Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

Системы искусственного интеллекта

Модуль №1

Базы знаний и онтологии

Группа: P3324

Выполнил: Маликов Глеб Игоревич

Преподаватель:

Королёва Юлия Александровна

Санкт-Петербург

2024г.

**Оглавление**

[Введение 3](#_Toc177210662)

[Анализ требований 4](#_Toc177210663)

[Изучение основных концепций и инструментов 5](#_Toc177210664)

[Обзор основных концепций баз знаний и онтологий 5](#_Toc177210665)

[Изучение Prolog 5](#_Toc177210666)

[Ознакомление с инструментами и библиотеками 5](#_Toc177210667)

[Реализация системы искусственного интеллекта 6](#_Toc177210668)

[Создание правил и логики вывода 6](#_Toc177210669)

[Создание онтологии в Protégé: 7](#_Toc177210670)

[Реализация рекомендательной системе на языке Python 7](#_Toc177210671)

[Инициализация и подключение Prolog 8](#_Toc177210672)

[Определение словарей ключевых слов 8](#_Toc177210673)

[Извлечение предпочтений 9](#_Toc177210674)

[Запрос к базе знаний Prolog 9](#_Toc177210675)

[Подсчет уверенности 10](#_Toc177210676)

[Оценка и интерпретация результатов 11](#_Toc177210677)

[Примеры запросов 11](#_Toc177210678)

[Примеры запросов к базе знаний 11](#_Toc177210679)

[Примеры запросов в Protégé 11](#_Toc177210680)

[Пример использования рекомендательной системы в Python 12](#_Toc177210681)

[Анализ результатов 13](#_Toc177210682)

[Заключение 14](#_Toc177210683)

# Введение

В данном проекте была поставлена цель разработать систему поддержки принятия решений с использованием баз знаний, онтологий и языка программирования Prolog. Задача важна для создания интеллектуальных систем, которые могут помочь автоматизировать процессы рассуждений и принимать решения на основе логических правил и фактов.

Проект направлен на изучение баз знаний, онтологий и их применения в системах поддержки принятия решений. Использование Prolog даёт возможность работать с логическими конструкциями и правилами, создавая интеллектуальные системы, способные решать задачи на основе заданных фактов и правил.

# Анализ требований

Для реализации системы поддержки принятия решений, были определены следующие требования:

Требования к базе знаний и онтологии:

* База знаний должна содержать факты и правила, описывающие отношения между объектами предметной области.
* Онтология должна включать концепции, атрибуты и отношения, определяющие структуру знаний.

Требования к системе поддержки принятия решений:

* Система должна уметь делать выводы на основе заданной базы знаний и онтологий.
* Реализованная система должна поддерживать интерфейс для ввода запросов и получения результатов в виде выводов или фактов.

# Изучение основных концепций и инструментов

## Обзор основных концепций баз знаний и онтологий

База знаний — это структурированное хранилище фактов и правил, которое используется для автоматизированного принятия решений. База знаний работает на основе логических взаимосвязей между элементами, позволяя системе принимать решения на основании заданных фактов.

Онтология представляет знания о предметной области в виде концепций, атрибутов и отношений. Она позволяет моделировать конкретные области знаний и обеспечивать логическую структуру для баз знаний.

## Изучение Prolog

Prolog — это язык программирования, ориентированный на обработку логических выражений. Он поддерживает работу с термами, фактами, правилами, предикатами и механизмом откатов для нахождения всех возможных решений. Важные концепции Prolog включают:

* Термы — основные структуры, такие как атомы, числа, переменные и составные термы.
* Факты — утверждения, описывающие связи между объектами.
* Предикаты — функции, возвращающие истину или ложь, которые используются для проверки условий и выполнения действий.
* Откат — механизм, позволяющий системе пробовать альтернативные решения, когда текущий путь не приводит к успешному результату.
* Унификация — процесс сопоставления термов, при котором переменные заменяются конкретными значениями.

## Ознакомление с инструментами и библиотеками

Для работы с онтологиями и базами знаний можно использовать такие инструменты, как Protégé, который предоставляет интерфейс для создания и визуализации онтологий. Protégé поддерживает создание концепций, экземпляров и отношений между ними.

Кроме того, для создания системы поддержки принятия решений с использованием Prolog в Python можно применять библиотеку «pyswip», которая предоставляет интерфейс для взаимодействия с движком Prolog.

# Реализация системы искусственного интеллекта

## Создание правил и логики вывода

Была создана база знаний на основе информации о чемпионах в игре «Лига Легенд». Свойства и отношения между чемпионами описаны фактами и правилами. Факты определяются как роль, позиция и рейтинги чемпионов, а правила описывают атрибуты чемпионов.

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4 | champion(garen).  champion(lee\_sin).  champion(lucian).  champion(lulu). | |  |

Рисунок 1 - Пример фактов: определение чемпионов

|  |  |
| --- | --- |
| 1  2  3  4 | role(singed, specialist).  role(kassadin, assassin).  role(karma, enchanter).  role(karma, burst). |

Рисунок 2 - Пример фактов: определение ролей

|  |  |
| --- | --- |
| 1  2  3  4 | position(garen, top).  position(lee\_sin, jungle).  position(lucian, bot).  position(lucian, mid). |

Рисунок 3 - Пример фактов: определение позиций

|  |  |
| --- | --- |
| 1  2  3  4  5 | rating(garen, utility, 1).  rating(garen, damage, 2).  rating(garen, toughness, 3).  rating(garen, control, 1).  rating(garen, mobility, 1). |

Рисунок 4 - Пример фактов: определение рейтингов

|  |  |
| --- | --- |
| 1  2  3  4  5 | good\_support(Champion) :-  rating(Champion, control, ControlRating),  rating(Champion, utility, UtilityRating),  ControlRating >= 2,  UtilityRating > 2. |

Рисунок 5 - Пример правила

Запросы в базе данных были написаны для поиска чемпионов с выбранными атрибутами:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2 | role(Champion, marksman), rating(Champion, control, 3).  findall(Champion, good\_support(Champion), Result). | |

Рисунок 6 - Пример запросов

Исходный код базы знаний доступен по ссылке:

<https://github.com/glebmavi/AIS/blob/main/Lab1/part1.pl>

## Создание онтологии в Protégé:

По аналогии с фактами и правилами в Prolog были созданы концепты и свойства в онтологии:

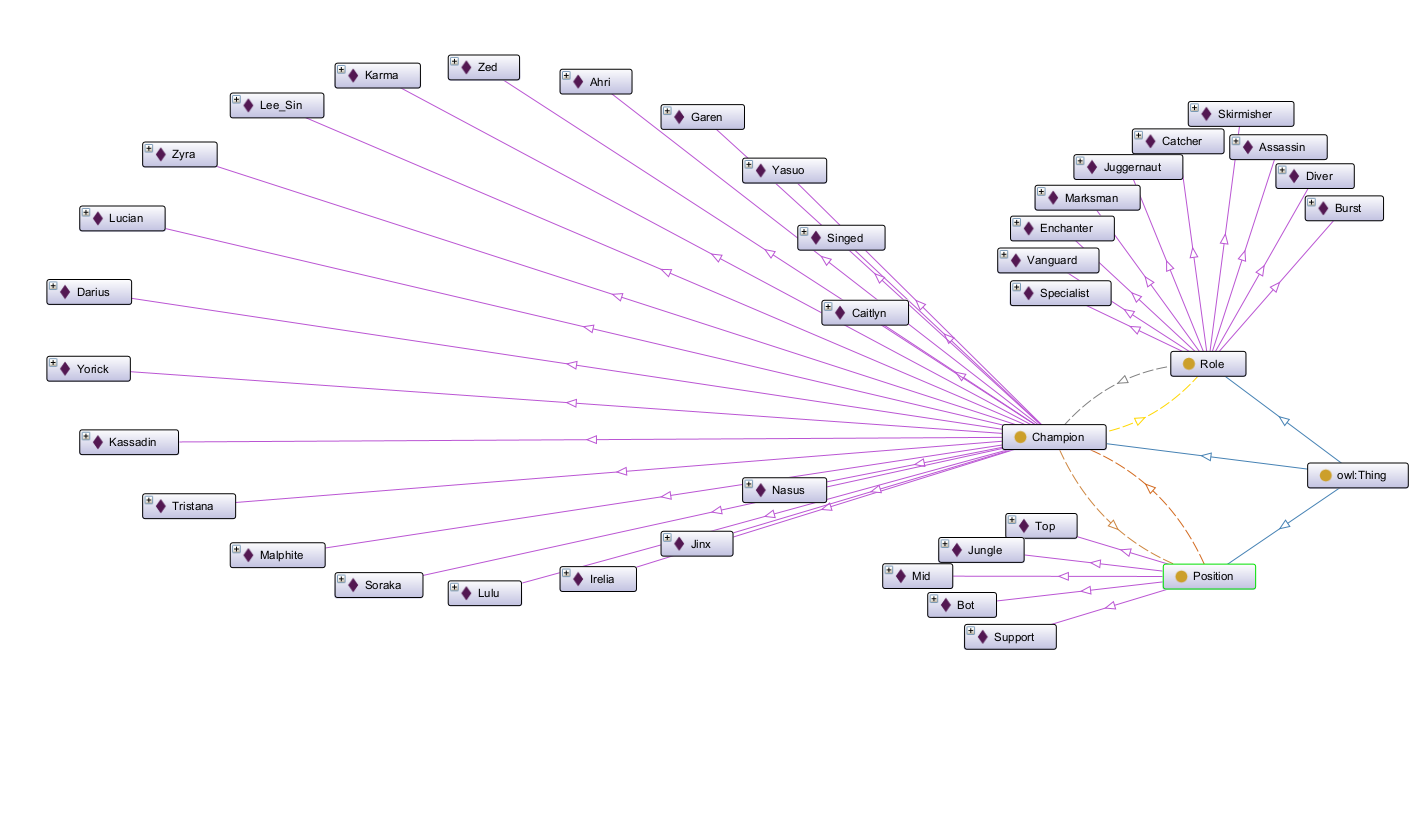


Рисунок 7 - Граф экземпляров и классов

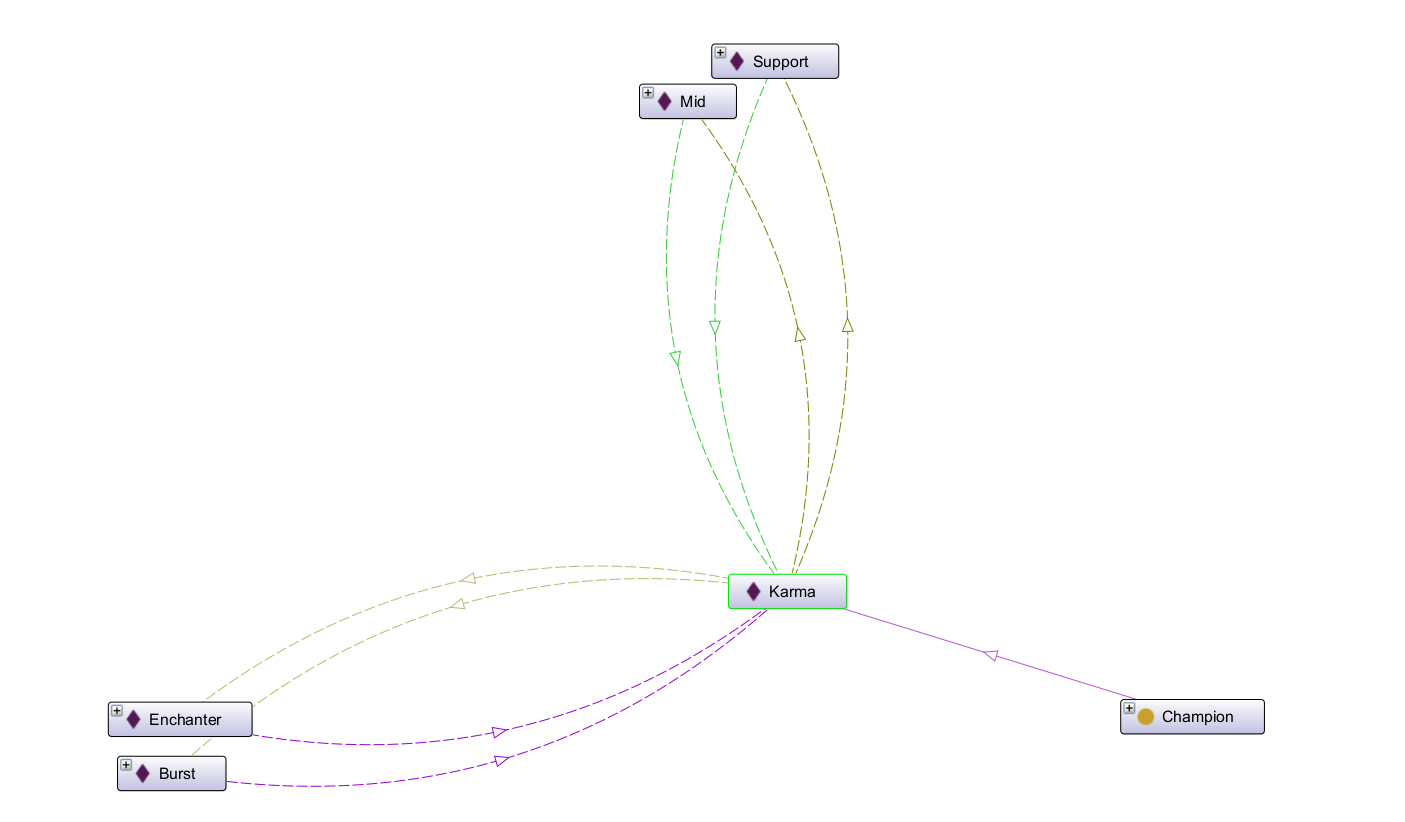


Рисунок 8 - Свойства чемпиона "Karma"

Исходный код онтологии доступен по ссылке:

<https://github.com/glebmavi/AIS/blob/main/Lab1/Ontology.rdf>

## Реализация рекомендательной системе на языке Python

Ниже представлена структура и объяснение создания рекомендательной системы с использованием библиотеки «pyswip», которая позволяет интегрировать Prolog в Python. В данном примере пользователь вводит свои предпочтения по роли, позиции и атрибутам, а система подбирает подходящих чемпионов, основываясь на базе знаний, реализованной в Prolog.

### Инициализация и подключение Prolog

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **from** os **import** putenv  putenv("SWI\_HOME\_DIR", "C:\\Program Files\\swipl")  **from** pyswip **import** Prolog  prolog = Prolog()  prolog.consult("part1.pl") |

Рисунок 9 - Инициализация и подключение Prolog

### Определение словарей ключевых слов

Ключевые слова для ролей, позиций и атрибутов нужны для сопоставления пользовательского ввода с запросами в Prolog. Некоторые слова, продублированные с корнем для получения слов с склонениями:

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | ROLES = {  'assassin': ['убийца', 'убийц', 'ассасин', 'assassin'],  'burst': ['бурст', 'burst'],  'catcher': ['ловец', 'ловц', 'catcher', 'cc', 'кэтчер'],  'diver': ['погружение', 'погруж', 'diver', 'дайвер'],  'enchanter': ['enchanter', 'чародейка', 'чарод'],  'juggernaut': ['танк', 'juggernaut', 'джаггернаут'],  'marksman': ['стрелок', 'стрел', 'адк', 'marksman', 'марксман'],  'skirmisher': ['дуэлянт', 'дуэл', 'skirmisher', 'скримишер'],  'specialist': ['специалист', 'специал', 'specialist'],  'vanguard': ['вангард', 'vanguard'],  }  POSITIONS = {  'top': ['топ', 'верх', 'top'],  'mid': ['средняя', 'средн', 'мид', 'mid'],  'jungle': ['джангл', 'лес', 'jungle'],  'bot': ['бот', 'нижняя', 'нижн', 'ботлейн', 'bot'],  'support': ['саппорт', 'сапп', 'поддержка', 'поддержк', 'supp'],  }  ATTRIBUTES = {  'damage': ['урон', 'damage'],  'mobility': ['мобильность', 'мобильн', 'mobility'],  'utility': ['утилита', 'утил', 'utility'],  'tankiness': ['выживаемость', 'выжив', 'tankiness'],  'control': ['контроль', 'контрол', 'control'],  'late\_game': ['поздняя', 'поздн', 'late', 'late game'],  'initiation': ['инициация', 'иници', 'initiation', 'инициа'],  'team\_fight': ['командные', 'команд', 'team fight'],  'one\_shot': ['ваншот', 'one shot'],  } | |  |

Рисунок 10 - Словари ключевых слов

### Извлечение предпочтений

Функция для извлечения ролей, позиций и атрибутов из пользовательского ввода:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **def** extract\_preferences(user\_input):  user\_input = user\_input.lower()  preferences = {  'roles': [],  'positions': [],  'attributes': []  }  **def** match\_keyword(phrase, keywords):  **for** keyword **in** keywords:  **if** keyword **in** phrase:  **return** **True**  **return** **False**  **for** role, keywords **in** ROLES.items():  **if** match\_keyword(user\_input, keywords):  preferences['roles'].append(role)  **for** position, keywords **in** POSITIONS.items():  **if** match\_keyword(user\_input, keywords):  preferences['positions'].append(position)  **for** attribute, keywords **in** ATTRIBUTES.items():  **if** match\_keyword(user\_input, keywords):  preferences['attributes'].append(attribute)  **return** preferences |

Рисунок 11 - Извлечение предпочтений

### Запрос к базе знаний Prolog

Функция для формирования запросов к базе знаний на основе предпочтений. Собираются все чемпионы, которые имели хотя бы одно совпадение с предпочтениями:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **def** query\_champions(preferences):  champions\_set = set()  **if** preferences['roles']:  **for** role **in** preferences['roles']:  role\_query = f"role(Champion, {role})"  role\_champions = list(prolog.query(role\_query))  champions\_set.update([champ['Champion'] **for** champ **in** role\_champions])  **if** preferences['positions']:  **for** position **in** preferences['positions']:  position\_query = f"position(Champion, {position})"  position\_champions = list(prolog.query(position\_query))  champions\_set.update([champ['Champion'] **for** champ **in** position\_champions])  **if** preferences['attributes']:  **for** attribute **in** preferences['attributes']:  prolog\_rule = ATTRIBUTE\_RULES.get(attribute)  **if** prolog\_rule:  attribute\_query = f"{prolog\_rule}(Champion)"  attribute\_champions = list(prolog.query(attribute\_query))  champions\_set.update([champ['Champion'] **for** champ **in** attribute\_champions])  champions = [{'Champion': champ} **for** champ **in** champions\_set]  **return** champions |

Рисунок 12 - Получение чемпионов из базы знаний

### Подсчет уверенности

Для каждого чемпиона оценивается уверенность (confidence), основываясь на процент совпадения предпочтений с найденными чемпионами:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **def** calculate\_confidence(preferences, champion):  confidence = 0  total\_criteria = len(preferences['roles']) + len(preferences['positions']) + len(preferences['attributes'])  **if** preferences['roles']:  **for** role **in** preferences['roles']:  **if** list(prolog.query(f"role({champion}, {role})")):  confidence += 1  **if** preferences['positions']:  **for** position **in** preferences['positions']:  **if** list(prolog.query(f"position({champion}, {position})")):  confidence += 1  **if** preferences['attributes']:  **for** attribute **in** preferences['attributes']:  prolog\_rule = ATTRIBUTE\_RULES.get(attribute)  **if** prolog\_rule **and** list(prolog.query(f"{prolog\_rule}({champion})")):  confidence += 1  **if** total\_criteria == 0:  **return** 0  **return** confidence / total\_criteria \* 100 |

Рисунок 13 - Подсчет уверенности

Исходный код программы доступен по ссылке:

<https://github.com/glebmavi/AIS/blob/main/Lab2/pythonProject/main.py>

# Оценка и интерпретация результатов

## Примеры запросов

В каждой системе будут протестированы два запроса поиска чемпионов. В первом случае ищутся чемпионы с высоким уроном, способностью «one shot» и средней позиции. Во втором случае ищутся чемпионы с высокой выживаемостью для верхней линии.

### Примеры запросов к базе знаний

Оба запроса выполняются с помощью готовых правил:

|  |
| --- |
| 1 ?- findall(Champion, (high\_damage(Champion), can\_one\_shot(Champion), position(Champion, mid)), Result).  Result = [lucian, tristana, ahri, kassadin, zed]. |

Рисунок 14 - Пример первого запроса в Prolog

|  |
| --- |
| 2 ?- findall(Champion, (high\_tankiness(Champion), position(Champion, top)), Result).  Result = [garen, malphite, nasus, singed, malphite, garen, yorick, nasus, darius]. |

Рисунок 15 - Пример второго запроса в Prolog

### Примеры запросов в Protégé

Запросы в Protégé были выполнены с использованием DL Query, что позволило воспользоваться онтологией и подклассами, созданными на основе тех же правил, что и в базе знаний Prolog:

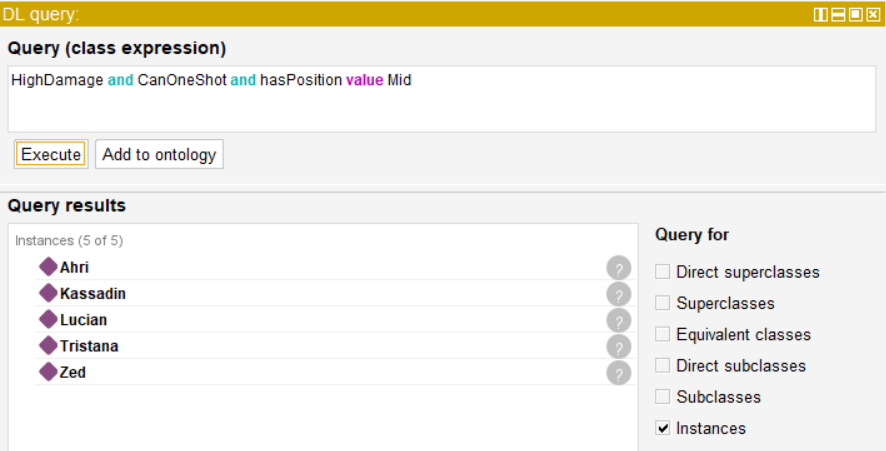


Рисунок 16 - Пример первого запроса в Protégé

A screenshot of a computer

Description automatically generated

Рисунок 17 - Пример второго запроса в Protégé

### Пример использования рекомендательной системы в Python

Пользователь взаимодействует с рекомендательной системой через текстовый интерфейс. Вводя свои предпочтения по роли, позиции и атрибутам чемпиона, пользователь получает список рекомендаций с указанием процента соответствия.

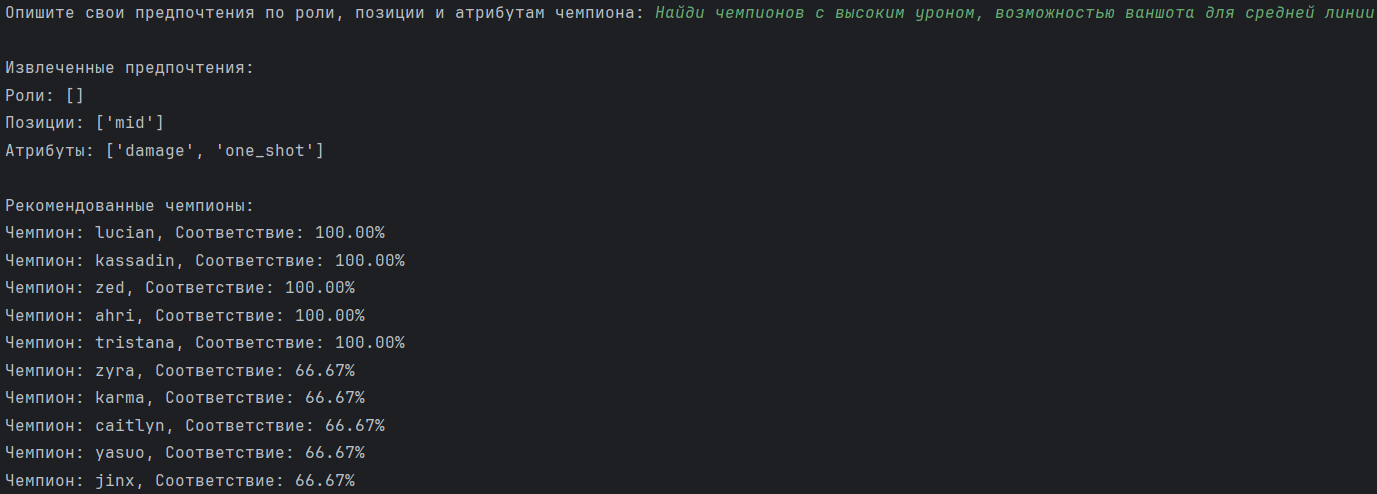


Рисунок 18 – Пример первого запроса в рекомендательной системе

Программа обрабатывает запрос, извлекая ключевые слова, соответствующие предпочтениям пользователя. Далее на основе извлеченных предпочтений система делает запросы к базе знаний и выводит список чемпионов, которые наиболее соответствуют указанным критериям. Каждый чемпион оценивается по проценту совпадения с предпочтениями.

A black screen with white text

Description automatically generated

Рисунок 19 - Пример второго запроса в рекомендательной системе

## Анализ результатов

Результаты запросов к базе знаний и онтологии подтверждают, что система эффективно поддерживает принятие решений, возвращая список чемпионов, соответствующих заданным критериям. Примеры с использованием Prolog и DL Query в Protégé показывают, что обе системы корректно обрабатывают запросы на основе заранее определённых правил.

Рекомендательная система на Python демонстрирует гибкость в адаптации к пользовательским предпочтениям. Механизм подсчёта соответствия позволяет пользователю понять степень релевантности предложенных вариантов, что особенно полезно, когда ни один чемпион не соответствует запросу на 100%, но предложенные варианты всё равно оказываются релевантными.

# Заключение

Разработанная система, баз знаний и онтологий обладают рядом значительных преимуществ. Использование Prolog позволяет эффективно формализовать логику рассуждений и правила вывода, что особенно полезно для систем поддержки принятия решений (СППР). База знаний, построенная с использованием онтологий, улучшает структуризацию данных и позволяет ясно представлять взаимосвязи между объектами.

Протестированные примеры запросов подтвердили, что система способна предоставлять результаты, которые соответствуют пользовательским требованиям, как в контексте Prolog, так и через онтологические запросы.

Возможность интеграции с Python также позволяет легко адаптировать её для более гибких и современных приложений, где требуется анализ данных и взаимодействие с пользователями в режиме реального времени.

Потенциальное применение разработанной системы выходит за рамки игр и рекомендательных систем. Она может быть использована в любой области, где требуется быстрая обработка логических запросов и принятие решений на основе больших объёмов данных, таких как медицина, управление ресурсами, разработка экспертных систем и обучение.