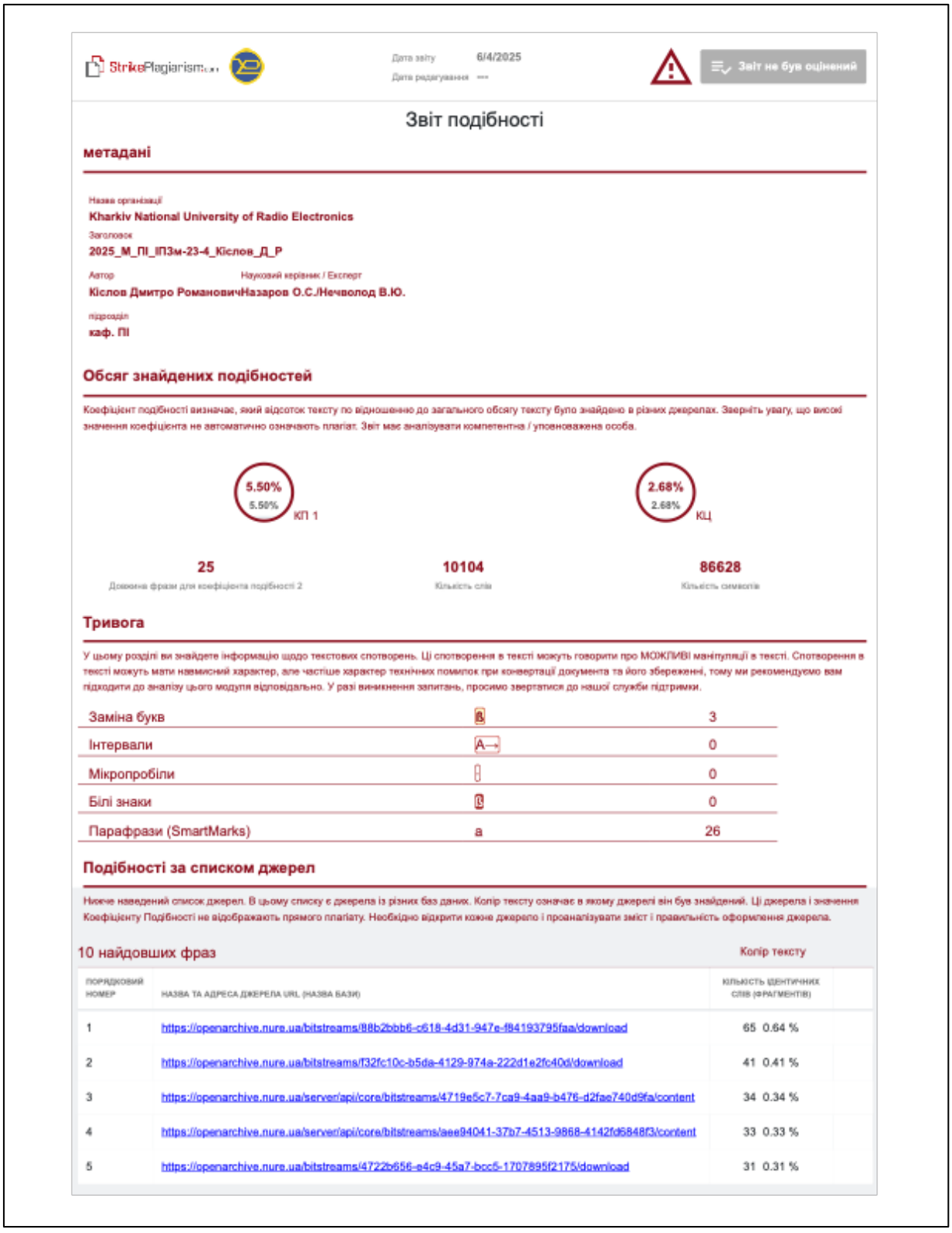
ДОДАТОК А

ЗВІТ РЕЗУЛЬТАТІВ ПЕРЕВІРКИ НА УНІКАЛЬНІСТЬ ТЕКСТУ В БАЗІ ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/4/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_M_ПІ_ІП3м-23-4_Кіслов_Д_Р
Автор
Кіслов Дмитро Романович
Науковий керівник / Експерт
Назаров О.С./Нечволод В.Ю.
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

5.50% 5.50% КП 1

2.68% 2.68% КЦ

25	10104	86628
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	в	3
Інтервали	A→	0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	26

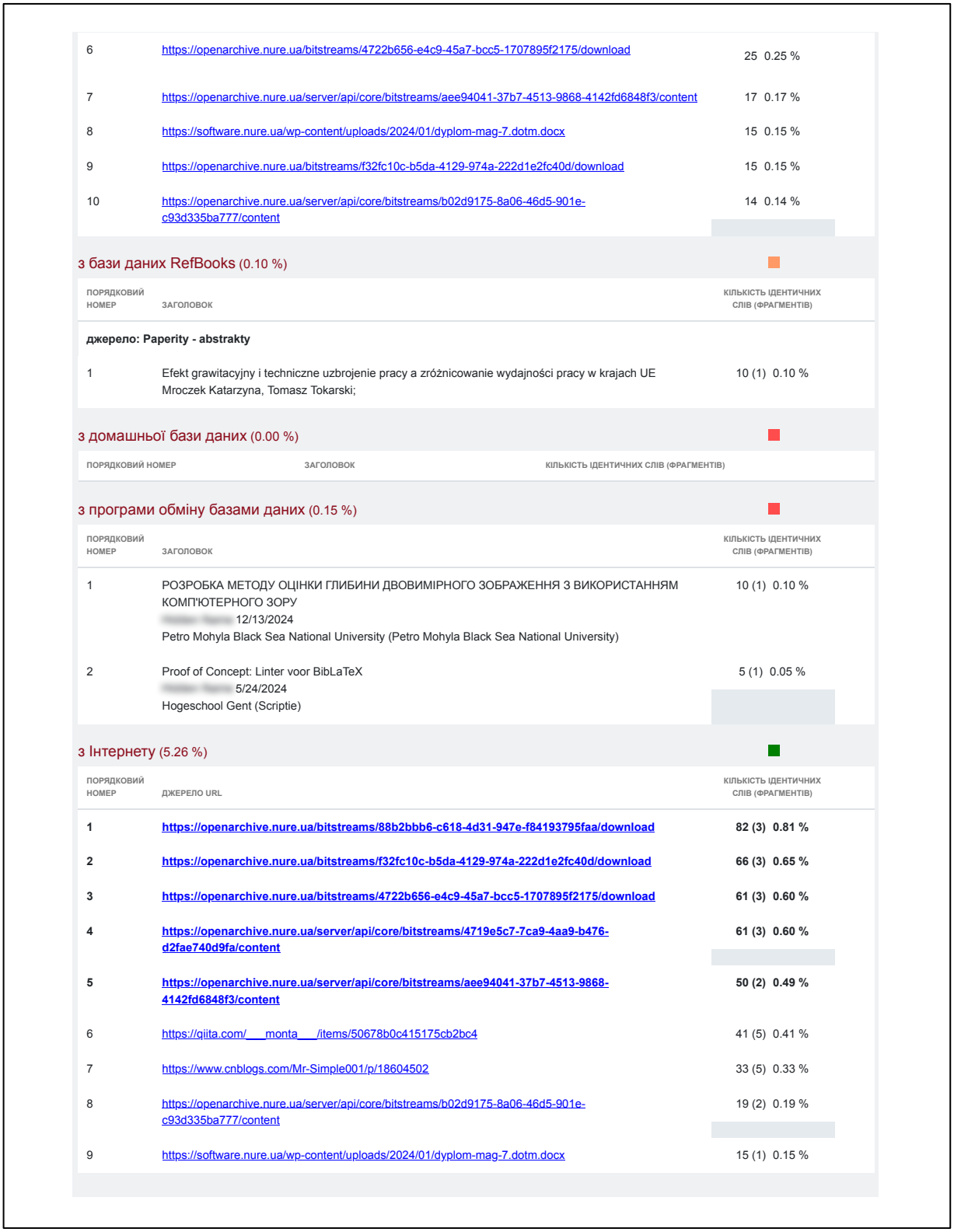
Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/88b2bbb6-c618-4d31-947e-f84193795faa/download	65 0.64 %
2	https://openarchive.nure.ua/bitstreams/f32fc10c-b5da-4129-974a-222d1e2fc40d/download	41 0.41 %
3	https://openarchive.nure.ua/server/api/core/bitstreams/4719e5c7-7ca9-4aa9-b476-d2fae740d9fa/content	34 0.34 %
4	https://openarchive.nure.ua/server/api/core/bitstreams/aee94041-37b7-4513-9868-4142fd6848f3/content	33 0.33 %
5	https://openarchive.nure.ua/bitstreams/4722b656-e4c9-45a7-bcc5-1707895f2175/download	31 0.31 %

Рисунок А.1 – Звіт з результатами перевірки роботи на академічну доброчесність, частина 1



6	https://openarchive.nure.ua/bitstreams/4722b656-e4c9-45a7-bcc5-1707895f2175/download	25 0.25 %
7	https://openarchive.nure.ua/server/api/core/bitstreams/aee94041-37b7-4513-9868-4142fd6848f3/content	17 0.17 %
8	https://software.nure.ua/wp-content/uploads/2024/01/dyplom-mag-7.dotm.docx	15 0.15 %
9	https://openarchive.nure.ua/bitstreams/f32fc10c-b5da-4129-974a-222d1e2fc40d/download	15 0.15 %
10	https://openarchive.nure.ua/server/api/core/bitstreams/b02d9175-8a06-46d5-901e-c93d335ba777/content	14 0.14 %

з бази даних RefBooks (0.10 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
джерело: Paperity - abstrakty		
1	Efekt grawitacyjny i techniczne uzbrojenie pracy a zróżnicowanie wydajności pracy w krajach UE Mroczek Katarzyna, Tomasz Tokarski;	10 (1) 0.10 %

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.15 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	РОЗРОБКА МЕТОДУ ОЦІНКИ ГЛИБИНИ ДВОВИМІРНОГО ЗОБРАЖЕННЯ З ВИКОРИСТАННЯМ КОМП'ЮТЕРНОГО ЗОРУ [redacted] 12/13/2024 Petro Mohyla Black Sea National University (Petro Mohyla Black Sea National University)	10 (1) 0.10 %
2	Proof of Concept: Linter voor BibLaTeX [redacted] 5/24/2024 Hogeschool Gent (Scriptie)	5 (1) 0.05 %

з Інтернету (5.26 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://openarchive.nure.ua/bitstreams/88b2bbb6-c618-4d31-947e-f84193795faa/download**	**82 (3) 0.81 %**
2	**https://openarchive.nure.ua/bitstreams/f32fc10c-b5da-4129-974a-222d1e2fc40d/download**	**66 (3) 0.65 %**
3	**https://openarchive.nure.ua/bitstreams/4722b656-e4c9-45a7-bcc5-1707895f2175/download**	**61 (3) 0.60 %**
4	**https://openarchive.nure.ua/server/api/core/bitstreams/4719e5c7-7ca9-4aa9-b476-d2fae740d9fa/content**	**61 (3) 0.60 %**
5	**https://openarchive.nure.ua/server/api/core/bitstreams/aee94041-37b7-4513-9868-4142fd6848f3/content**	**50 (2) 0.49 %**
6	https://qiita.com/___monta___/items/50678b0c415175cb2bc4	41 (5) 0.41 %
7	https://www.cnblogs.com/Mr-Simple001/p/18604502	33 (5) 0.33 %
8	https://openarchive.nure.ua/server/api/core/bitstreams/b02d9175-8a06-46d5-901e-c93d335ba777/content	19 (2) 0.19 %
9	https://software.nure.ua/wp-content/uploads/2024/01/dyplom-mag-7.dotm.docx	15 (1) 0.15 %

Рисунок А.2 – Звіт з результатами перевірки роботи на академічну доброчесність, частина 2

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кваліфікаційна робота магістра

Дослідження методів прийняття рішень у реальному часі широкого спектру в комп'ютерних іграх жанру RPG

Кіслов Дмитро Романович, ІПЗм-23-4
Керівник: доц. каф. ПІ
Назаров Олексій Сергійович

software engineering

2025

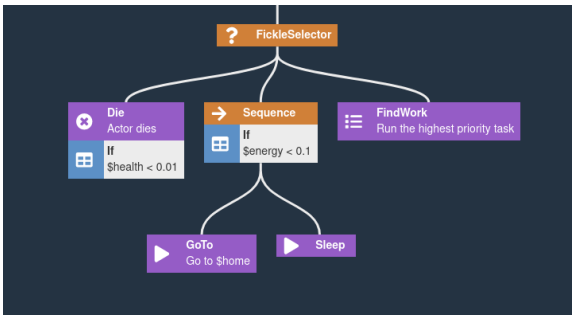
1

Рисунок Б.1 – Слайд 1

Аналіз предметної галузі

Актуальність дослідження:

- зростання обчислювальної складності ігрових систем
- підвищення вимог до реалістичності та ШІ
- необхідність оптимізації при високій якості ігрового процесу
- потреба у розробці масштабованих рішень

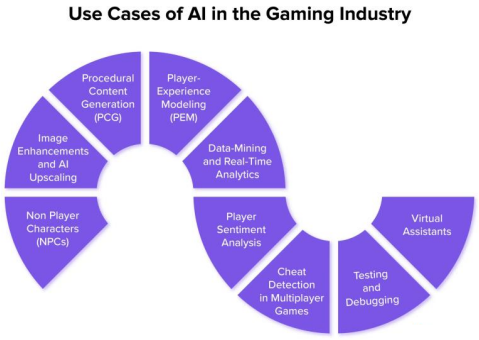


software engineering

2

Рисунок Б.2 – Слайд 2

Аналіз предметної галузі



Вимоги:

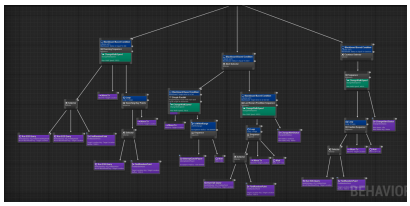
- загальна оптимізація
- мінімізація затримок
- реалістичність поведінки NPC
- надійність
- передбачуваність

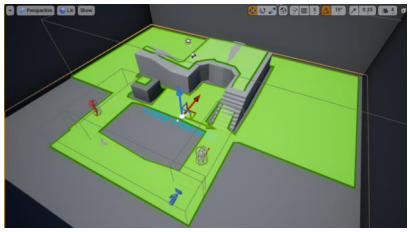
software engineering

3

Рисунок Б.3 – Слайд 3

Аналіз предметної галузі





Наявні рішення на ринку:

- Unreal Engine Behavior Trees
- Unity ML-Agents
- CryEngine Flowgraph
- TensorFlow
- PyTorch

software engineering

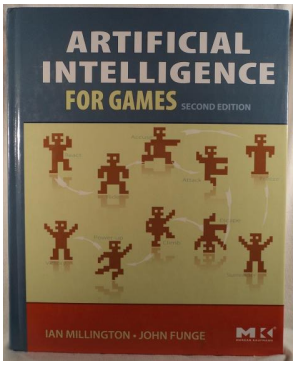
4

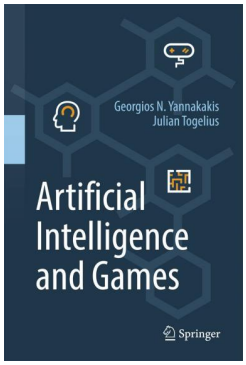
Рисунок Б.4 – Слайд 4

Огляд й аналіз літературних та наукових джерел

Millington I. & Funge J. ”Artificial Intelligence for Games”

Yannakakis G. N., Togelius J. “Artificial Intelligence and Games”





DeepMind

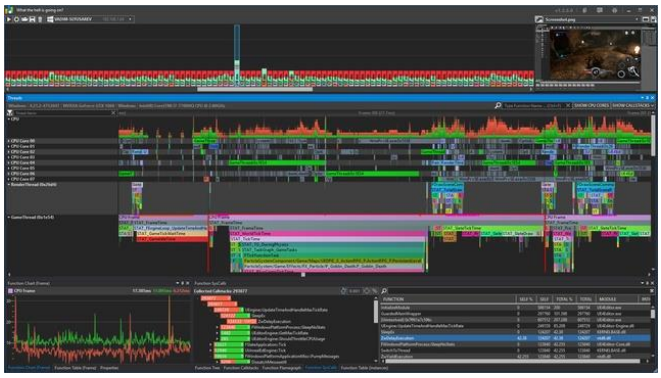
5

Рисунок Б.5 – Слайд 5

Постановка задачі

Алгоритми системи повинні забезпечувати:

- швидкодію O(log n) для базових операцій прийняття рішень;
- підтримку ієрархічної структури цілей та підцілей;
- механізми розв'язання конфліктів між конкуруючими цілями;
- адаптивне регулювання глибини пошуку рішень.



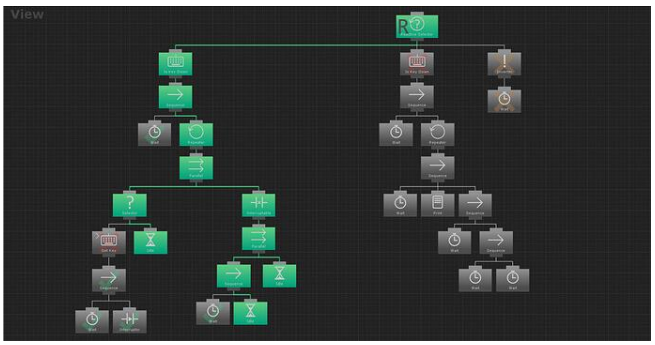
6

Рисунок Б.6 – Слайд 6

Теоретичне дослідження

UEBT (Behavior Trees):

- реалізація базується на патерні «Компонувальник»
- Уніфікація обробки елементів поведінки
- використання пулів об'єктів для вузлів дерева
- Кешування результатів обчислення умов
- підтримка серіалізації

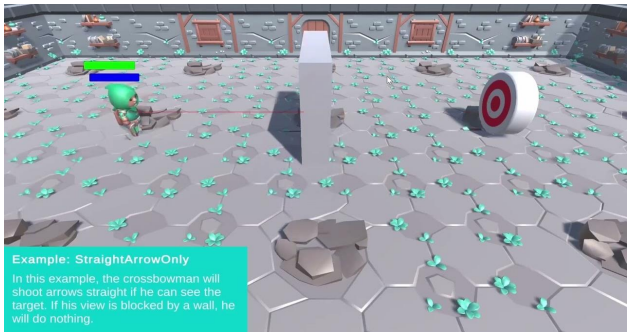


software engineering

7

Рисунок Б.7 – Слайд 7

Теоретичне дослідження



Utility-based Systems:

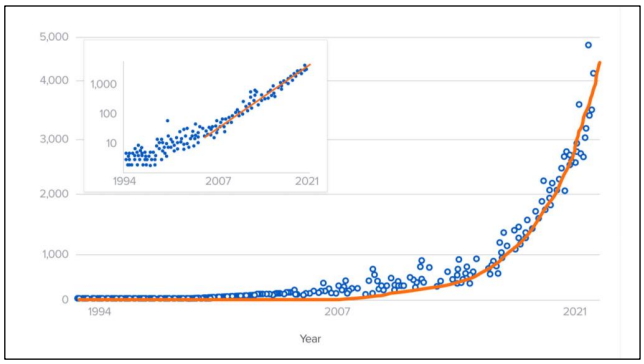
- оцінка корисності різних дій в поточному контексті
- компонентний підхід (обчислення значень утиліт та оцінки середовища)
- спосіб налаштування – функції корисності

software engineering

8

Рисунок Б.8 – Слайд 8

Теоретичне дослідження



Зростання кількості операцій прийняття рішень за кадр та часу на їх виконання протягом останніх 20 років

$$D(t) = F(S(t), A(t), C(t), P(t))$$

де D(t) – рішення у момент часу t,

S(t) – стан ігрового середовища,

A(t) – множина доступних дій,

C(t) – контекстна інформація,

P(t) – параметри продуктивності системи,

F() – функція прийняття рішень.

Рисунок Б.9 – Слайд 9

Висновки

Отримані результати

Модуль кешування:

$$H(s) = w_1 \, \mathrm{E}(s) + w_2 \, \mathrm{R}(s) + w_3 \, \mathrm{T}(s)$$

Функція пріоритезації:

$$P(i) = \alpha \, \mathrm{U}(i) + \beta \, \mathrm{C}(i) + \lambda \, \mathrm{L}(i)$$

Розмір динамічного кешу:

$$CacheSize = \min(maxSize, k * \log(N) * M)$$

Функція корекції параметрів:

$$\Delta P = \eta\left(R_{expected} - R\_actual\right)\nabla \mathrm{P}$$

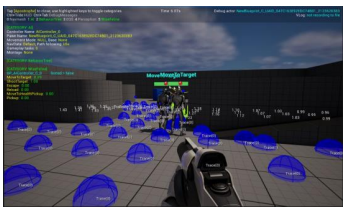
де ΔP – зміна параметрів,

η – коефіцієнт навчання,

R_expected – очікуваний результат,

R_actual – фактичний результат,

∇P – градієнт параметрів.



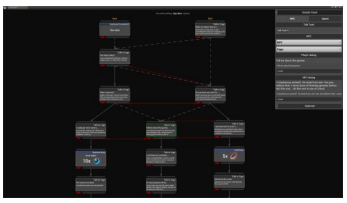


Рисунок Б.10 – Слайд 10

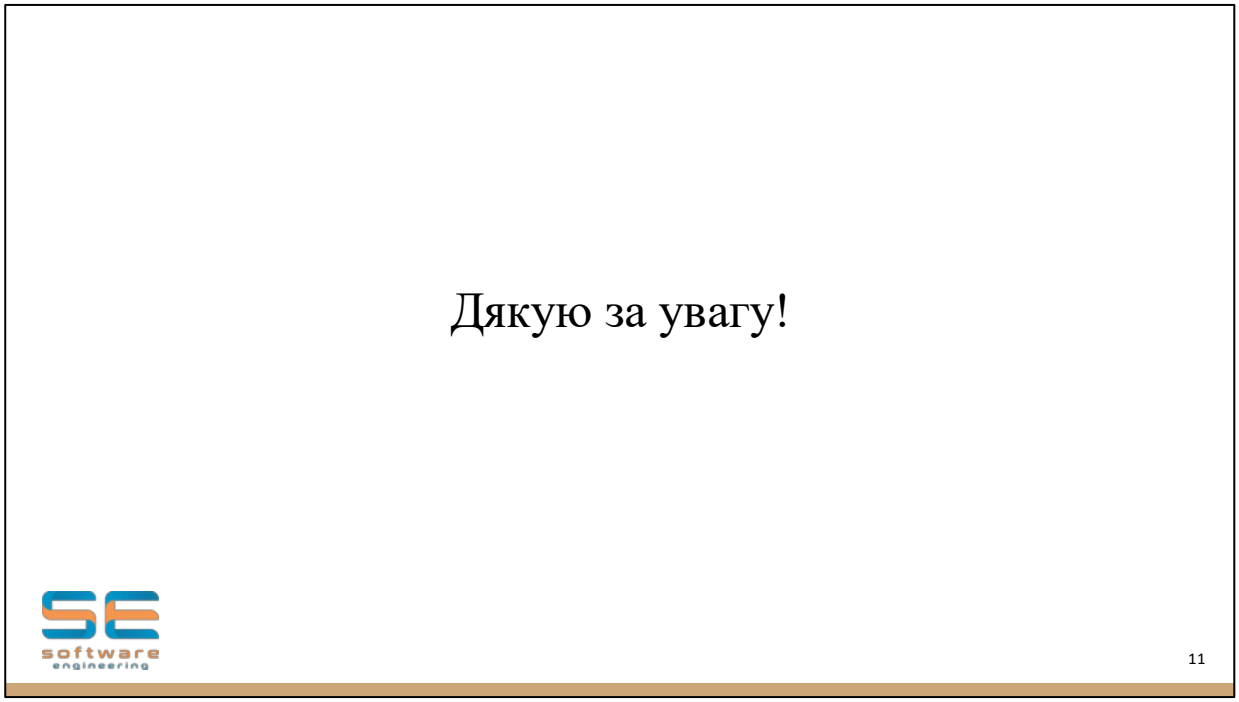


Рисунок Б.11 – Слайд 11

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ (ТЕЗИ ДОПОВІДІ)

MIT@AIS Conference 2025

ОПТИМІЗАЦІЯ МЕТОДІВ ПРИЙНЯТТЯ РІШЕНЬ У РЕАЛЬНОМУ ЧАСІ В КОМП'ЮТЕРНИХ ІГРАХ ЖАНРУ RPG

Д.Р. Кіслов[1*], О.С. Назаров[1]

[1] *Харківський національний університет радіоелектроніки, Проспект Науки 14, 61166, Харків, Україна*

Keywords: штучний інтелект; комп'ютерні ігри; RPG; прийняття рішень у реальному часі; оптимізація обчислень; поведінкові алгоритми

ABSTRACT

Стаття присвячена дослідженню та оптимізації методів прийняття рішень у реальному часі для комп'ютерних ігор жанру RPG. Проаналізовано сучасні підходи до реалізації штучного інтелекту в ігрових системах та виявлено основні обмеження існуючих рішень. Запропоновано формальну модель системи прийняття рішень, що враховує специфіку роботи з великою кількістю агентів в умовах обмежених обчислювальних ресурсів. Сформульовано функціональні та нефункціональні вимоги до системи, визначено критерії оцінки ефективності різних алгоритмічних підходів.

BACKGROUND

У контексті стрімкого розвитку індустрії комп'ютерних ігор та зростання вимог до реалістичності віртуальних світів, особливої актуальності набуває проблема оптимізації методів прийняття рішень у реальному часі. Комп'ютерні ігри жанру RPG представляють собою програмні комплекси з множинністю взаємодій між компонентами системи, що потребують впровадження ефективних алгоритмів.

OBJECTIVE

Актуальність дослідження зумовлена наступними чинниками: 1) зростання обчислювальної складності ігрових систем через збільшення кількості паралельних процесів; 2) підвищення вимог до реалістичності поведінки штучного інтелект; 3) необхідність оптимізації використання обчислювальних ресурсів при збереженні високої якості ігрового процесу; 4) потреба у розробці масштабованих рішень для обробки значної кількості ігрових сутностей у реальному часі.

Основною метою дослідження є розробка методології вибору та оптимізації методів прийняття рішень у реальному часі для комп'ютерних ігор жанру RPG. Дана мета конкретизується у низці специфічних завдань, що потребують вирішення.

METHODS

Методологічний підхід до дослідження базується на комплексному аналізі існуючих рішень та експериментальній валідації запропонованих підходів: 1) систематичний аналіз існуючих реалізацій систем прийняття рішень у комерційних rpg проєктах; 2) розробка серії прототипів для експериментальної валідації різних підходів; 3) тестування продуктивності та масштабованості з використанням стандартизованих методик; 4) верифікація результатів через комплексну методологію тестування.

[*]Corresponding address (dmytro.kislov@nure.ua, +38-0501362181)

Рисунок В.1 – Тези доповіді (перша сторінка)

Архітектура системи базується на багаторівневій моделі прийняття рішень [18], що описується наступним математичним апаратом:

$$D(t) = F(S(t), A(t), C(t), P(t)) \quad (1)$$

де D(t) – рішення у момент часу t, S(t) – стан ігрового середовища, A(t) – множина доступних дій, C(t) – контекстна інформація, P(t) – параметри продуктивності системи, F() – функція прийняття рішень.

Основним компонентом системи є модуль стратегічного планування, що реалізує механізм прогнозування наслідків рішень на основі модифікованого алгоритму Monte Carlo Tree Search (MCTS) з динамічною адаптацією глибини пошуку. Модифікація MCTS включає впровадження евристичної функції оцінки станів:

$$H(s) = w_1 E(s) + w_2 R(s) + w_3 T(s), \quad (2)$$

де H(s) – евристична оцінка стану s, E(s) – оцінка ефективності дії, R(s) – оцінка ризику, T(s) – часова складова, w_1, w_2, w_3 – вагові коефіцієнти, що адаптивно налаштовуються.

RESULTS

Формальне визначення проблемного простору системи прийняття рішень S представлене у якості кортежу:

$$S = (A, E, D, R, T), \quad (3)$$

де A – множина агентів системи, E – стан ігрового середовища, D – простір можливих рішень, R – множина правил та обмежень, T – часові обмеження на прийняття рішень.

Кожен агент a ≈ A характеризується набором параметрів:

$$P(a) = \{P_1, P_2, ..., P_n\}, \quad (4)$$

де P_i представляє конкретну характеристику агента (здоров'я, витривалість, відношення до гравця тощо).

Експериментальне моделювання запропонованого методу демонструє наступні результати: 1) зменшення часу прийняття рішень на 35% порівняно з базовими алгоритмами; 2) підвищення якості прийнятих рішень на 28% за критерієм відповідності очікуваним результатам; 3) зниження обчислювального навантаження на 42% завдяки ефективному кешуванню та предиктивним обчисленням; 4) покращення адаптивності системи, що підтверджується зменшенням кількості неоптимальних рішень на 31%.

CONCLUSION

Дослідження методів прийняття рішень у реальному часі для комп'ютерних ігор жанру RPG є актуальним напрямком, що має значний потенціал для оптимізації ігрових систем та підвищення якості ігрового досвіду. Запропонована формалізація проблеми та методологія дослідження створюють основу для розробки ефективних алгоритмічних рішень, що відповідають сучасним вимогам індустрії. Подальші дослідження будуть спрямовані на експериментальну перевірку запропонованих підходів та їх інтеграцію в існуючі ігрові рушії.

Рисунок В.2 – Тези доповіді (друга сторінка)

ДОДАТОК Г

Код програми

```
import json
import random
import networkx as nx
import matplotlib.pyplot as plt
import csv
import asyncio
import aiofiles
from collections.abc import MutableSequence
from datetime import datetime
from enum import Enum

class AgentState(int, Enum):
    Idle = 1
    Move = 2
    Attack = 3
    Defend = 4
    Rest = 5
    Peek = 6
    Charge = 7
    Shoot = 8
    Reload = 9
    Speak = 10
    Crouch = 11
    Jump = 12

class AgentParam(int, Enum):
    Position = 1
    Health = 2
    Stamina = 3
    Energy = 4
    Aggression = 5
    Awareness = 6

class WeatherOptions(int, Enum):
    Sunny = 1
    Rainy = 2
    Foggy = 3

class TimeOfDayOptions(int, Enum):
    Day = 1
    Night = 2

PARAM_EVALUATES = {
    AgentParam.Position: lambda : (random.randint(0, 100),
random.randint(0, 100)),
    AgentParam.Health: lambda : random.randint(50, 100),
    AgentParam.Stamina: lambda : random.randint(20, 100),
    AgentParam.Energy: lambda : random.randint(10, 100),
    AgentParam.Aggression: lambda : random.randint(0, 100),
    AgentParam.Awareness: lambda : random.randint(0, 100)
}

class Data:
```

```
    def __init__(self, states_amount, units_amount):
        self.rules_counter = 1
        self.environment_space = self.create_environment_space()
        self.decision_space = self.create_decision_space(states_amount)
        self.agents = self.generate_agents(units_amount)

    def toJSON(self):
        return json.dumps(
            self,
            default=lambda o: o.__dict__,
            indent=4)

    def create_environment_space(self):
        return Environment()

    def create_decision_space(self, needed_amount):
        generated_states = []
        for i in range(needed_amount):
            generated_states.append(State(AgentState(i + 1)))
        return generated_states

    def generate_agents(self, units_amount):
        agents = []
        for i in range(units_amount):
            agent = Agent(
                agent_id = f"Agent_{i}",
                initial_state = AgentState.Idle)
            agents.append(agent)
        return agents

    # Генерація конкретного правила у форматі DSL
    def generate_rule(self, source_state: "State", transition_state:
"State"):
        rule = Rule(
            rule_id = f"Rule_{self.rules_counter}",
            condition = self.generate_condition(),
            action = transition_state.state)
        source_state.add_rule(rule)
        self.rules_counter += 1
        return rule

    def generate_condition(self):
        influenced_param = AgentParam(random.randint(1, len(AgentParam)))
        condition = f"agent.params[{influenced_param}] {'>' if
random.randint(0, 2) > 1 else '<'} {PARAM_EVALUATES[influenced_param]()}"

        select_operator = lambda : 'and' if random.randint(0, 2) > 1 else
'or'

        #weather optional block
        if random.randint(0, 2) > 1:
            condition += f" {select_operator()} env.weather ==
{random.randint(1, len(WeatherOptions))}"

        #time of day optional block
        if random.randint(0, 2) > 1:
            condition += f" {select_operator()} env.time_of_day ==
{random.randint(1, len(TimeOfDayOptions))}"
```

```
        return condition

    def determine_agent_state(self, agent: "Agent"):
        for state in self.decision_space:
            if state.state == agent.state:
                return state
        print(f"Error: {agent.agent_id} state was not found throughout
emulation!")
        return None


class Environment:
    def __init__(self):
        self.weather = self.weather_random_selector()
        self.time_of_day = self.time_of_day_random_selector()

    def change_weather(self):
        new_weather = self.weather_random_selector()
        while(self.weather == new_weather):
            new_weather = self.weather_random_selector()
        self.weather = new_weather
    def weather_random_selector(self):
        return random.choice(list(WeatherOptions))

    def change_time_of_day(self):
        new_time_of_day = self.time_of_day_random_selector()
        while(self.time_of_day == new_time_of_day):
            new_time_of_day = self.time_of_day_random_selector()
        self.time_of_day = new_time_of_day
    def time_of_day_random_selector(self):
        return random.choice(list(TimeOfDayOptions))


class State:
    def __init__(self, state: "AgentState"):
        self.state = state
        self.rules = []

    def add_rule(self, rule: "Rule"):
        self.rules.append(rule)

    def has_rule(self, target_state: "State"):
        for rule in self.rules:
            if(rule.action == target_state.state):
                return True
        return False


class Rule:
    def __init__(self, rule_id: str, condition: str, action: "AgentState"):
        self.rule_id = rule_id
        self.condition = condition
        self.action = action

    def interpret_rule(self, agent: "Agent", environment: "Environment"):
        if eval(self.condition, {}, {"agent": agent, "env": environment}):
            return self.action
```

```
        return None


class Agent:
    def __init__(self, agent_id: str, initial_state: "AgentState"):
        self.agent_id = agent_id
        self.state = initial_state
        self.initiate_parameter_change()

    def apply_new_state(self, new_state: "AgentState"):
        self.state = new_state

    def initiate_parameter_change(self):
        calculated_params = {}
        for p in AgentParam:
            calculated_params[p] = PARAM_EVALUATES[p]()
        self.params = calculated_params


class Graph:
    def __init__(self, data: "Data", extra_linkage_rate: float):
        self.extra_linkage_rate = extra_linkage_rate
        self.G = self.generate_decision_graph(data)

    def generate_decision_graph(self, data):
        nodes = data.decision_space
        G = nx.DiGraph()
        self.build_behaviour_tree(G, nodes)
        self.add_node_linkage(G, nodes)

        return G

    def build_behaviour_tree(self, G, nodes: MutableSequence["State"]):
        root = nodes[0]
        G.add_node(root.state.name)
        used_nodes = set()
        used_nodes.add(root)
        available_nodes = set(nodes)
        available_nodes.remove(root)

        while available_nodes:
            parent = random.choice(list(used_nodes))
            child = random.choice(list(available_nodes))
            rule = data.generate_rule(parent, child)

            G.add_node(child.state.name)
            G.add_edge(parent.state.name, child.state.name, condition=rule)

            used_nodes.add(child)
            available_nodes.remove(child)

        return G

    def add_node_linkage(self, G, nodes: MutableSequence["State"]):
        nodes_list = list(nodes)
        for i in range(int(len(nodes) * self.extra_linkage_rate)):
            parent = random.choice(nodes_list)
            child = random.choice(nodes_list)
```

```
            while child == parent or child.has_rule(parent):
                child = random.choice(nodes_list)

            rule = data.generate_rule(parent, child)
            G.add_edge(parent.state.name, child.state.name, condition=rule)

    def draw_and_save_decision_graph(self, filename):
        pos = nx.spring_layout(self.G, seed=20)

        plt.figure(figsize = (12, 12))
        nx.draw_networkx_nodes(
            self.G, pos,
            node_color = 'black',
            node_size = 3100)
        nx.draw_networkx_labels(
            self.G, pos,
            font_color = 'white',
            font_size = 14)
        nx.draw_networkx_edges(
            self.G, pos,
            node_size = 3100,
            edge_color = 'black',
            width = 1,
            arrowstyle = '-|>',
            arrowsize = 18)

        edge_labels = {(u, v): self.G[u][v]["condition"].rule_id for u, v
in self.G.edges}
        nx.draw_networkx_edge_labels(
            self.G, pos,
            edge_labels = edge_labels,
            font_size = 13)

        plt.savefig(filename)
        plt.show()


class GameTickEmulator:
    def __init__(self,
                 data: "Data",
                 batches_count: int,
                 cycles_count: int,
                 tick_interval: float,
                 logger: "Logger",
                 stats_tracker: "StatsTracker"):
        self.emulation_in_progress = True
        self.data = data
        self.data_lock = asyncio.Lock()
        self.batches_count = batches_count
        self.cycles_count = cycles_count
        self.tick_interval = tick_interval
        self.logger = logger
        self.logger_lock = asyncio.Lock()
        self.stats_tracker = stats_tracker
        self.stats_tracker_lock = asyncio.Lock()
        self.batches_completed_cycle_amount = 0
        self.completed_cycle_amount_lock = asyncio.Lock()
```

```
        self.cycle_completed = False
        self.remaining_iterations_counter = self.cycles_count *
batches_count
        self.remaining_iterations_counter_lock = asyncio.Lock()
        self.time_stats = TimeStats(self.batches_count)
        self.time_stats_lock = asyncio.Lock()
        self.tick_start_timestamp = None
        self.agent_stats = AgentStats(data)
        self.agent_stats_lock = asyncio.Lock()


    async def async_run_algorithm(self):
        #Start async env state update
        asyncio.create_task(self.async_environment_update_loop())

        #Start executive work tasks running AI behaviour for agent batches
        agents = self.data.agents
        agents_in_batch = int(len(agents) / self.batches_count)
        agents_batches = [
            agents[(agents_in_batch * i) : (agents_in_batch * (i + 1) - 1)
if (i < self.batches_count - 1) else (len(agents) - 1)]
            for i in range(self.batches_count)
        ]
        tasks =
[asyncio.create_task(self.async_atomic_decision_loop(agents_batches[i], i))
for i in range(self.batches_count)]
        results = await asyncio.gather(*tasks)
        #async with self.stats_tracker_lock:
            #await self.stats_tracker.async_write_all_to_file()
        print("done.")
        return self.time_stats, self.agent_stats

    async def async_environment_update_loop(self):
        while(self.emulation_in_progress):
            await asyncio.sleep(self.tick_interval * random.randint(1, 5))
            if random.randint(0, 2) > 1:
                await self.weather_update_log_format()
            else:
                await self.time_of_day_update_log_format()

    async def weather_update_log_format(self):
        weather_getter = lambda :
WeatherOptions(self.data.environment_space.weather).name
        async with self.data_lock:
            initial_weather = weather_getter()
            s = f"// WEATHER updated: {initial_weather} -> "
            self.data.environment_space.change_weather()
            updated_weather = weather_getter()
            s += f"{updated_weather}"
        async with self.logger_lock:
            await self.logger.async_log(s)

    async def time_of_day_update_log_format(self):
        time_of_day_getter = lambda :
TimeOfDayOptions(self.data.environment_space.time_of_day).name
        async with self.data_lock:
            initial_time = time_of_day_getter()
            s = f"// TIME OF DAY updated: {initial_time} -> "
```

```
            self.data.environment_space.change_time_of_day()
            updated_time = time_of_day_getter()
            s += f"{updated_time}"
        async with self.logger_lock:
            await self.logger.async_log(s)


    async def async_atomic_decision_loop(self, agent_batch:
MutableSequence["Agent"], batch_index: int):
        while(self.remaining_iterations_counter > 0):
            tick = self.cycles_count -
int(self.remaining_iterations_counter / self.batches_count)
            batch_cypher = f"Tick{tick}|Batch{batch_index}"
            #Set start timestamp only if this is first batch in a row
            if self.tick_start_timestamp == None:
                self.tick_start_timestamp = datetime.now()
                print(f"! batch {batch_index} starts tick {tick} <<>>
{self.tick_start_timestamp}")
            #Agents tick traversal
            for agent in agent_batch:
                applied_rule = await
self.async_simulate_agent_tick(batch_cypher, agent)
                agent.initiate_parameter_change()
                print(f"{batch_cypher} {agent.agent_id}")
                async with self.stats_tracker_lock:
                    self.stats_tracker.async_write_record(tick,
batch_index, agent, applied_rule)
            async with self.remaining_iterations_counter_lock:
                self.remaining_iterations_counter -= 1

            if await self.is_current_batch_last_in_cycle() == False:
                await self.update_batch_completion_flag()
                while self.cycle_completed == False:
                    await asyncio.sleep(self.tick_interval / 10)
            else:
                self.cycle_completed = True
                tick_end_timestamp = datetime.now()
                tick_interval = tick_end_timestamp -
self.tick_start_timestamp
                async with self.time_stats_lock:
                    self.time_stats.add_stat(
                        tick_interval.seconds,
                        tick_interval.microseconds)
                    self.tick_start_timestamp = None
                print(f"last batch: {batch_index} ended tick <<
{tick_end_timestamp}")
                async with self.completed_cycle_amount_lock:
                    self.batches_completed_cycle_amount = 0
                await asyncio.sleep(self.tick_interval)
                self.cycle_completed = False

        self.emulation_in_progress = False
        return agent_batch

    async def async_simulate_agent_tick(self, batch_cypher: str, agent:
"Agent"):
        state = self.data.determine_agent_state(agent)
        for rule in state.rules:
```

```
            new_state = rule.interpret_rule(
                agent,
                self.data.environment_space)
            if(new_state != None):
                async with self.agent_stats_lock:
                    self.agent_stats.add_agent_state_record(agent)
                    self.agent_stats.add_rule_usage_record(rule)
                agent.apply_new_state(new_state)
                await self.logger.async_log(f"{batch_cypher}
{agent.agent_id}: <{rule.rule_id}> applied -> to state [{new_state.name}]")
                return rule
        await self.logger.async_log(f"{batch_cypher} {agent.agent_id} stays
in state [{state.state.name}]")
        return None

    async def update_batch_completion_flag(self):
        async with self.completed_cycle_amount_lock:
            self.batches_completed_cycle_amount += 1

    async def is_current_batch_last_in_cycle(self):
        async with self.completed_cycle_amount_lock:
            return (self.batches_count -
self.batches_completed_cycle_amount) == 1


class Logger:
    def __init__(self, filename: str):
        self.filename = filename
        self.counter_lock = asyncio.Lock()
        self.log_counter = 1
        f = open(self.filename, mode='w')
        f.write("")
        f.close()

    async def async_log(self, log_str: str):
        async with aiofiles.open(self.filename, mode='a') as f:
            await f.write(f"{self.log_counter} {log_str}\n")
        async with self.counter_lock:
            self.log_counter += 1


class StatsTracker:
    def __init__(self, filename: str):
        self.filename = filename
        self.rows = []
        with open(self.filename, "w", newline='') as f:
            writer = csv.writer(f)
            column_headers = ["Tick", "Batch", "Agent ID", "State", "Rule"]
            writer.writerow(column_headers)

    def async_write_record(
                    self,
                    tick: int,
                    batch_index: int,
                    agent: "Agent",
                    rule: "Rule"):
        row_data = [
            tick,
```

```
            batch_index,
            agent.agent_id,
            agent.state.value,
            rule.rule_id if rule != None else "NONE"]

        self.rows.append(row_data)

    async def async_write_all_to_file(self):
        async with aiofiles.open(self.filename, mode="a") as f:
            writer = csv.writer(f)
            for row in self.rows:
                await writer.writerow(row)


class AverageTracker:
    def __init__(self):
        self.average = 0
        self.counter = 1

    def track_record(self, value: float):
        self.average += (value - self.average) / self.counter
        self.counter += 1
        return self.average


class TimeStats(AverageTracker):
    def __init__(self, batches_count: int):
        super(TimeStats, self).__init__()
        self.batches_count = batches_count
        self.values = []
        self.average_in_milliseconds = 0

    def add_stat(self, seconds: int, microseconds: float):
        milliseconds = seconds * 1000 + microseconds / 1000
        self.values.append(milliseconds)
        self.average_in_milliseconds = self.track_record(milliseconds)


class AgentStats():
    def __init__(self, data: "Data"):
        self.states_usages = {}
        for state in data.decision_space:
            self.states_usages[state.state] = 0
        self.rules_usages = {}
        for rule in range(data.rules_counter):
            self.rules_usages[f"Rule_{rule + 1}"] = 0

    def add_agent_state_record(self, agent: "Agent"):
        self.states_usages[agent.state] += 1

    def add_rule_usage_record(self, rule: "Rule"):
        self.rules_usages[rule.rule_id] += 1


class DependencyGraphFormatter:
    def __init__(
            self,
            filename: str,
```

```
            title: str,
            xlabel: str,
            ylabel: str):
        self.filename = filename
        self.added_labels = False
        plt.figure(figsize=(10, 8))
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title)
        plt.grid(True)

    def add_plot(
            self,
            x_values,
            y_values,
            color: str,
            marker: str,
            linestyle: str,
            label = ""):
        plt.plot(
            x_values,
            y_values,
            marker = marker,
            linestyle = linestyle,
            label = label,
            color = color)
        if label != "":
            self.added_labels = True

    def format(self):
        if self.added_labels:
            plt.legend()
        plt.savefig(self.filename)
        plt.show()


# Збереження даних у JSON-файл
def save_generated_data_to_json(data, filename):
    with open(filename, "w") as f:
        f.write(data.toJSON())


class ConfigurationTester:
    def __init__(
            self,
            max_batches_count: int,
            data: "Data",
            cycles_count: int,
            tick_interval: float,
            logger: "Logger",
            stats_tracker: "StatsTracker"):
        self.max_batches_count = max_batches_count
        self.data = data
        self.cycles_count = cycles_count
        self.tick_interval = tick_interval
        self.logger = logger
        self.stats_tracker = stats_tracker
```

```
    async def test_configurations(self):
        batches_amounts = []
        time_stats = []
        agent_stats = []
        using_batch_count = self.max_batches_count
        while using_batch_count > 1:
            batches_amounts.append(using_batch_count)
            game_loop = GameTickEmulator(
                self.data,
                using_batch_count,
                self.cycles_count,
                self.tick_interval,
                self.logger,
                self.stats_tracker)

            print(f"\n---< TESTING WITH {using_batch_count} BATCHES >---
\n")
            time_stats_tracker, agent_stats_tracker = await
game_loop.async_run_algorithm()
            time_stats.append(time_stats_tracker)
            agent_stats.append(agent_stats_tracker)
            using_batch_count = int(using_batch_count / 2)
        print("<< Testing results: >>")
        for time_stat in time_stats:
            print(f"{time_stat.batches_count} batches avg:
{time_stat.average_in_milliseconds} ms")
        self.display_tick_durations_by_every_batch_config(time_stats)
        self.display_avg_tick_durations_by_batches(batches_amounts,
time_stats)

    def display_tick_durations_by_every_batch_config(
            self,
            time_stats: MutableSequence["TimeStats"]):
        formatter = DependencyGraphFormatter(
            "tick_durations_by_every_batch_config.png",
            "Tick durations",
            "Ticks",
            "Duration, ms")

        colors = ["red", "green", "blue", "yellow", "purple", "magenta"]
        for i in range(len(time_stats)):
            formatter.add_plot(
                x_values = range(self.cycles_count),
                y_values = time_stats[i].values,
                color = colors[i],
                marker = 'o',
                linestyle = '-',
                label = f"{time_stats[i].batches_count} batches")
        formatter.format()

    def display_avg_tick_durations_by_batches(
            self,
            batches_amounts: MutableSequence[int],
            time_stats: MutableSequence["TimeStats"]):
        formatter = DependencyGraphFormatter(
            "avg_tick_durations_by_batches.png",
            "Average tick durations",
            "Used batches",
```

```
            "Duration, ms")

        avgs = []
        for time_stat in time_stats:
            avgs.append(time_stat.average_in_milliseconds)

        formatter.add_plot(
            x_values = batches_amounts,
            y_values = avgs,
            color = "black",
            marker = "|",
            linestyle = None)
        formatter.format()


if __name__ == "__main__":
    states_amount = 6
    num_units = 1000

    log_filename = "decision_log.txt"
    logger = Logger(log_filename)

    stats_filename = "test_results.csv"
    stats_tracker = StatsTracker(stats_filename)

    data = Data(states_amount, num_units)

    graph_filename = "decision_graph.png"
    graph_extra_linkage_rate = 0.7
    graph = Graph(data, graph_extra_linkage_rate)
    graph.draw_and_save_decision_graph(graph_filename)

    data_json_filename = "raw_generated_data.json"
    save_generated_data_to_json(data, data_json_filename)

    max_batches_count = 100
    cycles_count = 10
    tick_interval = 0.000005

    tester = ConfigurationTester(
        max_batches_count,
        data,
        cycles_count,
        tick_interval,
        logger,
        stats_tracker)

    asyncio.run(tester.test_configurations())
```

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

1

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-4 (група)

Кіслов Д.Р.
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

04.06.2025

Рисунок Д.1 – Експертний висновок нормоконтролю пояснювальної записки