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.

```python
from typing import List
from log import Log as log
from pyswip import Prolog as SWIProlog
from pyswip.core import cleanupProlog


class Prolog:
    def __init__(self, preload: List[str]):
        self.__pl: SWIProlog = None
        self.__preload = preload

    def query(self, query: str):
        log.debug(query)
        return self.__pl.query(query)

    def is_proven(self, query: str) -> bool:
        log.debug(query)
        return bool(list(self.query(query)))

    def assume(self, fact: str):
        log.debug(fact)
        self.__pl.assertz(fact)

    def __enter__(self):
        self.__pl = SWIProlog()
        for file in self.__preload:
            self.__pl.consult(file)
        return self

    def __exit__(self, *args, **kvargs):
        # FIXME: segfault on duoble call inside a one process
        cleanupProlog()
```

Листинг 9: Слой взаимодействия с pyswip

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

```python
from enum import Enum

from parse import Expr


EvilBiome = Enum('EvilBiome', [
    'Crimson',
    'Corruption',
])
EvilBiome.__str__ = lambda self: f"evil_biome('{self.name}')"
EvilBiome.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"

Ore = Enum('Ore', [
    'Copper',
    'Tin',
    'Iron',
    'Lead',
    'Silver',
    'Tungsten',
    'Gold',
    'Platinum',
    'Meteorite',
    'Demonite',
```

```
    'Crimtane',
    'Obsidian',
    'Hellstone',
    'Cobalt',
    'Palladium',
    'Mythril',
    'Orichalcum',
    'Adamantite',
    'Titanium',
    'Chlorophyte',
    'Luminite',
])
Ore.__str__ = lambda self: f"ore('{self.name}')"
Ore.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"


Pickaxe = Enum('Pickaxe', [
    'Cactus',
    'Copper',
    'Tin',
    'Iron',
    'Lead',
    'Silver',
    'Tungsten',
    'Gold',
    'Platinum',
    'Nightmare',
    'Deathbringer',
    'Molten',
    'Cobalt',
    'Palladium',
    'Mythril',
    'Orichalcum',
    'Adamantite',
    'Titanium',
])
Pickaxe.__str__ = lambda self: f"pickaxe('{self.name}')"
Pickaxe.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"

Item = Pickaxe | Ore


NatureObject = EvilBiome | Ore

def existing(thing: object) -> str:
    return f'existing({thing})'


def found(thing: object) -> str:
    return f'found({thing})'


def available(item: Item) -> str:
    return f'available({item})'
```

Листинг 10: Доменная область

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

```
import sys
from typing import Generator, Optional, List
from dataclasses import dataclass

from lark import Lark, Transformer, v_args
from lark.ast_utils import Ast as LarkAST, WithMeta, create_transformer
from lark.tree import Meta


class AST(LarkAST):
    pass


@dataclass
```

```
class Name(AST):
    name: str


@dataclass(unsafe_hash=True)
class Expr(AST):
    name: Name
    expr: Optional['Expr'] = None

    @property
    def names(self) -> Generator[str, None, None]:
        node = self
        while node is not None:
            name = node.name
            s = name[0]
            if s != s.lower():
                name = f"'{name}'"
            yield name
            node = node.expr

    def __str__(self) -> str:
        result = ''

        i = 0
        for name in self.names:
            if 1 <= i:
                result += '('
            result += name
            i += 1
        result += ')' * (i - 1)

        return result

    @property
    def is_existing(self) -> bool:
        return 'existing' in self.names

    def __repr__(self) -> str:
        names = list(self.names)
        name = names[-1]
        type = names[-2]
        return f'{name} {type}'.replace('\'', '')

    @classmethod
    def parse(cls, input: str) -> 'Expr':
        result = transformer.transform(lark.parse(input))
        prev = result
        if isinstance(prev, str):
            return Expr(prev)
        curr = prev.expr
        while not isinstance(curr, str):
            prev = curr
            curr = curr.expr
        prev.expr = Expr(curr)
        return result


class ToAst(Transformer):
    def WORD(self, x):
        return x.value

    @v_args(inline=True)
    def start(self, x):
        return x


transformer = create_transformer(sys.modules[__name__], ToAst())

grammar = '''
start: expr

?expr: "'"? ID "'"? "(" expr ")"
     | "'"? ID "'"?
```

```
88
89 %import common.CNAME -> ID
90 %ignore " "
91 '''
92
93 lark = Lark(grammar)
```

Листинг 11: Парсер частного случая выражения на Prolog

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

```
1  from typing import Any, Generator, List, Set
2  from prolog import Prolog
3  from dsl import *
4  from parse import Expr
5  from itertools import chain
6
7
8  class TerrariaException(Exception):
9      pass
10
11
12 class Terraria:
13     def __init__(self, state: Prolog):
14         self._state = state
15
16     def cheat(self, item: Item):
17         self._state.assume(found(item))
18
19     def is_available(self, item: Item) -> bool:
20         return self._state.is_proven(available(item))
21
22     def take(self, item: Item):
23         if not self.is_available(item):
24             raise TerrariaException(
25                 'can not take an item as it is not available')
26         self.cheat(item)
27
28     def explore(self, object: NatureObject):
29         self._state.assume(found(existing(object)))
30
31     def all_items(self) -> Set[Expr]:
32         def generate():
33             for type in ('ore', 'pickaxe'):
34                 for item in self._state.query(f'{type}(A)'):
35                     item = Expr.parse(item['A'])
36                     item = Expr(type, item)
37                     yield item
38         return set(generate())
39
40     def inventory(self) -> Set[Expr]:
41         def generate():
42             for item in self._state.query(found('Item')):
43                 item = Expr.parse(item['Item'])
44                 if item.is_existing:
45                     continue
46                 yield item
47         return set(generate())
48
49     def available(self) -> Set[Expr]:
50         def generate():
51             inventory = set(self.inventory())
52             for item in self.all_items():
53                 if self.is_available(item) and item not in inventory:
54                     yield item
55         return set(generate())
56
57     def places(self) -> Set[Expr]:
58         def generate():
59             for item in self._state.query(found('Item')):
60                 item = Expr.parse(item['Item'])
61                 if item.is_existing:
62                     yield item
63         return set(generate())
```

```

    def is_ready_for_hardmode(self) -> bool:
        statement = "achived('Can break all found existing ores')"
        return self._state.is_proven(statement)

    def is_able_to_craft_both_evil_pickaxe(self) -> bool:
        statement = "achived('Theoretically can craft both Nightmare and Deathbringer')"
        return self._state.is_proven(statement)

    def is_able_to_break_all_explored_ores(self) -> bool:
        statement = "achived('Is ready to hardmode')"
        return self._state.is_proven(statement)
```

Листинг 12: Бизнес-логика

Слой бизнес логики используется слоем принятия и обработки текстовых запросов.

```

from typing import Callable
import random
import re

from terraria import Terraria, TerrariaException


def emotion(): return random.choice([
    '                      ',
    '      ',
    '      ',
    '                 ',
    '              ',
    '              ',
])


def on_show_inventory(terraria: Terraria, groups) -> str:
    inventory = terraria.inventory()
    if len(inventory) == 0:
        return random.choice([
            '    ,               ,                                   ',
            '                         .       -                    ...     ',
            '          !                                  ,           -
                                      ',
        ])
    else:
        return f'{emotion()}!                        {len(inventory)}
    !     \n'\
            + ',\n'.join(map(lambda w: '-> ' + w, map(repr, inventory)))


def on_show_places(terraria: Terraria, groups):
    places = terraria.places()
    if len(places) == 0:
        return random.choice([
            '
    ...     ',
            '                              (  )
                    ?!     ',
            '                                      (  )                       ...'
        ])
    return f'{emotion()}!                         {len(places)}           !     \n'\
        + ',\n'.join(map(lambda w: '-> ' + w, map(repr, places)))


def on_show_available(terraria: Terraria, groups):
    available = terraria.available()
    if len(available) == 0:
        return random.choice([
            '                              ...     ',
            '                                        ...     ',
            '                ,                                  ?             .     ',
        ])
    return f'                ,                                  ?
                  -                                 !\n'\
```

```
        + ',\n'.join(map(lambda w: '-> ' + w, map(repr, available)))


def on_action_take(terraria: Terraria, groups):
    name = groups[-2][0] + groups[-2][1:].lower()
    type = groups[-1].lower()
    try:
        if type not in ('ore', 'pickaxe'):
            return '         !                                ore         pickaxe!'
        terraria.take(f'{type}(\'{name}\')')
        return random.choice([
            f'{emotion()}!                     ,          !     ',
            f'    ,                  ',
            f'{emotion()}!               !',
        ])
    except TerrariaException as e:
        return random.choice([
            f'           .                  ?
                  ?',
            f'                              ',
            f'                          {name} {type}                    !
    ',
            f'                          (a)?     '
        ])


def on_action_cheat(terraria: Terraria, groups):
    name = groups[-2][0] + groups[-2][1:].lower()
    type = groups[-1].lower()
    if type not in ('ore', 'pickaxe'):
        return f'        !                                  ore         pickaxe,
              {type}!'
    terraria.cheat(f"{type}('{name}')")
    return random.choice([
        '                              ...     ,           ,
                          .     ',
        '           ,                                                    .
              ?..      ',
        '                                                           .          !
        ',
    ])


def on_action_explore(terraria: Terraria, groups):
    name = groups[-2][0] + groups[-2][1:].lower()
    type = groups[-1].lower()
    if type not in ('evil_biome', 'ore'):
        return f'     evil_biome     ore                          ,      {type}...
          ,              ?'
    terraria.explore(f'{type}(\'{name}\')')
    return random.choice([
        f'          {name} {type}?
                .     ',
        '                              .                              -
                ?',
        f'{emotion()}!                       !',
    ])


def on_check_is_ready_for_hardmode(terraria: Terraria, groups):
    if terraria.is_ready_for_hardmode():
        return random.choice([
            f'{emotion()}!               ,
                                                    !',
            '    ,                         ,           ,             !',
        ])
    return random.choice([
        '                      ,
              .',
        '                      ,                            (   )',
        '                                  ,            ?',
        '                              ',
    ])
```

```python


def on_check_is_able_to_craft_both_evil_pickaxe(terraria: Terraria, groups):
    if terraria.is_able_to_craft_both_evil_pickaxe():
        return random.choice([
            '                                        ?',
            f'{emotion()}!             !'
        ])
    return random.choice([
        '              ,               .',
        '    -   ',
        '    ',
    ])


def on_check_is_able_to_break_all_explored_ores(terraria: Terraria, groups):
    if terraria.is_able_to_break_all_explored_ores():
        return random.choice([
            '    ,                    ',
            '      ',
        ])
    return random.choice([
        '      ,                                    ',
        '                                                        ',
    ])


def on_error(terraria: Terraria, groups):
    return random.choice([
        '                                                        .
            .                        ,                                ,                      .
                    .',
        '                             ,                                      !     ',
        '             ...                                     ...    ',
        '            ,                                                    ,
                      ,
    ])


class Feedback:
    def __init__(self, terraria: Terraria):
        self.terraria = terraria
        self.patterns = {
            r'.*   .*           .*': on_show_inventory,
            r'.*           .*(                |               ).*': on_show_places,
            r'.*(                 |            |                 ).*': on_show_available,
            r'.*(    +)?.*   (      )? .*(             |              ).*        +   (\w+)
    (\w+).*': on_action_cheat,
            r'.*(          |             |             |         |           ) (\w+) (\w+).*
    ': on_action_take,
            r'.*    .*       (          ? )?((  )|(              )|(
      )) ([a-zA-z_]+) ([a-zA-z_]+).*': on_action_explore,
            r'.*        .*                     .*': on_check_is_ready_for_hardmode,
            r'.*                               .*':
    on_check_is_able_to_craft_both_evil_pickaxe,
            r'.*        (      |    ).*                                    .*':
    on_check_is_able_to_break_all_explored_ores,
        }
        self.fallback = on_error

    def answer(self, input: str) -> str:
        for pattern in self.patterns:
            match = re.match(pattern, input, re.IGNORECASE)
            if match is None:
                continue
            try:
                return self.patterns[pattern](
                    self.terraria, match.groups())
            except Exception as e:
                print('Error', e)
                return '               ,              ...              ,         !     '
        else:
```

```
            return self.fallback(self.terraria, None)
```

Листинг 13: Слой принятия и обработки текстовых запросов

Ну и вишенкой на торте стало CLI приложение.

```
import os
from typing import NamedTuple, List
import sys


class Args(NamedTuple):
    knowledge_path: str
    echo: bool = False

    @property
    def preload(self) -> List[str]:
        path = self.knowledge_path
        return [f'res/{file}' for file in os.listdir(path) if file.endswith('.pl')]

    @classmethod
    def parse(cls, args: List[str]) -> 'Args':
        return Args(
            knowledge_path=args[1],
            echo=True if len(args) > 2 and args[2] == 'echo' else False
        )

    @classmethod
    def system(cls) -> 'Args':
        return cls.parse(sys.argv)
```

Листинг 14: Аргументы командной строки

```
import random
from log import Log
from prolog import Prolog
from args import Args
from dsl import *
from terraria import Terraria
from feedback import Feedback


if __name__ == '__main__':
    Log.set_level(Log.Level.Info)

    args = Args.system()

    with Prolog(args.preload) as prolog:
        terraria = Terraria(prolog)

        feedback = Feedback(terraria)

        limit = 60
        for i in range(limit):
            if i == 0:
                query = input(
                    '           :             !                                    .
    '
                    + '
                          .'
                    + '                              !\n        : ')
            else:
                query = input(
                    '           : ' + random.choice([
                        '                            ,           ?',
                        '                         .',
                        '    ,                        ,              ',
                        '                        -                    ',
                        '                                          ?',
                    ]) + '\n        : ')
            query = query.strip()

            if args.echo:
                print(query)
```

```
            if query == '                        -           ?':
                print('             :             !                  ,                   ...')
                break
            elif query == '':
                print('             : ' + random.choice([
                    '                          ,            ?     ',
                    '        -                          ?..      ',
                    '       -                                           ...',
                    '',
                ]))
                print()
                continue

            print(f'            : {feedback.answer(query)}')
            print()

        else:
            print('             :              ,              ,                        .')
```

Листинг 15: Главный цикл

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

Было проведено модульное тестирование парсера.

```
import unittest
from unittest import TestCase

from parse import Expr


class ParserTest(TestCase):
    def test(self):
        for input, output in [
            ('hello', Expr('hello')),
            ('hello(hello)', Expr('hello', Expr('hello'))),
            ('a(b(c))', Expr('a', Expr('b', Expr('c')))),
        ]:
            self.assertEqual(Expr.parse(input), output)


if __name__ == '__main__':
    unittest.main()
```

Листинг 16: Тесты для парсера

Также слегка проверен был слой бизнес-логики.

```
from typing import Iterable, Set
import unittest
from unittest import TestCase

from args import Args
from log import Log
from prolog import Prolog
from terraria import Terraria, TerrariaException
from dsl import *


def reprset(iterable: Iterable[object]) -> Set[str]:
    return {repr(element) for element in iterable}


class TerrariaTest(TestCase):
    def test(self):
        Log.set_level(Log.Level.Debug)
        args = Args(knowledge_path='res')
        with Prolog(args.preload) as prolog:
            terraria = Terraria(prolog)

```

```
            # Initially empty
            self.assertEqual(reprset(terraria.inventory()), set())
            self.assertEqual(reprset(terraria.places()), set())

            # Explore
            terraria.explore(Ore.Iron)
            terraria.explore(Ore.Gold)

            self.assertEqual(reprset(terraria.inventory()), set())
            self.assertEqual(reprset(terraria.places()), {
                repr(Ore.Iron),
                repr(Ore.Gold),
            })

            # Explore Evil
            terraria.explore(EvilBiome.Corruption)

            self.assertEqual(reprset(terraria.inventory()), set())
            self.assertEqual(reprset(terraria.places()), {
                repr(Ore.Iron),
                repr(Ore.Gold),
                repr(EvilBiome.Corruption)
            })

            # Pick a pickaxe
            terraria.cheat(Pickaxe.Copper)

            self.assertEqual(reprset(terraria.inventory()), {
                repr(Pickaxe.Copper),
            })
            self.assertEqual(reprset(terraria.places()), {
                repr(Ore.Iron),
                repr(Ore.Gold),
                repr(EvilBiome.Corruption)
            })

            # Take
            terraria.take(Ore.Iron)
            terraria.take(Pickaxe.Iron)

            self.assertEqual(reprset(terraria.inventory()), {
                repr(Pickaxe.Copper),
                repr(Pickaxe.Iron),
                repr(Ore.Iron),
            })
            self.assertEqual(reprset(terraria.places()), {
                repr(Ore.Iron),
                repr(Ore.Gold),
                repr(EvilBiome.Corruption)
            })

            # Take failure
            self.assertRaises(
                TerrariaException,
                lambda: terraria.take(Ore.Gold))

            # Available
            terraria.explore(Ore.Copper)
            terraria.explore(Ore.Iron)
            terraria.explore(Ore.Gold)
            terraria.explore(Ore.Silver)

            self.assertEqual(reprset(terraria.available()), {
                repr(Ore.Silver),
                repr(Ore.Copper),
            })

            terraria.take(Ore.Silver)

            self.assertEqual(reprset(terraria.available()), {
                repr(Pickaxe.Silver),
                repr(Ore.Copper),
            })
```

```
96
97
98 if __name__ == '__main__':
99     unittest.main()
```

Листинг 17: Тесты для парсера

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

```
1  import random
2  import unittest
3  from unittest import TestCase
4  from args import Args
5  from feedback import Feedback
6  from log import Log
7  from prolog import Prolog
8
9  from terraria import Terraria
10
11
12 class FuzzingTest(TestCase):
13     def generate_normal(self) -> str:
14         return random.choice([
15             '                                    ?',
16             '                                          ?',
17             '                                               ',
18             '              Copper ore',
19             '              Iron ore',
20             '              Iron pickaxe',
21             '              Copper pickaxe',
22             '                           Copper ore',
23             '                           Iron ore',
24             '                           Iron pickaxe',
25             '                           Copper pickaxe',
26             '                                                ?',
27             # FIXME: disabled as it breaks tests
28             # '                                                        ',
29             '                              ',
30         ])
31
32     def generate_stupid(self) -> str:
33         size = random.randint(0, 10)
34         result = ''
35         for _ in range(size):
36             result += random.choice([
37                 lambda: '             ',
38                 lambda: '                          ',
39                 lambda: '                ',
40                 lambda: '           ',
41                 lambda: '             ',
42                 lambda: '                          ',
43                 lambda: '            ',
44                 lambda: '            ',
45                 lambda: '                       ',
46                 lambda: '                          ',
47                 # FIXME: disabled as it breaks tests
48                 # lambda: '                                ',
49                 lambda: '                                                  ',
50                 lambda: 'Copper ore',
51                 lambda: 'Iron ore',
52                 lambda: 'Iron pickaxe',
53                 lambda: 'Copper pickaxe',
54
55             ])() + ' '
56         return result
57
58     def test(self):
59         Log.set_level(Log.Level.Info)
60         args = Args(knowledge_path='res')
61         with Prolog(args.preload) as prolog:
62             terraria = Terraria(prolog)
63             feedback = Feedback(terraria)
64
```

```
            rounds = 200
            for i in range(rounds):
                if i % 2 == 0:
                    input = self.generate_stupid()
                else:
                    input = self.generate_normal()
                Log.info(f'input: {input}')
                Log.info(f'output: {feedback.answer(input)}')


if __name__ == '__main__':
    unittest.main()
```

Листинг 18: Фаззинг-тесты для чат-бота

4.3 Пример вывода программы

```
            :              !                                          .
                                                                                  .
                                  !
        :                        !
            :              ...                                  ...

            :     ,                        ,
        :                                  ?
            :            !                                  ,           -


            :                                  ,           ?
        :                                        ?
            :                                  (  )
                     ?!

            :                        .
        :                      ?
            :                                  ...

            :                                   ?
        :                Copper Pickaxe.
            :                              (a)?

            :                            .
        :                Copper Ore.
            :                              (a)?

            :                                  ,           ?
        :                                        Copper Pickaxe!
            :                                                                .
              !

            :                          -
        :                                  ?
            :                      !                       1                   !
-> Copper pickaxe

            :                                   ?
        :                                  ?
            :                                  ...

            :                           .
        :                        Copper Ore             -    !
            :             Copper ore?
                  .

            :                                   ?
        :                                  ?
            :                      !                       1           !
-> Copper ore

            :                          -
        :                                  Iron Ore!
```

```
            :                                 .                                  -
                  ?

            :     ,                        ,
        :                             Silver Ore!
            :            Silver ore?
                  .

            :                              ,           ?
        :                                   Gold Ore!
            :            Gold ore?
                  .

            :                       .
        :                                                  Demonite Ore.
            :            Demonite ore?
                   .

            :                              ,           ?
        :                                                                     ?
            :               !                        5           !
-> Iron ore,
-> Demonite ore,
-> Copper ore,
-> Gold ore,
-> Silver ore

            :                       .
        :                              ?
            :               !                        1                   !
-> Copper pickaxe

            :                      -
        :                                              ?
            :            ,                                    ?
                        -                                     !
-> Iron ore,
-> Copper ore

            :                                 ?
        :                                       ?
            :                                    ,           ?

            :                              ,           ?
        :                                                       ?
            :

            :     ,                        ,
        :                                                                  ?
            :       ,

            :                       .
        :                                    Crimtane Ore.
            :                                 .                                  -
                  ?

            :                                 ?
        :                                                       ?
            :                                    ?

            :                      -
        :              Iron Ore.
            :       !                      ,           !

            :                                 ?
        :                Iron Pickaxe.
            :                !                  !

            :                                 ?
        :                            Silver Ore.
            :       !                      ,           !

            :     ,                        ,
```

```
        :                   Silver Pickaxe.
            :       !                       ,           !

            :                             .
        :                                       Gold Ore.
            :               !                   !

            :                           -
        :                                   Gold Pickaxe.
            :       !                       ,           !

            :                             .
        :           Crimtane Ore.
            :     ,

            :                             .
        :           Demonite Ore.
            :       !                   !

            :                             .
        :               Deathbringer Pickaxe.
            :                       !                       ,           !

            :                           -
        :               Nightmare Pickaxe.
            :                       !                   !

            :                           -
        :             Hellstone Ore.
            :               .                   ?
                          ?

            :                             .
        :   ,           ,                                           Hellstone Ore.
            :                                       .                                   -
                    ?

            :                                       ?
        :                                 Hellstone Ore.
            :     ,

            :                           -
        :
            :           -                               ?..

            :                           -
        :                                                                       ?
            :               !                           12                  !
-> Hellstone ore,
-> Iron ore,
-> Silver pickaxe,
-> Copper pickaxe,
-> Gold ore,
-> Demonite ore,
-> Nightmare pickaxe,
-> Crimtane ore,
-> Iron pickaxe,
-> Gold pickaxe,
-> Deathbringer pickaxe,
-> Silver ore

            :                           -
        :                                           ?
            :     ,                           ,           ,           !

            :                           -
        :                       -           ?
            :             !                 ,                   ...
```

Листинг 19: Пример диалога с чат-ботом

5 Вывод

Вывод

Список литературы

[1] Б.П. Демидович, И.А. Марон Основы вычислительной математики: учебное пособие — 1966 год.