МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА БІБЛІОТЕКИ АГЕНТНОГО МОДЕЛЮВАННЯ»**

Виконав(ла): студент(ка) 2 курсу, групи 8.1218

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

І.С. Волгін
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 – інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.

(підпис)

« ____ » _____ 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Волгіна Ігоря Сергійовича

(прізвище, ім'я та по батькові)

1. Тема роботи Розробка бібліотеки агентного моделювання

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н.

(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » 05 2020 року № 576-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 30.05.2020

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	вересень 2020	Виконано
2.	Збір вихідних даних.	вересень 2020	Виконано
3.	Обробка методичних та теоретичних	жовтень 2020	Виконано
	джерел.		
4.	Розробка першого та другого розділу.	листопад 2020	Виконано
5.	Розробка третього розділу.	листопад 2020	Виконано
6.	Оформлення та нормоконтроль	листопад 2020	Виконано
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.12.2020	

Студент _____ І.С. Волгін
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка бібліотеки агентного моделювання»: 44 с., 18 рис., 18 джерел, 3 таблиці, 3 додатки.

AGENTS, INTELLIGENT, AI, MODELING

Об’єкт дослідження – процес агентного моделювання.

Мета роботи: розробка бібліотеки агентного моделювання.

Метод дослідження – теоретичний, аналітичний, практичний.

У першому розділі було проведене дослідження літератури, в результаті чого було дане визначення агенту, середовища, проаналізовані та наведені типи агентів.

Проаналізовані методології розробки агентів та їх використання. Зібрана інформація про поточні актуальні інструменти та рішення і проведено аналіз.

У другому розділі реалізована бібліотека на PHP для агентного моделювання, яка включає в себе абстрактний клас середовища та абстрактний клас агентів.

У третьому розділі на базі створеної бібліотеки реалізована модель поведінки еволюційних юнітів. Агенти мають специфічні методи взаємодії з середовищем яке, у свою чергу надає цілі для агентів. Були проведені тести які наглядно ілюструють результат роботи моделювання.

SUMMARY

Master's Qualification Thesis «Development of the Agent-Based Modelling Library» contains: 44 pages, 18 figures, 18 sources, 3 tables ,3 supplements.

AGENTS, INTELLIGENT, AI, MODELING

The object of the study is process of agent-based modelling.

The aim of the study is developing of methods of agent-based modelling.

Methods of research are theoretical, analytical, practical.

First chapter dedicated to literature review which helped to provide a definition to an agent, environment and also analyzed and provided types of agents.

Researched methodologies of agents developing and their usage. Gathered information about current agent-based instruments and solutions.

In second chapter was described PHP library for agent-based modelling described. It includes environment and agent abstract classes.

Third chapter is devoted to evolutional units modeling. Agents have specific methods for interaction with environment which provides goals for the agents. Finally, was made demonstrative tests of newly created model.

ЗМІСТ

ВСТУП

За останній час програмування отримало стрімкий розвиток та набуло велику кількість різноманітних мов, підходів, напрямків, використовується у найрізноманітніших галузях – від сім-карт до космонавтики.

Зараз є безліч готових рішень під майже будь-які звичні задачі, соціальні мережі, інформаційні мережі та науково-публіцистичний платформи.

Тим часом як частина програмістів створює та обслуговує такі продукти, інша частина працює над тим, з чим ми стикаємося, але не завжди розуміємо яких методів чи алгоритмів це створено.

Прогнозування погоди, проведення краш-тестів без участі реальних машин, встановлення діагнозу на базі опису досліджень, покращення фото, автопілоти, прогнозування розвитку пандемії та багато іншого працює на базі штучного інтелекту. Це – майбутнє цифрового віку, та саме розвиток цієї галузі надає нам змогу розвивати технології в цілому.

Якщо узагальнити поняття штучного інтелекту, то прийдемо до поняття інтелектуального агенту – що і буде головною темою даної роботи.

Метою кваліфікаційної кваліфікаційної роботи магістра є створення бібліотеки агентного моделювання та використання її на практиці.

Виходячи із мети, можна сформулювати наступні завдання:

а) визначення інтелектуального агента;

б) задачі та методи вирішення за допомогою інтелектуальних агентів;

в) створити бібліотеку агентного моделювання;

г) вирішити прикладну задачу за допомогою неї.

1 АНАЛІЗ ЛІТЕРАТУРИ

1.1 Визначення інтелектуального агента, середовища, типи агентів

Агентом є все, що може сприймати своє середовище за допомогою датчиків і впливає на це середовище за допомогою виконавчих механізмів. [2] Агенти можна вважати чеврів, собак, термостати, літаки, роботи, людей, компанії та держави.

Нас цікавить що агент робить – тобто ми судимо агентів по їх поведінці. Агент діє розумно, коли:

- те, що він робить відповідає обставинам та цілям;
- він реактивно реагує на змінення середовища та може мати або навіть змінювати свої цілі;
- вчиться на базі досвіду;
- також агент робить рішення враховуючи свої можливості та ліміти [1];

Обчислювальний агент - це агент, рішення якого можна пояснити з точки зору обчислення. Тобто рішення може бути розбито на примітивну операцію, яка може бути реалізована у фізичному пристрої. Це обчислення може мати різні форми. У людей це обчислення виконуються у «мокрому програмному забезпеченні» (мозок), в комп’ютерах - в «апаратному». Можна прийняти той факт, що існують агенти, які не є обчислювальними, наприклад вітер та дощ, що розмиває пейзаж, залишається відкритим питання, чи всі розумні агенти є обчислювальними.

Основною науковою метою штучного інтелекту є розуміння принципів, які роблять інтелектуальну поведінку можливою в природних або штучних системах, наприклад:

- аналіз природних та штучних речовин;

- формулювання та перевірка гіпотез щодо того, що потрібно для побудови інтелектуальних агентів;
- проектування, побудова та експерименти з обчислювальними системами, які виконують завдання, які зазвичай розглядаються як такі, що вимагають інтелекту.

Як частина науки, дослідники будують емпіричні системи для перевірки гіпотез або для вивчення простору можливостей. Вони цілком відрізняються від програм, які створені для того, щоб бути корисними для домену додатків.

Зверніть увагу, що визначення не стосується інтелектуальної думки, або рішення. Нас цікавить близьке до розумного мислення лише настільки, наскільки це призводить до кращих результатів. Роль думки полягає у впливі на дії.

Основною інженерною метою штучного інтелекту є розробка та синтез корисних, розумних артефактів. Ми насправді хочемо створити агенти, які діють розумно. Такі агенти корисні в багатьох додатках.

Підсумовуємо, що інтелектуальний агент має нести деяку користь. Набір датчиків, виконавчих механізмів та вміння аналізувати та використовувати механізми – можна назвати інтелектуальним агентом.

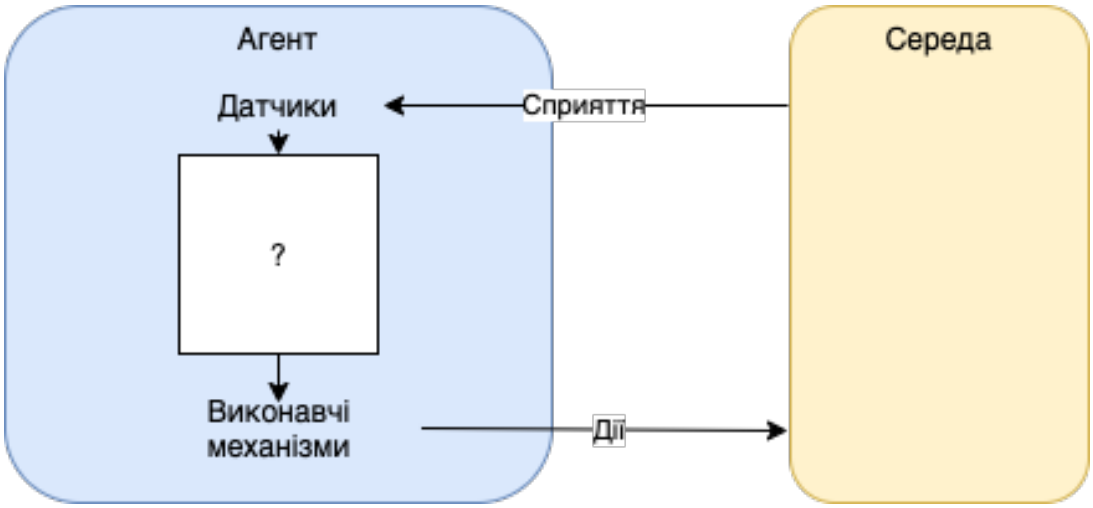

Рисунок 1.1 – Візуальне представлення базової моделі агента

Людина, як агент має вуха, очі тактильні органи почуття, а виконавчі механізми – це голос, руки, ноги, рот та інші частини тіла. Робот, у свою чергу,

може мати інфрачервоні сенсори, камери, звукові уловлювачі, а виконавчі механізми – це маніпулятори, двигуни та інші частини. Програмне забезпечення, таке, як агент, отримує на вході коди клавіш, файли, пакети даних, а взаємодія зі середою йде через пакети, інформацію на екрані, передачею даних іншим агентам.

Послідовністю актів сприйняття агента – це уся історія того, що агент сприйняв. Тобто якщо є змога визначити що агент буде робити при тому чи іншому стану середи – то математичною мовою це буде називатися функцією агенту.

Визначимо поняття програми агента – це реалізація у рамках архітектури агента. Тобто тут ми задаємо те, що агент буде спроможний робити [2]

Для прикладу візьмемо світ, показаний на рис. 1.2, де працює пилосос. Цей світ створений людиною, дуже простий, та ми можемо легко його описати. Є два квадрати A та B, у яких знаходиться наш агент. У квадратах є сміття, агент може пересуватися між A та B, аналізувати квадрат на наявність сміття та прийеняти рішення всмоктувати його. Тобто одна із його функцій якщо у квадраті є сміття – всмоктати його, якщо немає нічого – перейти у інший. Часткова табуляція функцій у таблиці на рисунку 1.3, а програма – рис. 1.4.

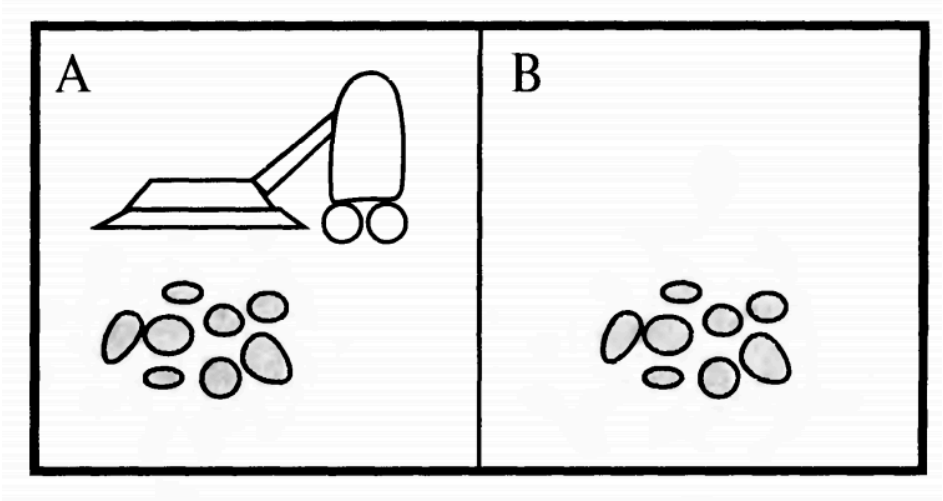

Рисунок 1.2 – Світ пилососу

[A,Clean]	Right
[A,Dirty]	Suck
[B,Clean]	Left
[B,Dirty]	Suck
[A,Clean],[A,Clean]	Right
[A,Clean],[A,Dirty]	Suck
…	…
[A,Clean],[A,Clean],[A,Clean]	Right
[A,Clean],[A,Clean],[A,Dirty]	Suck
…	…

Рисунок 1.3 – Часткова табуляція функції простого агенту пилососу, де зліва послідовність актів сприйняття, а справа прийняті дії

```
function Table-Driven-Agent (percept) returns дія action
        static percepts, послідовність актів сприйняття,
                         спочатку порожня
        table,  таблиця дій, індексована по послідовностям актів
                сприйняття та повністю задана з самого початку

        додати результати сприйняття percept до кінця
        послідовності percepts

        action <--- Lookup(percepts, table)
        return action
```

Рисунок 1.4 – Програма агента пилосос

У прикладі з пилососом ми надали йому ціль – прибрати сміття. Якщо він буде вміти його скидати, а також вливати на швидкість руху – тоді питання буде стояти так – що йому робити. Тут з’являється поняття раціональності агента. Раціональний агент – це той, у якого кожна запис у таблиці функцій буде вірною? Вірна функція – це та, яка повисить показник його продуктивності. Тобто раціональний агент повинен обирати найкращий шлях на базі той інформації о середі та минулих рішеннях, що він вже робив, тобто знань.

У прикладі з пилососом раціональний агент після всмоктування сміття перейде на іншу клітину та перевірить чи немає там роботи, а якщо є – зробить її. Але йому нічого не заважає просто кататися з клітини у клітину та нічого не

робити, тому ми повинні якось нагороджувати агента за вірні рішення, а за невірні штрафувати – таким чином буде реалізований раціональний агент.

Важлива складова світу агенту – це його середовище. Проблемне середовище – це те, для чого агент є рішенням.

Властивості середовища виділимо наступні: середовище може бути повністю чи частково спостерігаємо – де якщо агент сприймає одразу всі аспекти середовища – це повністю, а якщо тільки частину – частково.

Також можна виділити детерміноване або стохастичне середовище, де детерміноване – це якщо наступний стан середовища буде диктований дією виконаною агентом, у іншому випадку – таке середовище називається стохастичним.

Виділяють епізодичне чи послідовне середовище. Характеристикою такого середовища є наступна поведінка – агент приймає рішення епізод за епізодом без знань про те, що було раніше. Послідовним середовищем називають поведінку, якщо агент приймає рішення наперед, тобто прораховує усі можливі функції та обирає ту, яка повисить ефективність наступної.

Статичне чи динамічне – у статичному середовищі з кожним вибором агента нічого не змінюється, а у динамічному – середовище диктує агенту потребу вибору, тобто якщо з плином часу агент нічого не вирішив – це буде як вибір нічого не робити.

Дискретне та безперервне– у дискретному середовищі є кінцева кількість дій, а у безперервному – нескінченна, де агент повинен на протязі всього часу контролювати ситуацію навколо.

Важливим розділенням є на одноагентне чи багатоагентне. Хоча може здатися, що одноагентна система може бути простішою у вирішенні складної складної задачі, проте, часто буває навпаки. Насправді, коли контроль розподіляється між різними агентами, окремий агент може бути простішим.

Агентам у багатоагентній системі притаманні такі властивості, як неоднорідність знань агента – вона межує між зайвою та недостатньою.

Методи, що використовуються для розподілу контролю, можуть бути: доброзичливими або конкурентними, командними або ієрархічними, статичними або змінними ролями.

Агенти можуть спілкуватися між іншими агентами, на дошці або за допомогою повідомлень, це може бути зв'язок низького або високого рівня.

У багатоагентній системі є кілька агентів, здатних моделювати цілі та дії один одного.

У загальній мультиагентній системі між агентами можуть бути прямі взаємодії. Міжагентське спілкування розглядається окремо від спілкування з навколишнім середовищем. Основна відмінність від одноагентної системи полягає в тому, що в мультиагентних системах динаміка середовища може визначатися також іншими агентами, що може впливати на навколишнє середовище непередбачуваним чином. Таким чином, усі мультиагентні системи можна розглядати як такі, що мають динамічне середовище [3].

Інтелектуальних агентів можна класифікувати на декілька типів:

а) прості рефлексні агенти;

б) рефлексні агенти, базовані на моделі;

в) агенті, діючі на основі цілі;

г) агенти, діючі на базі корисності;

Нижче приведені приклади середовищ, які були розглянуті у роботі [2]. Тут показано які характеристики мають проблемні середовища по класифікації, показаній вище. Можемо побачити, що кросворд – це повністю спостережувана, послідовна, детермінована та дискретна, а керування таксі – динамічна, не контролюєма агентом, тобто стохатична система, так як на дорозі є інші автомобілі, які не підкорюються якоїсь єдиній системі керування, до того ж агент не може бачити усю систему цілком, а тільки те, що можуть дозволити сенсори, таким чином це частково спостережувана система.

Проблемная среда	Наблюдаемая полностью или частично	Детерминированная, стратегическая или стохастическая	Эпизодическая или последовательная	Статическая, динамическая или полудинамическая	Дискретная или непрерывная	Одноагентная или мультиагентная
Решение кроссворда	Полностью наблюдаемая	Детерминированная	Последовательная	Статическая	Дискретная	Одноагентная
Игра в шахматы с контролем времени	Полностью наблюдаемая	Стохастическая	Последовательная	Полудинамическая	Дискретная	Мультиагентная
Игра в покер	Частично наблюдаемая	Стохастическая	Последовательная	Статическая	Дискретная	Мультиагентная
Игра в нарды	Полностью наблюдаемая	Стохастическая	Последовательная	Статическая	Дискретная	Мультиагентная
Вождение такси	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Мультиагентная
Медицинская диагностика	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Одноагентная
Анализ изображений	Полностью наблюдаемая	Детерминированная	Эпизодическая	Полудинамическая	Непрерывная	Одноагентная
Робот-сортировщик деталей	Частично наблюдаемая	Стохастическая	Эпизодическая	Динамическая	Непрерывная	Одноагентная
Контроллер очистительной установки	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Непрерывная	Одноагентная
Интерактивная программа, обучающая английскому языку	Частично наблюдаемая	Стохастическая	Последовательная	Динамическая	Дискретная	Мультиагентная

Рисунок 1.5 – Варіанти середовищ та їх характеристик

Прості рефлексні агенти. Цей тип найпростіший – він базується на тому, що зараз бачить агент. Базується на правилі «умова-дія», тобто if – do.

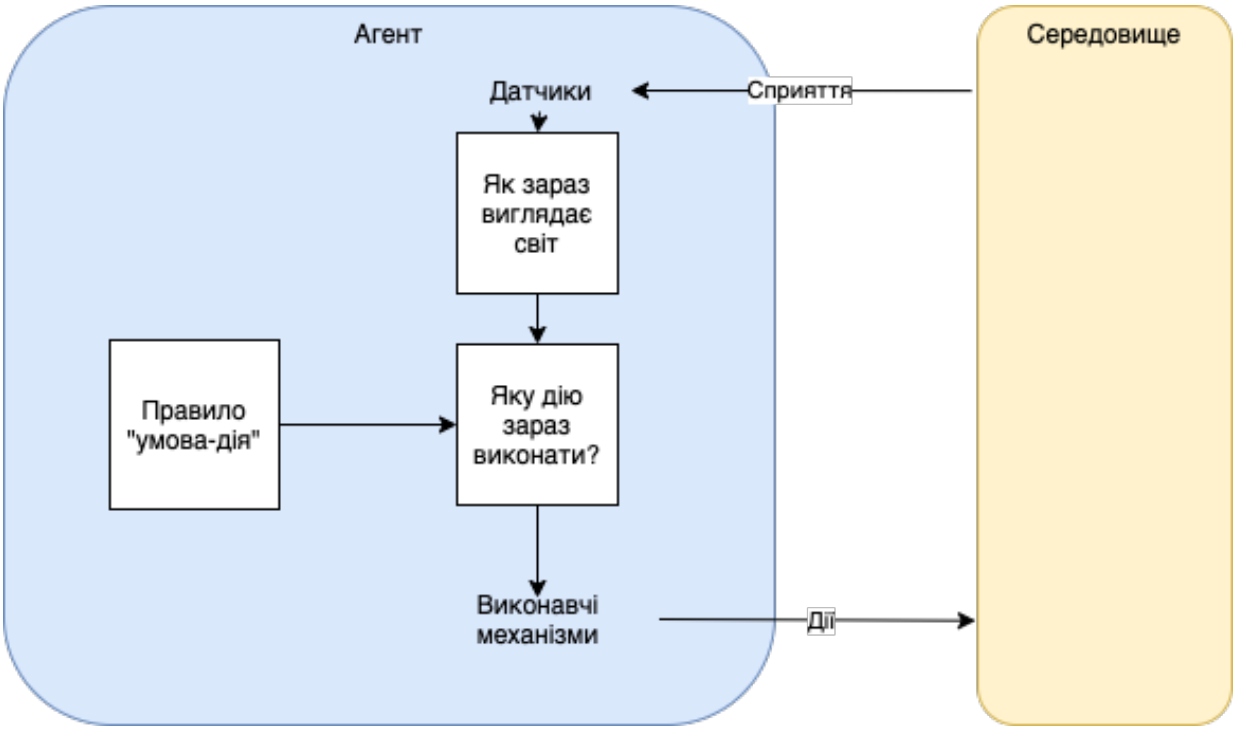

Рисунок 1.6 – Простий рефлексний агент

Рефлексні агенти, базовані на моделі. Такі агенти мають ідентичний набір правил, як рефлексні, але в них є поняття внутрішнього стану. Такий агент повинен мати модель світу навколо нього, що би мати можливість, приймаючи рішення, розуміти що буде після кожної із дій. Наприклад, якщо взяти автомобіль як агента, і уявити, що перед нами їде інший автомобіль, то агент має розуміти, що агент спереду рухається, тобто зберігати стан світу і звіряти з тим, що відбувається зараз. Таким чином, якщо з часом відстань між автомобілями скорочується – це буде означати, що автомобіль гальмує, а значить – треба також гальмувати. До того ж поняття, що виїзд на зустрічну полосу – це погано найкраще застосовується для рефлексного агента.

Нижче приведена структура такого агента.

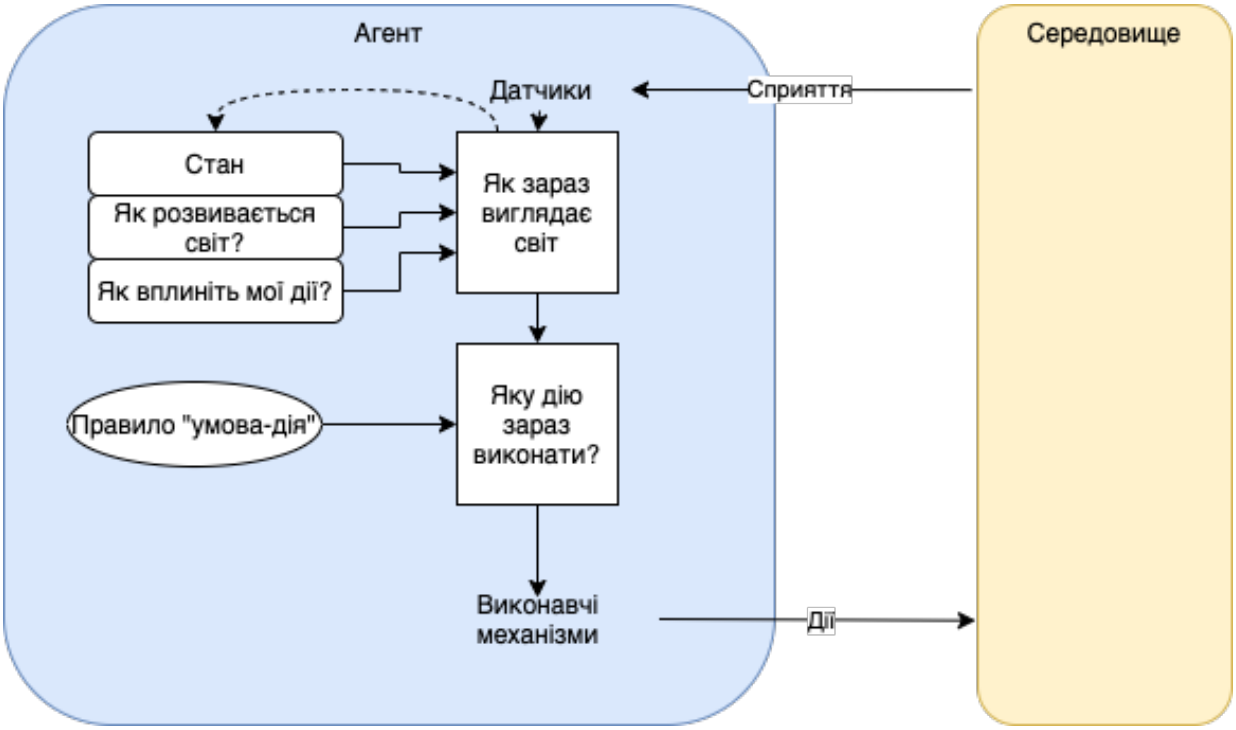

Рисунок 1.7 – Рефлексний агент, заснований на моделі

Агенти, діючі на основі цілі. Такі агенти мають ціль, та усі свої дії обирають так, що б досягнути її. Якщо взяти приклад з автомобілем – то тут автомобілю потрібно на перехресті зрозуміти куди повертати, більш того, йому буде потрібно завчасно змінити смугу, що б досягнути цілі. Агентів тільки цільових зазвичай не проектують, і використовують комбіновані типи (див. рис. 1.8).

На рисунку бачимо, що модель водія таксі – це тип заснований на моделі та цілі. В такого агента є ціль – добратися з точки А у точку Б, але також є уявлення про модель світу – правила дорожнього руху, знаки світлофорів, стоп-сигнали автомобілів, а також потік інформації з бортових сенсорів. Автопілот у цьому випадку ще й аналізує дистанцію ззаду, це також впливає на те як він вийде з можливої ситуації на дорозі. Найбільш яскравий приклад різниці з агентом на рис. 1.7. – агент таксі без урахування цілі не зможе прийняти рішення куди повертати на перехресті, якщо у методах не прописано раптовий вибір у таких ситуаціях, а агент з ціллю проаналізує який вибір більш вірний та приблизить до поставленої мети або цілі.

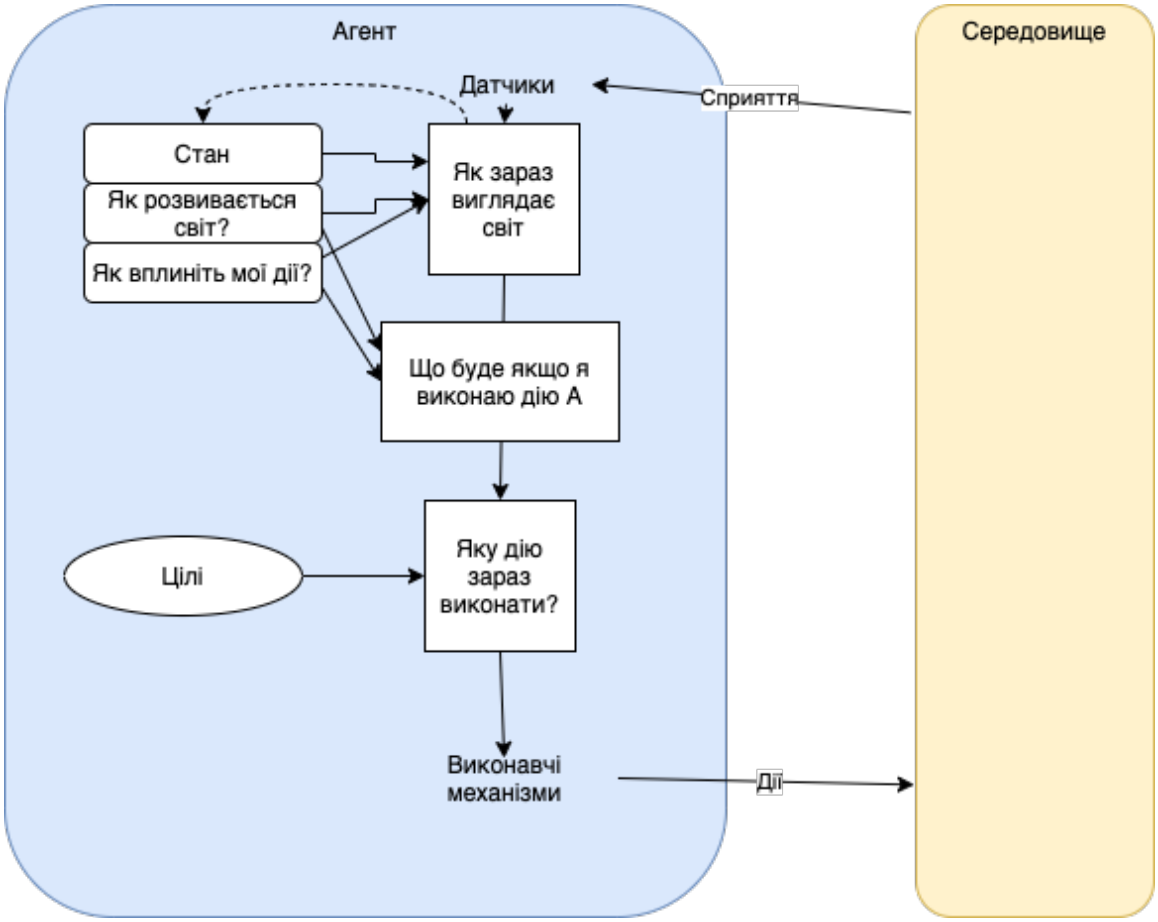

Рисунок 1.8 – Агент, заснований на моделі та цілі

Ще один вадливий тип – це агенти засновані на показнику корисності. Приклад такого агента на базі автомобіля – агент, який буде контролювати швидкість, та ефективно використовувати ресурси, у автомобіля це пальне. А якщо уявити таку ж саму ціль на космічному кораблі – то агент такого типу буде знаходити такі шляхи, які не пошкодять обшивку кораблю, та зекономлять паливо.

Приклад агента який у собі включає усі перелікові типи – це система контролювання у автомобілі Тесла. Рефлексний агент – це датчик дощу, автопілот – комбінація корисного та цільового агенту з моделлю, яка закладена у знання автопілоту.

Постає питання – як агенти отримують модель, за якою діяти. Для цієї цілі можна завдавати усі можливі умови на етапі проектування, але для складних систем використовується навчальні середовища, та модулі.

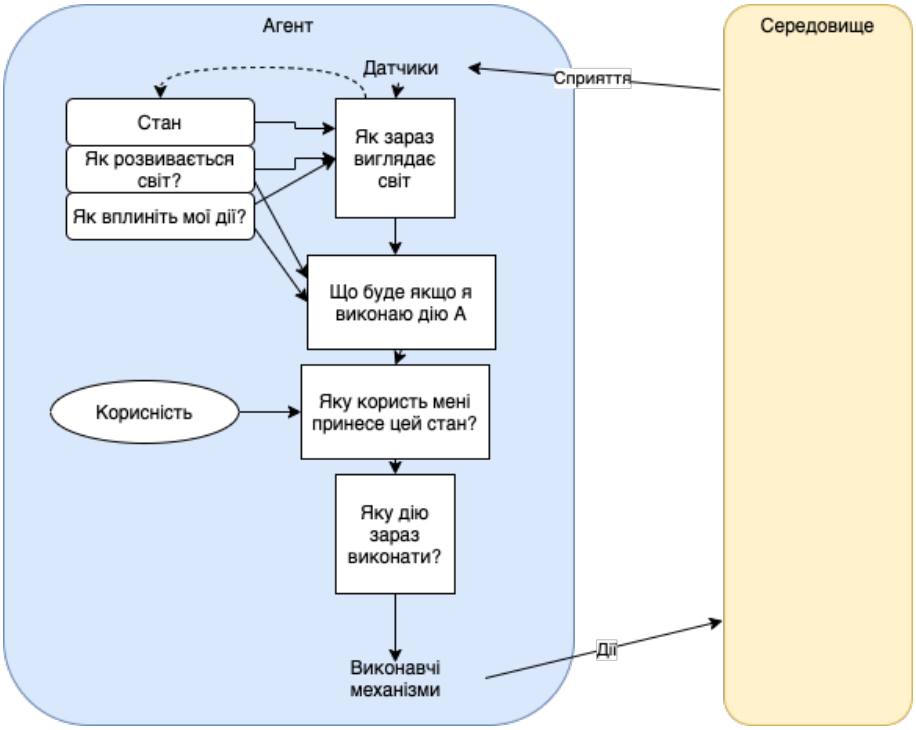

Рисунок 1.9 – Агент, заснований на моделі та корисності

Головні елементи, що мають агенти які можуть навчатися – навчаючий компонент , який відповідає за навчання, за те, що б агент отримав знання. Також є виконавчий компонент – це те, що приймає рішення що робити. Важливим елементом є генератор проблем та критики. Критик потрібен для того, щоб у процесі навчання навчаючий компонент міг розуміти як покращити виконавчий модуль. Частіше всього це зовнішні контролери, на які агент не може впливати. Адже найпростіше змінити межі потреб, а не знайти рішення для потреби.

1.2 Нечіткі агенти

Нечіткі агенти – це розумні агенти, які використовують нечіткі набори для представлення концептуальних категорій та нечіткої логіки як механізм міркувань. Нечіткі наборові / нечіткі логічні підходи є не просто альтернативою ймовірнісним підходам, вони зазвичай використовуються, коли концептуальні

категорії мають ступінь члена для своїх об'єктів, а не просте двійкове членство (а саме містить / не містить).

Факторне представлення, в якому кожен стан розділений на фіксований набір змінних та атрибутів, має ту перевагу, що агент має здатність представляти невизначеність, крім того, області штучного інтелекту, що використовують такі репрезентативні методи, включають алгоритми задоволення або нагород, пропозиційна логіка, байєсівські мережі та багато алгоритмів машинного навчання.

Справа не в тому, що факторизовані уявлення не в змозі представити аспекти навчання та міркування, але навпаки, міркування, які можна представляти, є обмеженими і можуть вимагати значно більших зусиль для впровадження та виконання порівняно з іншими представницькими структурами - в деяких випадках може знадобитися використовувати більш досконалі методи.

Отже, потреба в більш досконалих агентах повинна представляти світ не просто як речі, а й взаємозв'язок між ними, це змушує розробників агентських програм розглядати структуровані уявлення, в яких можна чітко виразити різні відносини. Такі структуровані уявлення лежать в основі таких областей, як логіка першого порядку, моделі ймовірності першого порядку, навчання на основі знань та розуміння природної мови.

1.3 Мультиагенти

Мультиагентний сценарій схожий на традиційну розподілену обчислювальну систему, і як такий, треба враховувати спосіб, за допомогою якого агенти спілкуватимуться між собою, крім того, як вони взаємодіють із своїм оточенням. Зазвичай для цього потрібно встановити відповідні протоколи. Тим не менш, звичайні протоколи, що використовуються у розподілених обчисленнях, такі як виклики віддалених процедур або виклик методу, які

можуть дозволити агентам безпосередньо змінювати внутрішній стан іншого агента або поведінку, просто непридатні в цьому сценарії.

Необхідні протоколи повинні бути набагато вищими, ніж звичайні транспортні протоколи, що використовують такі розподілені системи; як правило, це будуть системи, що базуються на повідомленнях, що дозволяють окремим агентам вирішувати, як відповідати на певне повідомлення або реагувати на нього. Більше того, ці протоколи, як правило, матимуть багатшу семантику, яка дозволить агентам взаємодіяти на навмисному рівні (або соціальному рівні). Це також забезпечує спільноту агентів механізмом, який допомагає їм співпрацювати та впливати один на одного, а отже, допомагає досягти спільної мети як колективного цілого.

Протягом багатьох років існував ряд підходів до моделювання та створення розподіленого обчислювального середовища, багатоагентні системи забезпечують додаткову парадигму моделювання, яка є досить чіткою, головним чином завдяки характеристикам її компонентів, а саме колекції інтелектуальних агентів. Тим не менше, ця парадигма не заважає конструктору системи використовувати інші розподілені поняття, наприклад, немає жодної причини, чому агент не може отримати доступ до послуги, пропонованої більш традиційною архітектурою, орієнтованою на обслуговування.

Важливо не плутати агентів з сервісами так як агенти автономні та мобільні, в тому сенсі, що вони можуть переходити між різними обчислювальними середовищами, що дозволяє їм приєднуватися до різних агентств для досягнення різних цілей.

Сервіси, як правило, призначені для повторного використання функцій, тоді як агенти повинні бути призначені для повторного використання цілих підсистем.

Агенти втілюють знання, міркування та часто навіть переконання за допомогою внутрішніх моделей і, отже, вважають, що сервіси, як правило, позбавлені статусів.

1.4 Методології розробки агентів

Розробка системи інтелектуальних мультиагентів являється більш складною справою ніж розробка одноагентної системи, тим більше, що кількість і складність мультиагентів зростає. Більша частина складності виникає спочатку з тих самих проблем, з якими ми стикаємося при розробці розподілених обчислювальних середовищ, а саме з питань зв'язку між різними компонентами .

У випадку з розумними мультиагентами проблема дещо інша, насамперед тому, що агенти здатні міркувати, вчитися та реагувати на різні події в своєму оточенні, що може призвести до непередбачуваної поведінки як колективу.

Проте це не означає, що ми не повинні встановлювати будь-які настанови чи методології моделювання. Методологія програмного моделювання, як правило, призначена для забезпечення системного підходу до аналізу, проектування та розробки для даної області; крім того, хороша методологія моделювання забезпечить необхідні (і відповідні) настанови, принципи та моделі.

Протягом багатьох років були люди, які намагалися надати такі рамки, хоча не існує єдиної думки, щодо якої слід користуватися. Ось вибір методологій, для створення агентів, які розширюють існуючі підходи:

- методології, які зазвичай надихаються об’єктно-орієнтованим програмним співтовариством:
    - GAIA [4]
    - BDI [5]
    - MaSE [6]
    - ADELFE [7]
    - ASEME [8]
    - Tropos [9]
    - Prometeus [10]
- підходи, які натхненні спільнотою інженерії знань:

- CoMoMAS [11]
- MAS-CommonKADS [12]
- Cassiopeia [13]

1.4.1 Gaia

GAIA - методологія, спочатку запропонована Вулдріджем, Дженнінгсом та Кінні [4]; остання оновлена версія була запропонована Вулдріджем, Дженнінгсом та Замбонеллі [14], яка поширює типи програм, до яких може застосовуватися GAIA.

Gaia - це ітеративна, орієнтована на агента методологія аналізу та проектування, яка на етапі аналізу є прототиповою моделлю ролі та взаємодії. Ця модель надає детальну інформацію про ключові ролі, окрім протоколів взаємодії, які будуть використовуватися агентами, коли їм потрібно буде спілкуватися та співпрацювати один з одним.

Етап проектування методології вимагає, щоб проектувальники згрупували ролі, раніше визначені за типами агентів, а потім визначили взаємозв'язок між цими типами. Після встановлення моделі всеосяжної ролі метод Gaia вимагає розробки моделі сервісу, яка визначає послуги, пов’язані з кожним типом агента.

Метою моделі сервісу є підтримка ролі та діяльності агента в цій системі. Завершальним етапом аналізу та проектування є розробка моделі ознайомлення з моделі взаємодії та агента. Повинно бути зрозуміло, що це не класична архітектура, орієнтована на сервіс, оскільки модель служби існує для підтримки ролей, які агенти повинні виконувати в певному середовищі. Більше того, вони існують для того, щоб запобігти одному агенту безпосередньо змінювати внутрішні стани іншого агента, а також не дозволяти одному агенту безпосередньо викликати операції іншого агента.

Отже, модель служби Gaia існує для визначення взаємозв'язку між агентом та службою, де ці служби призначені для реалізації ролі агента в середовищі.

Сервіс (у цій моделі) є частиною агента, і агент має повний контроль над ними, тому її можна розглядати як одну з видів діяльності, яку агент може виконувати.

Метод Gaia сприймає зовсім інший погляд на сервіси в порівнянні зі звичайними архітектурами веб-служб W3C. Тим не менше, нічого не заважає дизайнерам прийняти стандарт SOA для послуг, в деяких аспектах це цілком може допомогти вирішити нефункціональні інженерні вимоги до системи в цілому.

1.4.2 BDI

Belief-Desire-Intention (BDI) - це підхід до моделювання внутрішніх архітектурних станів агента з використанням таких елементів:

- Belief (переконання): це в сукупності представляє інформаційний стан агента, як правило, включає вірування у світ, вірування в себе, а також вірування в інших агентів. До цього набору також входить збірка правил умовиводу, які дозволяють агенту виконувати міркування на основі нових переконань.

- Desire (бажання): представляє мотиваційний стан агента. це можливі цілі, яких бажає досягти агент.

- Intention (наміри): представляють стан обговорення агента, а саме те, що вирішив зробити агент, вони, як правило, відображають прихильність агента до певної мети (або цілей). Зобов'язання агента в реальному вираженні означає, що він уже прийняв план / стратегію і почав його виконувати, що може включати проміжні цілі (і плани), необхідні для того, щоб агент міг здійснити обране бажання. Слово „план”, яке використовується в цьому контексті, стосується послідовностей дій, які агент може виконати завдання для досягнення заданої мети.

- Події: вони представляють тригери, необхідні для реалізації швидкої діяльності агента, яка може включати різноманітні дії, включаючи, але не обмежуючись цим, оновлення своїх переконань, правил висновку, ініціювання конкретного плану (або стратегії), зміни проміжних цілей тощо.

Хоча BDI є популярною моделлю, але вона має своїх критиків, проте метод BDI забезпечує системний підхід, що охоплює життєвий цикл, від моделювання до реалізації агентів BDI.

1.4.3 ASEME

Agent Systems Engineering Methodology (ASEME) - орієнтована на агента методологія інженерного програмного забезпечення (AOSE) для розробки MAS. ASEME використовує мову моделювання агента AMOLA [15], яка забезпечує синтаксис та семантику для створення моделей мультиагентних систем, які охоплюють фази аналізу та проектування процесу розробки програмного забезпечення.

Ця методологія дещо відрізняється від інших згаданих, оскільки вона використовує модельований інженерний підхід до розробки мультиагентних систем, який сумісний з парадигмою Model Driven Architecture (MDA).

1.4.4 MaSE та ADELFE

Подібно до того, як Gaia надихнула методологія Fusion, Adelfe походить від UML та RUP [16]

Чудовий огляд цих двох методологій можна переглянути по посиланню [17]. У якому автори представляють матеріал, який є частиною більш широкого проекту, який спрямований на аналіз важливих аспектів моделювання різних мультиагентних систем із використанням різних методологій, першим з яких було дослідження в галузі медицини за методологією Adelfe. Мультиагентні системи на базі цих методологій знайшли не менш широке використання, під них є готові бібліотеки на java та більш-менш наповнена документація, що дуже важливо у агентному моделюванні.

1.5 Foundation for Intelligent Physical Agents (FIPA)

FIPA - це, міжнародна організація, яка займається відкритою розробкою специфікацій, що підтримують взаємодію агентів та програм, заснованих на агентах. Відповідність специфікаціям FIPA забезпечує деякі конкретні переваги, наприклад, відповідність FIPA гарантує, що архітектура системи відповідає добре перевіреним протоколам, що забезпечують співпрацю агентів та сумісність. Тим не менше, окремий дослідник або організація вирішує, чи потрібно їх рішення відповідати FIPA.

1.6 Інструмент

Рішення для штучного інтелекту є досить складними, і рішення, які добре працюють в одному контексті, можуть погано працювати в іншому, що вимагає не просто повторної реалізації, а часто нового набору алгоритмів, схем подання тощо, що мають бути розроблені спеціально для цієї конкретної проблеми.

Спільнота ШІ збирає багато з цих алгоритмів, методів і підходів до представлення, навчання та міркування в різні системи або бібліотеки. Тим не менше, подібні бібліотеки часто корисні лише для тих, хто вже має досвід побудови таких систем, і покликані скоротити час, необхідний для розробки рішення, а не відкрити шлях кожному, хто бажає створити такий тип програмного забезпечення .

Порівняти інструменти - головне завдання, спочатку слід встановити критерій, який буде використаний для дослідження, а потім прийняти рішення про підмножину інструментів, доступних на той час. Потім потрібно дотримуватися процедур для кожного товару, щоб можна було звітувати про справедливе порівняння. Враховуючи, що таке велике дослідження [18] було проведено нещодавно, ми будемо згадувати лише деякі відповідні моменти цього дослідження.

Відповідність FIPA (див. табл. 1.1).

Огляд управління безпекою (див. табл. 1.2).

Поєднуючи перші дві таблиці, ми отримуємо продукти, які відповідають FIPA і безпечні (див. табл. 1.3).

Незважаючи на вибір двадцяти чотирьох платформ для розробки агентів, які були перевірено, JADE (середовище розробки агента Jave) залишається найпопулярнішою платформою, сумісною з FIPA та безпечною, підтримуючи при цьому різні операційні системи в Інтернеті. Більше того, це відкрите джерело і має жваву спільноту.

Таблиця 1.1 – Відповідність FIPA

Ступінь відповідності	*Інструмент*
Повна відповідність	JADE, Jadex, JACK, EMERALD
Часткова відповідність	JAS, Jason, AGLOBE, agent factory, SeSAm, GAMA
Не відповідає	Cougaar, swarm, MASON, INGENIAS development kit, cormas, repast, MaDKit, cybelePro, JIAC, agentscape, anylogic, net-logo, JAMES II

Таблиця 1.2 – Огляд управління безпекою

Назва	*End-to-end захист*	*Захист платформи*	*Створено*
Agentscape	Авторизація, приватний вхід	Добре	2010
AnyLogic	Авторизація	Дуже добре (закрита система)	2013
EMERALD	Підписи, шифрування, HTTPS	Дуже добре	2010

JACK	Авторизація, захист домену у JDK	Дуже добре	2005
JADE	HTTPS	Дуже добре (API)	2003
JADEX	Key sharing	Дуже добре	2013
MaDKit	Авторизація	Добре	1000

Таблиця 1.3 – Відповідають FIPA і безпечні

Назва	*End-to-end захист*	*Захист платформи*	*Створено*
EMERALD	Підписи, шифрування, HTTPS	Дуже добре	2010
JACK	Авторизація, захист домену у JDK	Дуже добре	2005
JADE	HTTPS	Дуже добре (API)	2003
JADEX	Key sharing	Дуже добре	2013

JADE написаний на мові програмування JAVA і, отже, забезпечує комплексний API (Інтерфейс програмування програм) у JAVA, що полегшує реалізацію різних функціональних аспектів програми-агента. Більше того, він включає реалізацію мови комунікації агента (ACL), як визначено FIPA, для багатоагентної координації, необхідної при впровадженні MAS.

Як можна побачити із обзору літератури моделювання на базі агентів має базові архітектурні вимоги, але вимог що до програмної реалізації немає. Потребуємий підхід та методи у процесі створення архітектури та розробці залежить від того, що буде моделюватися. Мінімальні потреби для агентного моделювання – це наявність середовища та агент, який зможе взаємодіяти с середовищем.

2 РЕАЛІЗАЦІЯ БІБЛІОТЕКИ

2.1 Створення середовища

Середовище може бути будь-яке, тому базовий абстрактний клас середовища має тільки конструктор у якому задано текст для того, щоб переконатися, що все успішно ініціалізувалося, при використанні це буде перевизначено.

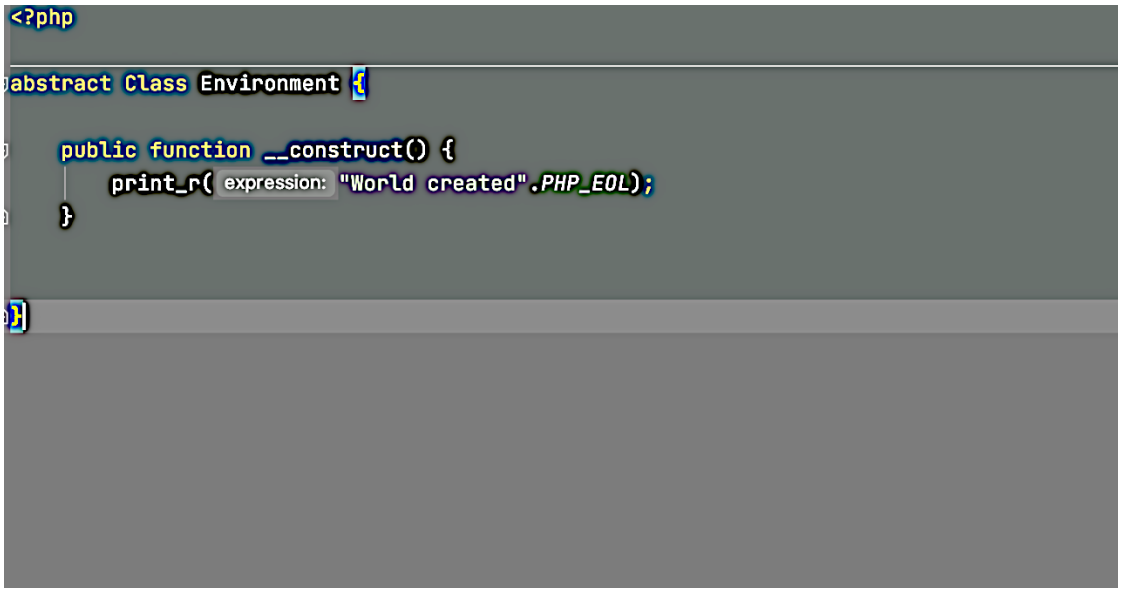

Рисунок 2.1 – Абстрактний клас середовища

2.2 Створення класу агента

Абстрактний клас агента має посилання на середовище яке потрібно передати у конструктор - __construct(&$env), набір функцій у виді масиву з анонімними функціями - functions , історію взаємозв’язку із навколишнім середовищем - percepts = [] , лог виконаних дій - actions та стан - state. Також реалізовані методи конструктора, метод взаємодії з середовищем та метод

виконання задач. Використовуючи цей абстрактний клас та архітектуру можна створювати як прості, так і базовані на цілі та моделі агенти.

```php
<?php

abstract class Agent {

    private $env;
    private $functions = [];
    private $percepts = [];
    private $actions = [];
    public $state = [];


    public function __construct( &$env ) {
        $this->env = $env;

        $this->functions = [
            'me' => function () {
                $this->actions[] = 'me';

                return 'I\'m agent and I exist';
            }
        ];
    }

    public function run(){
        $this->state[]   = $this->doAction();
    }

    private function checkWorld() {
        return 0;
    }

    private function doAction() {
        $this->percepts[] = $this->checkWorld();

        // intelligence should be here. but as soon as it's abstract - will be self introduce

        return call_user_func( $this->functions['me'] );
    }
}
```

Рисунок 2.2 – Абстрактний клас агенту

2.3 Створення точки ініціалізації бібліотеки

Нижче на рисунку показана точка ініціалізації бібліотеки.

Рисунок 2.3 – Точка ініціалізації

2.4 Висновок

Було створено бібліотеку, яка дає змогу на PHP створити середовище та у ньому використовувати агентів. Агент має усі мінімально обов’язкові елементи, а значить на базі цієї бібліотеки можливо створити модель.

3 СТВОРЕННЯ МОДЕЛІ

3.1 Логіка моделі

Логіка моделі буде середовище, у якому випадково з’являється їжа, яка і буде ціллю агента. Якщо агент до неї дійде – буде +1 к такому типу агента у середовищі.

Середовище розподіляє по краям карти та запускає раунд, який продовжується до тих пір, поки в агентів не закінчяться сили, або якщо усі агенти знайдуть їжу.

Агенти будуть мати параметри, та суттю моделювання буде знайти найбільш ефективний набір параметрів агентів. Мірою цього визначення буде кількість агентів після деякої кількості раундів.

3.2 Створення середовища

Середовище – це буде координатна сітка с розміром, який можна задати. Також в нього є поле з поточним станом кожної точки, а також масив з посиланнями на агентів.

```php
<?php

class Surv_Environment extends Environment {
    public $size;
    public $cells = [];
    private $agents = [];
    public $start_gun = 'wait';
```

Рисунок 3.1 – Поля середовища

У середовищі є метод для розташування агентів (див. рис. 3.2), який розташовує агентів по краям карти.

```
public function placeAgent( &$agent ) {
    $border_random_cells = array(
        $this->rand_cell(
        [
            'from' => 0,
            'to'   => $this->size - 1
        ],
        [
            'from' => 0,
            'to'   => 0
        ] ),
        $this->rand_cell(
            [
                'from' => 0,
                'to'   => 0
            ],
            [
                'from' => 0,
                'to'   => $this->size - 1
            ] ),
        $this->rand_cell(
            [
                'from' => $this->size - 1,
                'to'   => $this->size - 1
            ],
            [
                'from' => 0,
                'to'   => $this->size - 1
            ] ),
        $this->rand_cell(
            [
                'from' => 0,
                'to'   => $this->size - 1
            ],
            [
                'from' => $this->size - 1,
                'to'   => $this->size - 1
            ] ),
        );
    $cell = $border_random_cells[array_rand($border_random_cells)];
    $this->cells[ $cell['x'] ][ $cell['y'] ]['agents'][] = $agent;
    $agent->location = array($cell['x'], $cell['y']);
    $this->agents[] = $agent;
}
```

Рисунок 3.2 – Метод розташування агентів

Метод запуску раунду (див.рис. 3.3), отримання кількості агентів, методи для генерування випадкових координат (див. рис. 3.3), а також розташування їжі на мапі (див.рис. 3.4). Таким чином середовище розташовує агентів, має в собі послідовну систему а також надає цілі агентам. Також є конструктор, який при ініціалізації середовища створює мапу з базовою інформацією про кожну точку на ній. При старті заповнює тим, що точка порожня.

```php
/**
 * Surv_Environment constructor.
 *
 * @param int $size cube size world
 * @param int $food_amount food amount per round
 */
public function __construct( int $size ) {
    $this->size = $size;
    for ( $x = $size - 1; $x >= 0; $x -- ) {
        for ( $y = $size - 1; $y >= 0; $y -- ) {
            $this->cells[ $x ][ $y ] = [ 'food' => false ];
        }
    }
}

public function start_round(){
    do{
        foreach ($this->agents as $agent){
            $agent->run();
        }
        foreach ($this->agents as $key => $agent){
            if(in_array(array_pop( &array: $agent->state)['action'], array('died','found_food'))){
                var_dump(array_pop( &array: $agent->state)['action']);
                unset($this->agents[$key]);
            }
        }
    } while($this->getAgents() != 0);

}

private function rand_cell( $x, $y ) {
    $x = rand( $x['from'], $x['to'] ); //rand from 1 and up to size -2 because area near map borders are for resp of our agents
    $y = rand( $y['from'], $y['to'] ); //rand from 1 and up to size -2 because area near map borders are for resp of our agents

    return [ 'x' => $x, 'y' => $y ];
}
```

Рисунок 3.3 – Метод запуску раунду start_round та метод для генерування випадкових координат rand_cell

```
private function fill_with_food( $cell ) {
    $fill_cell = &$this->cells[ $cell['x'] ][ $cell['y'] ]['food'];
    if ( $fill_cell != true ) {
        $fill_cell = true;
    } else {
        $this->fill_with_food( $this->rand_cell(
            [
                'from' => 1,
                'to'   => $this->size - 2
            ],
            [
                'from' => 1,
                'to'   => $this->size - 2
            ]
        ) );
    }
}

public function seedFood( int $food_amount ) {
    while ( $food_amount != 0 ) {
        $this->fill_with_food( $this->rand_cell(
            [
                'from' => 1,
                'to'   => $this->size - 2
            ],
            [
                'from' => 1,
                'to'   => $this->size - 2
            ]
        ) );
        $food_amount --;
    }
}
```

Рисунок 3.4 – Метод розташування їжі на мапі - seedFood

3.3 Створення класу агента

У агента окрім полів – (див. рис. 3.4) з абстрактного класу, з’явилися поля його характеристик – витривалості, ваги та швидкості. Також є поле розташування та цілі.

Реалізован набір функцій, таких як рух, їсти та вмерти – рис 3.5. Move, eat та die. Метод move базований на випадковому напрямку, але обмежений мапою, а також не може бути шагом назад. Метод eat – це метод, коли агент знаходить ціль, а значить його популяція збільшується. Die – метод кінцевий, коли у агента закінчується сили. Такий агент надалі не буде існувати. Кожна функція зберігається у пам’яті агента .

Посилання на середовище дає змогу змінювати поля середовища, то б то цілком відповідає агентному підходу – це є вплив на середовище.

```php
Class Surv_Agent extends Agent {

    private $env;
    private $functions = [];
    private $percepts = [];
    private $actions = [];
    public $state = [];
    public $location;
    private $stamina;
    private $speed;
    private $sence_distance;
    private $weight;
    private $goal = false;
```

Рисунок 3.4 – Поля агента

```php
public function __construct( &$env ) {
    $this->env = $env;
    $this->stamina = 100;

    $this->functions = [
        'me' => function () {
            $this->actions[] = 'me';

            return 'I\'m agent and I exist';
        },
        'move' => function () {
            $rand = $this->randLocation();
            $smt = $this->env->size-2;
            if(
            $rand['x'] != $this->location['x'] &&
            $rand['y'] != $this->location['y'] &&
            $rand['x'] <= $smt &&
            $rand['x'] >= 1 &&
            $rand['y'] <= $smt &&
            $rand['y'] >= 1
            ){
                $this->location = $rand;
                $this->stamina = $this->stamina - 1;
                $this->actions[] = 'moved to' .$this->location['x'].' ' . $this->location['y']. ' -1 stamina';
                return 'moved, -1 stamina';
            } else{
                call_user_func( $this->functions['move'] );
            }
        },
        'eat' => function () {
            $this->actions[] = 'eat';
            $this->env->cells[$this->location['x']][$this->location['y']]['food'] = false;
            $this->goal = true;
            var_dump( expression: 'here');
            return 'found_food';
        },
        'die' => function () {
            $this->actions[] = 'die';
            return 'died';
        }
    ];

}
```

Рисунок 3.5 – Функції агента move, eat, die

```php
private function randLocation(){
    $rand = array(-1,1,0);
    $curx = $this->location['x'];
    $cury = $this->location['y'];
    $location = ['x' => $curx + $rand[array_rand($rand)], 'y' => $cury + $rand[array_rand($rand)]];
    return $location;
}

private function checkWorld() {
    return [
        'status' => $this->env->start_gun,
        'current_cell_food' => $this->env->cells[$this->location['x']][$this->location['y']]['food'],
        'radius',
    ];
}

public function run(){
    $this->state[]   = array(
        'action' => $this->doAction(),
        'location' => $this->location
    );
}

private function doAction() {
    $this->percepts[] = $this->checkWorld();
    if($this->goal != true){
        if(array_pop( &array: $this->percepts)['status'] == 'wait'){
            return 'wait';
        } elseif (array_pop( &array: $this->percepts)['status'] == 'start'){
            if($this->stamina > 0){
                if(array_pop( &array: $this->percepts)['current_cell_food'] == true){
                    return call_user_func( $this->functions['eat'] );
                } else{
                    return call_user_func( $this->functions['move'] );
                }
            }
            else{
                return call_user_func( $this->functions['die'] );
            }

        }
    } else{
        return 'found_food';

    }
```

Рисунок 3.6 – Методи агента randLocation, checkWorld, doAction, run

Тобто агент відповідає моделі на рисунку 1.8 – він заснований на моделі та цілі.

Кожну ітерацію перевіряє стан середовища, за що відповідає метод checkWorld(), робить перевірку «умова-дія» - doAction та діє. Головною ціллю агента є знайти їжу у середовищі.

3.4 Тести

Для результатів тестів було узято 2 моделювання з 2 парами.

Перша пара – це середні показники швидкості, сили та полю зору (S5ST5VR5), скорочено S5 проти генерації на одиницю швидше (S6ST5VR5), скорочено S6. Після 50 раундів генерації S5 залишилося 10 юнітів, а кількість юнітів другої генерації S6 зросло – стало 40, що є обмеженням середовища. (див. табл. 3.1)

Таблиця 3.1 – S5 проти S6

Час	*S5*	*S6*
Початок	25	25
25 раундів	15	30
50 раундів	10	40

У результаті моделювання бачимо очікуваний результат – більш швидкий юніт знаходив їжу швидше, аніж повільніший юніт, що дало змогу сімейству таких юнітів отримати превагу у кількості.

Друга пара - це переможець з першої пари, S6 та новий вид – (S5ST3VR7), скорочено ST3VR7. Його характерні риси – кращій зір, але менша витривалість. Після 50 раундів вияснилося, що кількість залишалась відносно однакова. (див. табл. 3.2)

Таблиця 3.2 – S6 проти ST3VR7

Час	*S6*	*ST3VR7*
Початок	25	25
10 раундів	20	29
20 раундів	24	25
25 раундів	23	27
50 раундів	22	25

ВИСНОВКИ

У результаті роботи над кваліфікаційною роботою магістра був зроблений аналітичний огляд публікацій наведений у переліку посилань, а також публікацій, на які звернень не має, але вони допомогли більш детально проаналізувати тему. Було дане визначення агенту, середовища, проаналізовані та наведені типи агентів. Проаналізовані методології розробки агентів та їх використання. Зібрана інформація про поточні актуальні інструменти та рішення.

Реалізована бібліотека, що дозволяє створити модель на базі інтелектуальних агентів на PHP. Бібліотека може використовуватися для симуляцій біологічних досліджень, симуляціях еволюційних поколінь, симуляцій моделі хижак та жертва. Також на базі неї можна зробити агентів з вмінням навчатися, що надасть змогу моделювати більш складні моделі.

Так як бібліотека зроблена на PHP – то її можна використовувати у різноманітних WEB-додатків. Перспективи подальшого розвитку – це дати можливість агентам буди багатопоточними, що дасть більше можливостей для моделювання.

ПЕРЕЛІК ПОСИЛАНЬ

1. David L. Poole. Artificial Intelligence Foundations of Computational Agents / David L. Poole, Alan K. Mackworth Cambridge: University Press, 2010. 682 p. ISBN-13 978-0-511-72946 (дата звернення: 28.09.2020)

2. Стюарт Рассел, Пітер Норвіг. Штучний інтелект: сучасний підхід = Artificial Intelligence: A Modern Approach, 2012. 1407 p. (дата звернення: 28.09.2020)

3. K. R. Chowdhary. Fundamentals of Artificial Intelligence, Jodhpur Institute of Engineering and Technology, Jodhpur, Rajasthan, India, 2020. 730 p. ISBN 978-81-322-3970-3 (дата звернення: 10.10.2020)

4. Wooldridge, M. Jennings, N.R., Kinny, D. (2000) “The Gaia Methodology for Agent-Oriented Analysis and Design” Journal of Autonomous Agents and Multi-Agent Systems (дата звернення: 11.10.2020)

5. Rao, M. and Georgeff, BDI-agents: From Theory to Practice. In Proceedings of the First International Conference on Multiagent Systems ICMAS. 1995. (дата звернення: 11.10.2020)

6. DeLoach, S. A Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems, Agent-Oriented Information Systems, Seattle WA. 1999. (дата звернення: 11.10.2020)

7. Henderson-Sellers, Brian & Giorgini, Paolo, Agent-oriented Methodologies. 1ed: Idea Group Inc, London, UK, 2005. 412 p. ISBN 1-59140-581-5. (дата звернення: 11.10.2020)

8. Spanoudakis N., Moraitis P., Model-Driven Agents Development with ASEME. In: 11th International Workshop on Agent Oriented Software Engineering (AOSE 2010), Toronto, Canada. 2010. (дата звернення: 11.10.2020)

9. Castro, J., Kolp, M. and Mylopoulos, J., A Requirements-Driven Development Methodology, In Proc of the 13th International Conference on Advanced Information Systems Engineering CAiSE 01, Interlaken, Switzerland. 2001. (дата звернення: 11.10.2020)

10. Padgham, L. and Winikoff, M., “Prometheus: A Methodology for Developing Intelligent Agents”. Proceedings of the Third International Workshop on Agent-Oriented Software Engineering : AAMAS’0. 2010. (дата звернення: 11.10.2020)

11. Glaser, N.,Contribution to Knowledge Modelling in a Multi-Agent Framework, Ph.D. Thesis, L’Université henri Poincaré, Nancy I, France. 1996. (дата звернення: 11.10.2020)

12. Analysis and design of multiagent systems using MAS-CommonKADS, In AAAI’97 Workshop on Agent Theories, Architectures and Languages, Providence, RI. ATAL. (An extended version of this paper has been published in INTELLIGENT AGENTS IV: Agent Theories Architectures, and Languages, Springer Verlag. 1998. (дата звернення: 11.10.2020)

13. Agent Oriented Design of a Soccer Robot Team, In Proc. of the Second Intl. Conf. on Multi-Agent Systems: Kyoto, Japan. 1996. (дата звернення: 11.10.2020)

14. Zambonelli, F. and Jennings, N.R. and Wooldridge, M.,Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3. 2003 (дата звернення: 11.10.2020)

15. Spanoudakis, N. and Moraitis, P., The Agent Modeling Language (AMOLA). In: Proceedings of the 13th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2008), Springer,

Lecture notes in Computer Science (LNCS), Volume 5253/2008: Varna, Bulgaria. 2008. (дата звернення: 11.10.2020)

16. Rumbaugh, J., Jacobson, I. and Booch, G., The Unified Modelling Language Reference Manual, Second edition, Addison-Wesley. 2004 (дата звернення: 11.10.2020)

17. Vera Maria B. Werneck, Rosa Maria E. Moreira Costa and Luiz Marcio Cysneiros, Modelling Multi-Agent Systems using Different Methodologies. 2011. (дата звернення: 11.10.2020)

18. Kravari, Kalliopi and Bassiliades, Nick ‘A Survey of Agent Platforms’ Journal of Artificial Societies and Social Simulation 18 (1) 11 http://jasss.soc.surrey.ac.uk/18/1/11.html., 2015 (дата звернення: 11.10.2020)

ДОДАТОК А

Програмний код – бібліотека агентів

agent-class.php

```
abstract class Agent {

	private $env;
	private $functions = [];
	private $percepts = [];
	private $actions = [];
	public $state = [];


	public function __construct( &$env ) {
		$this->env = $env;

		$this->functions = [
			'me' => function () {
				$this->actions[] = 'me';

				return 'I\'m agent and I exist';
			}
		];
	}

	public function run(){
		$this->state[]   = $this->doAction();
	}

	private function checkWorld() {
		return 0;
	}

	private function doAction() {
		$this->percepts[] = $this->checkWorld();

		// intelligence should be here. but as soon as it's abstract - will be self introduce

		return call_user_func( $this->functions['me'] );
	}
}
```

env-class.php

```
<?php
abstract Class Environment {
	public function __construct() {
		print_r("World created".PHP_EOL);
	}
}
```

ДОДАТОК Б

Програмний код – тестова модель

agent.php

```
<?php

Class Surv_Agent extends Agent {

	private $env;
	private $functions = [];
	private $percepts = [];
	private $actions = [];
	public $state = [];
	public $location;
	private $stamina;
	private $speed;
	private $sence_distance;
	private $weight;
	private $goal = false;

	public function __construct( &$env ) {
		$this->env = $env;
		$this->stamina = 100;

		$this->functions = [
			'me' => function () {
				$this->actions[] = 'me';

				return 'I\'m agent and I exist';
			},
			'move' => function () {
				$rand = $this->randLocation();
				$smt = $this->env->size-2;
				if(
				$rand['x'] != $this->location['x'] &&
				$rand['y'] != $this->location['y'] &&
				$rand['x'] <= $smt &&
				$rand['x'] >= 1 &&
				$rand['y'] <= $smt &&
				$rand['y'] >= 1
				){
					$this->location = $rand;
					$this->stamina = $this->stamina - 1;
					$this->actions[] = 'moved to' .$this->location['x'].' ' . $this->location['y']. ' -
1 stamina';
					return 'moved, -1 stamina';
				} else{
					call_user_func( $this->functions['move'] );
				}
			},
			'eat' => function () {
				$this->actions[] = 'eat';
				$this->env->cells[$this->location['x']][$this->location['y']]['food'] = false;
				$this->goal = true;
				var_dump('here');
				return 'found_food';
			},
```

```
                        'die' => function () {
                                $this->actions[] = 'die';
                                return 'died';
                        }
                ];

        }

        private function randLocation(){
                $rand = array(-1,1,0);
                $curx = $this->location['x'];
                $cury = $this->location['y'];
                $location = ['x' => $curx + $rand[array_rand($rand)], 'y' => $cury + $rand[array_rand($rand)]];
                return $location;
        }

        private function checkWorld() {
                return [
                        'status' => $this->env->start_gun,
                        'current_cell_food' => $this->env->cells[$this->location['x']][$this->location['y']]['food'],
                        'radius',
                ];
        }

        public function run(){
                $this->state[]   = array(
                        'action' => $this->doAction(),
                        'location' => $this->location
                );
        }

        private function doAction() {
                $this->percepts[] = $this->checkWorld();
                if($this->goal != true){
                        if(array_pop($this->percepts)['status'] == 'wait'){
                                return 'wait';
                        } elseif (array_pop($this->percepts)['status'] == 'start'){
                                if($this->stamina > 0){
                                        if(array_pop($this->percepts)['current_cell_food'] == true){
                                                return call_user_func( $this->functions['eat'] );
                                        } else{
                                                return call_user_func( $this->functions['move'] );
                                        }
                                }
                                else{
                                        return call_user_func( $this->functions['die'] );
                                }

                        }
                } else{
                        return 'found_food';

                }

        }
}
```

ДОДАТОК В

Програмний код – клас середовища тестової моделі

Surv-env.php

```
<?php

class Surv_Environment extends Environment {
	public $size;
	public $cells = [];
	private $agents = [];
	public $start_gun = 'wait';

	/**
	 * Surv_Environment constructor.
	 *
	 * @param int $size cube size world
	 * @param int $food_amount food amount per raund
	 */
	public function __construct( int $size ) {
		$this->size = $size;
		for ( $x = $size - 1; $x >= 0; $x -- ) {
			for ( $y = $size - 1; $y >= 0; $y -- ) {
				$this->cells[ $x ][ $y ] = [ 'food' => false ];
			}
		}
	}

	public function start_round(){
		do{
			foreach ($this->agents as $agent){
				$agent->run();
			}
			foreach ($this->agents as $key => $agent){
				if(in_array(array_pop($agent->state)['action'], array('died','found_food'))){
					var_dump(array_pop($agent->state)['action']);
					unset($this->agents[$key]);
				}
			}
		} while($this->getAgents() != 0);

	}

	private function rand_cell( $x, $y ) {
		$x = rand( $x['from'], $x['to'] ); //rand from 1 and up to size -2 because area near map borders are for
resp of our agents
		$y = rand( $y['from'], $y['to'] ); //rand from 1 and up to size -2 because area near map borders are for
resp of our agents

		return [ 'x' => $x, 'y' => $y ];
	}

	private function fill_with_food( $cell ) {
		$fill_cell = &$this->cells[ $cell['x'] ][ $cell['y'] ]['food'];
		if ( $fill_cell != true ) {
			$fill_cell = true;
		} else {
			$this->fill_with_food( $this->rand_cell(
				[
```

```
                                        'from' => 1,
                                        'to'   => $this->size - 2
                                ],
                                [
                                        'from' => 1,
                                        'to'   => $this->size - 2
                                ]
                        ) );
                }
        }

        public function seedFood( int $food_amount ) {
                while ( $food_amount != 0 ) {
                        $this->fill_with_food( $this->rand_cell(
                                [
                                        'from' => 1,
                                        'to'   => $this->size - 2
                                ],
                                [
                                        'from' => 1,
                                        'to'   => $this->size - 2
                                ]
                        ) );
                        $food_amount --;
                }
        }

        public function placeAgent( &$agent ) {
                $border_random_cells = array(
                        $this->rand_cell(
                        [
                                'from' => 0,
                                'to'   => $this->size - 1
                        ],
                        [
                                'from' => 0,
                                'to'   => 0
                        ] ),
                        $this->rand_cell(
                                [
                                        'from' => 0,
                                        'to'   => 0
                                ],
                                [
                                        'from' => 0,
                                        'to'   => $this->size - 1
                                ] ),
                        $this->rand_cell(
                                [
                                        'from' => $this->size - 1,
                                        'to'   => $this->size - 1
                                ],
                                [
                                        'from' => 0,
                                        'to'   => $this->size - 1
                                ] ),
                        $this->rand_cell(
                                [
                                        'from' => 0,
                                        'to'   => $this->size - 1
                                ],
                                [
```

```
                                        'from' => $this->size - 1,
                                        'to'   => $this->size - 1
                                    ] ),
                                );
                $cell = $border_random_cells[array_rand($border_random_cells)];
                $this->cells[ $cell['x'] ][ $cell['y'] ]['agents'][] = $agent;
                $agent->location = array($cell['x'], $cell['y']);
                $this->agents[] = $agent;
        }


        public function getAgents() {
                return count($this->agents);
        }

        public function getCells() {
                $size = $this->size;
                for ( $x = $size - 1; $x >= 0; $x -- ) {
                        for ( $y = $size - 1; $y >= 0; $y -- ) {
                                if ( $this->cells[ $x ][ $y ]['food'] == true) {
                                }
                        }
                }

                for ( $x = $size - 1; $x >= 0; $x -- ) {
                        for ( $y = $size - 1; $y >= 0; $y -- ) {
                                if (isset($this->cells[ $x ][ $y ]['agents'])) {
//                                      var_dump( $this->cells[ $x ][ $y ] );
//                                      var_dump( 'x '.$x.' y '.$y );
                                }
                        }
                }

        }


}
```

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, _____

студент(ка) 2 курсу, денної форми навчання, математичного факультету, спеціальності програмна інженерія, адреса електронної пошти igorvolgin1@gmail.com , – підтверджую, що написана мною кваліфікаційна робота магістра на тему «Розробка бібліотеки агентного моделювання» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Студент

_____	_____	Волгін І. С.
(дата)	(підпис)	(прізвище, ініціали)

Науковий керівник

_____	_____	Кудін О. В.
(дата)	(підпис)	(прізвище, ініціали)