Abstract - give a few sentences INCLUDING 1-3 claims
We have created an ontology for the online multiplayer video game League of Legends that can infer an optimal build based on the currently played champion and all other champions in that specific game. The ontology stores all champions and items in order to allow for inferences to be made on champion role, a set of optimal items to be built, and the best items to build at any point in the game.

Introduction/Motivation - give a few sentences
League of Legends is multiplayer online battle arena game, played by several millions of people around the world. As such, there are many, many rules and colloquialisms that have grown around this game, both formally and informally. Many of these rules deal with assigning roles to in-game characters, or champions, as well as buying the items needed for success within the game. These can be daunting for the beginning player, who can find this lack of information discouraging. By creating an ontology to furnish a plugin displaying suggested builds throughout gameplay, a player may overcome this lack of knowledge and play the game enjoyably.

Use case - summary and pointer to your online use case
The objective of this ontology is to build a plugin to help a new player (secondary actor) learn the basics of playing a champion for the first time. The basics include providing a list of recommended items to build based on the users, allied, and enemy champions, and several basic ability combinations the champion can do or terminology uniquely related to that champion. These suggested builds would continue throughout the game, depending on the amount of gold the user has, as well as the items the user has in their inventory at the time.

When the game first loads, the plugin should already be running and retrieve information about the user’s game, including what champions are in the game, which identifies the user’s champion as well. This data would come from a site like lolnexus, which is a public interface between Riot, the company that hosts the game, and the players. lolnexus displays information about players in a game in real time. This data could also be sourced directly from Riot using their public API, then translated into a format that the ontology can understand. After the agent retrieves name of the user’s champion, it should be able to tell that the user has not played this champion before. From there, it would pull data from the ontology containing data on champion and role terminology and ability combinations, as a champion usually plays one or two roles, but determining which depends on the context of the other champions on the team. It would also pull data on recommended build paths within context of champion, role, and the enemy champions. The top levels of things in this ontology would have the categories of Champion, Item, and Term. This ontology would be built on information scraped automatically from the wikipedia page defining champions and items, community sites dedicated to
champion builds (like mobafire), and sites containing statistics on what role a champion usually plays (like op.gg or the Riot API), all translated into a standard format and aggregated into one data structure.

A link to the current version of the use case can be found here.

Technical Approach

Our approach, focused on providing appropriate build paths for champions, involves following a two-stage path: inferring team roles and suggesting item builds. The first, inferring team roles, is taken care of by our Team and Champion classes, while suggesting item builds is appropriately handled by the Item Class.

In order to furnish the inference of team roles, the Champion master class has several properties and subclasses. Any given instance of the Champion class is related to an instance of the Statistic class, which for any champion contain the properties hasGoalStatisticValue and hasBaseStatisticValue, the former indicating which statistics should be prioritized and the latter indicating statistics given at the beginning of the game. Champions also have an object property whose class, PartyInRole, has the properties of hasLikelihood, hasDamageTypeString, and four properties referring to instances of the earlier mentioned Statistic class. Each of these Statistic object properties subdivide all statistics a champion may have into categories which help in item selecting inferences. These other properties, particularly the hasLikelihood property to indicate likelihood of a certain role, influence deciding which role a champion has within a team. These are indicated by the subclasses of PartyInRole, which detail the 6 roles a champion can play for any given game. The Champion class then relates to the overall Team class by the latter's hasChampion property, which allows for a given champion to influence the team damage type, as given by the property hasTeamDamageType.

The architecture of the ontology that allows item build inferences is structured in such a way that the Item class's subclasses and various properties contain all the information needed to make those inferences. Any given item may be defined by a subclass categorizing the statistical information of what it does for a champion (magic damage, defense, etc.). These items may also be obtained at the beginning of the game or be consumable, also represented by the appropriate subclasses. Each item also has descriptors represented by properties -- hasGoldCost, hasItemIdentifier, hasStatistic, nextItemInBuild, and buildsInto. The property describing the amount of gold cost is of particular advantage to making inferences, since it contains the amount of gold a champion has. Items also have properties declaring which of the four tiers they fall under, indicating the amount of advantage a property confers.

To test the main ontology’s correctness, an ontology of individuals was created with five example champion individuals who were set as being part of one example team individual. Along with this, around thirty item individuals were created with relations organized in such a way as to produce an example ‘item build tree’ for one of the champion individuals (Jinx). The team and champion individuals were created to test inferring each champion's role based on likelihoods that they will play
a role in a given game. The item individuals were created to test inferring which items a user should buy at any point in an ongoing game, given the items the user currently has and the user's current gold.

There are two groupings of inferences that are made: one for determining champion role, and the other for pulling buyable items in a build. To actually make an inference, a Java program is used to generate a series of queries whose results, when run on the individuals ontology, are aggregated by the program to get the final inference. This resulting inference answers its respective competency question. When a user requests a list of items they can buy at some point during a game, three queries are generated from the inputted list of items and amount of gold the user currently has. The first query outputs the gold value of all items in the user’s inventory, and each item is mapped to its respective gold cost. The second query returns the next items in the build path, excluding all items the user currently owns. The third query returns all items each currently owned item can be built into.

In order to use all gold currently available, we test if the sum of all unique items returned by the last two queries is less than gold available. If this is true, we generate and run the two queries again for each item returned in the first iteration, with the addition of that item to the owned items list. The function continues recursing until a list that uses all current gold is generated. In this way, we take the list of items currently owned, assume one of the next discovered items is bought, and run the entire search again to see if we can find new items that are farther along in the build tree. Each item in this aggregated result list is an inference. All queries generated are uniquely saved to individual files, and may be run in order to produce the same result list, though items may appear in results more than once.

Related Work
Ontologies for other specific games have been built before (notably Skyrim); however, due to the uniqueness of each game’s architecture and rules, this League of Legends ontology is not based on the other ontology.

Evaluation (against competency questions)
Competency questions can be broken down into two categories: questions about item inferences, and questions about team or champion inferences.

- Competency questions 1, 13, and the first part of competency questions 2, 3, and 4 answered by query 1 on the ‘Demonstration and Queries’ page
  - The second part of competency questions 2, 3, and 4 involve construction of the item build knowledge graph, which we show is possible using queries 2 and 3. Actual implementation of this feature is discussed in detail in the ‘Future Work’ section.
- Competency question 5 is directly answered with query 4.
- Competency questions 6, 7, 8, 11, and 12 can be correctly answered using the Java program with the appropriate inputs.
• Competency questions 9 and 10 are possible to answer with the current ontology, but currently there is no query pattern that answers them (again discussed in 'Future Work').

For a more in-depth analysis on how each query answers competency questions and produces inferences, please refer to the ‘Demonstration and Queries’ page of the website.

Discussion

Value of Semantics - make sure to have a separate section in the discussion highlighting the impact of the semantics

Include a link to your project website

Support your claims

Semantics were of central importance at several points within the ontology, but not when interacting with the user. The latter is so because of how the range of inputs to the ontology is limited by the number of champions the user can select, and is later limited by the finite set of items a user can buy in game. Within the structure of the ontology, semantics came into play when constructing the class hierarchies for the Champion, Item, Team, and Statistic classes, most significantly in the Champion class. The PartyInRole class, an object property of the Champion class, contained the semantics of champions. Subclasses of PartyInRole outline all the possible roles a champion can play, but determining these subclasses had to ultimately be based on semantics, since all roles were creations of the League of Legends community, and were not originally part of the game. Even after the game adopted some community defined roles, opinions on what champions could play those roles was left to each player's subjective assessment. For this project is was therefore necessary to take the natural language semantics regarding those varying opinions into account when designing how one champion could pick one of several roles, not only based on the other champions on that user's team, but also based on the user's opinion.

The current ontology, including the query generating Java program, is well positioned to support the proposed plugin. Given a knowledge graph representing the user’s goal build, the items the user currently has, and gold available, the java program is able to return all buyable items at any given point in the game. The ontology is utilized when the program generates and runs a series of queries on it to infer all the information the plugin requested. The ultimate purpose of the java program is to generate queries and aggregate results, the queries on the ontology are what produce all the inferences (refer to the ‘Technical Approach’ section for a more in-depth discussion.

More details on how the Java program works and how to run it can be found on the ‘Demonstration and Queries’ page of the project web site, found here.

To produce this knowledge graph, the ontology is currently in a position where it can infer the user’s role in a game, which should then be used to produce a statistical ‘goal’ profile that the ontology should then try to most closely approximate by selecting a set of six items, similar to the well-known knapsack problem. The structure of the current ontology contains all the classes and relations
pertaining to statistics, but more work is necessary to fully utilize statistics (refer to the ‘Future Work’ section for a more in-depth discussion).

**Future Work**

The ontology as it is currently structured is well positioned to be the centerpiece of all reasoning that is outlined in the use case, and can support future work that more fully encompasses and refines the process of selecting optimal items in a build. That work would be primarily be focused on taking more advantage of the Statistics class within the ontology in order to consider a larger breadth of parameter types when selecting optimal items, as well as the ability to prioritize some statistics over others when selecting items. Another unused aspect of the ontology is the increase of some champion statistics over time, in place to be implemented in the queries.

As it stands now, the ontologies need the following to reach completion:

- A complete ontology containing all champions and items, along with their respective statistics.
  - There is a system currently implemented in the Java code that can read in champion and item JSON files, but needs to be expanded to be able to construct or add to an ontology containing all that information.
- Linking the sections of the Java code that find a champion’s role to a new section that maximizes desired statistics based on that role.
- Generating a new ontology for selected items, and having in game item selection queries use that ontology.

Since what is built currently already demonstrates correctness in answering all competency questions, these points seek to add robustness to the project.

Long term, the ontology has the ability to be expanded to allow it to be the centerpiece of a more general plugin for new players, as discussed in the first several iterations of the use case. This may come to include a tip system for players based on the champion they are playing, as well as enriched text tips for terms that come up in the in-game chat box. There also exists the possibility to expand functionality to encompass any aspect of the game revolving around champion or item statistics, since that knowledge already exists in the ontology. An example of new possible functionality is the potential for the plugin to adapt the player’s generated build based on how well or badly they are doing in their current game, as an experienced player does. The plugin could also be expanded to include the ability for the plugin to suggest optimal champions for a user to pick during champion select (before every game starts) based on champions already selected by allied and enemy team members.

**Conclusion**

Overall, this ontology accurately provides a simple method for new players to navigate several of the most difficult parts of the game, which include choosing item builds and champion roles within teams. By creating both a base ontology to hold the overarching structure and an individuals ontology to hold
instances, competency questions to evaluate whether these ontologies accomplished the overall goal were successful. Queries behind these resulted in accurate builds and roles for the user. Despite the success of this set of ontologies, there are several additions and expansions that would add to the robustness and reach in accomplishing the goal. These include proper utilization of the Statistics class in determining builds and roles and an expansive knowledge base to handle all user situations.

References