Университет ИТМО
Факультет программной инженерии и компьютерной техники

Системы искусственного интеллекта. Лабораторные работы 1-3. Базы знаний и онтологии

Группа : P33131
Студент: Смирнов Виктор Игоревич
Тема : Игра Terraria

2023

Ключевые слова

База знаний, Онтология.

Содержание

1 Введение

1.1 Цели

Целью лабораторных работ первого модуля была разработка базы знаний на базе языка Prolog, а также фронтэнда к ней на Python для взаимодействия пользователя с системой. Прежде всего это был просто образовательный проект с целью ознакомления с онтологиями.

1.2 Значимость

Значимость заключается в полученных мною знаний и создании автоматиеского чат-бота для получения персональных рекомендаций по игре Terraria.

2 Анализ требований

2.1 Требования к фронтэнду

1. Фронтэнд должен обрабатывать запросы пользователя и выдавать на них ответы, соответвующие состоянию базы знаний.
2. Фронтэнд должен быть устойчив к неккоректному пользовательскому вводу.
3. Фронтэнд должен быть реализован на языке Python.
4. Фронтэнд должен иметь CLI интерфейс.
5. Фронтэнд должен инициировать взаимодействие с базой знаний.

2.2 Требования к базе знаний

1. База зананий должна быть реализована на базе системы Prolog.
2. База знаний должна содержать информацию об игре Terraria.
3. База знаний должна содержать факты и правила различной сложности.

3 Изучение основных концепций и инструментов

3.1 Обзор основных концепций баз знаний и онтологий

По определению из википедии база знаний - база данных, содержащая правила вывода и информацию о знаниях в некоторой предметной области.

Онтология – иерархический способ представления в базе знаний набора понятий и их связей.

Важно, что онтология сама по себе содержит только абстрактные понятия и связи. Она становится базой знаний, только когда там появляются конкретные объекты.

Базы знаний способны делать автоматические умозаключения об уже имеющихся или вновь вводимых фактах и тем самым производить как бы осмысленную обработку информации.

База знаний содержит правила вывода – логические конструкции, определяющие отношения между фактами и позволяющие делать выводы.

База знаний обрабатывает запросы к себе, выдавая информацию, удовлетворяющую заданным в запросе условиям.

Онтологии состоят из

1. экземпляров - конкретных или абстрактных объектов,

2. понятий - абстрактных наборов объектов,

3. атрибутов - некоторых значений, связанных с конкретным экземпляром,

4. отношений - завсимостей между объектами онтологии.

Онтология позволяет формализовать факты и правила и обрабатывать их автоматически, например, в системах искусственного интеллекта или для анализа данных.

Онтология задает смысл данных, а база знаний их хранит и предоставляет доступ.

3.2 Prolog и его возможностей для разработки систем искусственного интеллекта

Prolog (Programming in Logic) - это декларативный язык программирования, который используется для решения задач искусственного интеллекта и логического программирования. Prolog основан на логике первого порядка и предназначен для решения задач, в которых логические отношения и правила играют ключевую роль.

3.3 Инструменты и библиотеки, подходящие для работы с базами знаний и онтологиями на Prolog

.

Инструмент для Базы знаний - https://swish.swi-prolog.org/.
Инструмент для Онтологий - https://protege.stanford.edu/.
Инструмент для взаимодействия с базой знаний - https://github.com/yuce/pyswip/.

4 Создание правил и логики вывода для принятия решений на основе базы знаний и онтологии

4.1 Создание правил и логики вывода для принятия решений на основе базы знаний и онтологии

Была составлена база знаний на языке Prolog.

```
:- module(achievements, [achived/1]).

:- use_module(ore).
:- use_module(pickaxe).
:- use_module(available).
:- use_module(reachable).

achived('Can break all found existing ores') :-
    forall(
        found(existing(ore(Ore))),
        available(ore(Ore))
    ).

achived('Theoretically can craft both Nightmare and Deathbringer') :-
    reachable(pickaxe('Nightmare')),
    reachable(pickaxe('Deathbringer')).

achived('Is ready to hardmode') :-
    available(pickaxe('Molten')).
```

Листинг 1: Достижения игрока

```
:- module(available, [available/1]).

:- use_module(progress).
:- use_module(requirement).

available(Item) :-
    Item, found(Item).

available(ore(Ore)) :-
```

```
    found(existing(ore(Ore))),
    ore_power(ore(Ore), OrePower),
    findall(Pickaxe, (
        pickaxe(Pickaxe),
        pickaxe_power(pickaxe(Pickaxe), PickaxePower),
        (PickaxePower < OrePower -> false; true),
        found(pickaxe(Pickaxe))
    ), Pickaxes),
    member(_, Pickaxes).

available(Item) :-
    Item,
    findall(R, requirement(Item, R), Requirements),
    member(_, Requirements),
    forall(member(R, Requirements), found(R)).
```

Листинг 2: Доступность предмета игроку в один шаг

```
:- module(reachable, [reachable/1]).

:- use_module(requirement).
:- use_module(available).

:- use_module(pickaxe).

reachable(Item) :-
    Item, available(Item).

reachable(pickaxe_with_power(TargetPower)) :-
    findall(Pickaxe, (
        pickaxe(Pickaxe),
        pickaxe_power(pickaxe(Pickaxe), PickaxePower),
        (PickaxePower < TargetPower -> false; true)
    ), Pickaxes),
    member(FirstPickaxe, Pickaxes),
    reachable(pickaxe(FirstPickaxe)).

reachable(Item) :-
    Item,
    findall(R, requirement(Item, R), Requirements),
    member(_, Requirements),
    forall(member(R, Requirements), reachable(R)).
```

Листинг 3: Доступность предмета игроку в несколько шагов

```
:- module(evil, [evil_biome/1]).

evil_biome('Corruption').
evil_biome('Crimson').
```

Листинг 4: Злые биомы Террарии

```
:- module(ore, [ore/1, ore_power/2, existing/1]).

ore('Copper').
ore('Tin').
ore('Iron').
ore('Lead').
ore('Silver').
ore('Tungsten').
ore('Gold').
ore('Platinum').
ore('Meteorite').
ore('Demonite').
ore('Crimtane').
ore('Obsidian').
ore('Hellstone').
ore('Cobalt').
ore('Palladium').
ore('Mythril').
ore('Orichalcum').
ore('Adamantite').
ore('Titanium').
```

```
ore('Chlorophyte').
ore('Luminite').

existing(ore(Ore)) :- ore(Ore).

ore_power(ore('Copper'), 1).
ore_power(ore('Tin'), 1).
ore_power(ore('Iron'), 2).
ore_power(ore('Lead'), 2).
ore_power(ore('Silver'), 3).
ore_power(ore('Tungsten'), 3).
ore_power(ore('Gold'), 4).
ore_power(ore('Platinum'), 4).
ore_power(ore('Meteorite'), 4).
ore_power(ore('Demonite'), 5).
ore_power(ore('Crimtane'), 5).
ore_power(ore('Obsidian'), 5).
ore_power(ore('Hellstone'), 6).
ore_power(ore('Cobalt'), 7).
ore_power(ore('Palladium'), 7).
ore_power(ore('Mythril'), 8).
ore_power(ore('Orichalcum'), 9).
ore_power(ore('Adamantite'), 10).
ore_power(ore('Titanium'), 11).
```

Листинг 5: Руды Террарии

```
:- module(pickaxe, [
    pickaxe/1,
    pickaxe_power/2,
    pickaxe_with_power/1
]).

pickaxe('Cactus').
pickaxe('Copper').
pickaxe('Tin').
pickaxe('Iron').
pickaxe('Lead').
pickaxe('Silver').
pickaxe('Tungsten').
pickaxe('Gold').
pickaxe('Platinum').
pickaxe('Nightmare').
pickaxe('Deathbringer').
pickaxe('Molten').
pickaxe('Cobalt').
pickaxe('Palladium').
pickaxe('Mythril').
pickaxe('Orichalcum').
pickaxe('Adamantite').
pickaxe('Titanium').

pickaxe_power(pickaxe('Cactus'), 2).
pickaxe_power(pickaxe('Copper'), 2).
pickaxe_power(pickaxe('Tin'), 2).
pickaxe_power(pickaxe('Iron'), 3).
pickaxe_power(pickaxe('Lead'), 3).
pickaxe_power(pickaxe('Silver'), 4).
pickaxe_power(pickaxe('Tungsten'), 4).
pickaxe_power(pickaxe('Gold'), 5).
pickaxe_power(pickaxe('Platinum'), 5).
pickaxe_power(pickaxe('Nightmare'), 6).
pickaxe_power(pickaxe('Deathbringer'), 6).
pickaxe_power(pickaxe('Molten'), 7).
pickaxe_power(pickaxe('Cobalt'), 8).
pickaxe_power(pickaxe('Palladium'), 8).
pickaxe_power(pickaxe('Mythril'), 9).
pickaxe_power(pickaxe('Orichalcum'), 9).
pickaxe_power(pickaxe('Adamantite'), 10).
pickaxe_power(pickaxe('Titanium'), 10).

pickaxe_with_power(1).
pickaxe_with_power(2).
```

```
pickaxe_with_power(3).
pickaxe_with_power(4).
pickaxe_with_power(5).
pickaxe_with_power(6).
pickaxe_with_power(7).
pickaxe_with_power(8).
```

Листинг 6: Кирки Террарии

```
:- module(progress, [found/1]).

:- use_module(pickaxe).
:- use_module(ore).

:- dynamic(found/1).

% found(evil_biome('Crimson')).
%
% found(existing(ore('Crimtane'))) :-
%     found(evil_biome('Crimson')).
% found(existing(ore('Demonite'))) :-
%     found(evil_biome('Corruption')).
%
% found(existing(ore('Demonite'))).
%
% found(existing(ore('Tin'))).
% found(existing(ore('Iron'))).
% found(existing(ore('Silver'))).
% found(existing(ore('Gold'))).
%
% found(existing(ore('Hellstone'))).
%
% found(ore('Copper')).
% found(pickaxe('Copper')).
% found(pickaxe('Gold')).
% found(pickaxe('Deathbringer')).
% found(pickaxe('Molten')).
%

found(pickaxe_with_power(Power)) :-
    number(Power),
    found(pickaxe(Pickaxe)),
    pickaxe_power(pickaxe(Pickaxe), PickaxePower),
    (PickaxePower < Power -> false ; true).
```

Листинг 7: Прогресс игрока

```
:- module(requirement, [requirement/2]).

:- use_module(pickaxe).
:- use_module(ore).
:- use_module(evil).

requirement(pickaxe('Copper'), ore('Copper')).
requirement(pickaxe('Tin'), ore('Tin')).
requirement(pickaxe('Iron'), ore('Iron')).
requirement(pickaxe('Lead'), ore('Lead')).
requirement(pickaxe('Silver'), ore('Silver')).
requirement(pickaxe('Tungsten'), ore('Tungsten')).
requirement(pickaxe('Gold'), ore('Gold')).
requirement(pickaxe('Platinum'), ore('Platinum')).
requirement(pickaxe('Nightmare'), ore('Demonite')).
requirement(pickaxe('Deathbringer'), ore('Crimtane')).
requirement(pickaxe('Molten'), ore('Hellstone')).
requirement(pickaxe('Cobalt'), ore('Cobalt')).
requirement(pickaxe('Palladium'), ore('Palladium')).
requirement(pickaxe('Mythril'), ore('Mythril')).
requirement(pickaxe('Orichalcum'), ore('Orichalcum')).
requirement(pickaxe('Adamantite'), ore('Adamantite')).
requirement(pickaxe('Titanium'), ore('Titanium')).

requirement(ore(Ore), pickaxe_with_power(PickaxePower)) :-
    ore_power(ore(Ore), PickaxePower).
```

```
27
28 requirement(ore(Ore), existing(ore(Ore))).
```

Листинг 8: Требования для добычи предметов

4.2 Тестирование и отладка системы, обеспечение ее функциональности и эффективности

Была написана система принятия решений в виде CLI чат-бота на языке Python.

Запросы к базе зананий выполнялись с помощью библиотеки pyswip.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/prolog.py

Также в Python была выражена часть предметной области для функции для построения запросов к серверу Prolog и удобного тестирования.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/dsl.py

Для разбора ответов сервера Prolog был сделан простенький парсер выражений на основе Python Lark.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/parse.py

На базе описанного выше удалось спрятать Prolog за слоем бизнес логики.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/terraria.py

Слой бизнес логики используется слоем принятия и обработки текстовых запросов.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/feedback.py

Ну и вишенкой на торте стало CLI приложение.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/args.py

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/main.py

Программное обеспечение всегда страшно разрабатывать без тестов, поэтому без них не обошлось.

Было проведено модульное тестирование парсера.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/test$_{p}arse.py$

Также слегка проверен был слой бизнес-логики.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/test$_{t}erraria.py$

Чтобы быть более уверенным в корректной обработки даже неправильных запросов пользователя была реализована некоторая пародия на фаззинг-тестирование.

https://github.com/vityaman-edu/ai-prolog/blob/trunk/src/test$_{f}uzzing.py$

4.3 Пример вывода программы

https://github.com/vityaman-edu/ai-prolog/blob/trunk/res/example.txt

5 Оценка и интерпретация результатов

Наша система соответвует заявленным требованиям.

Цель была достигнута - мы разработали CLI чат-бота предоставляющего доступ к базе знаний на языке Prolog, он помогает человеку играть в игру Terraria, советуя, какие шаги предпринять и давая комментарии по поводу прогресса в игре.

В дальнейшем можно расширять базу знаний, добавлять новые аспекты игры. Расширить вариативность как ввода, так и вывода. Сделать бота более гибким и похожим на человека. Также следует исправить существующие проблемы приложения (баги) и повысить тестовое покрытие, доработать фаззинг.

6 Заключение

Разработанное приложение поможет игрокам в Terraria понять правила игры и получить подсказки.

Список литературы

[1] https://ru.wikipedia.org/wiki/

[2] https://ru.wikipedia.org/wiki/

[3] https://www.swi-prolog.org/

[4] https://protege.stanford.edu/

[5] https://github.com/yuce/pyswip