Университет ИТМО

Модуль 1: Базы знаний и онтологии

Дисциплина:
Системы искусственного интеллекта

Выполнил:
Студент группы P3331
Нодири Хисравхон

Проверила:
Преподаватель
Авдюшина Анна Евгеньевна

Санкт-Петербург, 2024

Содержание

1 Введение

В рамках модуля "Базы знаний и онтологии"были выполнены две лабораторные работы. Первая работа посвящена созданию базы знаний в языке программирования Prolog и выполнению различных запросов к ней. Вторая работа направлена на создание онтологии на основе базы знаний и разработку системы поддержки принятия решений с использованием этой онтологии.

2 Лабораторная работа 1: Создание базы знаний и выполнение запросов в Prolog

2.1 Цель работы

Целью данной работы является изучение основ логического программирования на языке Prolog, создание базы знаний с фактами и правилами, а также выполнение различных запросов к этой базе знаний.

2.2 Задание

- Создать базу знаний, включающую:
  - Не менее 20 фактов с одним аргументом.
  - 10-15 фактов с двумя аргументами.
  - 5-7 правил.
- Написать несколько запросов разной сложности к базе знаний.

2.3 Реализация

2.3.1 Создание базы знаний

В качестве предметной области для базы знаний выбрана тематика видеоигр.

Факты с одним аргументом Факты с одним аргументом описывают жанры видеоигр.

Листинг 1: Факты с одним аргументом

```
% (
              )
genre("The Witcher 3", rpg).
genre("Dark Souls", rpg).
genre("Overwatch", fps).
genre("Counter-Strike", fps).
genre("FIFA", sports).
genre("NBA 2K", sports).
genre("Civilization VI", strategy).
genre("Age of Empires II", strategy).
genre("Minecraft", sandbox).
genre("Terraria", sandbox).
genre("League of Legends", moba).
genre("Dota 2", moba).
genre("Fortnite", battle_royale).
genre("PUBG", battle_royale).
genre("Red Dead Redemption 2", action_adventure).
genre("GTA V", action_adventure).
genre("Super Mario Odyssey", platformer).
genre("Hades", roguelike).
genre("Among Us", party).
genre("Fall Guys", party).
genre("Assassin's Creed", action_adventure).
genre("Call of Duty", fps).
genre("Starcraft", strategy).
genre("Apex Legends", battle_royale).
```

```
genre("The Sims", simulation).
genre("Cities: Skylines", simulation).
genre("Rocket League", sports).
```

Факты с двумя аргументами Факты с двумя аргументами связывают игры с их разработчиками.

Листинг 2: Факты с двумя аргументами

```
% (
)
developer("The Witcher 3", "CD Projekt Red").
developer("Dark Souls", "FromSoftware").
developer("Overwatch", "Blizzard").
developer("Counter-Strike", "Valve").
developer("FIFA", "EA Sports").
developer("NBA 2K", "2K Sports").
developer("Civilization VI", "Firaxis Games").
developer("Age of Empires II", "Ensemble Studios").
developer("Minecraft", "Mojang").
developer("Terraria", "Re-Logic").
developer("League of Legends", "Riot Games").
developer("Dota 2", "Valve").
developer("Fortnite", "Epic Games").
developer("PUBG", "PUBG Corporation").
developer("Red Dead Redemption 2", "Rockstar Games").
developer("GTA V", "Rockstar Games").
developer("Super Mario Odyssey", "Nintendo").
developer("Assassin's Creed", "Ubisoft").
developer("Call of Duty", "Activision").
developer("Starcraft", "Blizzard").
developer("Apex Legends", "Respawn Entertainment").
developer("The Sims", "EA").
developer("Cities: Skylines", "Colossal Order").
developer("Rocket League", "Psyonix").
```

Факты с тремя аргументами Факты с тремя аргументами описывают платформы, на которых доступны игры.

Листинг 3: Факты с тремя аргументами

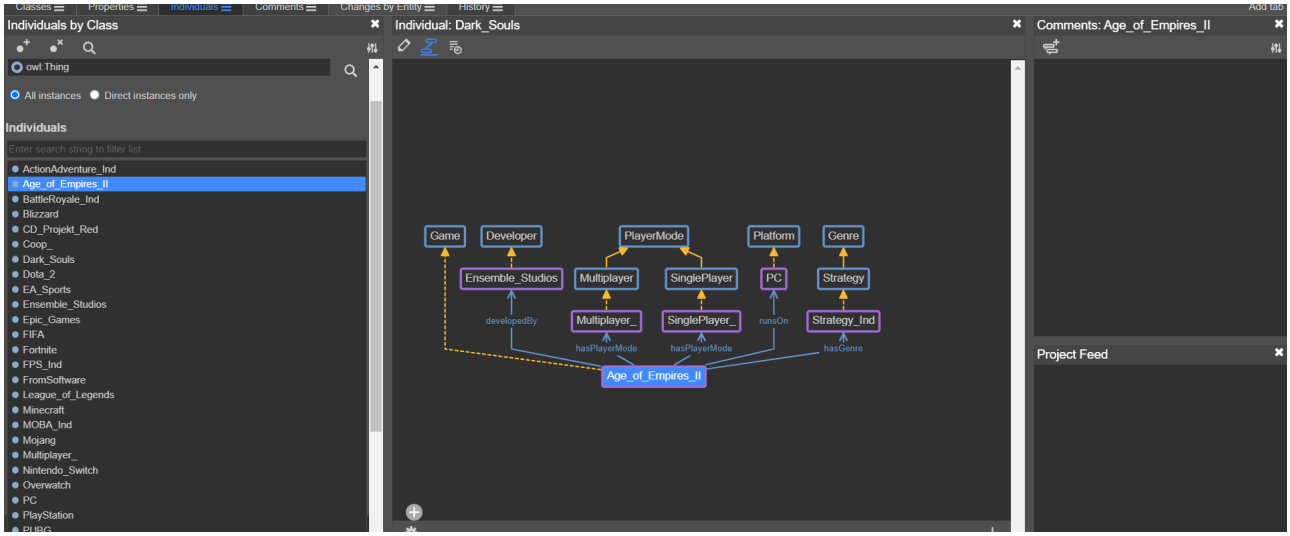
```
% ( )
platform("The Witcher 3", "PC", "PS4").
platform("Dark Souls", "PC", "PS4").
platform("Overwatch", "PC", "PS4").
platform("Counter-Strike", "PC").
platform("FIFA", "PC", "PS4").
platform("NBA 2K", "PC", "PS4").
platform("Civilization VI", "PC").
platform("Age of Empires II", "PC").
platform("Minecraft", "PC", "Xbox").
platform("Terraria", "PC", "PS4").
platform("League of Legends", "PC").
platform("Dota 2", "PC").
platform("Fortnite", "PC", "PS4").
platform("PUBG", "PC", "PS4").
platform("Red Dead Redemption 2", "PC", "PS4").
platform("GTA V", "PC", "PS4").
platform("Super Mario Odyssey", "Switch").
platform("Assassin's Creed", "PC", "PS4").
platform("Call of Duty", "PC", "PS4").
platform("Starcraft", "PC").
platform("Apex Legends", "PC", "PS4").
platform("The Sims", "PC").
```

```
platform("Cities: Skylines", "PC").
platform("Rocket League", "PC", "PS4").
```

Правила Созданы правила для определения различных свойств игр и их связей.

Листинг 4: Правила

```
%
is_rpg_game(X) :- genre(X, rpg).
is_fps_game(X) :- genre(X, fps).
is_strategy_game(X) :- genre(X, strategy).
is_battle_royale_game(X) :- genre(X, battle_royale).
is_sports_game(X) :- genre(X, sports).
is_sandbox_game(X) :- genre(X, sandbox).
is_simulation_game(X) :- genre(X, simulation).
is_action_adventure_game(X) :- genre(X, action_adventure).

%                                                    ,
same_developer(X, Y) :- developer(X, Z), developer(Y, Z).

%
multigenre_game(X) :- genre(X, Genre1), genre(X, Genre2), Genre1 \= Genre2.

%
popular_game(X) :- genre(X, rpg); genre(X, fps); genre(X, battle_royale).

%
                                ,
strategic_developer(Y) :- developer(X, Y), genre(X, strategy).

%
available_on_pc(X) :- platform(X, "PC", _).
available_on_console(X) :- platform(X, "PS4", _).

%
multiplayer_game(X) :- genre(X, moba); genre(X, fps); genre(X, battle_royale).

%                                                      ,
is_simulator(X) :- genre(X, simulation).

%                                                    ,
    "Blizzard"
blizzard_game(X) :- developer(X, "Blizzard").

%                                                              Nintendo Switch
available_on_switch(X) :- platform(X, "Switch", _).
```



2.3.2 Выполнение запросов

Простые запросы

Листинг 5: Простой запрос: поиск всех RPG игр

```
?- is_rpg_game(Game).
```

Результат:

```
Game = "The Witcher 3";
Game = "Dark Souls";
```

Запросы с логическими операторами

Листинг 6: Запрос: игры жанра RPG или FPS

```
?- genre(Game, rpg); genre(Game, fps).
```

Результат:

```
Game = "The Witcher 3";
Game = "Dark Souls";
Game = "Overwatch";
Game = "Counter-Strike";
Game = "Call of Duty";
```

Запросы с переменными

Листинг 7: Запрос: разработчики стратегических игр

```
?- genre(Game, strategy), developer(Game, Dev).
```

Результат:

```
Game = "Civilization VI", Dev = "Firaxis Games";
Game = "Age of Empires II", Dev = "Ensemble Studios";
Game = "Starcraft", Dev = "Blizzard";
```

Запросы, требующие выполнения правил

Листинг 8: Запрос: игры, доступные на ПК

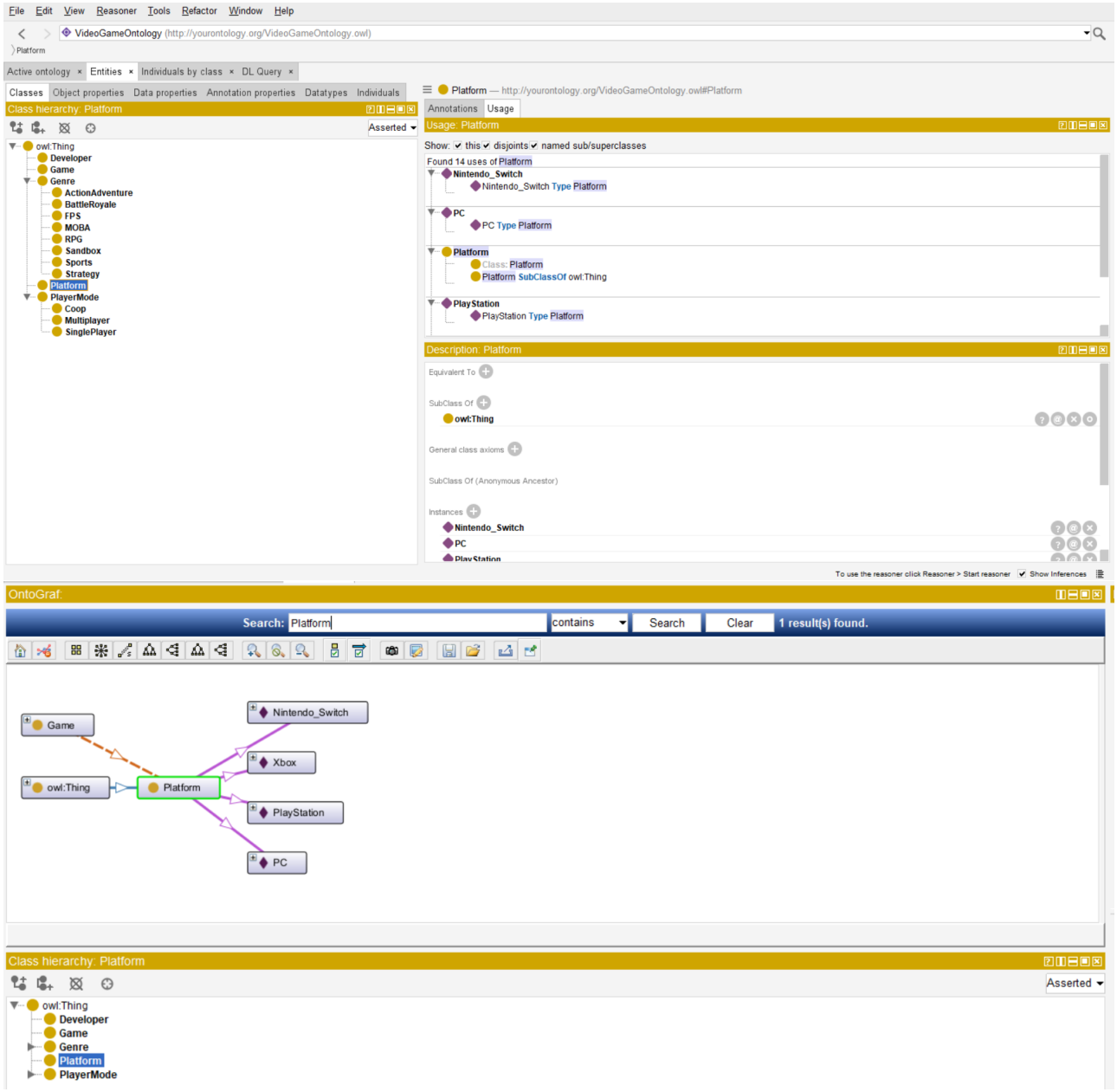
```
?- available_on_pc(Game).
```

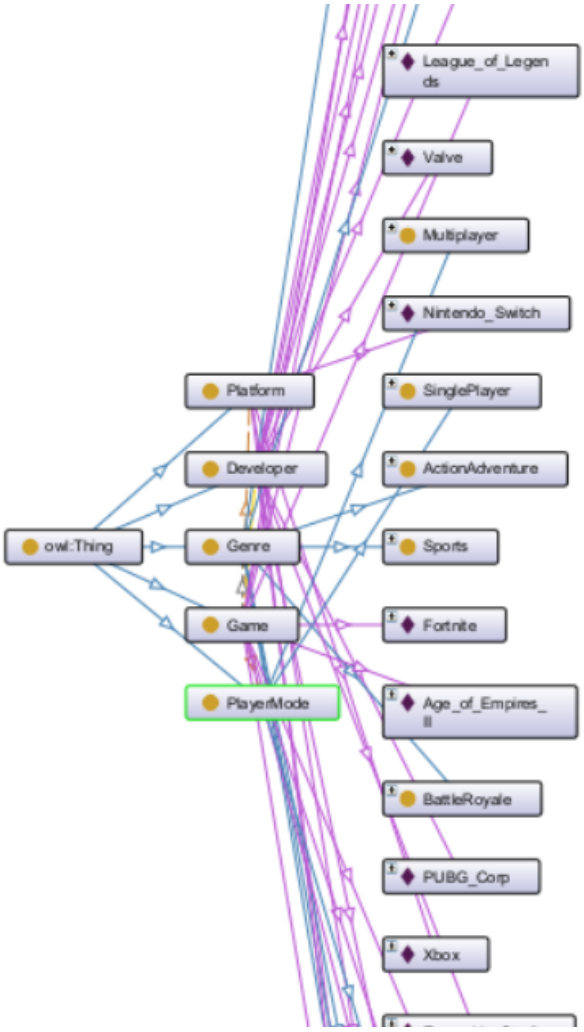
Результат (список всех игр, доступных на ПК).

2.4 Выводы

В результате выполнения первой лабораторной работы была создана база знаний в Prolog, описывающая различные аспекты видеоигр. Были реализованы разнообразные запросы, демонстрирующие работу с фактами, логическими операторами, переменными и правилами.

2.5 Создание онтологии в Protege




League_of_Legen
ds
Valve
Multiplayer
Nintendo_Switch
Platform
SinglePlayer
Developer
ActionAdventure
owl:Thing
Genre
Sports
Game
Fortnite
PlayerMode
Age_of_Empires_
II
BattleRoyale
PUBG_Corp
Xbox

3 Лабораторная работа 2: Создание онтологии в Protege и разработка системы поддержки принятия решений

3.1 Цель работы

Целью данной работы является преобразование базы знаний из Prolog в онтологию с использованием инструмента Protege, а также разработка программы для предоставления рекомендаций на основе этой онтологии.

3.2 Задание

- Создать онтологию, отражающую факты и отношения из базы знаний Prolog.
- Разработать программу, которая взаимодействует с онтологией и предоставляет рекомендации пользователю на основе введённых данных.

3.3 Реализация онтологии

3.3.1 Создание онтологии

Онтология создана с использованием библиотеки `owlready2` в Python и сохранена в файл `VideoGameOntology.owl`.

Основные классы и свойства

- **Классы:** Game, Genre, Developer, Platform, PlayerMode.
- **Свойства:**
  - hasGenre (Game → Genre)
  - developedBy (Game → Developer)
  - runsOn (Game → Platform)
  - hasPlayerMode (Game → PlayerMode)

3.4 Разработка системы поддержки принятия решений

3.4.1 Описание программы

Программа на Python позволяет пользователю ввести информацию о своих предпочтениях и предоставляет рекомендации игр на основе онтологии.

```python
from owlready2 import get_ontology, sync_reasoner_pellet
import re

onto = get_ontology("VideoGameOntology.owl").load()

with onto:
    sync_reasoner_pellet()

def parse_user_input(input_str):
    age = None
    preferences = []

    age_match = re.search(r"Мне (\d+) лет", input_str)
    if age_match:
        age = int(age_match.group(1))

    prefs_match = re.search(r"мне нравятся: (.+)", input_str)
    if prefs_match:
        prefs_str = prefs_match.group(1)
        preferences = [pref.strip() for pref in prefs_str.split(',')]
```

```
    return age, preferences

def parse_platform_input(input_str, valid_platforms):
    platforms = [plat.strip() for plat in input_str.split(',')]
    user_platforms = [plat for plat in platforms if plat in valid_platforms]
    return user_platforms

def parse_player_mode_input(input_str, valid_player_modes):
    modes = [mode.strip() for mode in input_str.split(',')]
    user_modes = [mode for mode in modes if mode in valid_player_modes]
    return user_modes

def main():
    valid_genres = ['RPG', 'FPS', 'Strategy', 'ActionAdventure', 'MOBA', 'BattleRoyale', 'Sandbox', 'Spor

    print(f"Доступные жанры: {', '.join(valid_genres)}")
    user_input = input("Введите информацию о себе и своих предпочтениях (например, 'Мне 13 лет, мне нравя

    age, preferences = parse_user_input(user_input)

    if age is None or not preferences:
        print("Пожалуйста, введите информацию в правильном формате.")
        return

    print("Возраст пользователя:", age)
    print("Предпочтения пользователя:", preferences)

    genre_mapping = {
        'RPG': onto.RPG_Ind,
        'FPS': onto.FPS_Ind,
        'Strategy': onto.Strategy_Ind,
        'ActionAdventure': onto.ActionAdventure_Ind,
        'MOBA': onto.MOBA_Ind,
        'BattleRoyale': onto.BattleRoyale_Ind,
        'Sandbox': onto.Sandbox_Ind,
        'Sports': onto.Sports_Ind
    }

    user_genres = [pref for pref in preferences if pref in valid_genres]

    if not user_genres:
        print("К сожалению, ваши предпочтения не совпадают с доступными жанрами.")
        return
    else:
        print("Жанры пользователя:", user_genres)

    valid_platforms = ['PC', 'PlayStation', 'Xbox', 'Nintendo_Switch']
    print(f"Доступные платформы: {', '.join(valid_platforms)}")
    platform_input = input("На каких платформах вы предпочитаете играть? (например, 'PC, PlayStation'): "
    user_platforms = parse_platform_input(platform_input, valid_platforms)

    if not user_platforms:
        print("Вы не указали доступные платформы. Будут учитываться все платформы.")

    platform_mapping = {
        'PC': onto.PC,
        'PlayStation': onto.PlayStation,
```

```
        'Xbox': onto.Xbox,
        'Nintendo_Switch': onto.Nintendo_Switch
    }

    valid_player_modes = ['SinglePlayer', 'Multiplayer', 'Coop']
    print(f"Доступные режимы игры: {', '.join(valid_player_modes)}")
    player_mode_input = input("Какой режим игры вы предпочитаете? (например, 'SinglePlayer, Multiplayer')
    user_player_modes = parse_player_mode_input(player_mode_input, valid_player_modes)

    if not user_player_modes:
        print("Вы не указали доступные режимы игры. Будут учитываться все режимы.")

    player_mode_mapping = {
        'SinglePlayer': onto.SinglePlayer_,
        'Multiplayer': onto.Multiplayer_,
        'Coop': onto.Coop_
    }

    def find_matching_games(consider_platforms=True, consider_player_modes=True):
        matching_games = set()
        for genre_name in user_genres:
            genre_individual = genre_mapping[genre_name]
            games_with_genre = onto.search(hasGenre=genre_individual)
            matching_games.update(games_with_genre)

        final_matching_games = set()
        for game in matching_games:
            if consider_platforms and user_platforms:
                game_platforms = [plat.name for plat in game.runsOn]
                if not any(plat in game_platforms for plat in user_platforms):
                    continue
            if consider_player_modes and user_player_modes:
                game_player_modes = [mode.name for mode in game.hasPlayerMode]
                if not any(mode in game_player_modes for mode in user_player_modes):
                    continue
            final_matching_games.add(game)
        return final_matching_games

    games = find_matching_games()
    if len(games) >= 2:
        recommended_games = games
    else:
        print("\nИщем игры, соответствующие вашим жанрам и платформам, независимо от режима игры...")
        games = find_matching_games(consider_player_modes=False)
        if len(games) >= 2:
            recommended_games = games
        else:
            print("\nИщем игры, соответствующие вашим жанрам, независимо от платформ и режима игры...")
            games = find_matching_games(consider_platforms=False, consider_player_modes=False)
            if len(games) >= 2:
                recommended_games = games
            else:
                print("\nИщем любые игры, соответствующие вашим жанрам...")
                recommended_games = set()
                for genre_name in user_genres:
                    genre_individual = genre_mapping[genre_name]
                    games_with_genre = onto.search(hasGenre=genre_individual)
                    recommended_games.update(games_with_genre)
```

```
                if len(recommended_games) < 2:
                    print("\nК сожалению, не удалось найти достаточно игр по вашим предпочтениям. Рекомен
                    recommended_games = set(onto.Game.instances())  # All games

    recommended_games = list(recommended_games)[:2]

    if recommended_games:
        print("\nМы рекомендуем вам следующие игры:")
        for game in recommended_games:
            print("- {}".format(game.name.replace('_', ' ')))
    else:
        print("К сожалению, мы не нашли игр, соответствующих вашим предпочтениям.")

if __name__ == "__main__":
    main()
```

3.4.2 Логика работы программы

1. Пользователь вводит информацию о себе, например: "Мне 25 лет, мне нравятся: RPG, FPS".
2. Программа парсит ввод и извлекает возраст и предпочтения.
3. Пользователь выбирает предпочитаемые платформы и режимы игры.
4. Программа ищет в онтологии игры, соответствующие критериям.
5. Если найдено менее двух игр, программа ослабляет критерии поиска.
6. Выводится список рекомендованных игр.

3.4.3 Пример работы программы

```
Доступные жанры: RPG, FPS, Strategy, ActionAdventure, MOBA, BattleRoyale, Sandbox, Sports
Введите информацию о себе и своих предпочтениях (например, 'Мне 13 лет, мне нравятся: RPG, FPS'):
-> Мне 18 лет, мне нравятся: Sandbox, Sports
Возраст пользователя: 18
Предпочтения пользователя: ['Sandbox', 'Sports']
Жанры пользователя: ['Sandbox', 'Sports']
Доступные платформы: PC, PlayStation, Xbox, Nintendo_Switch
На каких платформах вы предпочитаете играть? (например, 'PC, PlayStation'):
-> PC, Xbox
Доступные режимы игры: SinglePlayer, Multiplayer, Coop
Какой режим игры вы предпочитаете? (например, 'SinglePlayer, Multiplayer'):
-> Multiplayer

Мы рекомендуем вам следующие игры:
- Minecraft
- FIFA
```

3.5 Выводы

В результате выполнения второй лабораторной работы была создана онтология, отражающая базу знаний из Prolog. Разработанная программа успешно предоставляет рекомендации пользователю на основе его предпочтений, демонстрируя практическое применение онтологий в системах поддержки принятия решений.

4 Заключение

В ходе выполнения лабораторных работ были приобретены практические навыки создания баз знаний в Prolog и онтологий в Protege. Разработка системы поддержки принятия решений показала, как знания, структурированные в онтологии, могут использоваться для предоставления пользователю полезной информации на основе его запросов.

5 Список литературы

1. Bratko, I. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 2001.
2. *OWL Web Ontology Language Reference*. `https://www.w3.org/TR/owl-ref/`
3. *Protege Ontology Editor*. `https://protege.stanford.edu/`
4. *owlready2 Documentation*. `https://owlready2.readthedocs.io/en/latest/`