ДОДАТОК А

Скрипт, що реалізує запропоновану модель управління переміщенням

```
import numpy as np
import pandas as pd
import skfuzzy

from time import time
from skfuzzy import trimf, trapmf
from datetime import datetime

class FactBase:
    def __init__(self):
        self._fact_base = {}

    def __len__(self):
        return len(self._fact_base)

    def __str__(self):
        facts = ''
        for fact in sorted(self._fact_base):
            facts += fact
            facts += '\t - \t'
            facts += str(self._fact_base[fact])
            facts += '\n'
        return facts

    def __repr__(self):
        return self.__str__()

    def return_facts(self):
        return self._fact_base.copy()

    def fill_from_rule_base(self, rule_base):
        for rule in rule_base:
            fields = list(rule['if'].keys()) + list(rule['th'].keys())
            for field in fields:
                self._fact_base[field] = 0.0
        for fact in self._fact_base:
            if 'event_Start' in fact:
                self._fact_base[fact] = 0.0
            if 'sat_Start' in fact:
```

```
                self._fact_base[fact] = 1.0
            if 'sat_Start_k' in fact:
                continue

    def update(self, fact, value):
        try:
            self._fact_base[fact] = float(value)
        except ValueError:
            self._fact_base[fact] = value

class RuleBase:
    def __init__(self, epsilon=0.75):
        self._rule_base = []
        self.epsilon = epsilon

    def __len__(self):
        return len(self._rule_base)

    def __str__(self):
        rules = ''
        for rule in self._rule_base:
            rules += 'IF '
            for part in rule['if']:
                rules += part
                rules += ' is '
                rules += rule['if'][part]
                rules += ' and '
            rules = rules[:-5] + ', THEN '
            for part in rule['th']:
                rules += part
                rules += ' = '
                rules += str(rule['th'][part])
                rules += ', '
            rules = rules[:-2]
            rules += '\n'
        return rules
    def __repr__(self):
        return self.__str__()

    def add_one_rule(self, rule_txt):
        rule = rule_txt
        while rule[-1] == ';' or rule[-1] == ' ':
            rule = rule[:-1]
```

```
        if_field = rule.split(';>;')[0]
        th_field = rule.split(';>;')[1]
        rule = {}
        rule['if'] = {}
        rule['th'] = {}
        for part in if_field.split(';'):
            fact  = part[:part.rfind(' is ')]
            value = part[part.rfind(' is ') + 4:]
            rule['if'][fact] = value
        for part in th_field.split(';'):
            fact  = part[:part.rfind(' = ')]
            value = part[part.rfind(' = ') + 3:]
            try:
                rule['th'][fact] = float(value)
            except ValueError:
                rule['th'][fact] = value
        self._rule_base.append(rule)

    def add_rules(self, rules_txt):
        for rule_txt in rules_txt:
            self.add_one_rule(rule_txt.strip())

    def add_rules_from_file(self, file_path):
        with open(file_path) as rules:
            self.add_rules(rules)

    def return_rules(self):
        return self._rule_base.copy()

    def check(self, fact_base):
        for rule in self._rule_base:
            facts = fact_base.return_facts()
            fuzzy_values = []
            for fact in rule['if']:
                fuzzy_val = self.fuzzification(rule['if'][fact], facts[fact])
                fuzzy_values.append(fuzzy_val)
            output = min(fuzzy_values)
            if abs(output) < self.epsilon:
                continue
            for fact in rule['th']:
                if rule['th'][fact] == 'RUN':
                    fact_base.update(fact, 1.0)
                    continue
```

```
                if 'sat_Start_k' in fact:
                    for base_fact in facts:
                        if 'sat_Start' in base_fact:
                            fact_base.update(base_fact, rule['th'][fact])
                fact_base.update(fact, float(rule['th'][fact]) * output)

    @staticmethod
    def fuzzification(term, value):
        value = np.array([value])
        hig_term_param = np.array([0, 0.6 1, 1])
        mid_term_param = np.array([-0.6, 0, 0.6])
        low_term_param = np.array([-1, -1, -0.6, 0])
        if term == 'high' and trapmf(value, hig_term_param)[0] > 0:
            return trapmf(value, hig_term_param)[0]
        if term == 'middle' and trimf(value, mid_term_param)[0]:
            return trapmf(value, mid_term_param)[0]
        if term == 'low' and trapmf(value, low_term_param)[0]:
            return trapmf(value, low_term_param)[0]
        return False

class Engine:
    def __init__(self, facts, rules, epsilon=0.75):
        self.epsilon = epsilon

        self.facts = facts
        self.rules = rules

        self.rules.add_rules_from_file('rules_type4.csv')
        self.rules.add_rules_from_file('rules_type3.csv')
        self.rules.add_rules_from_file('rules_type2_1.csv')
        self.rules.add_rules_from_file('rules_type2_2.csv')
        self.rules.add_rules_from_file('rules_type1_1.csv')
        self.rules.add_rules_from_file('rules_type1_2.csv')
        self.rules.add_rules_from_file('rules_type0.csv')

        self.facts.fill_from_rule_base(self.rules.return_rules())

    def simulation(self, environmental_change_path):
        path = environmental_change_path
        self.fact_memory_logging()
        with open(path) as environmental_change:
            #Емуляція змін в оточенні
            for chg in environmental_change:
```

```
                #Якщо в рядку, що має містити зміни в оточенні міститься рядок 'next',
                #то одночасні зміни в оточенні закінчились і виконується перехід до
                #перевірки правил
                chg = chg.strip()
                if chg[:-1] == 'next':

                    self.rules.check(self.facts)
                    #Логування стану пам'яті фактів
                    self.fact_memory_logging()
                    #Опускання фактів подій (event), та фактів, що базуються на ознаках
                    #після перевірки усіх правил,
                    for fact in self.facts.return_facts():
                        if 'sat_' not in fact:
                            self.facts.update(fact, 0.0)
                    continue
                #Парсинг рядка подій і відповідна зміна бази фактів
                chg = chg.split(';')
                self.facts.update(chg[0], chg[1])


    def fact_memory_logging(self):
        '''Логування стану бази фактів '''
        with open('LOG_facts.csv', 'a') as log_facts:
            log_facts.write(str(self.facts.return_facts()).replace(', ', ';').replace(':
', ';').replace("'", '').replace('{', '').replace('}', '').replace('.', ','))
            log_facts.write('\n')

facts = FactBase()
rules = RuleBase()
engine = Engine(facts, rules)
engine.simulation(environmental_change_path='environmental_change.csv')
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	79 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

		Прізвище та ініціали.	Підп.	Дата	

Розробив	Іванюк О.І.			Розробка та дослідження моделі інтелектуального управління автономними об’єктами	Шифр групи	Код напр./спец.
Перевірив	Шевченко О.Ю.				СШІздм-19-1	122
Н.контр.	Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.	Філатов В.О.					