builds a game-agnostic MOBA ontology using OWL 2. Instead of tying the ontology to one specific game, it uses generic terms so the same structure can work for different MOBA titles. The ontology covers heroes, skills, items, matches, and strategic relationships. It has 139 classes, 88 object properties, and 111 data properties. The ontology was populated with about 50,000 data points from League of Legends. A chatbot interface lets users ask questions in plain language, which get translated into SPARQL queries behind the scenes. There's also a snapshot feature where users can upload their current match state and get suggestions.

Testing showed that responses were 55% better when the system used ontology-derived context compared to relying on the language model alone. Human judges saw a 78% improvement in accuracy specifically. Overall, the results demonstrate that a helpful chat assistant for MOBA games has been successfully implemented.

# Contents

<b>1</b>	<b>Introduction and Motivation</b>	<b>6</b>
1.1	Background . . . . .	6
1.2	Motivation . . . . .	6
1.3	Contributions . . . . .	7
<b>2</b>	<b>Related Work</b>	<b>8</b>
2.1	General Game Ontology Frameworks . . . . .	8
2.2	MOBA-Specific Approaches . . . . .	9
2.3	Identified Gaps and Project Positioning . . . . .	9
<b>3</b>	<b>System Overview</b>	<b>11</b>
3.1	Problem Definition . . . . .	11
3.2	System Architecture . . . . .	11
3.2.1	Data Sources . . . . .	12
3.2.2	Ontology Layer . . . . .	13
3.2.3	Mapping and Population . . . . .	13
3.2.4	Query Interface . . . . .	13
3.2.5	Component Interaction . . . . .	14
<b>4</b>	<b>Modeling and Ontology Design</b>	<b>15</b>
4.1	Modeling Approach . . . . .	15
4.2	Ontology Scope and Design Principles . . . . .	16
4.2.1	Scope Definition . . . . .	16
4.2.2	Design Principles . . . . .	16
4.3	Classes . . . . .	16
4.3.1	Hero Class Hierarchy . . . . .	17
4.3.2	Skill Class Hierarchy . . . . .	18
4.3.3	Item Class Hierarchy . . . . .	18
4.3.4	Crowd Control Hierarchy . . . . .	18
4.4	Properties . . . . .	18
4.4.1	Object Properties . . . . .	19
4.4.2	Data Properties . . . . .	20
4.5	Special Individuals . . . . .	21
4.5.1	Activation Types . . . . .	21

4.5.2	Lanes . . . . .	21
4.5.3	Damage Types . . . . .	21
4.5.4	Attack Types . . . . .	22
4.5.5	Complexity Levels . . . . .	22
4.5.6	Item Types . . . . .	22
4.5.7	Resource Types . . . . .	22
4.5.8	Effect-Related Individuals . . . . .	22
4.5.9	Area of Effect Shapes . . . . .	22
4.5.10	Game Phases and Team Compositions . . . . .	22
4.5.11	Role Individuals . . . . .	22
4.6	Rules and Constraints . . . . .	22
4.6.1	Disjointness . . . . .	23
4.6.2	Functional Properties . . . . .	23
4.6.3	Inverse Properties . . . . .	23
4.6.4	Symmetric Properties . . . . .	23
4.6.5	Cardinality Restrictions . . . . .	23
4.6.6	Value Restrictions . . . . .	24
4.6.7	SWRL Rules . . . . .	24
4.6.8	What the Constraints Enable . . . . .	24
4.7	Summary . . . . .	25
<b>5</b>	<b>Usage and Predefined Queries</b>	<b>26</b>
5.1	Running the System . . . . .	26
5.2	Predefined Queries . . . . .	26
<b>6</b>	<b>Evaluation</b>	<b>30</b>
6.1	Evaluation Methodology . . . . .	30
6.1.1	Experimental Conditions . . . . .	30
6.1.2	Evaluation Criteria . . . . .	30
6.2	Results . . . . .	30
6.3	Analysis . . . . .	31
<b>7</b>	<b>Discussion and Conclusion</b>	<b>32</b>
7.1	Discussion . . . . .	32
7.2	Conclusion . . . . .	32
7.3	Future Work . . . . .	33
<b>8</b>	<b>Resources and Ethics Declaration</b>	<b>34</b>
8.1	External Resources . . . . .	34
8.1.1	Data Sources . . . . .	34
8.1.2	Libraries and Tools . . . . .	34
8.2	AI Tools and Prompt Usage . . . . .	34
8.3	Academic Integrity Statement . . . . .	35

<b>A</b>	<b>Appendix</b>	<b>36</b>
A.1	LLM Judge Prompt . . . . .	36
A.2	Benchmark Questions . . . . .	37

# List of Figures

3.1	Overview of system architecture . . . . .	12
-----	---	----

# List of Tables

4.1	Core Class Categories in the MOBA Game Ontology . . . . .	16
4.2	Hero-Related Object Properties . . . . .	19
6.1	Evaluation results comparing system performance with and without ontology access. . . . .	31

# Chapter 1

## Introduction and Motivation

### 1.1 Background

MOBA games (Multiplayer Online Battle Arenas) have grown into one of the most popular competitive gaming genres. League of Legends alone has over a hundred million monthly players, and DOTA 2 tournaments offer prize pools worth tens of millions of dollars. These aren't niche titles; they've become a significant part of global gaming culture.

What makes MOBAs particularly interesting from a knowledge representation standpoint is the sheer volume of interconnected information they involve. A single game like League of Legends has over 160 playable characters, each with four unique abilities that scale differently as players level up. There are hundreds of items, many of which combine from simpler components into more powerful versions. Characters fall into roles such as tank, mage, assassin, or support, though many can flex between multiple roles depending on the situation. On top of all this sits the strategic layer: which characters counter which, who synergizes well with whom, and what team compositions tend to succeed.

For players trying to get better, this complexity creates real difficulties. Finding out who counters a particular hero usually means digging through community forums or watching tutorial videos. Looking for optimal item builds requires checking multiple statistics websites that may offer conflicting recommendations. Understanding why certain hero pairs work well together isn't really documented systematically anywhere; it's mostly knowledge passed around within the community.

The data does exist. It's just scattered. Hero statistics sit in one database, match histories in another, item information somewhere else entirely. Nothing links together in a machine-readable way. There's no straightforward method to ask something like "show me tanky heroes with good crowd control who work well with aggressive supports" and get a structured answer.

### 1.2 Motivation

This project addresses a gap between two existing approaches to handling MOBA game knowledge. On one side, general game ontologies, such as the Game Ontology Project, describe games at an abstract level. They're useful for academic analysis but don't capture MOBA-specific concepts. Terms like "lane," "jungle camp," or "last-hitting" simply don't appear in genre-agnostic



frameworks.

On the other side, game-specific tools track detailed statistics for individual titles. Platforms dedicated to League of Legends offer champion win rates, matchup data, and build recommendations, but their data structures are proprietary and completely tied to that one game. There’s no way to compare strategies across different MOBAs or reuse the knowledge elsewhere.

What’s missing is something in between: an ontology that captures the concepts MOBAs share while staying abstract enough to work across different titles. Players who switch from League of Legends to DOTA 2 already understand the core ideas (laning, team fights, objectives) even if the specific heroes and items differ. A well-designed ontology should reflect that same kind of transferability.

There’s also practical value to consider. Esports analysts could compare strategies across games. Researchers studying competitive gaming could work with structured data rather than scraping wikis manually. New players could receive recommendations tailored to their skill level and preferred playstyle. All of this becomes easier when the underlying knowledge is organized properly.

## 1.3 Contributions

This project makes several contributions:

- **A game-agnostic MOBA ontology** built using OWL 2 and stored in RDF/XML format. The ontology uses generic terms like “Hero” rather than game-specific vocabulary like “Champion,” making it applicable across different MOBA titles. It covers heroes, skills, items, matches, and strategic relationships.
- **Rich semantic relationships** between entities. Counter-picks, synergies, lane matchups, and item build paths are modeled as proper OWL properties. Symmetric relationships capture mutual synergy; inverse relationships allow the system to derive that if Hero A counters Hero B, then Hero B is countered by Hero A.
- **Reasoning capabilities** through OWL restrictions and SWRL rules. Heroes meeting certain statistical thresholds, such as high base health, get automatically classified as tanks. More complex inferences, like determining which abilities counter which crowd control effects, are handled through SWRL rules.
- **Support for practical queries** that address what players actually want to know. The ontology is structured so that questions like “find heroes with crowd control who are strong early” or “recommend items against a magic-heavy team” can be expressed naturally in SPARQL.
- **A populated knowledge base** using League of Legends data. Real champion statistics, ability information, and item properties demonstrate that the abstract ontology works when instantiated with actual game data.

## Chapter 2

# Related Work

Games have attracted considerable research interest in recent years, with scholars trying to find systematic ways to describe what makes games work. This chapter looks at prior efforts to build formal frameworks for understanding games, paying special attention to work that’s relevant for designing a MOBA Game Ontology that isn’t tied to any single game.

### 2.1 General Game Ontology Frameworks

The Game Ontology Project (GOP) by Zagal, Mateas, Fernandez-Vara, Hochhalter, and Lichti [16] stands out as one of the first serious attempts at creating a comprehensive game description framework. Their ontology has over 150 elements arranged hierarchically under five main categories: Interface, Rules, Goals, Entities, and Entity Manipulation. Rather than using rigid definitions, the GOP relies on prototype theory, elements get categorized based on how similar they are to typical examples. This turns out to be quite practical since games rarely fit perfectly into neat boxes.

What’s notable about the GOP is its bottom-up methodology. The researchers built their framework by analyzing actual games instead of starting from abstract theory. This grounded approach helped them notice subtle differences that might otherwise be missed. Take spatiality, for instance: they distinguish between represented space (the visuals), gameworld space (where things actually happen), and gameplay space (where players can act). For MOBAs, this kind of distinction matters, the map you see isn’t quite the same as the strategic territory that affects winning or losing.

That said, the GOP was meant to cover all videogames. It doesn’t get into genre-specific details, and the framework itself isn’t machine-readable. It can’t be queried by SPARQL queries.

Debus [4] tackled a different problem with the Unifying Game Ontology (UGO). His observation was that researchers kept proposing different classification schemes that didn’t really talk to each other. The UGO tries to fix this by identifying common dimensions, which he calls facets, that cut across different models: mechanics, space, time, entities, goals, and randomness. Each facet gives you a different lens for looking at games.

The UGO’s main value is in how carefully it compares existing approaches. Different scholars have categorized game goals in incompatible ways. Having a unified vocabulary helps avoid

confusion. This idea of shared terminology resonates with semantic web thinking, where interoperability depends on everyone using the same concepts.

Zagal and Mateas [15] also contributed a detailed look at time in games. They propose four temporal frames: real-world time, gameworld time, coordination time, and fictive time. These frames interact in interesting ways. In MOBAs, time is everywhere, cooldowns, respawn timers, match phases. Understanding how these different temporal layers work together is pretty important for modeling the domain.

## 2.2 MOBA-Specific Approaches

General frameworks offer useful foundations but don't really capture what makes MOBAs distinctive. The most directly related prior work is a DOTA 2 ontology built on the Basic Formal Ontology (BFO) [1]. This one focuses entirely on a single game, organizing knowledge around four entity types: items, heroes, creeps, and structures.

Some of the design choices in the DOTA 2 ontology are worth mentioning. Structures get classified as independent continuants rather than spatial regions—the reasoning being that whether a tower exists matters more than where it sits on the map (since locations don't change between matches anyway). The ontology also separates character states like “pushing” a lane from game states like victory or defeat. These decisions show the kind of domain thinking that goes into building a proper game ontology.

The work validates its design using what's called the “true path” rule, checking that parent-child relationships stay consistent throughout. That kind of logical rigor is necessary if you want an ontology that actually works with reasoners or SPARQL endpoints.

But the DOTA 2 ontology only works for DOTA 2. Everything is specific to that game class names like “Dire\_Base” or “Ring\_of\_Health” don't mean anything outside of it. Trying to use this ontology for League of Legends or Smite would require basically starting over.

There's also the League of Legends Match Data Compactor (LoL-MDC) [2], which takes a completely different approach. Instead of building an ontology, it compresses match data into summaries that large language models can process. The tool extracts match overviews, player stats, timelines, and key events. It even calculates momentum shifts and win probabilities using a modified Pythagorean formula.

The LoL-MDC isn't really competing with ontology work, it's solving a different problem. Still, it shows just how complex MOBA data can be and why structured approaches are needed. The authors do mention that their framework could potentially be adapted for other games by swapping out the game-specific metrics, though they do not actually do this.

## 2.3 Identified Gaps and Project Positioning

Looking at what exists, there's an obvious gap. General ontologies like the GOP and UGO apply to any game but miss MOBA-specific concepts: lanes, jungles, hero roles, ability cooldowns, item builds, team compositions. Game-specific ontologies like the DOTA 2 one handle these details but can't transfer to other titles.

The missing piece is something in between an ontology that captures what MOBAs have in common without being locked to one particular game. Abstract classes like “Hero” or “Lane” would have properties that work universally, while the actual data could come from whichever game you’re interested in.

That’s exactly the gap this project fills. The ontology abstracts away game-specific details while keeping the semantics that matter for MOBAs. A query like “find heroes with crowd control abilities” should work regardless of whether the data underneath comes from League of Legends, DOTA 2, or something else, as long as the data maps to the shared ontology.

There’s also a practical emphasis here that differs from prior work. Beyond just being theoretically complete, this ontology targets real use cases: champion recommendations, counter-pick suggestions, synergy analysis. The focus isn’t just on what exists in MOBA games, but on what players actually care about when trying to improve.

## Chapter 3

# System Overview

This chapter presents a high-level view of the system, starting with a formal definition of the problem and then describing how the various components work together.

### 3.1 Problem Definition

The problem is straightforward to state: MOBA game data lives across many disconnected sources, stored in different formats, with no common vocabulary linking them together. The goal is to build a knowledge representation that organizes this information so it can be queried systematically, and that works across different games, not just one.

Several requirements follow from this goal:

1. **Unified representation.** A query about “heroes with crowd control abilities” should work whether the underlying data comes from League of Legends, DOTA 2, or something else. This requires abstracting away game-specific terminology into a shared vocabulary.
2. **Relationship modeling.** Knowing that a hero exists isn’t very useful on its own. What matters are the relationships: who counters whom, which heroes synergize together, what items build from what components. The system needs to capture these connections.
3. **Reasoning support.** If the system knows that Hero A counters Hero B, it should figure out on its own that Hero B is countered by Hero A. If a hero has unusually high base health, the system should classify them as a tank without someone having to add that label manually.
4. **Query capability.** The knowledge base should answer the kinds of questions players actually ask, like “which assassins beat mages in lane” or “what items give both armor and magic resist.”
5. **Extensibility.** Adding a new hero, item, or even a new game shouldn’t require restructuring the ontology. New data should fit naturally into the existing schema.

### 3.2 System Architecture

Four main components make up the system. Figure 3.1 shows how they relate to each other.

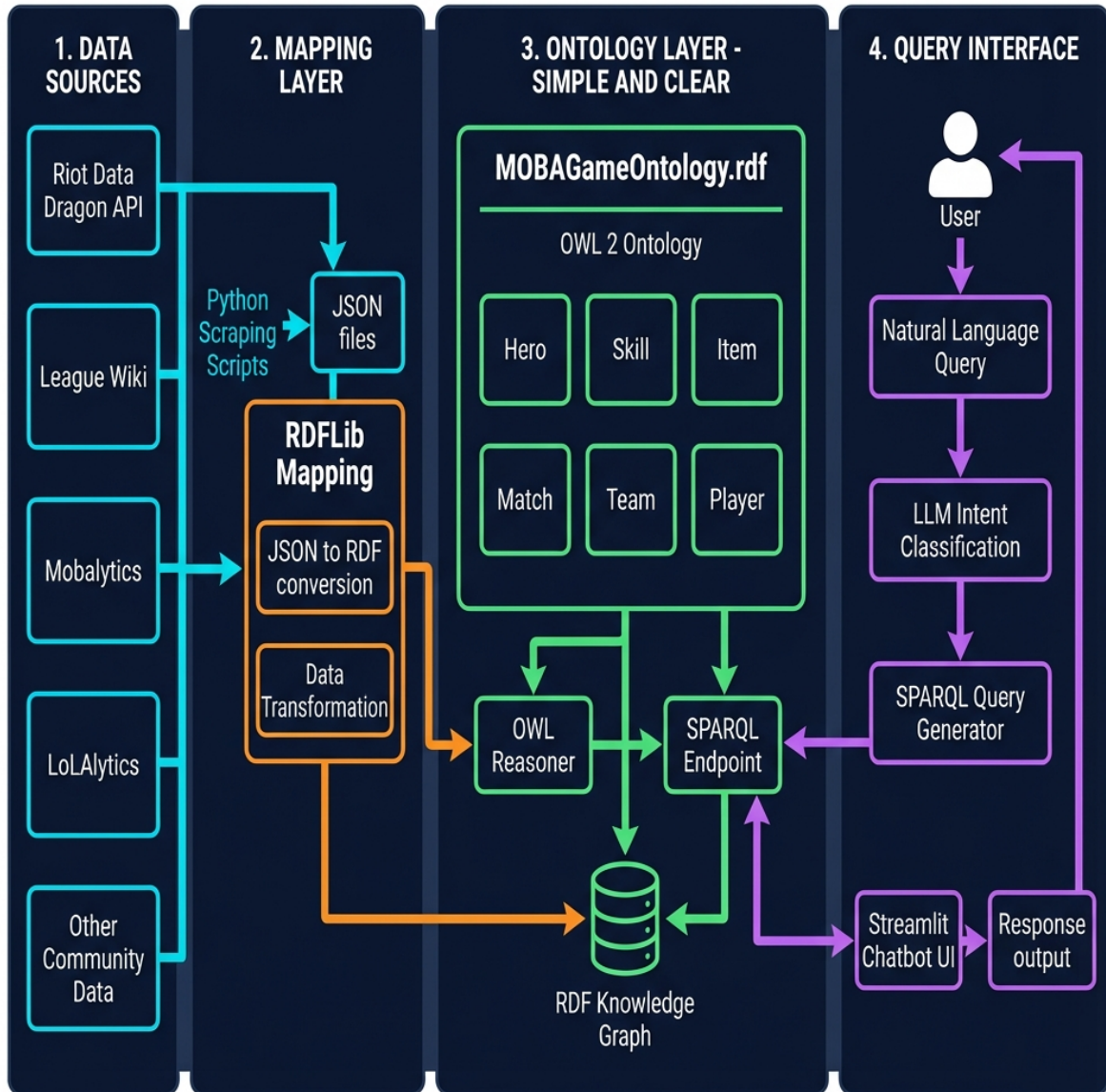


Figure 3.1: Overview of system architecture

### 3.2.1 Data Sources

The system pulls information from several different sources:

**Static game data** covers hero attributes, skill descriptions, base statistics, and item properties, things that don't change from match to match. For League of Legends, this primarily comes from Riot's Data Dragon API along with community wikis. Static data updates when patches release, but otherwise stays constant.

**Strategic data** includes counter-pick relationships, synergy information, and matchup win rates derived from analyzing actual gameplay. Analytics platforms tracking millions of matches provide this data. Unlike static information, strategic knowledge shifts as the meta evolves.

**Build data** consists of recommended items, skill orders, and role assignments based on high-ranked player behavior and professional matches. This changes frequently as players discover new strategies.

Python scripts handle the collection work, fetching data through APIs or scraping web sources. Everything gets stored as JSON files for the next processing stage.

### 3.2.2 Ontology Layer

At the core of the system sits the MOBA Game Ontology. Built using OWL 2 and stored in RDF/XML format, it defines the conceptual structure for all knowledge in the system.

**Classes** establish what types of things exist: Hero, Skill, Item, Team, Match, and various subtypes. These form hierarchies, where TankHero and MageHero are both kinds of Hero, and AdvancedItem and ComponentItem are both kinds of Item.

**Properties** define relationships and attributes. Object properties link entities together (hasSkill connects heroes to their abilities; counters links one hero to another). Data properties attach values (baseHealth, cooldown, goldCost).

**Individuals** serve as predefined instances for enumeration-like concepts: damage types, lane positions, game phases, complexity levels.

**Constraints and rules** enforce validity and enable inference. OWL restrictions specify what makes something a valid TankHero, for instance. SWRL rules handle more complex logic, like determining that a cleanse ability counters crowd control effects.

The ontology is compatible with standard semantic web tools for editing and reasoning.

### 3.2.3 Mapping and Population

Raw JSON data doesn't automatically become part of the knowledge base. It needs to be transformed into RDF triples that follow the ontology's structure.

The mapping component reads JSON files containing hero, skill, item, and strategic data. It creates RDF instances that populate ontology classes with actual values. It also translates game-specific terms into the generic vocabulary, so League of Legends "Champions" become generic "Hero" instances.

Output is written as Turtle (.ttl) files containing the populated knowledge graph. The implementation uses Python with RDFLib, which offers straightforward APIs for constructing and serializing RDF.

### 3.2.4 Query Interface

The final layer provides ways for users to interact with the knowledge base:

**SPARQL endpoint.** Users comfortable with query syntax can access the knowledge base directly. Queries can find heroes matching various criteria, retrieve all counters for a given matchup, or trace item build paths.

**Chatbot interface.** Not everyone knows SPARQL. A Streamlit-based front end accepts natural language questions and translates them into queries behind the scenes. General game information or strategic advice with a given snapshot of the match can be queryable by the chatbot.

**Reasoning integration.** Before queries run, an OWL reasoner processes the ontology to derive implicit facts. Results can include both explicitly stated information and inferred

relationships.

### **3.2.5 Component Interaction**

The workflow connects these pieces in sequence:

Data collection scripts run first, fetching current game information and writing it to JSON files. Mapping scripts then read those files and produce RDF triples conforming to the ontology schema. The generated triples load into the knowledge base alongside the ontology definitions. A reasoner processes everything to compute inferences and verify consistency. At that point, the knowledge base is ready for querying through either SPARQL or the chatbot interface. Results may include both asserted facts and derived knowledge.

This pipeline is designed to be repeatable. When patches drop or new statistics become available, rerunning the collection and mapping stages updates the knowledge base without manual intervention.



## Chapter 4

# Modeling and Ontology Design

This chapter presents the MOBA Game Ontology developed for this project. We begin by discussing how the ontology was designed and what goals guided its development. Then, we examine the classes, properties, and individuals that make up the ontology, along with the constraints and rules that allow for automated reasoning.

### 4.1 Modeling Approach

The ontology was developed using OWL 2 (Web Ontology Language) and stored in RDF/XML format, ensuring compatibility with standard semantic web tools and reasoners. OWL 2 was selected because it supports not only the definition of concepts but also the specification of relationships and constraints between them.

Several design principles guided the construction of the ontology:

- **Game-Agnostic Design:** Rather than focusing on a single MOBA title, the ontology was designed to represent concepts common across the genre. Generic terminology such as “Hero” is used instead of game-specific terms like “Champion” (League of Legends), allowing the ontology to be applicable to different games.
- **Rich Relationships:** Various OWL property types were utilized to capture different kinds of relationships. Symmetric properties represent mutual relationships, such as synergy between two heroes. Inverse properties enable automatic inference; for example, asserting “Hero A counters Hero B” allows the system to derive “Hero B is countered by Hero A” without explicit declaration.
- **Defined Classes:** Many classes include formal definitions through OWL restrictions rather than serving as mere labels. The class `TankHero`, for instance, requires a minimum base health value, enabling a reasoner to classify heroes automatically based on their statistical properties.
- **Static and Dynamic Data:** The ontology accommodates both permanent game information, such as hero abilities and item statistics, and transient match-state data, including current gold and active game phase.

The ontology uses the namespace <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology> and is at version 2.0.

## 4.2 Ontology Scope and Design Principles

### 4.2.1 Scope Definition

The ontology covers these main areas:

1. **Heroes:** Hero attributes, statistics, skills, roles, and character types.
2. **Skills:** Abilities with their damage values, cooldowns, resource costs, and how they scale with levels.
3. **Items:** Purchasable equipment, how items are built from components, and what stats they provide.
4. **Matches:** Team composition, player state, game phases, and map objectives.
5. **Strategic Relationships:** Counter-picks, synergies between heroes, lane matchups, and power spikes.

### 4.2.2 Design Principles

The ontology follows these best practices:

- **Modularity:** Related concepts are grouped together (heroes, items, matches), making the ontology easier to understand and maintain.
- **Extensibility:** New hero types, damage categories, or effects can be added as subclasses without changing existing definitions.
- **Reusability:** Common patterns are reused. For example, the **Stats** class is used by both heroes and items.
- **Documentation:** Classes and properties have labels and comments explaining their purpose.

## 4.3 Classes

The ontology defines 139 classes organized in a hierarchy. Table 4.1 shows the main categories.

Table 4.1: Core Class Categories in the MOBA Game Ontology

Category	Classes
Game Entities	Hero, Item, Skill, Team, Player, Match
Roles	Role, AssassinRole, MageRole, TankRole, Support-Role, CarryRole, JunglerRole

Category	Classes
Hero Types	MeleeHero, RangedHero, TankHero, MageHero, AssassinHero, SupportHero
Item Types	OffensiveItem, DefensiveItem, UtilityItem, ConsumableItem, ComponentItem, AdvancedItem
Skill Types	ActiveSkill, PassiveSkill, UltimateSkill, DamageSkill, HealingSkill, MobilitySkill
Effect Types	DamageEffect, ControlEffect, HealingEffect, MobilityEffect, MitigationEffect
Crowd Control	HardCC, SoftCC, Stun, Root, Silence, Knockup, Slow, Blind
Damage Types	PhysicalDamage, MagicalDamage, TrueDamage, MixedDamage
Statistics	Stats, BaseStats, BonusStats, StatGrowth
Game State	Match, GamePhase, EarlyGame, MidGame, LateGame, Objective
Team Concepts	Team, AllyTeam, EnemyTeam, TeamComposition, TeamSlot

### 4.3.1 Hero Class Hierarchy

The **Hero** class is central to the ontology. It has a restriction stating that every hero must have at least four skills (three basic abilities plus an ultimate).

Listing 4.1: Hero class definition

```

1 <owl:Class rdf:about="#Hero">
2   <rdfs:subClassOf>
3     <owl:Restriction>
4       <owl:onProperty rdf:resource="#hasSkill"/>
5       <owl:minCardinality>4</owl:minCardinality>
6     </owl:Restriction>
7   </rdfs:subClassOf>
8   <rdfs:comment>Playable character in a MOBA game</rdfs:comment>
9 </owl:Class>

```

Heroes are classified in two ways:

- **By Attack Type:** **MeleeHero** (close-range) and **RangedHero** (long-range). These are disjoint, meaning a hero cannot be both.
- **By Role:** **AssassinHero**, **MageHero**, **TankHero**, **SupportHero**, **CarryHero**, **WarriorHero**. These are not disjoint, so a hero can have multiple roles.

The **TankHero** class has an additional restriction: the hero's base health must be above 600.

### 4.3.2 Skill Class Hierarchy

Skills are classified in several ways:

- **By Activation:** `ActiveSkill` (must be activated manually), `PassiveSkill` (always active), and `ToggleSkill` (can be turned on/off).
- **By Effect:** `DamageSkill`, `HealingSkill`, `MobilitySkill`, `CrowdControlSkill`.
- **Ultimate Skills:** A special subclass of `ActiveSkill` that can only have up to 3 levels instead of 5.

The base `Skill` class has a restriction limiting it to at most 5 skill levels.

### 4.3.3 Item Class Hierarchy

Items are classified by function and build status:

**By Function:**

- **OffensiveItem:** Items that increase damage output. Includes `PhysicalDamageItem` and `MagicalDamageItem`.
- **DefensiveItem:** Items that increase survivability. Includes `ArmorItem` and `MagicResistItem`.
- **UtilityItem:** Items with special active effects.
- **ConsumableItem:** Items that are used up (like potions).

**By Build Status:**

- **ComponentItem:** Basic items that can be combined.
- **AdvancedItem:** Completed items built from components.

### 4.3.4 Crowd Control Hierarchy

Crowd control effects are divided into two disjoint categories:

- **Hard CC:** Effects that completely disable the target (Stun, Root, Silence, Knockup, Knockback, Fear, Taunt, Sleep, Disarm, Polymorph, Pull).
- **Soft CC:** Effects that impair but don't fully disable (Slow, Blind).

## 4.4 Properties

The ontology defines 88 object properties (relationships between entities) and 111 data properties (attributes with literal values).

### 4.4.1 Object Properties

#### Hero Properties

Table 4.2 shows the main properties related to heroes.

Table 4.2: Hero-Related Object Properties

Property	Domain	Range	Description
hasSkill	Hero	Skill	Links a hero to their abilities
hasBaseStats	Hero	BaseStats	Hero's starting statistics
hasAttackType	Hero	AttackType	Melee or Ranged
hasComplexity	Hero	ComplexityLevel	How difficult the hero is to play
playsRole	Hero	Role	Hero's strategic role
typicalLane	Hero	Lane	Where the hero usually plays
dealsDamageType	Hero	DamageType	Main damage type

Some properties are marked as functional, meaning each hero can have only one value. For example, each hero has exactly one attack type.

#### Counter and Synergy Properties

The ontology represents strategic relationships between heroes:

##### Counter Properties:

- **counters**: Hero A has an advantage over Hero B.
- **hardCounters**: Hero A has a strong advantage.
- **softCounters**: Hero A has a slight advantage.
- **counteredBy**: The inverse (automatically derived).
- **dominatesInLane / weakInLaneAgainst**: Lane-specific matchups.
- **outscales / outscaledBy**: Late-game comparisons.

##### Synergy Properties:

- **synergyWith**: Heroes that work well together (symmetric).
- **strongSynergyWith / weakSynergyWith**: Degrees of synergy.
- **combosWith**: Heroes whose abilities combine well.

#### Item Properties

##### Build Path:

- **buildPath**: Links an advanced item to its components.

- `upgradesInto`: The inverse relationship.
- `requiresComponent`: Specifies required component items.

#### **Recommendations:**

- `recommendedItem`: Suggested items for a hero.
- `coreItem`: Essential items for a hero.
- `situationalItem`: Items useful in certain situations.

### **Match and Team Properties**

- `hasAllyTeam` / `hasEnemyTeam`: Links a match to its two teams.
- `hasHero` / `memberOf`: Team-hero relationships.
- `currentPhase`: Current game phase.
- `isAllyOf` / `isEnemyOf`: Relationships between heroes (these are disjoint: two heroes cannot be both allies and enemies).

### **4.4.2 Data Properties**

Data properties store numeric and text values.

#### **Hero Statistics**

Base stats on the `Stats` class:

- `baseHealth`, `baseMana`, `baseArmor`, `baseMagicResist`, `baseAttackDamage`, `baseAttackSpeed`, `baseMovementSpeed`, `attackRange`

Per-level growth on `StatGrowth`:

- `healthPerLevel`, `manaPerLevel`, `armorPerLevel`, `attackDamagePerLevel`

Hero metadata:

- `heroName`: The hero's name.
- `winRate`, `pickRate`, `banRate`: Game statistics.
- Boolean flags: `hasCC`, `hasDash`, `hasHeal`, `hasShield`, `hasStealth`

## Skill Properties

- `skillName`: Skill identifier.
- `baseDamage`, `cooldown`, `castRange`: Base values.
- `costAmount`, `costType`: Resource cost information.
- `adScaling`, `apScaling`: Scaling coefficients.

The `SkillLevel` class stores level-specific values:

- `skillLevelNumber`: The level (1-5).
- `damageAtSkillLevel`, `cooldownAtSkillLevel`, `manaCostAtSkillLevel`

## Item Properties

- `itemName`, `goldCost`: Basic information.
- `goldEfficiency`: Value for money.
- `uniquePassive`, `stackable`: Boolean flags.

## Match State Properties

- Player: `currentLevel`, `currentGold`, `kills`, `deaths`, `assists`
- Team: `teamGold`, `teamKills`, `goldLead`, `towersDestroyed`
- Match: `matchTime`

## 4.5 Special Individuals

The ontology includes 62 predefined individuals that represent fixed values. These are like enumeration values that can be used throughout the ontology.

### 4.5.1 Activation Types

- **Active**: Skills that require manual activation.
- **Passive**: Skills that are always active.
- **Toggle**: Skills that can be switched on/off.

### 4.5.2 Lanes

- `TopLane`, `MidLane`, `BottomLane`, `Jungle`

### 4.5.3 Damage Types

- `PhysicalDamage`, `MagicalDamage`, `TrueDamage`, `MixedDamage`

#### 4.5.4 Attack Types

- Melee, Ranged

#### 4.5.5 Complexity Levels

- Beginner, Intermediate, Advanced

#### 4.5.6 Item Types

- Offensive, Defensive, Utility, Consumable

#### 4.5.7 Resource Types

- Mana, Energy, Health, NoResource

#### 4.5.8 Effect-Related Individuals

- Effect activations: `ActiveEffect`, `PassiveEffect`, `AuraEffect`
- Effect ranges: `AllAllies`, `NearbyAllies`, `SelfRange`
- Scaling types: `AbilityPowerScaling`, `AttackDamageScaling`, `Agility`, `Strength`, `Intelligence`
- Effects: `DamageEffect`, `HealingEffect`, `DamageBoost`, `Dash`, `Blink`, `Knockup`, `Shield`, `Stealth`, `Reveal`, `ResourceDrain`

#### 4.5.9 Area of Effect Shapes

- `Circle`, `Cone`, `Line`, `Global`, `SingleTarget`

#### 4.5.10 Game Phases and Team Compositions

- Phases: `EarlyGame`, `MidGame`, `LateGame`
- Compositions: `EngageComposition`, `PokeComposition`, `PickComposition`, `SplitPushComposition`, `TeamfightComposition`, `SiegeComposition`

#### 4.5.11 Role Individuals

- `CarryRole`, `MageRole`, `SupportRole`, `TankRole`, `RoamerRole`

### 4.6 Rules and Constraints

The ontology uses OWL's built-in mechanisms to enforce rules and constraints.



### 4.6.1 Disjointness

Some classes cannot overlap:

- `ActiveSkill` and `PassiveSkill` are disjoint (a skill cannot be both).
- `HardCC` and `SoftCC` are disjoint.
- `MeleeHero` and `RangedHero` are disjoint.
- `AllyTeam` and `EnemyTeam` are disjoint.
- `PhysicalDamage`, `MagicalDamage`, and `TrueDamage` are all disjoint from each other.

### 4.6.2 Functional Properties

Some properties can only have one value per subject:

- `hasAttackType`: Each hero has exactly one attack type.
- `hasAllyTeam`, `hasEnemyTeam`: Each match has exactly one of each.
- `heroName`, `itemName`, `skillName`: Unique identifiers.

### 4.6.3 Inverse Properties

Some properties automatically create reverse relationships:

- `counters` ↔ `counteredBy`
- `hasHero` ↔ `memberOf`
- `buildPath` ↔ `upgradesInto`
- `dominatesInLane` ↔ `weakInLaneAgainst`

### 4.6.4 Symmetric Properties

Some properties are bidirectional by nature:

- `synergyWith`: If Hero A has synergy with Hero B, then Hero B has synergy with Hero A.
- `isAllyOf`, `isEnemyOf`: Team relationships are mutual.
- `conflictsWith`: Resource competition is mutual.

### 4.6.5 Cardinality Restrictions

Limits on how many values a property can have:

- `Hero hasSkill` min 4: Every hero must have at least 4 skills.
- `Skill hasSkillLevel` max 5: Skills can have at most 5 levels.

- `UltimateSkill hasSkillLevel max 3`: Ultimate skills can have at most 3 levels.
- `Team hasHero exactly 5`: Teams have exactly 5 heroes.
- `Match hasAllyTeam exactly 1`: Matches have exactly one ally team.

#### 4.6.6 Value Restrictions

The `TankHero` class has a restriction requiring base health above 600:

Listing 4.2: Value restriction on `TankHero`

```

1 <rdfs:subClassOf>
2   <owl:Restriction>
3     <owl:onProperty rdf:resource="#baseHealth"/>
4     <owl:someValuesFrom>
5       <rdfs:Datatype>
6         <owl:onDatatype rdf:resource="xsd:float"/>
7         <owl:withRestrictions>
8           <xsd:minInclusive>600.0</xsd:minInclusive>
9         </owl:withRestrictions>
10      </rdfs:Datatype>
11    </owl:someValuesFrom>
12  </owl:Restriction>
13 </rdfs:subClassOf>

```

This means a hero with base health above 600 can be automatically classified as a `TankHero`.

#### 4.6.7 SWRL Rules

The ontology includes 3 SWRL (Semantic Web Rule Language) rules for advanced reasoning:

1. **Classify Ranged Hero**: If a hero has attack type `Ranged`, the system automatically classifies it as a `RangedHero`. This rule demonstrates how SWRL can infer class membership from property values.
2. **Transitive Counter**: If Hero A hard-counters Hero B, and Hero B hard-counters Hero C, then Hero A soft-counters Hero C. This rule shows how SWRL can derive indirect relationships through transitivity.
3. **CC and Damage Synergy**: If one hero has hard crowd control abilities and another hero deals high damage, the two heroes have synergy. This rule enables automatic detection of team composition synergies.

#### 4.6.8 What the Constraints Enable

These rules allow the ontology to:

1. **Automatic Classification**: Heroes can be automatically categorized based on their properties.

2. **Consistency Checking:** Invalid data (like a skill marked as both active and passive) can be detected.
3. **Inference:** If we know “Hero A counters Hero B”, the system automatically knows “Hero B is countered by Hero A”.

## 4.7 Summary

The MOBA Game Ontology provides a formal model of the MOBA game domain. It contains:

- 139 classes organized in hierarchies
- 88 object properties (relationships)
- 111 data properties (attributes)
- 62 predefined individuals
- 3 SWRL rules for advanced reasoning

The ontology covers heroes, skills, items, matches, and strategic relationships. It uses OWL features like class restrictions, property characteristics, and predefined individuals to create a rich, queryable knowledge base that can support applications like game analysis and recommendation systems.

## Chapter 5

# Usage and Predefined Queries

### 5.1 Running the System

The application can be executed by following the instructions in the “Installation” section of the repository’s `README.md` file.

### 5.2 Predefined Queries

The application dynamically creates SPARQL queries based on natural language input; therefore, the user does not need to explicitly type SPARQL queries. Below are examples demonstrating this workflow.

#### Example 1: Counter-Picking

**User Query:** "Who should I pick to counter the enemy team: Yasuo mid, Zed jungle, Jinx ADC?"

**Generated SPARQL:** For this input, the following SPARQL queries are generated and executed:

```
PREFIX moba: <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology#>
SELECT DISTINCT ?counter_name
WHERE {
    ?enemy moba:heroName ?enemy_name .
    FILTER(LCASE(?enemy_name) = "yasuo")
    ?enemy moba:counteredBy ?counter .
    ?counter moba:heroName ?counter_name .
}
```

```
PREFIX moba: <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology#>
SELECT DISTINCT ?counter_name
WHERE {
    ?enemy moba:heroName ?enemy_name .
```

```

    FILTER(LCASE(?enemy_name) = "zed")
    ?enemy moba:counteredBy ?counter .
    ?counter moba:heroName ?counter_name .
}

PREFIX moba: <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology#>
SELECT DISTINCT ?counter_name
WHERE {
    ?enemy moba:heroName ?enemy_name .
    FILTER(LCASE(?enemy_name) = "jinx")
    ?enemy moba:counteredBy ?counter .
    ?counter moba:heroName ?counter_name .
}

```

**System Output:** The LLM generates the following answer based on the context retrieved by the SPARQL queries:

“To counter an enemy team of Yasuo (mid), Zed (jungle), and Jinx (ADC), you should consider picking champions who specifically counter Yasuo and Zed, as they are strong all-in threats. The best options based on counter data are:

**Fizz:** Counters both Yasuo and Zed with his strong burst, mobility, and ability to dodge their key abilities.

**Diana:** Matches well into both Yasuo and Zed due to her tankiness, sustained damage, and ability to duel assassins.

**Xerath:** Keeps Yasuo and Zed at range with poke and crowd control, making it hard for them to engage.

Twisted Fate and Aurora are also strong picks, offering map presence and crowd control.

All these picks are effective at shutting down Yasuo and Zed’s engage and assassination potential, while also scaling well into teamfights where Jinx is a threat. Consider your playstyle—choose Fizz or Diana for aggressive play, or Xerath/Twisted Fate for safer, utility-focused approaches.”

## Example 2: Feature Retrieval (Stealth)

**User Query:** "Which assassins have stealth in their kit?"

**Generated SPARQL:**

```

PREFIX moba: <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology#>
SELECT DISTINCT ?name WHERE {
    ?champion moba:heroName ?name .
    ?champion moba:playsRole moba:AssassinRole .
}

```

```

    ?champion moba:hasAbilityEffect moba:StealthEffect .
}
ORDER BY ?name

```

### System Output:

Several assassin champions in League of Legends have stealth abilities in their kits. These include:

- **Akali** (Twilight Shroud grants invisibility within the smoke)
- **Akshan** (Passive: Goes camouflaged while near terrain)
- **Aurora** (Has stealth in kit)
- **Evelynn** (Passive: Permanent camouflage after level 6)
- **Kayn** (Shadow Assassin form gains stealth with E)
- **Kha’Zix** (Ultimate grants invisibility during casts)
- **LeBlanc** (Passive: Mirror Image grants invisibility briefly)
- **Neeko** (Passive: Can disguise and stealth on W)
- **Pyke** (W grants camouflage)
- **Qiyana** (W with brush element grants invisibility)
- **Rengar** (Ultimate provides true invisibility)
- **Shaco** (Q grants invisibility)
- **Talon** (Ultimate grants invisibility)
- **Twitch** (Q grants invisibility)
- **Vayne** (Ultimate + Q grants invisibility)

These champions use stealth to reposition, engage, or escape, making them hard to track and counter. Note that “stealth” includes both invisibility (completely unseen) and camouflage (hidden but can be seen within a certain range or by control wards).

### Example 3: Complex Criteria (Tanks with Hard CC)

**User Query:** "Which tanks have hard CC?"

#### Generated SPARQL:

```

PREFIX moba: <http://www.semanticweb.org/gizemyilmaz/ontologies/moba/ontology#>
SELECT DISTINCT ?name WHERE {
    ?champion moba:heroName ?name .
    ?champion moba:playsRole moba:TankRole .
    ?champion moba:hasCrowdControl moba:StunCC .
    ?champion moba:hasCrowdControl moba:RootCC .
    ?champion moba:hasCrowdControl moba:KnockupCC .
}

```

```

?champion moba:hasCrowdControl moba:SuppressCC .
?champion moba:hasCrowdControl moba:CharmCC .
?champion moba:hasCrowdControl moba:TauntCC .
?champion moba:hasCrowdControl moba:SleepCC .
?champion moba:hasCrowdControl moba:FearCC .
}
ORDER BY ?name

```

### System Output:

Here are some tank champions in League of Legends with hard crowd control (CC) abilities such as stuns, knockups, roots, suppressions, charms, taunts, sleep, or fear:

- **Malphite** (Knockup, Slow): His ultimate, Unstoppable Force, is a strong AoE knockup.
- **Leona** (Stun, Root): Her Q (Shield of Daybreak) and R (Solar Flare) both provide hard CC.
- **Nautilus** (Stun, Root, Knockup): Passive auto attacks root, Q (Dredge Line) stuns, and R (Depth Charge) knocks up.
- **Amumu** (Stun): His Q (Bandage Toss) and R (Curse of the Sad Mummy) both stun.
- **Sejuani** (Stun, Freeze): Her Q and R can stun, and her passive applies a freeze effect.
- **Rammus** (Taunt): His E (Frenzying Taunt) is a reliable single-target taunt.
- **Ornn** (Knockup, Brittle): His R (Call of the Forge God) and Q+E combo both knock up.
- **Shen** (Taunt): His E (Shadow Dash) is a multi-target taunt.
- **Sion** (Knockup, Stun): His Q (Decimating Smash) and R (Unstoppable Onslaught) can knock up/stun.
- **Maokai** (Root, Knockback): W (Twisted Advance) roots, Q (Bramble Smash) knocks back and slows.
- **Alistar** (Knockup, Stun): Q (Pulverize) knocks up, W (Headbutt) displaces.
- **Zac** (Knockup, Slow): Q (Stretching Strikes) and E (Elastic Slingshot) knock up or displace.

These champions are strong frontliners, excel at teamfighting, and can reliably initiate or peel with hard CC. If you need more specifics about any champion's abilities or best roles, let me know!

=====

# Chapter 6

## Evaluation

### 6.1 Evaluation Methodology

To evaluate the contribution of SPARQL-based ontology querying to the system’s response quality, we designed a comparative experiment with the following setup:

#### 6.1.1 Experimental Conditions

1. **Baseline (Without Ontology):** The system responds to user queries using only the language model’s pre-trained knowledge, without access to the ontology.
2. **With Ontology:** The system uses SPARQL queries to retrieve relevant context from the ontology, which is then provided to the language model to generate responses.

#### 6.1.2 Evaluation Criteria

A set of 20 questions about the game was prepared for testing. Each response was evaluated by two independent judges:

- **Human Judge:** A domain expert with extensive knowledge of League of Legends.
- **LLM Judge:** Gemini 2.5 Pro, prompted to evaluate response quality.

Both judges rated each response on a scale of 1 to 5 based on the following criteria (see Appendix for the full LLM judge prompt):

- **Accuracy:** Correctness of the information provided.
- **Informativeness:** Depth and relevance of the content.
- **Clarity:** How well-written and understandable the response is.

### 6.2 Results

Table 6.1 summarizes the average scores across all 20 questions for each condition.



Condition	Human Judge	LLM Judge	Average
Without Ontology	2.3	3.2	2.75
With Ontology	4.1	4.4	4.25
<b>Improvement</b>	+1.8 (+78%)	+1.2 (+38%)	+1.5 (+55%)

Table 6.1: Evaluation results comparing system performance with and without ontology access.

### 6.3 Analysis

Based on the experimental results, we draw the following conclusions:

- **Ontology significantly improves response quality:** The average score increased from 2.75 to 4.25 (a 55% improvement) when the system had access to ontology-derived context. This demonstrates the value of structured knowledge retrieval in enhancing LLM responses.
- **Human judges are more critical:** The human judge consistently gave lower scores than the LLM judge in both conditions. This suggests that human evaluators apply stricter standards, particularly regarding domain-specific accuracy.
- **Greater improvement observed with human evaluation:** The human judge’s scores improved by 78% with ontology access, compared to 38% for the LLM judge. This indicates that the ontology particularly helps with factual accuracy, which human experts are better at detecting.

## Chapter 7

# Discussion and Conclusion

### 7.1 Discussion

**Strengths:** We designed a comprehensive ontology with extensive coverage of the game. The language model is able to generate high-quality answers with the help of context retrieved from SPARQL queries.

**Limitations:** MOBA games like League of Legends are extremely complex. To demonstrate, the official Wiki of the game has over 11,000 articles. As such, it was not possible to cover all mechanics of the game. In addition, the game often has exceptional cases. For example, all champions have 4 abilities, but a recent champion Hwei has 13 abilities, which made the modeling work more difficult.

**Lessons Learned:** When designing an ontology, it is important to pay attention to all the details from the start. Starting with a vague ontology and improving it along the way makes things more difficult.

### 7.2 Conclusion

In this project, we were able to:

- Create a comprehensive ontology of MOBA games.
- Enrich it with approximately 50,000 data points, fetched and scraped from various resources.
- Design an intent classification system where the appropriate SPARQL query for the user request is generated.
- Use the context retrieved by the SPARQL query to enable the LLM to provide high-quality answers to the user.
- Implement snapshot analysis, a tool where users can upload state of a game in JSON format and receive helpful suggestions.

This project allowed us to gain hands-on experience with the topics taught in the Social Semantic Web course, including designing and building ontologies and using SPARQL queries

to retrieve results. We were also able to build a chat interface that we believe represents the state of the art in the gaming industry.

## 7.3 Future Work

The following are planned for future development:

- Support for Dota 2 and other MOBA games.
- Enriching the ontology with more mechanics and data.
- Fine-tuning an open-source language model specifically for answering user questions about MOBA games.

## Chapter 8

# Resources and Ethics Declaration

### 8.1 External Resources

The following open-source resources were used in the development of this project:

#### 8.1.1 Data Sources

- **Mobalytics** [9]: Champion builds, counters, and synergy data. Open source.
- **League of Legends Wiki** [5]: Champion abilities and game mechanics documentation. Licensed under CC BY-SA.
- **DataDragon** [13]: Static game data and assets provided by Riot Games. Open source.
- **Leaguepedia** [6]: Esports and game data API. Licensed under CC BY-SA.
- **LoLalytics** [8]: Champion statistics and analytics platform. Open source.

#### 8.1.2 Libraries and Tools

- **RDFLib** [12]: Python library for parsing RDF data and executing SPARQL queries. Licensed under BSD-3-Clause.
- **Streamlit** [14]: Web application framework used for the chatbot’s user interface. Licensed under Apache 2.0.
- **Protégé** [10]: Ontology editor used for designing and validating RDF/OWL ontologies. Licensed under BSD.
- **OpenAI API** [11]: Language model API used for intent classification and natural language response generation.

### 8.2 AI Tools and Prompt Usage

In this project, the following AI-based tools were used:

- **Gemini** [7]: Used to search for data sources for League of Legends.

- **Claude** [3]: Used to assist with coding the application and user interface, as well as formatting the LaTeX of this report. All content is the authors' own words; Claude was only used for LaTeX formatting assistance.

**Example prompts:**

- “Add this image to the background of the Streamlit UI.”
- “Fix the bug causing the button to freeze when pressed.”
- “Add bullet points to this LaTeX section of the paper.”

### **8.3 Academic Integrity Statement**

We confirm that all work presented in this report is our own, and any external resources or AI tools used have been properly acknowledged. The AI assistants mentioned above were used only for supportive tasks such as searching for data sources, coding assistance, and LaTeX formatting - not for generating the intellectual content of this project. All data sources and libraries comply with their respective licenses.

# Appendix A

## Appendix

### A.1 LLM Judge Prompt

The following prompt was used with Gemini 2.5 Pro to evaluate the quality of system responses:

You are an expert evaluator for a League of Legends question-answering system. Your task is to rate the quality of the system's response to a user question.

**Question:** {question}

**System Response:** {response}

Please rate the response on a scale of 1 to 5 for each of the following criteria:

1. **Accuracy** (1-5): Is the information factually correct?
  - 1: Mostly incorrect or misleading
  - 3: Partially correct with some errors
  - 5: Fully accurate
2. **Informativeness** (1-5): Does the response provide sufficient depth and relevant details?
  - 1: Vague or missing key information
  - 3: Adequate but could include more details
  - 5: Comprehensive and thorough
3. **Clarity** (1-5): Is the response well-written and easy to understand?
  - 1: Confusing or poorly structured
  - 3: Understandable but could be clearer
  - 5: Clear, well-organized, and easy to follow

Provide your ratings in the following format:

Accuracy: [score]

Informativeness: [score]

Clarity: [score]

## A.2 Benchmark Questions

The following questions were used to evaluate the system:

1. Who should I pick to counter an enemy team of: Yasuo mid, Zed jungle, Jinx ADC?
2. Which assassins have stealth in their kit?
3. How much damage does Evelynn's Q do at level 3?
4. Which one is a better tank, Malphite or Rengar?
5. For Vayne, should I buy BotRK or IE?
6. For Alistar, what is the best starting item?
7. If Olaf and Yuumi fight at Level 1, which one will win?
8. How does Ornn's base health scale per level?
9. How can I counter a fed Aatrox?
10. How can I counter a fed Yi & Taric pairing?
11. Is Rammus a good counter to Yasuo?
12. Who should I pick to counter an enemy team comp of: Malphite (Top), Amumu (Jungle), Orianna (Mid)?
13. Is Kassadin a good pick into a heavy AD team (Zed Mid, Talon Jungle)?
14. How can I counter a fed Kha'Zix if my team has no hard CC?
15. Which support is better at peeling against a Leona & Samira kill lane: Janna or Yuumi?
16. Which champions have resets on their abilities upon getting a takedown?
17. Which Marksmen (ADCs) deal True Damage with their abilities?
18. Does Teemo's blind (Q) stop Garen's Q damage?
19. Which champions possess a Global Ultimate (can reach anywhere on the map)?
20. Can Soraka's Equinox (E) silence a Karthus while he is in his passive (undead) form?

# Bibliography

- [1] Anonymous. A DOTA 2 ontology. Unpublished manuscript using Basic Formal Ontology (BFO) v1.1 framework, 2014.
- [2] Anonymous. LoL-MDC: League of Legends match data compactor for large language model analysis. *Applied Sciences*, 15(7190):1–16, 2025.
- [3] Anthropic. Claude: Ai assistant. <https://www.anthropic.com/claude>, 2024. Commercial AI assistant by Anthropic.
- [4] Michael S. Debus. *Unifying Game Ontology: A Faceted Classification of Game Elements*. Ph.d. dissertation, IT University of Copenhagen, 2019.
- [5] Fandom Community. League of legends wiki. <https://wiki.leagueoflegends.com/en-us/>, 2024. Licensed under CC BY-SA.
- [6] Fandom Community. Leaguepedia api. [https://lol.fandom.com/wiki/Help:Leaguepedia\\_API](https://lol.fandom.com/wiki/Help:Leaguepedia_API), 2024. Licensed under CC BY-SA.
- [7] Google DeepMind. Gemini: A family of highly capable multimodal models. <https://gemini.google.com>, 2024. Commercial AI assistant by Google.
- [8] LoLalytics. Lolalytics: Champion statistics and analytics. <https://lolalytics.com>, 2024. Open source analytics platform.
- [9] Mobalytics. Mobalytics: League of legends stats, builds, and analytics. <https://mobalytics.gg/lol>, 2024. Open source game data platform.
- [10] Mark A. Musen. The protégé project: A look back and a look forward. In *AI Matters*, volume 1, pages 4–12. ACM, 2015.
- [11] OpenAI. Openai api. <https://openai.com/api/>, 2024. Commercial API for language models.
- [12] RDFLib Team. Rdfliib: A python library for working with rdf. <https://github.com/RDFLib/rdfliib>, 2024. Licensed under BSD-3-Clause.
- [13] Riot Games. Datadragon: League of legends static data. <https://github.com/meraki-analytics/lolstaticdata>, 2024. Open source static game data.



- [14] Streamlit Inc. Streamlit: The fastest way to build data apps. <https://github.com/streamlit/streamlit>, 2024. Licensed under Apache 2.0.
- [15] José P. Zagal and Michael Mateas. Temporal frames: A unifying framework for the analysis of game temporality. In *Situated Play, Proceedings of DiGRA 2007 Conference*, Tokyo, Japan, 2007. Digital Games Research Association.
- [16] José P. Zagal, Michael Mateas, Clara Fernandez-Vara, Brian Hochhalter, and Nolan Lichti. Towards an ontological language for game analysis. In *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play*, Vancouver, Canada, 2005. Digital Games Research Association.