I. Use Case Description

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Developing a League of Legends application for first-time players</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Identifier</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>Point of Contact</td>
<td>Anders Maraviglia (<a href="mailto:anders.maraviglia@gmail.com">anders.maraviglia@gmail.com</a>), Hannah De los Santos (<a href="mailto:hdelossantos653@gmail.com">hdelossantos653@gmail.com</a>)</td>
</tr>
<tr>
<td>Creation / Revision Date</td>
<td>April 29, 2017</td>
</tr>
<tr>
<td>Associated Documents</td>
<td><a href="https://tw.rpi.edu/web/Courses/Ontologies/2017/LeagueofLegends">https://tw.rpi.edu/web/Courses/Ontologies/2017/LeagueofLegends</a></td>
</tr>
</tbody>
</table>

II. Use Case Summary

| Goal | To develop an integrated League of Legends plugin to aid first-time players. This plugin will display suggested builds on screen throughout gameplay. |
| Requirements | This use case must generate a plugin to aid first-time players, with suggested builds throughout gameplay. By creating this plugin to aid first-time players, this will allow new players to get up to speed quickly. New players are often stymied by the amount of necessary knowledge to simply play the game, discouraging them from playing further. Requirements of the plugin include: real-time data about the game the user is playing from an online data site or directly from the company hosting the game; access to aggregated data from online on how the community plays that champion, where this data is almost all in natural language; access to data about item hierarchies and statistics. |
| Scope | This use case focuses on beginner League of Legends players. It will only care about data related to the user's champion and role. Because of the focus on the user, it will also not concentrate on the relationships between users. |
| Priority | |
| Stakeholders | The user's teammates, as they want the user to know terminology so that the user plays better and improves the team's chances of winning. The developers, Anders Maraviglia and Hannah De los Santos. Riot, the company that owns League of Legends, as they want their game to have less of a steep learning curve and be made more accessible. |
| Description | The objective of this use case 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 a couple 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.

Once this information is retrieved, display it to the user when they are loaded into the game.

### Actors / Interfaces

**Primary Actors:**
- Programmer: He/she is the designer of the plugin/ontology.

**Secondary Actors:**
- Player: He/she drives the use case by defining champion and role
- Programming language and interface
- League of Legends plugin
- lolnexus (website that retrieves real time game information)
- Riot API (allows access to game, player, and basic champion data)
- League of Legends Wiki (contains information on champion matchups, and community opinions on many aspects of the game, will build ontology from this)
- mobafire (site dedicated to champion builds, matchups, and playstyle. Since content is user driven, quality is not assured and thus needs aggregation within the ontology to be considered reliable)
- Prov ontology from W3C (to add reliability)

### Pre-conditions

- Programmer has access to relevant League of Legends data repositories and ontologies.
- Programmer has access to sufficient programming language and interface to code League of Legends plugin.
- Player is a beginner League of Legends Player.
- Player has sufficiently capable system to download and run League of Legends and developed plugin.

### Post-conditions

- Programmer has developed League of Legends plugin.
- Player has copy of League of Legends plugin.

### Triggers

- The programmer decides to code a League of Legends plugin to aid first-time users.

### III. Usage Scenarios

**Scenario 1: New Player Nick**

Nick is a college student who has casually played computer games in his spare time for the past several years. His friends recently told him about the game League of Legends and he decides to give it a try. He installs the game as well as the agent, which he heard was a good resource for new players looking to learn the game. After playing the tutorials, his friends convince him to
play a real game with them. They try to help him out by telling him to pick the champion Annie and to go mid. Nick agrees and picks Annie, but does not know what “go mid” means and is too embarrassed at his ignorance to ask his friends. Once the game starts loading, the agent looks up Nick’s player information, recognizes that it’s his first time playing Annie, and retrieves data to help him from Riot and the ontology. Once the game is loaded up and begun, the agent displays a list of items it recommends Nick build. This is inferred from having determined that he is playing in the mid lane against the champion Zed, using data stored in the ontology representing the “meta”. Throughout the game, the plugin continues to suggest item combinations based on what Nick has in his inventory to optimize his champion’s statistics.

Scenario 2: Tim the Troll

Tim is an 8th grader that has played League of Legends for 5 months, but has recently decided to try to improve his gameplay by downloading the agent. However, Tim has his own views on what is viable, which differ from the meta. He gets into a game and picks the champion Fiddlesticks for the ad carry role. While the game is loading the agent attempts to automatically determine Tim’s role, what he should build, etc. but the ontology doesn’t contain any information for that champion with respect to that role since Tim’s pick does not fall into categorizations outlines in the meta. When the game is loaded the agent therefore displays a message that no build or playstyle information is available for this champion in this role.

Scenario 3: Swords for Selena

Selena is 23-year old woman and a first-time video game player; she’s begun playing League of Legends because her boyfriend seems to be obsessed with it. At this point, she has downloaded the game and played a few matches. However, she is often spoken to in all capital letters by her teammates since she does so badly. After detailing her woes to her boyfriend, he suggests that she download the plugin for some help. She does so and loads up the game. Immediately, the plugin suggests that she choose Ashe, and she does so. The plugin suggests which items she should start off with to play the game well, inferred from her character choice, as well as her enemies and teammates’ characters. The plugin’s underlying knowledge base then restricts itself to viewing items relevant to creating good builds for Ashe. As she begins her first game, she begins to accrue gold and items, including Brawler’s Gloves. As she already has a dagger in her inventory, the plugin suggests that she build Zeal, which will give her champion an increased attack speed and critical strike chance, inferred from her champion’s optimal build path. Her champion is now a better fighter and she is now able to kill more enemies throughout the game. Throughout the game, the plugin suggests more and more builds in a similar manner, and soon her champion is one of the foremost players of her group. Her team wins, and she is happy to have contributed so highly to their win.

Scenario 4: Eager Eleanor

Eleanor is a 14-year old “gamer girl,” as she calls herself, though she has never played League of Legends before. Upon downloading the game and plugin by suggestion from her friends, she chooses the champion Annie. The plugin’s underlying knowledge base then restricts itself to viewing items relevant to creating good builds for Annie. She becomes accustomed to the playstyle, but continues to have no idea which items are best for her champion to succeed. As she progresses and accrues gold and items, she buys an Amplifying Tome. Though she thinks this
item is cool, she particularly dislikes the way the Sapphire Crystal looks in her inventory and gets rid of it. This Sapphire Crystal is necessary, however, to create a Lost Tome, which contributes to one of Annie’s best items. In response, the plugin limits its knowledge base to only consider items that do not require the Sapphire Crystal. It instead suggests that she build a Fiendish Codex, as it requires only the Amplifying Tome. Eleanor does so, unaware that she has lost out on a large advantage. The plugin continues suggesting builds throughout the game and limiting its knowledge base as necessary.

IV. Basic Flow of Events

Narrative: Often referred to as the primary scenario or course of events, the basic flow defines the process/data/work flow that would be followed if the use case were to follow its main plot from start to end. Error states or alternate states that might occur as a matter of course in fulfilling the use case should be included under Alternate Flow of Events, below. The basic flow should provide any reviewer a quick overview of how an implementation is intended to work. A summary paragraph should be included that provides such an overview (which can include lists, conversational analysis that captures stakeholder interview information, etc.), followed by more detail expressed via the table structure.

In cases where the user scenarios are sufficiently different from one another, it may be helpful to describe the flow for each scenario independently, and then merge them together in a composite flow.

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor (Person)</th>
<th>Actor (System)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Programmer</td>
<td>Lolnexus, Riot API, League of Legends subreddit, mobafire, League of Legends plugin</td>
<td>Programmer conceives of League of Legends plugin for first time players.</td>
</tr>
<tr>
<td>2</td>
<td>Programmer</td>
<td>Lolnexus, Riot API, League of Legends subreddit, mobafire, League of Legends plugin</td>
<td>Programmer collects League of Legends data (lolnexus, Riot API, League of Legends Wiki, mobafire), then codes plugin to access these data and be integrated with League of Legends.</td>
</tr>
<tr>
<td>3</td>
<td>Player</td>
<td>League of Legends, League of Legends plugin</td>
<td>Player downloads League of Legends plugin and begins to play League of Legends.</td>
</tr>
<tr>
<td>4</td>
<td>Player</td>
<td>League of Legends, League of Legends plugin</td>
<td>Player picks new champion.</td>
</tr>
<tr>
<td>5</td>
<td>Player</td>
<td>League of Legends, League of Legends plugin</td>
<td>League of Legends plugin looks up champion information, and the ontology infers the correct build.</td>
</tr>
<tr>
<td>6</td>
<td>Player</td>
<td>League of Legends, League of Legends plugin</td>
<td>League of Legends plugin displays suggested build information to player at beginning of game. The ontology infers this based on the frequent play styles of players (metagame).</td>
</tr>
</tbody>
</table>

V. Alternate Flow of Events

Narrative: The alternate flow defines the process/data/work flow that would be followed if the use case enters an error or alternate state from the basic flow defined, above. A summary paragraph should be included that provides
an overview of each alternate flow, followed by more detail expressed via the table structure.

*Note: Steps 1-2 from normal flow are the same. Likely induced by “troll” players.

### Alternate Flow of Events 1

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor (Person)</th>
<th>Actor (System)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Player picks non meta champion.</td>
</tr>
<tr>
<td>2</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Agent cannot find champion information.</td>
</tr>
<tr>
<td>3</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Agent displays failure message (no recommended builds).</td>
</tr>
</tbody>
</table>

### Alternate Flow of Events 2

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor (Person)</th>
<th>Actor (System)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 3</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Same as basic flow.</td>
</tr>
<tr>
<td>4</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>User sells item needed in build path.</td>
</tr>
<tr>
<td>5</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Knowledge base moves to earlier point in build path, suggesting that the user create item recently sold.</td>
</tr>
</tbody>
</table>

### Alternate Flow of Events 3

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor (Person)</th>
<th>Actor (System)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 3</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Same as basic flow.</td>
</tr>
<tr>
<td>4</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>User buys item not in knowledge base.</td>
</tr>
<tr>
<td>5</td>
<td>Player</td>
<td>Legends, League of Legends plugin</td>
<td>Knowledge base reduces amount of inventory space available and gold available, basing future suggestions on this lack of space.</td>
</tr>
</tbody>
</table>

VI. Use Case and Activity Diagram(s)

Note: updated to complete Joseph’s missing work

Provide the primary use case diagram, including actors, and a high-level activity diagram to show the flow of primary events that include/surround the use case. Subordinate diagrams that map the flow for each usage scenario should be included as appropriate.
1. General Flow

- Data
  - Gets
  - Creates
  - Plugin
  - Uses
  - Player

2. Data

- Programmer
  - Gets
  - Metadata
    - Riot API
    - League Wiki
    - Lolnexus
    - Database
    - op.gg

3. Plugin

- Programmer
  - Creates
  - Plugin
  - Has feature
  - Suggested Builds
  - Uses
  - Metadata
4. Player

- Player installs League of Legends.
- Player chooses Champion.
- Player looks up Plugin.
- Plugin creates Optimal Build.
5: Overarching Detailed Diagram
5. Activity Diagrams: What happens when the Player does something with the plugin:

VII. Competency Questions

Ontology I/O categorization key:
All inputs and outputs to the ontology are categorized by the following descriptors:
- WEB LOOKUP = Information that came from one of the web-based resources listed in the resources section.
- PLUGIN LOOKUP = Information the plugin pulls from a game the user is in, real time.
- ONTOLOGY LOOKUP = Information retrieved from a lookup in the ontology that does not require the ontology to do any sort of reasoning.
- INFERRED = Information that was reached based on some combination of the four categorizations used in logical operations performed on the ontology.

1. Identify Role Question: What role is the champion Azir when the rest of the team is made up of the champions Garen, Jinx, Nidalee, and Thresh?
Ontology: The plugin would know what these champions are by making a call requesting them from the Riot API or lolnexus. It then requests the role of the user’s champion from the ontology, giving the rest of the champions as context for the search.
Ontology Inputs:
- Users champion (from Riot API or lolnexus) (WEB LOOKUP)
- All other champions on users team (again from Riot API or lolnexus) (WEB LOOKUP)

Answer: Mid lane (maybe Top lane)
Ontology: The plugin would return this answer after finding the user’s and his/her teammates champions in the ontology. For every champion, the ontology will contain percentages on how often that champion is played in each role, where that data is extracted from a site dedicated to such statistics, like champion.gg or just the Riot API. Given this information for every champion on the user’s team, the ontology must infer each champions most statistically probable role. So in the context of this question, Azir usually fills the mid lane role 80% of the time, top lane 18%, and the other three roles 2%. The only other champion that comes close to this likelihood of going mid is Garen, who goes mid 15% of the time, and top 75%. Since Azir is the champion who goes mid most often, the ontology designates him as being mid and returns that conclusion to the plugin. (Note: there percentages were made up just for this example.)
Ontology Outputs:
- User’s role (INFERRED)
- The roles of all other champions on the user’s team (INFERRED)
2. Find Correct Build Question: What build path should the champion Malphite (in the top lane role) take against a team comprising the champions Zed, Aatrox, Caitlyn, Olaf, and Braum?

Ontology: The plugin would find the champions in question the same way as in question 1, by querying either Riot API or lolnexus. It would then find the user's role the same way it did in question 1, by querying the ontology.

Ontology Inputs:
- User’s role (INFERRRED)
- Champions on enemy team (WEB LOOKUP)

Answer: He should itemize armor, with items like sunfire cape, dead man’s plate, and frozen heart.

Ontology: After the plugin has inferred the user’s role, we now need to look up what job the user’s champion has on his or her team (does he do physical or magic damage, or is he a tank?). We can find this by accessing a field in the champion class in the ontology for Malphite, which has his statistically highest played job (extracted from online data on frequency of champion builds and their win rates, which the Riot API contains) as being a tank. The ontology must then find the jobs of all the enemy team’s champions the same way, and make the inference that they only do physical damage. The ontology then infers several goal statistics from both those that the champion needs most for that game (inferred from which stats counter the enemy damage types) and statistics that work best for that champion in general (inferred from the champions base statistics). From there, the ontology selects a set of items that most maximize these goal statistics while minimizing gold cost and ‘weight’ per item. These items are then expanded into their full upgrade trees and each tree is joined together to form a directed acyclic graph such that a topological sort on it will produce a list of items that represent what the user should build as the game progresses (how this is done is handled in a later competency question). This subgraph is returned to the plugin and becomes the knowledge base for that game.

Ontology Outputs:
- User’s job (ONTOMETRY LOOKUP)
- Enemy team damage type (INFERRRED)
- Recommended build subgraph for user (INFERRRED) (based on previous inference)

3. No Variation Build Question: What build path should the champion Jinx (in the AD carry role) take against a team comprising the champions Zed, Aatrox, Caitlyn, Olaf, and Braum?

Ontology: Just like the Find Correct Build question, the plugin finds the enemy champions from lolnexus or Riot API.

Ontology Inputs:
- User’s role (INFERRRED)
- Champions on enemy team (WEB LOOKUP)

Answer: She should itemize physical damage and attack speed with items like infinity edge, statik shiv, phantom dancer, and rapid fire cannon.

Ontology: The process of finding the user’s role and job are the same as discussed in the Find Correct Build question, but now the number of ‘good’ builds for Jinx is minimal (one or two), and for beginners should always be the same, because Jinx is the AD carry and is responsible for doing most of her team’s damage. Therefore all her items should be focused on doing as much physical damage as possible, and thus she has no room for items to counter the enemy team’s damage type(s). The ontology must be able to recognize this and return that there is only one build for Jinx in all situations.

Ontology Outputs:
- User’s job (ONTOMETRY LOOKUP)
- Recommended build subgraph for user (INFERRRED) (special case when there is only one build for a champion, does not need statistics from enemy team to build item subgraph)

4. Troll Identify Role and Build Question: What role is the champion Fiddlesticks, when the rest of his team is Garen, Nidalee, Azir, and Thresh, and what should he build?

Ontology Inputs:
- User’s champion (from Riot API or lolnexus) (WEB LOOKUP)
- All other champions on users team (again from Riot API or lolnexus) (WEB LOOKUP)
- Champions on enemy team (WEB LOOKUP)

Answer: Return failure, as Fiddlesticks does not fit into this team composition in the current meta.

Ontology Outputs:
- User’s champion has no defined role (INFERRED)
- A ‘no recommended builds’ note (INFERRED)

5. Present Build to the User Question: What items (from the build the ontology selected) should the plugin present to the user (playing the champion Jinx) when he/she opens the shop for the first time that game with 550 gold?

Knowledge Base Inputs:
- Users item inventory (PLUGIN LOOKUP)
- User's current gold (PLUGIN LOOKUP)

Answer: Return the item list ‘doran’s blade, health potion x2, warding totem’ in that order.

Knowledge Base: From having earlier requested the most optimal build from the ontology, the plugin would have already build a knowledge graph of all the items it wants to recommend to the user throughout the game. Once the plugin retrieves the user's current inventory (set of items) and gold the user has, the plugin updates the knowledge base with what items have been completed in the build path (which in this case is the starter item), and infers a set of items that are next in the knowledge base (build path) that maximizes statistical benefit while minimizing gold cost. It makes this inference by referencing the item list that represents the topological sort of the knowledge base, and takes a subset of items from the index of the current completed point in the build to the furthest index reachable with the current amount of gold the user has.

Knowledge Base Outputs:
- List of most gold efficient items that are currently buyable (INFERRED)

6. User Sells Component Item Question: What should the plugin display when the user sells the item ‘Brawler’s Gloves’ while already having ‘Boots of Speed’ and ‘Berzerker’s Greaves’?

Knowledge Base Inputs:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should return a list of purchasable items that may still include ‘Brawler’s Gloves’, and any newly purchasable items from the knowledge base build path. In this case, ‘Dagger’, ‘Dagger 2’, and ‘Dagger 3’.

Knowledge Base Outputs:
- List of most gold efficient items that are currently buyable (INFERRED) (All that is done is just moving the startpoint of the item subset back to point before the sold item.)

7. User Buys Item Not In Build Path Question: What should the plugin display when the user buys the item ‘wits end’, where that item is not present in the knowledge base?

Knowledge Base Inputs:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should return buyable items from the current point in the build path stored in the knowledge base, taking into account there is one less item slot available.

Knowledge Base Outputs:
- List of most gold efficient items that are currently buyable (INFERRED) (we ignore the item not present in the knowledge base and only factor in that there is one less item slot available)

8. Recommend Last Item in Build Question: What should the plugin display when the user has 5 of the 6 completed items from the build path in their inventory, and the starting item taking up the last slot?
Knowledge Base Inputs:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should display a note to sell the starter item and buy a component or the entire 6th item, depending on how much gold the user has at that point.

Knowledge Base Outputs:
- Note to sell starter item (INFERRED) (no space left in inventory and starter item present)
- Part of or the entire last item (INFERRED) (done by normal procedure, stated above)

9. Recommend Stacking Item in Build Question: What build should the plugin suggest when the user picks the champion Cassiopeia?

Ontology Inputs:
- User’s role (INFERRED)
- Champions on enemy team (WEB LOOKUP)

Answer: The build the plugin should recommend to the user should include the item ‘Archangel’s Staff’, one of whose component items should be displayed as one of the first to be built, with the full item appearing much later. Ontology: The ontology should construct a subgraph representing the most optimal build path just as normal, but should recognize that one of the items it wants to build has a stacking component item. The ontology then infers that this component should be built earlier than the rest of the item, and thus puts that component item far before the rest of the item in the directed acyclic graph. Thus when a topological sort is performed on the knowledge graph, the component shows up much earlier in the list than the rest of the item.

Ontology Outputs:
- User’s job (ONTOMETRY LOOKUP)
- Enemy team damage type (INFERRED)
- Recommended build subgraph for user (INFERRED) (stacking item should appear early)

10. Building Multiple Items at the Same Time Question: What items should the plugin recommend the user build when playing the champion Maokai when they return to the shop for the first time with 2300 gold?

Knowledge Base Inputs:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should recommend the user build the component items ‘specter’s cowl’ and ‘bambi’s cinder’ with the gold the user has.

Knowledge Base: The knowledge graph built by inferences from the Ontology for the user’s champion should represent edges to component items ‘specter’s cowl’ and ‘bambi’s cinder’ with very high weights relative to other purchasable items. The knowledge base will have these components weighted more highly than even their completed parent items because it infers that the breadth of stats they give maximize gold efficiency more than completing either item before the other is started would. This will affect the subsequent topological ordering in such a way where the component items appear first.

Knowledge Base Outputs:
- List of most gold efficient items that are currently buyable (INFERRED)

11. What items should the plugin recommend the user build if they only have Doran’s Blade, Warding Totem, and 100 gold available?

Knowledge Base Input:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should recommend the user build ‘Health Potion 2’ and ‘Health Potion’ with the gold the user would require to buy these items.

Knowledge Base: The knowledge graph built by inferences from the Ontology for the user’s champion should represent edges to component items ‘health potion’ and ‘health potion 2’ with very high weights relative to other
purchasable items at the time. The knowledge base will have these components weighted more highly than even their completed parent items because it infers that the breadth of stats they give maximize gold efficiency more than completing either item before the other is started would. This will affect the subsequent topological ordering in such a way where the component items appear first.

Knowledge Base Outputs:
- List of most gold efficient items that are currently buyable (INFERRED)

12. What items should the plugin recommend the user build when they have the Warding Totem and a limited amount of gold, enough to only buy one item?

Knowledge Base Input:
- Users item inventory (PLUGIN LOOKUP)
- User’s current gold (PLUGIN LOOKUP)

Answer: The plugin should recommend the user build ‘Cloak of Agility’ with the gold the user would require to buy this items.

Knowledge Base: The knowledge graph built by inferences from the Ontology for the user’s champion should represent edges to component items ‘cloak of agility’, ‘pickaxe’, and ‘bf sword’ with very high weights relative to other purchasable items at the time. The knowledge base will have these components weighted more highly than even their completed parent items because it infers that the breadth of stats they give maximize gold efficiency more than completing either item before the other is started would. This will affect the subsequent topological ordering in such a way where the component items appear first. The ontology should then sort these items by their costs, only displaying the lowest-cost item.

Knowledge Base Outputs:
- The most gold efficient item that is currently buyable (INFERRED)

13. What role should the champion Azir play in any given game played by the user?

Ontology: The plugin would know what this champion is by making a call requesting them from the Riot API or lolnexus. It then requests the role of the user’s champion from the ontology, giving the rest of the champions as context for the search.

Ontology Inputs:
- Users champion (from Riot API or lolnexus) (WEB LOOKUP)

Answer: Mage

Ontology: The plugin would return this answer after finding the user’s champions in the ontology. For every champion, the ontology will contain percentages on how often that champion is played in each role, where that data is extracted from a site dedicated to such statistics, like champion.gg or just the Riot API. Given this information for every champion on the user’s team, the ontology must infer each champions most statistically probable role. Since Azir is the champion who is most likely to be a mage, the ontology designates him as being a mage and returns that conclusion to the plugin.

Ontology Outputs:
- User’s role (INFERRED)

VIII. Resources

In order to support the capabilities described in this Use Case, a set of resources must be available and/or configured. These resources include the set of actors listed above, with additional detail, and any other ancillary systems, sensors, or services that are relevant to the problem/use case.

Knowledge Bases, Repositories, or other Data Sources

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
<th>Characteristics</th>
<th>Description</th>
<th>Owner</th>
<th>Source</th>
<th>Access Policies &amp; Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(dataset or repository name)</td>
<td>(remote, local/in situ, etc.)</td>
<td>e.g. – no cloud cover</td>
<td>Short description of the dataset, possibly including rationale of the usage characteristics</td>
<td>Source (possibly a system, or remote site) for discovery and access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Riot API</td>
<td>Remote</td>
<td></td>
<td>Contains player history stats, champion stats, etc.</td>
<td>Riot Games</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobafire</td>
<td>Remote</td>
<td>Third party, community driven site</td>
<td>Champion build, playstyle, matchup info</td>
<td>Private</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lolnexus</td>
<td>Remote</td>
<td>Third party site</td>
<td>Real time game info about players in an ongoing match</td>
<td>Private</td>
<td></td>
<td></td>
</tr>
<tr>
<td>League Wiki</td>
<td>Remote</td>
<td>Third party, community driven site</td>
<td>Defines champions, terminology, and how the two may relate</td>
<td>Private</td>
<td></td>
<td></td>
</tr>
<tr>
<td>op.gg</td>
<td>Remote</td>
<td>Third party</td>
<td>Alternative to Riot API for gathering statistics on champions and what roles they usually occupy</td>
<td>Private</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**External Ontologies, Vocabularies, or other Model Services**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Language</th>
<th>Description</th>
<th>Owner</th>
<th>Source</th>
<th>Describes/Uses</th>
<th>Access Policies &amp; Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ontology, vocabulary, or model name)</td>
<td>(ontology language and syntactic form, e.g., RDFS - N3)</td>
<td>If the service is one that runs a given ontology or model-based application at a given frequency, state that in addition to the basic description</td>
<td>Source (link to the registry or directly to the ontology, vocabulary, or model where that model is maintained, if available)</td>
<td>List of one or more data sources described by and/or used by the model</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Other Resources, Service, or Triggers** (e.g., event notification services, application services, ...)
<table>
<thead>
<tr>
<th>Resource</th>
<th>Type</th>
<th>Description</th>
<th>Owner</th>
<th>Source</th>
<th>Access Policies &amp; Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(sensor or external service name)</td>
<td></td>
<td>Include a description of the resource as well as availability, if applicable</td>
<td>Primary owner of the service</td>
<td>Application or service URL; if subscription based, include subscription and any subscription owner</td>
<td></td>
</tr>
<tr>
<td>Programming language and interface</td>
<td>service</td>
<td>Capable of building the League of Legends</td>
<td>The owner of the programming language and interface</td>
<td>N/A</td>
<td>This should be available to the programmer</td>
</tr>
</tbody>
</table>