

Университет ИТМО  
Факультет программной инженерии и компьютерной техники

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

Группа : Р33131  
Студент: Смирнов Виктор Игоревич  
Тема : Игра Terraria

# 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 автоматического 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, соответствующие состоянию базы знаний.
2. Фронтэнд должен быть устойчив к некорректному 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 знаний 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 - зависимостей 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.

```
1 :- module(achievements, [achived/1]).
2
3 :- use_module(ore).
4 :- use_module(pickaxe).
5 :- use_module(available).
6 :- use_module(reachable).
7
8 achived('Can break all found existing ores') :-
9     forall(
10         found(existing(ore(Ore))),
11         available(ore(Ore))
12     ).
13
14 achived('Theoretically can craft both Nightmare and Deathbringer') :-
15     reachable(pickaxe('Nightmare')),
16     reachable(pickaxe('Deathbringer')).
17
18 achived('Is ready to hardmode') :-
19     available(pickaxe('Molten')).
```

Листинг 1: Достижения игрока

```
1 :- module(available, [available/1]).
2
3 :- use_module(progress).
4 :- use_module(requirement).
5
6 available(Item) :-
7     Item, found(Item).
8
9 available(ore(Ore)) :-
```

```

10     found(existing(ore(Ore))),
11     ore_power(ore(Ore), OrePower),
12     findall(Pickaxe, (
13         pickaxe(Pickaxe),
14         pickaxe_power(pickaxe(Pickaxe), PickaxePower),
15         (PickaxePower < OrePower -> false; true),
16         found(pickaxe(Pickaxe))
17     ), Pickaxes),
18     member(_, Pickaxes).
19
20 available(Item) :-
21     Item,
22     findall(R, requirement(Item, R), Requirements),
23     member(_, Requirements),
24     forall(member(R, Requirements), found(R)).

```

Листинг 2: Доступность предмета игроку в один шаг

```

1 :- module(reachable, [reachable/1]).
2
3 :- use_module(requirement).
4 :- use_module(available).
5
6 :- use_module(pickaxe).
7
8 reachable(Item) :-
9     Item, available(Item).
10
11 reachable(pickaxe_with_power(TargetPower)) :-
12     findall(Pickaxe, (
13         pickaxe(Pickaxe),
14         pickaxe_power(pickaxe(Pickaxe), PickaxePower),
15         (PickaxePower < TargetPower -> false; true)
16     ), Pickaxes),
17     member(FirstPickaxe, Pickaxes),
18     reachable(pickaxe(FirstPickaxe)).
19
20 reachable(Item) :-
21     Item,
22     findall(R, requirement(Item, R), Requirements),
23     member(_, Requirements),
24     forall(member(R, Requirements), reachable(R)).

```

Листинг 3: Доступность предмета игроку в несколько шагов

```

1 :- module(evil, [evil_biome/1]).
2
3 evil_biome('Corruption').
4 evil_biome('Crimson').

```

Листинг 4: Злые биомы Террарии

```

1 :- module(ore, [ore/1, ore_power/2, existing/1]).
2
3 ore('Copper').
4 ore('Tin').
5 ore('Iron').
6 ore('Lead').
7 ore('Silver').
8 ore('Tungsten').
9 ore('Gold').
10 ore('Platinum').
11 ore('Meteorite').
12 ore('Demonite').
13 ore('Crimtane').
14 ore('Obsidian').
15 ore('Hellstone').
16 ore('Cobalt').
17 ore('Palladium').
18 ore('Mythril').
19 ore('Orichalcum').
20 ore('Adamantite').
21 ore('Titanium').

```

```

22 ore('Chlorophyte').
23 ore('Luminite').
24
25 existing(ore(Ore)) :- ore(Ore).
26
27 ore_power(ore('Copper'), 1).
28 ore_power(ore('Tin'), 1).
29 ore_power(ore('Iron'), 2).
30 ore_power(ore('Lead'), 2).
31 ore_power(ore('Silver'), 3).
32 ore_power(ore('Tungsten'), 3).
33 ore_power(ore('Gold'), 4).
34 ore_power(ore('Platinum'), 4).
35 ore_power(ore('Meteorite'), 4).
36 ore_power(ore('Demonite'), 5).
37 ore_power(ore('Crimtane'), 5).
38 ore_power(ore('Obsidian'), 5).
39 ore_power(ore('Hellstone'), 6).
40 ore_power(ore('Cobalt'), 7).
41 ore_power(ore('Palladium'), 7).
42 ore_power(ore('Mythril'), 8).
43 ore_power(ore('Orichalcum'), 9).
44 ore_power(ore('Adamantite'), 10).
45 ore_power(ore('Titanium'), 11).

```

Листинг 5: Руды Террарии

```

1 :- module(pickaxe, [
2     pickaxe/1,
3     pickaxe_power/2,
4     pickaxe_with_power/1
5 ]).
6
7 pickaxe('Cactus').
8 pickaxe('Copper').
9 pickaxe('Tin').
10 pickaxe('Iron').
11 pickaxe('Lead').
12 pickaxe('Silver').
13 pickaxe('Tungsten').
14 pickaxe('Gold').
15 pickaxe('Platinum').
16 pickaxe('Nightmare').
17 pickaxe('Deathbringer').
18 pickaxe('Molten').
19 pickaxe('Cobalt').
20 pickaxe('Palladium').
21 pickaxe('Mythril').
22 pickaxe('Orichalcum').
23 pickaxe('Adamantite').
24 pickaxe('Titanium').
25
26 pickaxe_power(pickaxe('Cactus'), 2).
27 pickaxe_power(pickaxe('Copper'), 2).
28 pickaxe_power(pickaxe('Tin'), 2).
29 pickaxe_power(pickaxe('Iron'), 3).
30 pickaxe_power(pickaxe('Lead'), 3).
31 pickaxe_power(pickaxe('Silver'), 4).
32 pickaxe_power(pickaxe('Tungsten'), 4).
33 pickaxe_power(pickaxe('Gold'), 5).
34 pickaxe_power(pickaxe('Platinum'), 5).
35 pickaxe_power(pickaxe('Nightmare'), 6).
36 pickaxe_power(pickaxe('Deathbringer'), 6).
37 pickaxe_power(pickaxe('Molten'), 7).
38 pickaxe_power(pickaxe('Cobalt'), 8).
39 pickaxe_power(pickaxe('Palladium'), 8).
40 pickaxe_power(pickaxe('Mythril'), 9).
41 pickaxe_power(pickaxe('Orichalcum'), 9).
42 pickaxe_power(pickaxe('Adamantite'), 10).
43 pickaxe_power(pickaxe('Titanium'), 10).
44
45 pickaxe_with_power(1).
46 pickaxe_with_power(2).

```

```

47 pickaxe_with_power(3).
48 pickaxe_with_power(4).
49 pickaxe_with_power(5).
50 pickaxe_with_power(6).
51 pickaxe_with_power(7).
52 pickaxe_with_power(8).

```

Листинг 6: Кирки Террарии

```

1 :- module(progress, [found/1]).
2
3 :- use_module(pickaxe).
4 :- use_module(ore).
5
6 :- dynamic(found/1).
7
8 % found(evil_biome('Crimson')).
9 %
10 % found(existing(ore('Crimtane'))):-
11 %     found(evil_biome('Crimson')).
12 % found(existing(ore('Demonite'))):-
13 %     found(evil_biome('Corruption')).
14 %
15 % found(existing(ore('Demonite'))).
16 %
17 % found(existing(ore('Tin'))).
18 % found(existing(ore('Iron'))).
19 % found(existing(ore('Silver'))).
20 % found(existing(ore('Gold'))).
21 %
22 % found(existing(ore('Hellstone'))).
23 %
24 % found(ore('Copper')).
25 % found(pickaxe('Copper')).
26 % found(pickaxe('Gold')).
27 % found(pickaxe('Deathbringer')).
28 % found(pickaxe('Molten')).
29 %
30
31 found(pickaxe_with_power(Power)) :-
32     number(Power),
33     found(pickaxe(Pickaxe)),
34     pickaxe_power(pickaxe(Pickaxe), PickaxePower),
35     (PickaxePower < Power -> false ; true).

```

Листинг 7: Прогресс игрока

```

1 :- module(requirement, [requirement/2]).
2
3 :- use_module(pickaxe).
4 :- use_module(ore).
5 :- use_module(evil).
6
7 requirement(pickaxe('Copper'), ore('Copper')).
8 requirement(pickaxe('Tin'), ore('Tin')).
9 requirement(pickaxe('Iron'), ore('Iron')).
10 requirement(pickaxe('Lead'), ore('Lead')).
11 requirement(pickaxe('Silver'), ore('Silver')).
12 requirement(pickaxe('Tungsten'), ore('Tungsten')).
13 requirement(pickaxe('Gold'), ore('Gold')).
14 requirement(pickaxe('Platinum'), ore('Platinum')).
15 requirement(pickaxe('Nightmare'), ore('Demonite')).
16 requirement(pickaxe('Deathbringer'), ore('Crimtane')).
17 requirement(pickaxe('Molten'), ore('Hellstone')).
18 requirement(pickaxe('Cobalt'), ore('Cobalt')).
19 requirement(pickaxe('Palladium'), ore('Palladium')).
20 requirement(pickaxe('Mythril'), ore('Mythril')).
21 requirement(pickaxe('Orichalcum'), ore('Orichalcum')).
22 requirement(pickaxe('Adamantite'), ore('Adamantite')).
23 requirement(pickaxe('Titanium'), ore('Titanium')).
24
25 requirement(ore(Ore), pickaxe_with_power(PickaxePower)) :-
26     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 знаний выполнялись с помощью библиотеки `pyswip`.

```

1 from typing import List
2 from log import Log as log
3 from pyswip import Prolog as SWIProlog
4 from pyswip.core import cleanupProlog
5
6
7 class Prolog:
8     def __init__(self, preload: List[str]):
9         self.__pl: SWIProlog = None
10        self.__preload = preload
11
12    def query(self, query: str):
13        log.debug(query)
14        return self.__pl.query(query)
15
16    def is_proven(self, query: str) -> bool:
17        log.debug(query)
18        return bool(list(self.query(query)))
19
20    def assume(self, fact: str):
21        log.debug(fact)
22        self.__pl.assertz(fact)
23
24    def __enter__(self):
25        self.__pl = SWIProlog()
26        for file in self.__preload:
27            self.__pl.consult(file)
28        return self
29
30    def __exit__(self, *args, **kwargs):
31        # FIXME: segfault on duoble call inside a one process
32        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.

```

1 from enum import Enum
2
3 from parse import Expr
4
5
6 EvilBiome = Enum('EvilBiome', [
7     'Crimson',
8     'Corruption',
9 ])
10 EvilBiome.__str__ = lambda self: f"evil_biome('{self.name}')"
11 EvilBiome.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"
12
13 Ore = Enum('Ore', [
14     'Copper',
15     'Tin',
16     'Iron',
17     'Lead',
18     'Silver',
19     'Tungsten',
20     'Gold',
21     'Platinum',
22     'Meteorite',
23     'Demonite',

```

```

24     'Crimtane',
25     'Obsidian',
26     'Hellstone',
27     'Cobalt',
28     'Palladium',
29     'Mythril',
30     'Orichalcum',
31     'Adamantite',
32     'Titanium',
33     'Chlorophyte',
34     'Luminite',
35 ]
36 Ore.__str__ = lambda self: f"ore('{self.name}')"
37 Ore.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"
38
39
40 Pickaxe = Enum('Pickaxe', [
41     'Cactus',
42     'Copper',
43     'Tin',
44     'Iron',
45     'Lead',
46     'Silver',
47     'Tungsten',
48     'Gold',
49     'Platinum',
50     'Nightmare',
51     'Deathbringer',
52     'Molten',
53     'Cobalt',
54     'Palladium',
55     'Mythril',
56     'Orichalcum',
57     'Adamantite',
58     'Titanium',
59 ])
60 Pickaxe.__str__ = lambda self: f"pickaxe('{self.name}')"
61 Pickaxe.__repr__ = lambda self: f"{repr(Expr.parse(str(self)))}"
62
63 Item = Pickaxe | Ore
64
65
66 NatureObject = EvilBiome | Ore
67
68 def existing(thing: object) -> str:
69     return f'existing({thing})'
70
71
72 def found(thing: object) -> str:
73     return f'found({thing})'
74
75
76 def available(item: Item) -> str:
77     return f'available({item})'

```

Листинг 10: Доменная область

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

```

1 import sys
2 from typing import Generator, Optional, List
3 from dataclasses import dataclass
4
5 from lark import Lark, Transformer, v_args
6 from lark.ast_utils import Ast as LarkAST, WithMeta, create_transformer
7 from lark.tree import Meta
8
9
10 class AST(LarkAST):
11     pass
12
13
14 @dataclass

```



```

15 class Name(AST):
16     name: str
17
18
19 @dataclass(unsafe_hash=True)
20 class Expr(AST):
21     name: Name
22     expr: Optional['Expr'] = None
23
24     @property
25     def names(self) -> Generator[str, None, None]:
26         node = self
27         while node is not None:
28             name = node.name
29             s = name[0]
30             if s != s.lower():
31                 name = f'_{name}'
32             yield name
33             node = node.expr
34
35     def __str__(self) -> str:
36         result = ''
37
38         i = 0
39         for name in self.names:
40             if 1 <= i:
41                 result += '('
42                 result += name
43                 i += 1
44             result += ')' * (i - 1)
45
46         return result
47
48     @property
49     def is_existing(self) -> bool:
50         return 'existing' in self.names
51
52     def __repr__(self) -> str:
53         names = list(self.names)
54         name = names[-1]
55         type = names[-2]
56         return f'{name} {type}'.replace('\'', '')
57
58     @classmethod
59     def parse(cls, input: str) -> 'Expr':
60         result = transformer.transform(lark.parse(input))
61         prev = result
62         if isinstance(prev, str):
63             return Expr(prev)
64         curr = prev.expr
65         while not isinstance(curr, str):
66             prev = curr
67             curr = curr.expr
68         prev.expr = Expr(curr)
69         return result
70
71
72 class ToAst(Transformer):
73     def WORD(self, x):
74         return x.value
75
76     @v_args(inline=True)
77     def start(self, x):
78         return x
79
80
81 transformer = create_transformer(sys.modules[__name__], ToAst())
82
83 grammar = '''
84 start: expr
85
86 ?expr: '"'? ID '"'? "(" expr ")"
87       | '"'? 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())

```

```

64
65     def is_ready_for_hardmode(self) -> bool:
66         statement = "achived('Can break all found existing ores')"
67         return self._state.is_proven(statement)
68
69     def is_able_to_craft_both_evil_pickaxe(self) -> bool:
70         statement = "achived('Theoretically can craft both Nightmare and Deathbringer')"
71         return self._state.is_proven(statement)
72
73     def is_able_to_break_all_explored_ores(self) -> bool:
74         statement = "achived('Is ready to hardmode')"
75         return self._state.is_proven(statement)

```

Листинг 12: Бизнес-логика

Слой бизнес логики используется слоем принятия и обработки текстовых запросов.

```

1
2 from typing import Callable
3 import random
4 import re
5
6 from terraria import Terraria, TerrariaException
7
8
9 def emotion(): return random.choice([
10     ', ',
11     ', ',
12     ', ',
13     ', ',
14     ', ',
15     ', '
16 ])
17
18
19 def on_show_inventory(terraria: Terraria, groups) -> str:
20     inventory = terraria.inventory()
21     if len(inventory) == 0:
22         return random.choice([
23             ', ',
24             ', ',
25             ', ',
26             ', ',
27             ', ',
28             ', '
29         ])
30     else:
31         return f'{emotion()}! {len(inventory)}\n' + ',\n'.join(map(lambda w: '-> ' + w, map(repr, inventory)))
32
33
34 def on_show_places(terraria: Terraria, groups):
35     places = terraria.places()
36     if len(places) == 0:
37         return random.choice([
38             ', ',
39             ', ',
40             ', ',
41             ', ',
42             ', ',
43             ', '
44         ])
45     else:
46         return f'{emotion()}! {len(places)}\n' + ',\n'.join(map(lambda w: '-> ' + w, map(repr, places)))
47
48
49 def on_show_available(terraria: Terraria, groups):
50     available = terraria.available()
51     if len(available) == 0:
52         return random.choice([
53             ', ',
54             ', ',
55             ', ',
56             ', ',
57             ', ',
58             ', '
59         ])
60     else:
61         return f'{emotion()}! {len(available)}\n' + ',\n'.join(map(lambda w: '-> ' + w, map(repr, available)))

```





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

Листинг 13: Слой принятия и обработки текстовых запросов

Ну и вишенкой на торте стало CLI приложение.

```
1 import os
2 from typing import NamedTuple, List
3 import sys
4
5
6 class Args(NamedTuple):
7     knowledge_path: str
8     echo: bool = False
9
10     @property
11     def preload(self) -> List[str]:
12         path = self.knowledge_path
13         return [f'res/{file}' for file in os.listdir(path) if file.endswith('.pl')]
14
15     @classmethod
16     def parse(cls, args: List[str]) -> 'Args':
17         return Args(
18             knowledge_path=args[1],
19             echo=True if len(args) > 2 and args[2] == 'echo' else False
20         )
21
22     @classmethod
23     def system(cls) -> 'Args':
24         return cls.parse(sys.argv)
```

Листинг 14: Аргументы командной строки

```
1 import random
2 from log import Log
3 from prolog import Prolog
4 from args import Args
5 from dsl import *
6 from terraria import Terraria
7 from feedback import Feedback
8
9
10 if __name__ == '__main__':
11     Log.set_level(Log.Level.Info)
12
13     args = Args.system()
14
15     with Prolog(args.preload) as prolog:
16         terraria = Terraria(prolog)
17
18         feedback = Feedback(terraria)
19
20         limit = 60
21         for i in range(limit):
22             if i == 0:
23                 query = input(
24                     ',
25                     + '
26                     + '
27                 else:
28                     query = input(
29                         ',
30                         ',
31                         ',
32                         ',
33                         ',
34                     ]) + '\n
35                 query = query.strip()
36
37             if args.echo:
38                 print(query)
```

```

40
41         if query == '':
42             print('...')
43             break
44         elif query == '':
45             print('...')
46             print('...')
47             print('...')
48             print('...')
49             print('...')
50             print('...')
51             print('...')
52             print('...')
53             print('...')
54             print('...')
55             print('...')
56             print('...')
57             print('...')
58             print('...')

```

Листинг 15: Главный цикл

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

Было проведено модульное тестирование парсера.

```

1 import unittest
2 from unittest import TestCase
3
4 from parse import Expr
5
6
7 class ParserTest(TestCase):
8     def test(self):
9         for input, output in [
10             ('hello', Expr('hello')),
11             ('hello(hello)', Expr('hello', Expr('hello'))),
12             ('a(b(c))', Expr('a', Expr('b', Expr('c')))),
13         ]:
14             self.assertEqual(Expr.parse(input), output)
15
16
17 if __name__ == '__main__':
18     unittest.main()

```

Листинг 16: Тесты для парсера

Также слегка проверен был слой бизнес-логики.

```

1 from typing import Iterable, Set
2 import unittest
3 from unittest import TestCase
4
5 from args import Args
6 from log import Log
7 from prolog import Prolog
8 from terraria import Terraria, TerrariaException
9 from dsl import *
10
11
12 def reprset(iterable: Iterable[object]) -> Set[str]:
13     return {repr(element) for element in iterable}
14
15
16 class TerrariaTest(TestCase):
17     def test(self):
18         Log.set_level(Log.Level.Debug)
19         args = Args(knowledge_path='res')
20         with Prolog(args.preload) as prolog:
21             terraria = Terraria(prolog)
22

```

```

23     # Initially empty
24     self.assertEqual(reprset(terraria.inventory()), set())
25     self.assertEqual(reprset(terraria.places()), set())
26
27     # Explore
28     terraria.explore(Ore.Iron)
29     terraria.explore(Ore.Gold)
30
31     self.assertEqual(reprset(terraria.inventory()), set())
32     self.assertEqual(reprset(terraria.places()), {
33         repr(Ore.Iron),
34         repr(Ore.Gold),
35     })
36
37     # Explore Evil
38     terraria.explore(EvilBiome.Corruption)
39
40     self.assertEqual(reprset(terraria.inventory()), set())
41     self.assertEqual(reprset(terraria.places()), {
42         repr(Ore.Iron),
43         repr(Ore.Gold),
44         repr(EvilBiome.Corruption)
45     })
46
47     # Pick a pickaxe
48     terraria.cheat(Pickaxe.Copper)
49
50     self.assertEqual(reprset(terraria.inventory()), {
51         repr(Pickaxe.Copper),
52     })
53     self.assertEqual(reprset(terraria.places()), {
54         repr(Ore.Iron),
55         repr(Ore.Gold),
56         repr(EvilBiome.Corruption)
57     })
58
59     # Take
60     terraria.take(Ore.Iron)
61     terraria.take(Pickaxe.Iron)
62
63     self.assertEqual(reprset(terraria.inventory()), {
64         repr(Pickaxe.Copper),
65         repr(Pickaxe.Iron),
66         repr(Ore.Iron),
67     })
68     self.assertEqual(reprset(terraria.places()), {
69         repr(Ore.Iron),
70         repr(Ore.Gold),
71         repr(EvilBiome.Corruption)
72     })
73
74     # Take failure
75     self.assertRaises(
76         TerrariaException,
77         lambda: terraria.take(Ore.Gold))
78
79     # Available
80     terraria.explore(Ore.Copper)
81     terraria.explore(Ore.Iron)
82     terraria.explore(Ore.Gold)
83     terraria.explore(Ore.Silver)
84
85     self.assertEqual(reprset(terraria.available()), {
86         repr(Ore.Silver),
87         repr(Ore.Copper),
88     })
89
90     terraria.take(Ore.Silver)
91
92     self.assertEqual(reprset(terraria.available()), {
93         repr(Pickaxe.Silver),
94         repr(Ore.Copper),
95     })

```



```

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         return result + ' '
56
57     def test(self):
58         Log.set_level(Log.Level.Info)
59         args = Args(knowledge_path='res')
60         with Prolog(args.preload) as prolog:
61             terraria = Terraria(prolog)
62             feedback = Feedback(terraria)
63
64

```

```

65         rounds = 200
66         for i in range(rounds):
67             if i % 2 == 0:
68                 input = self.generate_stupid()
69             else:
70                 input = self.generate_normal()
71             Log.info(f'input: {input}')
72             Log.info(f'output: {feedback.answer(input)}')
73
74
75 if __name__ == '__main__':
76     unittest.main()

```

Листинг 18: Фаззинг-тесты для чат-бота

### 4.3 Пример вывода программы

## 5 Вывод

Вывод

## Список литературы

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