Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів ройового інтелекту в агентних системах

(тема)

Виконав:
студент 2 курсу, групи СШІм-19-2
Верхогляд І.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник доц. Золотухін О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Верхогляду Ігорю Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів ройового інтелекту і агентних системах

затверджена наказом університету від 29 березня 20 21 р. № 390Ст

2. Термін подання студентом роботи до екзаменаційної комісії _____ _____ 20 ___ р.

3. Вихідні дані до роботи науково-технічні публікації, лані статей, експериментальних досліджень по технологіям, методам, моделям, алгоритмам ройового інтелекту та агентних систем

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної галузі і постановка задачі
2. Теоретичні дослідження ройового інтелекту
3. Використання бібліотеки для моделювання поведінки роботів
4. Розробка системи координації взаємодії роботів на основі алгоритму

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1 – Алгоритм рою частинок, Рисунок 2 – Рух до центру мас, Рисунок 3 – Вектор руху частинки від найближчих сусідів, Рисунок 4 – Рух до цілі, Рисунок 5 – Рух від перешкоди, Рисунок 6 – Блок-схема модифікованого алгоритму Boids, Рисунок 7 – Функціональна схема взаємодії обладнання. Рисунок 8–13 Екранні форми програмної реалізації

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основний розділ	доц. Золотухін О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	29.03.2021	виконано
2	Аналіз предметної галузі та постановка задачі	30.03.2021-04.04.2021	виконано
3	Теоретичні дослідження ройового інтелекту та	05.04.2021-12.04.2021	виконано
	агентних систем		
4	Розробка системи взаємодії та координації роботів на	13.04.2021-23.04.2021	виконано
	основі алгоритму Boids		
5	Практична реалізація	24.04.2021-04.05.2021	виконано
6	Підготовка пояснювальної записки	05.05.2021-12.05.2021	виконано
7	Надання пояснювальної записки на перевірку	13.05.2021	виконано
	керівнику		
8	Захист перед ЕК		

Дата видачі завдання 29 березня 20 21 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 123 с., 16 рис., 2 дод., 27 джерел.

КОЛЕКТИВ, РІЙ, РОБОТ, РОБОТОТЕХНІКА, РОЙ ЧАСТОК, РОЙОВИЙ ІНТЕЛЕКТ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єктом дослідження є системи координації взаємодії роботів.

Предметом дослідження є методи колективного розуму, ройові методи.

Методи дослідження – методи системного та об’єктно-орієнтованого аналізу, методи колективного розуму, ройові методи.

Метою даної роботи є дослідження алгоритмів ройового інтелекту у робототехніці для вирішення завдання координації та комунікації роботів.

Результатом практичної реалізації роботи є програмний додаток, що реалізує алгоритм Boids. Для наочності розроблене середовище, яке симулює взаємодію роботів в різноманітніх ситуаціях. Також в даній роботі проведена оцінка ефективності командної взаємодії в порівнянні з індівідуальними діямі агентів.

Для розробки програмного додатка використана фізична бібліотека Box2D, а також середовище розробки Microsoft Visual Studio.

РЕФЕРАТ

Пояснительная записка: 123 с., 16 рис., 2 прил., 27 источников.

КОЛЛЕКТИВ, РОБОТ, РОБОТОТЕХНИКА, РОЙ, РОЙ ЧАСТИЦ, РОЕВОЙ ИНТЕЛЛЕКТ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ.

Объектом исследования являются системы координации взаимодействия роботов.

Предметом исследования являются методы коллективного разума, роевые методы.

Методы исследования – методы системного и объектно-ориентированного анализа, методы коллективного разума, роевые методы.

Целью данной работы является исследование алгоритмов роевого интеллекта в робототехнике для решения задачи координации и коммуникации роботов.

Результатом практической реализации работы является программное приложение, которое реализует алгоритм Boids. Для наглядности разработанное среду, симулирует взаимодействие роботов в разнообразных ситуациях. Также в данной работе проведена оценка эффективности командного взаимодействия по сравнению с индивидуальным действиям агентов.

Для разработки программного приложения использована физическая библиотека Box2D, а также среда разработки Microsoft Visual Studio.

ABSTRACT

Explanatory note: 123 p., 16 fig., 2 ann., 27 references.

ARTIFICIAL INTELLIGENCE, COLLECTIVE, ROBOT, ROBOTIC TECHNOLOGY, SWARM, SWARM INTELLIGENCE, SWARM OF PARTICLES.

The object of a research is a system for interaction coordination of robots.

The subject of research is the methods of collective intelligence, swarm methods.

Research methods – methods of system and object-oriented analysis, methods of collective intelligence, swarm methods.

The aim of this work is a research of the swarm intelligence algorithms in robotic technology for the solution of tasks of robot coordination and communication.

The result of the work is the software application realizing Boids algorithm. For descriptive reasons the developed environment simulates an interaction of robots in various situations. Also this work contains efficiency assessment of team interaction in comparison with personal actions.

A physical library Box2D and also a development environment Microsoft Visual Studio were used for development of a software application.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 9
Вступ 10
1 Аналіз предметної області та постановка задачі 12
1.1 Мета кваліфікаційної роботи 12
1.2 Ройовий інтелект в природі 13
1.3 Актуальність застосування ройового інтелекту 17
1.4 Групова робототехніка 19
1.5 Постановка задачі 21
2 Теоретичні дослідження колективного інтелекту 23
2.1 Алгоритм рою частинок 23
2.1.1 Алгоритм Boids 24
2.1.2 Алгоритм рою частинок GBEST 25
2.2 Модифікації алгоритму рою частинок 28
2.2.1 Модифікація LBEST 28
2.2.2 Модифікація Inertia Weighted PSO 29
2.2.3 Модифікація Time-Varying Inertia Weighted PSO 30
2.2.4 Модифікація Canonical PSO 30
2.2.5 Модифікація Fully Informed Particle Swarm 31
2.3 Мурашиний алгоритм 32
2.4 Бджолиний алгоритм 33
3 Використання фізичної бібліотеки Box2D для моделювання поведінки роботів 35
3.1 Опис Box2D 35
3.2 Основні поняття Box2D 35
3.3 Створення фізичних об'єктів в Box2D 36
3.3.1 Створення світу в Box2D 36
3.3.2 Створення землі в Box2D 37
3.3.3 Створення динамічних тел в Box2D 38

3.4 Інтерфейс Box2D 39
3.4.1 Управління динамічної пам'яттю в Box2D 39
3.4.2 Фабрики і визначення в Box2D 40
3.4.3 Одиниці виміру в Box2D 41
3.4.4 Користувальницькі дані в Box2D 41
3.4.5 Використання мови програмування C ++ в Box2D 42
3.5 Світ в Box2D 43
3.5.1 Створення і знищення світу в Box2D 43
3.5.2 Використання світу в Box2D 43
3.6 Фізичні тіла в Box2D 45
3.6.1 Визначення фізичних тіл в Box2D 45
3.6.1.1 Маса 46
3.6.1.2 Позиція і кут 46
3.6.1.3 Гальмування 46
3.6.1.4 Параметри сну 47
3.6.1.5 Снаряди 47
3.6.2 Видалення фізичних тіл в Box2D 49
3.6.3 Використання фізичних тіл в Box2D 50
3.7 Фігури 51
3.7.1 Визначення фігури в Box2D 51
3.7.1.1 Коефіцієнти тертя і пружності 52
3.7.1.2 Щільність 52
3.7.1.3 Фільтрація зіткнень 53
3.7.1.4 Сенсори 54
3.7.1.5 Геометричні фігури 54
3.7.2 Фабрики фігур в Box2D 55
3.8 З'єднання 55
3.8.1 Визначення сполук в Box2D 55
3.8.1.1 З'єднання «жорсткий відрізок» 57
3.8.1.2 Болтове з'єднання 57

3.8.1.3 Призматичне з'єднання 58
3.8.1.4 Талеві з'єднання 59
3.8.1.5 Передаточне з'єднання 60
3.8.2 Фабрика з'єднань в Box2D 61
3.8.3 Використання сполук в Box2D 61
3.9 Контакти в Box2D 62
3.9.1 Опис контактів в Box2D 62
3.9.2 Оброблювач контактів в Box2D 63
3.9.3 Фільтрація контактів в Box2D 64
3.10 Знищення тел і кордони світу в Box2D 65
3.11 Налаштування Box2D 66
4. Розробка системи координації взаємодія роботів на основі алгоритму Boids 68
4.1 Аналіз існуючих алгоритмів ройового інтелекту 68
4.2 Реалізація алгоритму Boids 69
4.3 Опис роботи системи і оцінка результатів 74
Висновки 80
Перелік джерел посилання 82
Додаток А Вихідний код програми 84
Додаток Б Відомість кваліфікаційної роботи 123

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Боід – елементарний об'єкт рою;

Багатоагентна система – це система, утворена декількома взаємодіючими інтелектуальними агентами;

НДЛ – науково-дослідна лабораторія;

ПК – персональний комп'ютер;

ПЕОМ – персональна електронна обчислювальна машина;

Рій – група, колектив;

РІ – ройовий інтелект;

РЧ – рій частинок;

ЦФ – цільова функція;

PSO – particle swarm optimization – алгоритм рою частинок.

ВСТУП

Ройовий інтелект описує колективну поведінку децентралізованої системи, що самоорганізовується. Розглядається в теорії штучного інтелекту як метод оптимізації.

З точки зору інформатики, ройовий інтелект (РІ) – це розділ комп'ютерних наук, який досліджує ефективні чисельні методи вирішення проблем засобом, схожим з поведінкою «колективу» живих організмів.

Системи ройового інтелекту, як правило, складаються з безлічі агентів (багатоагентна система), локально взаємодіючих між собою і з навколишнім середовищем. Ідеї поведінки, як правило, виходять від природи, а особливо, від біологічних систем. Кожен боід слідує дуже простим правилам і, незважаючи на те, що немає якоїсь централізованої системи управління поведінки, яка б вказувала кожному з них на те, що йому слід робити, локальні і, в деякій мірі, випадкові взаємодії призводять до виникнення глобально інтелектуальної поведінки, неконтрольованої окремими боідамі. Таким чином, в цілому РІ (ройовий інтелект) являє собою багатоагентну систему, що володіє самоорганизованою поведінкою. Прикладом в природі може служити колонія мурашок, рій бджіл, зграя птахів, косяк риб.

Такі характеристики ройових алгоритмів, як можливість самоорганізації, відсутність централізованої системи управління і порівняльна простота реалізації, часто роблять їх незамінними при вирішенні завдань оптимізації.

Ройовий інтелект актуальний тим, що завдяки простим локальним взаємодій частинок, стає можливим створити складне групове поведінку.

Ройовий інтелект привабливий з інтуїтивної точки зору, бо він заснований на біологічній моделі поведінки тварин. В майбутньому розвиток ройових технологій може призвести до створення роботів зі складною колективною взаємодією, такі роботи стануть у пригоді в

медицині, військовій промисловості, рятувальних і пошукових операціях, а також в сфері розваг та послуг.

У сучасному світі алгоритми ройового інтелекту знаходять застосування в багатьох областях науки і техніки, і найбільш перспективною з них є робототехніка.

Проблемами координації і взаємодії великої кількості відносно простих роботів займається групова робототехніка. Закладені правила індивідуальної поведінки можуть створювати складну організовану поведінку всієї групи. Крім того, на відміну від розподілених робототехнічних систем, групова робототехніка підкреслює велику кількість роботів, а також передбачає масштабованість, наприклад, з використанням тільки локальної зв'язку.

Алгоритми ройового інтелекту в груповій робототехніці дозволяють реалізувати координацію руху і пошук оптимального шляху, обмін і використання корисної інформації, а також прийняття рішень, що відносяться до конкретної розв'язуваної проблеми групою роботів.

Слід зауважити, що, незважаючи на активний розвиток ройового інтелекту і перспективи його застосування, такі алгоритми досить обмежені у використанні на практиці, так як рівень технологій поки не досяг потрібних висот, і рішення проблем РІ в області робототехніки обмежується тільки теоретичними і експериментальними дослідженнями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Мета кваліфікаційної роботи

Метою даної кваліфікаційної роботи є дослідження використання ройового інтелекту в робототехніці для вирішення завдань координації та комунікації роботів.

Практичний результат дослідження − розроблений програмний додаток, що реалізує один з алгоритмів ройового інтелекту. Для наочності передбачається розробка середовища, що симулює взаємодію боідів в різноманітних ситуаціях. Перед кожним роботом ставиться завдання пошуку предметів, їх транспортування, а також захист від зовнішніх впливів у вигляді всіляких перешкод і рухомих об'єктів.

У даній роботі проведена оцінка ефективності командної взаємодії в порівнянні з індивідуальними діями агентів. В якості оцінюваних параметрів передбачається вибрати кількість знайдених цілей і кількість боідів в кожній групі.

Для виконання даного завдання проведено дослідження існуючих алгоритмів ройового інтелекту, таких як алгоритм рою частинок, мурашиний алгоритм, бджолиний алгоритм і багато інших. Кожен алгоритм оцінений за критеріями ефективності, універсальності, адаптації до нових ситуацій і простоти реалізації.

Для розробки середовища взаємодії боідов передбачається використовувати бібліотеку Box2D, яка дозволяє реалізувати фізичну і графічну складові симулятора. Точна розробка такого середовища є важливою частиною роботи, так як в подальшому готова програма може бути з легкістю перенесена на фізичні боіди.

1.2 Ройовий інтелект в природі

Груповий інтелект тварин часто перевершує розумові здібності однієї особи [1]. З боку може здатися, що, наприклад, мурахи завжди знають, що їм слід робити. Вони виглядають впевнено, наче у них є ясний план дій, і вони чітко уявляють собі, куди йти і навіщо.

Насправді це не так. Кожен взятий окремо мураха не володіє ніякими навичками і знаннями. Коли потрібно вирішити, що робити в конкретній ситуації, вони впадають в розгубленість. Саме завдяки колективній взаємодії приблизно дванадцяти тисячам відомих видів мурах вдалося успішно проіснувати на Землі цілих сто сорок мільйонів років.

Колонія може вирішувати завдання, немислимі для одного мурашки: знаходити найкоротший шлях до джерела їжі, розподіляти обов'язки між працівниками, захищати свою територію від ворогів. І якщо окремо взятий мураха – досить проста тварина, колонія здатна швидко і ефективно реагувати на виникаючі проблеми. Все це стає можливим завдяки так званому колективному інтелекту.

Походження цього інтелекту пов'язано з однією з головних проблем біології: як завдяки простим діям окремих особ складається складна поведінку групи? Здатність живих істот діяти узгоджено в інтересах всієї групи, з урахуванням того, що кінцева мета для них прихована, є загадкою для вчених [2]. Однак за останні десятиліття зроблено чимало дивовижних відкриттів.

Наприклад, один із ключів до розуміння устрою колонії в тому, що там немає керуючих і підлеглих. І навіть якщо в колонії налічується півмільйона особин, вона прекрасно функціонує. Життя колонії побудоване на безперервній взаємодії між окремими тваринами, кожен з яких дотримується набору простих правил. Така система називається самоорганізованою [3].

Спостерігаючи за розподілом обов'язків у мурах, можна помітити,

що ці комахи спілкуються за допомогою дотику і нюху, передаючи важливу інформацію про знайдену їжу і можливих ворогів. Коли мураха зустрічає свого побратима, він «обнюхує» його вусиками, щоб з'ясувати, чи є вони з ним сусідами по гнізду, і дізнатися, де той щойно працював (мурахи, що працюють за межами мурашника, пахнуть інакше, ніж ті, які залишаються всередині). Кожен день, перш ніж покинути гніздо, фуражири зустрічаються з патрульними мурахами, що повертаються додому, і отримують цю важливу інформацію.

Для того, щоб фуражир прийняв рішення покинути мурашник, йому необхідно зустрітися з патрульним певну кількість разів, і проміжок між зустрічами не повинен перевищувати десяти секунд. Частота зустрічей з патрульними мурахами допомагає фуражирам оцінити безпеку обстановки за межами гнізда. Коли мурахи, що відправилися за провіантом, повертаються назад з видобутком, родичі їм допоможуть, якщо вони бачилися один з одним досить часто. Це означає, що рішення поповнити їстівні запаси, мурахи приймають всім колективом [4].

Тепер принцип роботи ройового інтелекту зрозумілий: комахи слідують простим правилам, діючи на підставі наявної у них обмеженої інформації. Така поведінка надихнула Марко Дориго, фахівця з комп'ютерних технологій з Брюсселя. Для створення математичних алгоритмів, які дозволили б вирішити такі складні завдання, як складання оптимальних маршрутів вантажоперевезень і розкладу авіарейсів, а також управління військовими роботами, він використовував накопичені відомості про поведінку мурах.

Подібну програму використовує американська компанія American Air Liquide, яка виробляє гази для промислових і медичних цілей [5]. Постачання готової продукції здійснюється по шести тисячам адрес по трубопроводах, залізниці і на чотирьох сотнях вантажівок. Завдання ускладнюється тим, що ціни на ринках енергоносіїв в деяких регіонах можуть змінюватися кожні п'ятнадцять хвилин. Потрібно знайти спосіб

зводити всю цю інформацію воєдино.

У співпраці з компанією NuTech Solutions, що займається питаннями штучного інтелекту, Air Liquide розробила комп'ютерну модель, в основу якої лягли алгоритми, отримані в результаті дослідження поведінки аргентинських мурашок, які б виробляли феромони. Тепер компанія може враховувати найменші зміни графіка роботи заводів, погодних умов і маршрутів вантажоперевезень [6].

Ройовий інтелект властивий не тільки мурашкам. Також відома вражаюча здатність медоносних бджіл до колективного взаємодії і спільним рішення.

Коли в колонії виникає перенаселення, рій розділяється на дві частини: матка, трутні і приблизно половина робочих бджіл відлітають на невелику відстань і розбивають табір на гілці дерева. Решта відправляються на пошуки іншого житла. У ньому повинно вистачати місця для потомства і запасів меду, так що права на помилку у них немає.

Як тільки біля входу в нову оселю збирається достатня кількість бджіл, вони подають рою сигнал, що вибір зроблено і прийшла пора рухатися до нового місця.

Вважається, що бджолиний метод прийняття рішень (виявлення наявних можливостей, вільна конкуренція ідей і відмова від непродуктивних варіантів) був би дуже доречним людям [7]. Наприклад, представники таких професій, як інвестори на фондовому ринку або вчені-дослідники, можуть приймати правильні колективні рішення, за умови, якщо члени колективу один на одного не схожі, незалежні у своїх судженнях і якщо вони вдаються до голосування або вибирають усереднений варіант.

Наприклад, гравці на скачках можуть досить точно прогнозувати результати заїздів. Прогнози майже завжди виявляються вірними [8]. Кінь, на яку поставило більшість гравців, приходить першої, наступна за списком – другий і так далі. Причина полягає в тому, що тоталізатор – це

практично ідеальний приклад прояву інтелекту натовпу. Подібно бджолам, які намагаються прийняти вірне рішення, гравці збирають інформацію, розходяться один з одним в думках, а потім виносять колективне судження, роблячи ставки.

У природі тварини можуть жити групами, що налічують величезну кількість осіб. У членів великої колонії, будь то зграя, косяк або табун, набагато більше шансів вчасно помітити хижака, знайти собі прожиток, обзавестися парою або зробити сезонні міграції. Від того, наскільки успішно такі тварини вміють координувати свої дії, залежить їхнє життя.

Ройова поведінка також притаманна і птахам. Кожна з них, кружляючи в небі, пильно спостерігає за поведінкою своїх найближчих сусідів і слідує декільком простим правилам. Ці правила складають основу іншої форми ройового інтелекту, що служить не стільки для прийняття правильного рішення, скільки для точної координації рухів.

Подібні можливості зацікавили Крейга Рейнолдса, фахівця в галузі комп'ютерної графіки. У 1986 році він створив просту на перший погляд програму, в якій брали участь схожі на птахів роботи [9]. Вони повинні були дотримуватися таких трьох правил:

− не стикатися з іншими членами групи;

− вибирати середню траєкторію польоту, орієнтуючись на сусідів;

− триматися поблизу від інших роботів.

Продемонструвавши здатність самоорганізованих моделей копіювати поведінку зграї, Рейнолдс вказав шлях і фахівцям з робототехніки. Команда роботів, здатна координувати свої дії подібно до птахів, працювала б значно ефективніше. Розосередившись по великій території, ці роботи служили б пересувною сенсорною мережею: збираючи інформацію, вони могли б контролювати обстановку. Натрапивши на щось несподіване, група могла б швидко зорієнтуватися і вчасно відреагувати, навіть якщо роботи були б влаштовані досить просто. До того ж у разі поломки одного робота його місце зайняли б інші. Але найголовнішою

перевагою подібної системи є децентралізоване управління: працездатність групи не залежала б від команд центрального робота.

У майбутньому можна буде посилати роботів цілими групами для ураження терористів або позиціонування заложників. Стане можливим відправляти роботів в зруйновані землетрусом будівлі шукати під уламками людей, а також в зони витоку отруйних хімічних речовин, брати проби і усувати неполадки або допомагати прикордонним військам, відстежувати порушників [10].

Чи говоримо ми про мурах, бджіл, або про птахів, у наявності незмінні принципи «розумної» групової поведінки: відсутність «центру управління», вміння оцінювати обстановку, дотримання простих правил. Саме так народжується ефективна стратегія, що дозволяє цим істотам з успіхом виходити зі складних ситуацій. В цьому і полягає ефективність ройового інтелекту.

1.3 Актуальність застосування ройового інтелекту

Методи колективного руху агентів в колонії використовуються при проектуванні систем координованої роботи роботів. Також розподілена взаємодія між агентами призвела до створення деяких кластерних алгоритмів і алгоритмів упорядкування [11]. Моделі поділу праці між агентами колонії були використані для регулювання спільної роботи систем робототехніки.

Дизайнери використовують технології рою як засіб створення складних інтерактивних систем і моделювання натовпу. «Розбиваючи лід» – перший фільм, який використовував технології колективного інтелекту для візуалізації реалістичного зображення руху груп риб і птахів з використанням алгоритму Boids. Тім Бертон створив фільм «Повернення Бетмена» також з використанням технології колективного інтелекту для демонстрації руху груп кажанів. У фільмі «Володар кілець»

використовували подібну технологію, відому як технологія масивів, під час батальних сцен. Такі технології є особливо привабливими, тому що використання колективного інтелекту – це дешевий, надійний і простий спосіб.

Авіакомпанії використовують теорію колективного інтелекту при моделюванні розташування місць пасажирів перед посадкою в літак. Дослідник Дуглас Лоусон використовував алгоритм мурах на основі комп'ютерного моделювання і виявив наявність тільки шести правил взаємодії пасажирів і зміг оцінити час посадки з використанням різних методів розміщення пасажирів [12].

Колективний інтелект може бути використаний в цілому ряді програм. Збройні сили США використовують методи колективного інтелекту для управління безпілотними транспортними засобами. У Європейського космічного агентства в планах розробка «орбітального рою» для самостійної збірки інформації та інтерферометрії. NASA досліджує використання технології колективного інтелекту для створення планетарних карт. У 1992 році робота М. Е. Льюїса і Д. А. Беки довела можливість використання ройової розвідки, за допомогою колективного інтелекту для контролю нанороботів в тілі з метою знищення ракових пухлин. Також колективний інтелект застосовується для інтелектуального аналізу даних [13].

Використання колективного інтелекту в телекомунікаційних мережах також застосовується у вигляді основ мурашиної маршрутизації. Вперше такі алгоритми застосували компанії Dorigo і Hewlett Packard в середині 1990-х років з рядом змін з тих пір. В основному використовувалися імовірнісні таблиці маршрутизації з використанням «нагородження» – зміцнення успішно пройденого маршруту кожного «мурашки» (невеликий пакет управління), який проходить в мережі. Мобільні засоби масової інформації і нові технології також багато в чому отримали розвиток завдяки розвитку колективних систем і застосування

алгоритмів ройового інтелекту.

За словами творців Google, їх пошукова система «використовує колективний розум мережі для визначення якості веб-сторінок». Посилання користувачів вона розглядає як голоси на користь даного сайту. У підсумку ми отримуємо список сторінок, відсортованих системою в порядку убування їх популярності [14].

Іншим не менш успішним проектом виявилася «Вікіпедія». У цій безкоштовної енциклопедії колективу авторів містяться мільйони статей про все на світі більш ніж на двохстах мовами. «Одна людина не може знати, як вирішувати проблеми, що виникають перед суспільством, наприклад, як вдосконалити систему охорони здоров'я або зупинити зміни клімату на планеті. Але всі разом ми володіємо величезним запасом знань, які до сих пір нам не вдавалося успішно застосувати на практиці», – каже Томас Мелоун з Центру досліджень колективного інтелекту при Массачусетському технологічному інституті.

Слід зауважити ще одну важливу властивість колективного розуму: натовп може бути розумним, тільки якщо індивіди несуть відповідальність за свої дії і самостійно приймають рішення. Якщо ж вони будуть сліпо копіювати поведінку один одного, задовольняти тільки свої бажання або чекати, що хтось підкаже їм, що робити, колектив не зможе функціонувати успішно.

1.4 Групова робототехніка

Групова робототехніка є новий підхід до координації систем багатьох роботів, які складаються з великого числа в основному простих фізичних роботів. Передбачається, що бажана колективна поведінка виникає з взаємодії роботів між собою і їх взаємодії з навколишнім середовищем. Такий підхід відноситься до наукового напрямку штучного колективного інтелекту, який з'явився при проведенні біологічних

досліджень комах, зокрема, мурах, бджіл, а також при дослідженні в інших областях природи, де має місце ройова поведінка.

Дослідження групової робототехніки – це вивчення конструкції роботів, їх зовнішнього вигляду і контролю поведінки. Її поява пов'язана (але не обмежується) з системним ефектом поведінки, що спостерігається у соціальних комах і називається ройовим інтелектом. Відносно прості правила індивідуальної поведінки можуть створювати складну організовану поведінку всього рою. Ключовим моментом є взаємодія між членами групи, яка створює систему постійного зворотного зв'язку. Поведінка рою включає постійну зміну учасників, що взаємодіють один з одним, а також поведінку всієї групи в цілому.

На відміну від просто розподілених робототехнічних систем, групова робототехніка містить велику кількість роботів, а також передбачає масштабованість, наприклад, з використанням тільки локального зв'язку. Такий локальний зв'язок може бути реалізований, наприклад, за допомогою бездротових систем передачі даних в радіочастотному або інфрачервоному діапазонах [15].

Важливим інструментом для систематичного вивчення поведінки групи є відеотрекінг, хоча є й інші методи відстеження. Нещодавно в лабораторії робототехніки Брістоля розробили ультразвукову систему стеження за роєм для дослідницьких цілей. Незважаючи на те, що необхідні подальші дослідження для знаходження методик, які забезпечать достовірний прогноз поведінки групи (при заданих властивостях тільки окремих її членів), такі результати можна вважати великим досягненням.

Ключовими факторами в груповій робототехніці є мініатюризація і вартість. Це дві головні проблеми в створенні великих груп роботів, тому простоті кожного члена команди повинна приділятися особлива увага, і виправданим є підхід з використанням ройового інтелекту для досягнення значущої поведінки на рівні групи, а не на індивідуальному рівні.

Потенційні додатки групової робототехніки включають завдання, які

вимагають мініатюризації (нанороботів, мікроботів), а також рішення розподілених задач зондування в мікроелектромеханічних системах або в людському тілі. З іншого боку, групова робототехніка може підходити для вирішення завдань, які вимагають дешевих виробів, наприклад, при розмінуванні небезпечних територій або заготівлі для сільськогосподарських тварин. Крім того, деякі художники використовують методи групової робототехніки для реалізації нових форм інтерактивного мистецтва.

Основа програмного коду майбутнього – дифузний прикладний код. Він базується на трьох основних принципах:

− взаємодія між кодами двох об'єктів стає слабкішою, якщо кількість об'єктів збільшується. Тому, завдяки паралельній роботі один з одним, несинхронізовані взаємодії – це майбутнє програм, заснованих на ройовому інтелекту;

− поняття мікрокомпонентів тісно пов'язане з поширенням коду, який контролюється на макроскопічному рівні;

− алгоритми необхідно адаптувати до певних проблем, тобто вони повинні знаходити способи вирішення проблем самостійно.

Майбутні програми будуть розвиватися відповідно до завдань, які вони здійснюють в рамках свого середовища. Така концепція стає можливою завдяки використанню мутацій додатків.

1.5 Постановка задачі

Основна задача даної роботи полягає в дослідженні використання ройового інтелекту для вирішення завдань координації та комунікації роботів.

Практичною частиною роботи є програмний додаток, що реалізує один з алгоритмів ройового інтелекту. Планується розробка середовища, що імітує взаємодію роботів. Перед кожним боідом ставиться завдання

пошуку джерел енергії, їх транспортування, а також захист від ворогів у вигляді інших груп боідів.

Попередньо слід вибрати мову програмування, середовище розробки та допоміжні бібліотеки з урахуванням таких показників, як швидкість і зручність розробки, висока продуктивність при роботі алгоритму, а також максимальна простота і зручність для користувача.

Також в ході виконання магістерської роботи слід провести дослідження існуючих алгоритмів ройового інтелекту, таких як мурашиний алгоритм, бджолиний алгоритм, алгоритм рою частинок і його модифікації. Кожен алгоритм буде оцінений за критеріями ефективності, універсальності, простоти реалізації і застосування для задач комунікації і координації руху роботів.

Середовище взаємодії боідів слід розробляти за допомогою бібліотеки Box2D. Box2D є фізичним движком реального часу і призначена для роботи з двомірними фізичними об'єктами. Точна симуляція взаємодії роботів є важливою частиною роботи, так як готова віртуальна модель може бути легко застосована до реальних роботів.

Принцип роботи алгоритму рою частинок можна описати наступними пунктами:

− створення рою частинок;

− знаходження кращого рішення для кожної частинки;

− знаходження кращого рішення серед усіх частинок;

− корекція швидкості кожної частки;

− переміщення кожної частки.

У готовому програмному додатку користувач зможе поспостерігати діяльність роботів, а також провести оцінку ефективності їх дій.

Таким чином, в даній роботі ставиться практичне завдання реалізувати один з алгоритмів ройового інтелекту, стосовно галузі робототехніки, а також розробити природне середовище – симулятор для їх діяльності та взаємодії.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ КОЛЕКТИВНОГО ІНТЕЛЕКТУ

2.1 Алгоритм рою частинок

Метод оптимізації за допомогою рою частинок базується на моделюванні поведінки популяції частинок в просторі параметрів задачі оптимізації [16].

Пропонований метод привабливий простотою реалізації і тим, що в процесі обчислення не використовується градієнт. Метод може використовуватися для вирішення багатьох завдань, включаючи навчання нейромереж, завдання пошуку мінімуму функції, а також завдань, типових для генетичних алгоритмів.

Метод рою часток, як і всі алгоритми, що належать до сімейства еволюційних алгоритмів, є стохастичним, що не вимагає обчислення градієнта, що дозволяє використовувати його в випадках, де обчислення градієнта неможливо, або має високу обчислювальну складність.

В даному алгоритмі агентами є частки в просторі параметрів задачі оптимізації. На кожній ітерації частки мають деякий стан і вектор швидкості. Для кожного положення частинки обчислюється відповідне значення цільової функції, і на цій основі за певними правилами частка змінює своє положення і швидкість в просторі пошуку.

Оптимізація з використанням рою частинок базується на понятті популяції і моделює поведінку птахів у зграї або косяків риб. Первинною метою використання концепції рою частинок була графічна імітація непередбачуваного руху птахів у зграї з метою виявлення базових принципів, завдяки яким птахи літають синхронно і вміють змінювати напрямок руху з перегрупуванням в оптимальні формації.

Зграя птахів являє собою хороший приклад колективної поведінки тварин. Літаючи великими групами, вони майже ніколи не стикаються в повітрі, зграя рухається скоординовано і плавно.

Справа тут не в тому, що хтось віддає накази – в зграях багатьох видів птахів ватажків просто немає. Як і колонія мурашок або бджіл, зграя є ройовий інтелект. Птахи в ній діють згідно з визначеними, досить простим правилам. У польоті, кожна з птахів стежить за своїми родичами і координує свій рух згідно їх положенню. А знайшовши джерело їжі або небезпека, вона сповістить їх про це.

Одним з найбільш точних пояснень цього феномена є те, що переваги від такої поведінки кожної особини зграї більше, ніж такі очевидні недоліки, як необхідність боротьби за знайдену їжу з іншими особинами [17].

Джерела їжі зазвичай розташовані випадковим чином, тому на самоті птах цілком може загинути. Однак якщо всі птахи будуть ділитися з родичами інформацією про знахідки, то шанси кожної з них на виживання різко підвищуються. Таким чином, будучи неефективною для окремої особини, така стратегія є запорукою ефективності зграї і виду в цілому.

2.1.1 Алгоритм Boids

Спостереження за птахами надихнуло Крейга Рейнольдса на створення в 1986 році комп'ютерної моделі, яку він назвав Boids. Для імітації поведінки зграї птахів Рейнольдс задав поведінку і взаємодію кожної з них окремо. При цьому він використовував три простих принципу:

− кожен птах в його моделі прагнула уникнути зіткнень з іншими птахами;

− кожен птах рухалася в тому ж напрямку, що і знаходяться неподалік птиці;

− птиці прагнули рухатися на однаковій відстані одна від одної.

Результати перших же симуляцій показали, що, незважаючи на простоту алгоритмів в основі програми, зграя при цьому виглядала вкрай

правдоподібно. Птахи збивалися в групи, йшли від зіткнень і навіть хаотично металися як справжні.

Як фахівець в області комп'ютерної графіки, Крейг Рейнольдс був в першу чергу зацікавлений візуальної стороною результатів створеної ним імітації. Однак в присвяченій алгоритму Boids статті він також зазначив, що розроблена ним поведінкова модель може бути розширена введенням додаткових чинників – таких, як пошук їжі або боязнь хижаків [18].

2.1.2 Алгоритм рою частинок GBEST

У 1995 році Джеймс Кеннеді і Рассел Еберхарт запропонували метод для оптимізації безперервних нелінійних функцій, який назвали алгоритмом рою частинок (рисунок 2.1). Алгоритм був розроблений на основі імітаційної моделі Рейнольдса, а також роботи Хеппнер і гренадери. Кеннеді і Еберхарт відзначили, що обидві моделі засновані на управлінні дистанціями між птахами, а, відповідно, синхронність зграї є в них функцією від зусиль, які птиці докладають для збереження оптимальної дистанції.

Розроблений ними алгоритм досить простий і може бути реалізований буквально в декількох десятках рядків коду на будь-якій високорівневій мові програмування. Він моделює багатоагентного систему, де агенти-частинки рухаються до оптимальних рішень, обмінюючись при цьому інформацією з іншими частинками.

Поточний стан частинки характеризується координатами в просторі рішень, а також вектором швидкості переміщення. Обидва цих параметра вибираються випадковим чином на етапі ініціалізації. Крім того, кожна частка зберігає координати одного з найкращих знайдених їй рішень, а також найкраще з пройдених усіма частинками рішень – цим імітується миттєвий обмін інформацією між птахами.

Рисунок 2.1 – Алгоритм рою частинок

На кожній ітерації алгоритму напрямок і довжина вектора швидкості кожної з частинок змінюються в відповідність до відомостей про знайдених оптимумах:

$$v_i = v_i + a_1 * rand() * (pbest_i - x_i) + a_2 * rand() * (gbest_i - x_i), \quad (3.1)$$

де v – вектор швидкості частинки (vi – його i-а компонента);

a1, a2 – постійні прискорення;

rand () – функція, яка повертає випадкове число від 0 до 1 включно;

pbest – найкраща знайдена часткою точка;

x – поточний стан частинки;

gbest – найкраща точка з пройдених усіма частинками системи.

Після обчислення напрямку вектора v, частка переміщається в точку.

У разі необхідності, оновлюються значення кращих точок для кожної частки і для всіх частинок в цілому. Після цього цикл повторюється, $x = x + v$.

У своїй статті, яка описує алгоритм рою частинок, Джеймс Кеннеді і Рассел Еберхарт запропонували використовувати алгоритм для імітації соціальної поведінки [19]. Однак найбільше поширення алгоритм отримав в задачах оптимізації складних багатовимірних нелінійних функцій.

Алгоритм рою частинок широко застосовується в багатьох сферах науки і техніки:

− в задачах машинного навчання (зокрема, для навчання нейромереж і розпізнавання зображень);

− в галузі проектування параметричної і структурної оптимізації (форм, розмірів і топологій);

− в галузях біохімії і біомеханіки.

За ефективністю він може змагатися з іншими методами глобальної оптимізації, а низька алгоритмічна складність сприяє простоті його вивчення і реалізації.

Найбільш перспективними напрямками подальших досліджень в даній області слід вважати:

− теоретичні дослідження збіжності алгоритму рою частинок і пов'язаних з цим питань з областей ройового інтелекту і теорії хаосу;

− комбінування різних модифікацій алгоритму для вирішення складних завдань;

− розгляд алгоритму рою частинок як багатоагентній обчислювальної системи;

− дослідження можливостей включення в даний алгоритм аналогів більш складних природних механізмів.

2.2 Модифікації алгоритму рою частинок

Алгоритм рою частинок з'явився порівняно недавно, проте різними дослідниками вже був запропонований цілий ряд його модифікацій, крім того нові роботи на цю тему не перестають публікуватися. Можна виділити кілька шляхів поліпшення класичного алгоритму, реалізованих в більшості з них:

− з'єднання алгоритму з іншими алгоритмами оптимізації;

− зменшення ймовірності передчасної збіжності шляхом зміни характеристик руху частинок;

− динамічна зміна параметрів алгоритму під час оптимізації.

2.2.1 Модифікація LBEST

У тому ж 1995 року пізніше була опублікована стаття Кеннеді і Еберхарта, в якій вони описали оригінальний алгоритм GBEST, оскільки він використовує глобальне краще рішення для формування векторів швидкостей, а також запропонували його модифікацію, названу ними LBEST. При оновленні напрямку і швидкості руху частинки в алгоритмі LBEST використовується інформація про рішення сусідніх з ними частинок:

$$v_i = v_i + a_1 * rand() * (pbest_i - x_i) + a_2 * rand() * (lbest_i - x_i), \quad (3.2)$$

де v – вектор швидкості частинки (vi – його i-а компонента);

a1, a2 – постійні прискорення;

rand () – функція, яка повертає випадкове число від 0 до 1 включно;

pbest – найкраща знайдена часткою точка;

x – поточний стан частинки;

lbest – кращий результат серед частинки і її сусідів.

Сусідніми вважаються або частки, що відрізняються від поточного елемента індексом не більше ніж на деяке задане значення, або частки, відстань до яких не перевищує заданого порогу.

Даний алгоритм ретельноїї досліджує простір пошуку, проте є більш повільним, ніж оригінальний. При цьому, чим менше число сусідніх частинок враховується при формуванні вектора швидкості, тим нижче швидкість збіжності алгоритму, але тим ефективніше він уникає субоптимальних рішень [20].

2.2.2 Модифікація Inertia Weighted PSO

У 1998 році Юхи Ши і Рассел Еберхарт запропонували модифікацію, на перший погляд зовсім незначно відрізняється від класичного алгоритму. У своїй статті Ши і Еберхарт помітили, що однією з головних проблем при вирішенні задач оптимізації є баланс міжретельністю дослідженням простору пошуку та швидкістю збіжності алгоритму [21]. Залежно від завдання і характеристик пошукового простору в ній, цей баланс повинен бути різлічним.С урахуванням цього Ши і Еберхарт запропонували змінити правило поновлення векторів швидкостей частинок:

$$v_i = w_i * v_i + a_1 * rand() * (pbest_i - x_i) + a_2 * rand() * (gbest_i - x_i), \quad (3.3)$$

де v – вектор швидкості частинки (vi – його і-а компонента);

w – коефіцієнт інерції, який визначає баланс між широтою дослідження і увагою до знайденим субоптимальних рішень;

a1, a2 – постійні прискорення;

rand () – функція, яка повертає випадкове число від 0 до 1 включно;

pbest – найкраща знайдена часткою точка;

x – поточний стан частинки;

gbest – найкраща точка з пройдених усіма частинками системи.

У разі, коли, швидкості частинок збільшуються, далі вони розлітаються в сторони і досліджують простір більш ретельно. В іншому випадку, швидкості частинок з часом зменшуються, швидкість збіжності в такому випадку залежить від вибору параметрів a1 і a2.$w > 1$

2.2.3 Модифікація Time-Varying Inertia Weighted PSO

У 1998 році в своїй роботі Ши і Еберхарт відзначили, що інерція не обов'язково повинна бути позитивною константою, вони вважали, що вона може змінюватися під час роботи алгоритму за лінійним або навіть нелінійному закону. У більш пізніх роботах вони часто використовували лінійний закон убування, як досить ефективний і в той же час простий [22]. Проте, розроблялися і успішно застосовувалися і інші закони зміни інерції.

Значення коефіцієнта інерції може як зменшуватися, так і рости. При його убування, частинки спочатку досліджують область пошуку екстенсивно, знаходячи безліч субоптимальних рішень, іпотім з часом все більше концентруються на дослідженні їх околиць. Зростання інерції сприяє збіжність алгоритму на пізніх стадіях роботи.

2.2.4 Модифікація Canonical PSO

У 2002 році Маріс Клер і Джеймс Кеннеді запропонували свою модифікацію алгоритму рою частинок, яка стала настільки популярною, що її прийнято називати канонічним алгоритмом рою частинок. Він дозволяє позбутися від необхідності підбирати підходящі значення регульованих параметрів алгоритму, контролюючи збіжність частинок.

Клер і Кеннеді змінили спосіб обчислення векторів швидкостей частинок, ввівши в нього додатковий множник:

$$v_i = X * [v_i + a_1 * rand() * (pbest_i - x_i) + a_2 * rand() * (gbest_i - x_i)], \quad (3.4)$$

де v – вектор швидкості частинки (vi – його і-а компонента);

X – коефіцієнт стиснення,;$X = 2 * k / \left|2 - a - \sqrt{a^2 - 4 * a}\right|$

a1, a2 – постійні прискорення;

a – сумарне прискорення,;$a = a_1 + a_2 > 4$

rand () – функція, яка повертає випадкове число від 0 до 1 включно;

pbest – найкраща знайдена часткою точка;

x – поточний стан частинки;

gbest – найкраща точка з пройдених усіма частинками системи.

Такий підхід гарантує збіжність алгоритму без необхідності явно контролювати швидкість частинок.

2.2.5 Модифікація Fully Informed Particle Swarm

У квітні 2004 року Руї Мендес, Джеймс Кеннеді і Жозе Невес помітили своїй роботі, що прийняте в канонічному алгоритмі припущення про те, що на кожну з частинок впливає тільки найбільш успішна, не відповідає які лежать в його основі природним принципам і, можливо, веде до зниження ефективності алгоритму. Вони прийшли до рішення, що з-за надмірної уваги алгоритму до єдиного рішення може бути втрачена важлива інформація про структуру простору пошуку.

Виходячи з цього, вони запропонували зробити все частки «повністю поінформованими», тобто отримують інформацію від усіх сусідніх частинок. Для цього вони змінили в канонічному алгоритмі закон зміни швидкості:

$$v_i = X * [v_i + \sum_{k \epsilon N} a_k * W(k) * rand() * \left(pbest_{k,i} - x_i\right), \quad (3.5)$$

де v – вектор швидкості частинки (vi – його і-а компонента);

X – коефіцієнт стиснення,;$X = (2 * k)/\left|2 - a - \sqrt{a^2 - 4 * a}\right|$

a1, a2 – постійні прискорення;

a – сумарне прискорення,;$a = a_1 + a_2 > 4$

N – безліч сусідів частки;

W (k) – вагова функція, яка може відображати будь-яку характеристику k-ой частки, яка вважається важливою, а саме значення цільової функції в точці, в якій вона знаходиться, дистанцію від неї до даної частинки і так далі;

rand () – функція, яка повертає випадкове число від 0 до 1 включно;

pbestk – найкраща з пройдених k-им сусідом точок;

x – поточний стан частинки;

gbest – найкраща точка з пройдених усіма частинками системи.

2.3 Мурашиний алгоритм

Мурашиний алгоритм – один з ефективних поліноміальних алгоритмів для знаходження наближених рішень завдання комівояжера, а також аналогічних завдань пошуку маршрутів на графах. Підхід запропонований бельгійським дослідником Марко Дориго.

Суть підходу полягає в аналізі та використанні моделі поведінкимурах, Що шукають шляхи від колонії до їжі. В основі алгоритму лежить поведінка мурашиної колонії – маркування вдалих доріг великою кількістюферомонів. Робота починається з розміщення мурашок у вершинахграфа, Потім починається рух мурашок – напрям визначається імовірнісним методом, на підставі формули:

$$P_i = \left(l_i^q * f_i^p\right) / \left(\sum_{k=0}^{N} l_k^q * f_k^p\right), \quad (3.6)$$

де P_i – ймовірність переходу дорогої i;

l_i – довжинаi-ого переходу;

f_i – кількість феромонів наi-ом переході;

q – величина, що визначає «жадібність» алгоритму;

p – величина, яка визначає «стадність» алгоритму, $q + p = 1$.

2.4 Бджолиний алгоритм

Бджолиний алгоритм є досить молодим алгоритмом для знаходження глобальних екстремумів складних багатовимірних функцій.

Даний алгоритм моделює поведінку бджіл в природному середовищі. Ідея бджолиного алгоритму полягає в тому, що всі бджоли на кожному кроці будуть вибирати як елітні ділянки для дослідження, так і ділянки в околиці елітних, що дозволить, по-перше, урізноманітнити популяцію рішень на наступних ітераціях, по-друге, збільшити ймовірність виявлення близьких до оптимальних рішень. Основні поняття бджолиного алгоритму полягають в наступних пунктах:

− джерело нектару (квітка, ділянка);

− фуражири (робочі бджоли);

− бджоли-розвідники.

Джерело нектару характеризується значимістю, яка визначається різними параметрами. Фуражири закріплені за джерелами нектару. Кількість всіх бджіл у цих ділянках більше, ніж на інших. Середня кількість розвідників в рої становить 5–10%. Сам алгоритм можна описати наступними пунктами:

− генерація ділянок для пошуку нектару;

− оцінка корисності ділянок;

− вибір ділянок для пошуку в їх околиці;

− відправка фуражирів;

− пошук в околицях джерел нектару;

− відправка бджіл-розвідників;

− випадковий пошук;

− оцінка корисності нових ділянок.

Таким чином, ключовою операцією алгоритму бджіл є спільне дослідження перспективних областей і їх околиць. В кінці роботи алгоритму популяція рішень буде складатися з двох частин: бджоли з кращими значеннями цільової функції (ЦФ) елітних ділянок, а також групи робочих бджіл з випадковими значеннями ЦФ.

3 ВИКОРИСТАННЯ ФІЗИЧНОЇ БІБЛІОТЕКИ BOX2D ДЛЯ МОДЕЛЮВАННЯ ПОВЕДІНКИ РОБОТІВ

3.1 Опис Box2D

Box2D – це бібліотека для моделювання поведінки твердих тіл в двовимірному просторі. З її допомогою програміст може змусити об'єкти рухатися реалістично з точки зору фізики, а світ виглядати більш інтерактивним.

Бібліотека Box2D є фізичним движком реального часу і призначена для роботи з двомірними фізичними об'єктами. Движок Box2D є кросплатформним програмним забезпеченням, він написаний на мові програмування C ++, тому може працювати на будь-якій платформі, на якій присутній компілятор C ++.

Бібліотека Box2D призначена для симуляції механіки твердих тіл з урахуванням обмежень. Движок може симулювати фізичні тіла, складені з опуклих багатокутників, кіл і ліній. Тіла можуть бути пов'язані обмежувачами в кінематичні пари і піддаватися впливу різних фізичних сил, таких як гравітація, тертя і удар. Тіла також можуть зазнати впливу внутрішніх сил, таких як пружність.

3.2 Основні поняття Box2D

Box2D працює з наступними базовими об'єктами:

− тверде тіло – предмет, що складається з матеріалу настільки міцного, що відстань між будь-якими двома точками цього предмета є постійним при будь-яких умовах;

− фігура – плоска геометрична фігура, жорстко прикріплена до тіла, застосовується для розрахунків зіткнень. Фігури мають властивості притаманні матеріалами, такі як коефіцієнт тертя і пружність;

− обмеження – фізичне з'єднання, що обмежує свободу переміщення і обертання тіла. На площині тіло має 3 ступеня свободи (рух по вертикалі, рух по горизонталі, обертання);

− контактне обмеження – спеціальне обмеження, призначене для запобігання проникнення тіл один в одного, а також моделювання тертя і пружності тіл;

− з'єднання – це спеціальне обмеження, яке використовується для з'єднання двох і більш тел;

− обмежене з'єднання – як випливає з назви, діапазон рухів такого з'єднання обмежений. Як приклад можна привести лікоть людини;

− з'єднання-мускул – приводить в рух усіх під'єднаних до нього тіла відповідно до ступенями свободи їх з'єднань. Наприклад, можна використовувати м'язів, щоб змусити лікоть згинатися і розгинатися;

− світ – світ об'єднує всі тіла, фігури і обмеження, які взаємодіють один з одним. Box2D підтримує створення декількох світів, але зазвичай це не потрібно.

3.3 Створення фізичних об'єктів в Box2D

3.3.1 Створення світу в Box2D

Кожна програма з використанням движка Box2D починається зі створення об'єкта світу. Цей об'єкт центр управління пам'яттю, об'єктами і процесами моделювання.

Для його створення потрібно визначити обмежуючий бокс. Обмежуючий бокс використовуються в Box2D для прискорення розрахунків зіткнень тіл. Розмір боксу не критичний, але хороший його підбір збільшить продуктивність.

Далі слід визначити вектор гравітації і повідомити світу, що тілам дозволено «засипати», коли вони знаходяться в стані спокою. Сплячий

об'єкт не потребує моделюванні поведінки.

3.3.2 Створення землі в Box2D

Для створення фізичного тіла потрібно виконати наступні кроки:

− сформувати визначення тіла: позиція, амортизація і т.д;

− створити тіло за допомогою об'єкта світу;

− сформувати визначення фігур: геометрія, тертя і щільність;

− створити фігури на тілі;

− встановити масу, відповідну прикріпленим до тіла фігурам (опціонально).

Для створення тіла слід створити бокс землі. Для цього нам необхідно визначення тіла. У ньому потрібно задати початкову позицію тіла. Далі визначення тіла передається в об'єкт світу для створення самого тіла. Об'єкт світу не зберігає посилання на визначення тіла. Бокс землі створюється як статичне тіло.

Статичне тіло не стикається з іншими статичними тілами і не здатне рухатися. Box2D визначає, що тіло статичне, якщо його маса дорівнює нулю. Тіла мають нульову масу за замовчуванням, отже, вони за умовчанням статичні.

Далі потрібно створити визначення полігону землі. Для цього використовується спрощена функція SetAsBox, щоб сформувати полігон землі як бокс, центр якого знаходиться на початку координат батьківського тіла.

Кожна фігура повинна мати батьківське тіло, навіть якщо ця фігура статична. Однак можна приєднати всі статичні фігури до одного тіла. Це було зроблено для статичних тел для того, щоб зробити код всередині Box2D однаковим, універсальним, зменшивши таким чином кількість потенційних помилок.

3.3.3 Створення динамічних тел в Box2D

Для створення динамічного тіла використовується така ж техніка. Основна відмінність, крім розмірів, полягає в тому, що потрібно встановити для нього безліч параметрів, властивих динамічним об'єктам.

Спочатку створюється тіло, використовуючи CreateBody. Потім встановлюється фігура-полігон і прикріплюється до тіла. При цьому щільність тіла за замовчуванням дорівнює нулю. Після приєднання фігури потрібно викликати метод SetMassFromShapes, щоб перерахувати масу тіла на основі даних про приєднаних фігурах. Якщо маса тіла після перерахунку виявиться нульовою, то тіло стає статичним. Слід зауважити, що тіла мають нульову масу за замовчуванням.

Для моделювання руху тел Box2D використовує чисельне диференціювання (метод Ейлера). Виглядає це таким чином, що світ Box2D містить функцію Step, що має два аргументи: час, яке необхідно змоделювати в світі, і кількість ітерацій для вирішення взаємодій між об'єктами.

Зазвичай, фізичні движки для ігор використовують timeStep не більше 1/60 секунди, такого інтервалу цілком достатньо для реалістичного моделювання фізики в іграх. Можна задати інший timeStep, але в такому випадку доведеться обережніше налаштовувати параметри вашого світу. Тому timeStep змінювати не слід, але при цьому дуже небажано пов'язувати тимчасової крок з частотою зміни кадрів графічної підсистеми.

У коді Box2D більшу частину займає обробник обмежень. Завдання даного обробника полягає в прорахунку обмежень для правильної поведінки тел при зіткненнях. При прорахунку одного обмеження не виникає особливих труднощів. Однак при прорахунку кількох обмежень, дозвіл одного обмеження злегка порушує інші. Тому для отримання хорошого результату необхідно провести прорахунок всіх обмежень кілька разів (тобто зробити кілька ітерацій).

У Box2D пропонується використовувати 10 ітерацій. Можна задати кількість ітерацій на свій розсуд, але необхідно пам'ятати наступне правило: «чим більше кількість ітерацій, тим більше точність і менше швидкість прорахунку обмежень» та навпаки, «чим менше кількість ітерацій, тим менше точність і більше швидкість».

Слід зауважити, що параметри timeStep і кількість ітерацій ніяк не пов'язані між собою, ітерація – це не крок. Одна ітерація – це один розрахунок всіх обмежень в кожен момент часу. Можна використовувати кілька проходів в кожен момент часу для збільшення точності.

Тепер можна почати цикл моделювання. Цикл моделювання може бути об'єднаний з програмним циклом. В такому випадку на кожній ітерації програмного циклу необхідно викликати функцію Step. Зазвичай достатньо одного виклику, але це залежить від частоти кадрів і величини timeStep.

Коли об'єкт світу покидає зону видимості або знищується внаслідок видалення покажчика на нього, вся пам'ять, зарезервована світом для тіл і з'єднань між ними, звільняється. Таким чином, розробник повинен обнулити покажчики на всі тіла, фігури і з'єднання, які зберігаються в програмному забезпеченні, так як ці покажчики стануть недійсними.

3.4 Інтерфейс Box2D

3.4.1 Управління динамічної пам'яттю в Box2D

Необхідність у швидкій і ефективній роботі з пам'яттю всередині Box2D стала причиною великої кількості дискусій.

Box2D потребує виділення пам'яті для великої кількості маленьких об'єктів. Використання системного Хіпа через функції malloc або new для невеликих об'єктів неефективно і призводить до фрагментації пам'яті. При цьому велика частина цих об'єктів має невеликий час життя (наприклад,

контакти), але при цьому може існувати протягом декількох фізичних циклів. Тому необхідний аллокатор, який зможе ефективно забезпечувати додаток динамічною пам'яттю для великої кількості маленьких об'єктів.

У Box2D для вирішення цієї проблеми використовується аллокатор маленьких об'єктів (SOA). SOA зберігає кілька блоків пам'яті різного розміру. При вимозі виділити пам'ять SOA надає блок пам'яті, найкращим чином відповідний необхідного розміру.

Якщо блок більше не потрібен, то він просто позначається як вільний. Обидві операції виконуються дуже швидко і не призводять до фрагментації пам'яті.

Так як Box2D використовує власний SOA, забороняється виділяти пам'ять для тіл, фігур або з'єднань вручну. Виділення пам'яті вручну дозволено тільки для b2World. Клас b2World надає фабрики (спеціальні функції для створення об'єктів) для створення тіл, фігур і з'єднань. Фабрики дозволяють використовувати ефективний механізм виділення пам'яті, при цьому приховуючи непотрібні деталі. Аналогічно, забороняється використовувати delete або free для тіл, фігур і з'єднань.

При прорахунку поточного часового інтервалу, Box2D потребує тимчасової робочої пам'яті. Для цього він використовує стековий аллокатор, щоб кожного разу не звертатися до хіпу.

3.4.2 Фабрики і визначення в Box2D

Управління пам'яттю грає дуже важливу роль в інтерфейсі Box2D. Тому для створення b2Body або b2Joint, необхідно викликати функції-фабрики класу b2World (за якими ховається швидкий і ефективний аллокатор пам'яті).

При створенні тіла або з'єднання, необхідно передавати у функцію-фабрику покажчик на визначення тіла. Визначення містять всю інформацію, необхідну фабриці для створення тіла або з'єднання.

Використання визначень дозволяє запобігти помилкам при створенні тіл і з'єднань, скоротити кількість аргументів у функцій-фабрик і зменшити число аксессор (функцій для доступу до членів класу). Так як будь-яка фігура повинна бути приєднана до тіла-батькові, то і створюються вони фабричними методами тіла-батька, тобто класу b2Body. Фабрики незберігають покажчики на визначення тел або з'єднань, тому визначення можна оголошувати як локальні змінні.

3.4.3 Одиниці виміру в Box2D

Так як Box2D використовує числа з плаваючою точкою, необхідно відразу встановити допустимі розміри і відхилення від них. Параметри Box2D підібрані для використання одиниць МКС (метр-кілограм-секунда) і роботи з динамічними об'єктами, розмір яких складає від 0.1 до10 метрів.

Ідея використовувати пікселі як одиниці виміру дуже приваблива, але, на жаль це призведе до неточних моделювання і, можливо, дуже дивній поведінці об'єктів.

Так як Box2D використовує одиниці МКС, слід використовувати розміри рухомих об'єктів, в інтервалі від 0.1 до 10 метрів. Потрібно скористатися масштабуванням при виведенні результату в графічному

вигляді. У Box2D використовується OpenGL-перетворення поля виведення.

3.4.4 Користувальницькі дані в Box2D

Класи b2Shape, b2Body і b2Joint дозволяють зберегти покажчик на якісь інші дані у вигляді покажчика на довільний тип void. Це зручно, коли при використанні об'єктів Box2D необхідно ставити відповідні їм програмні об'єкти. Наприклад, часто цей покажчик зберігає посилання на відповідний «програмний об'єкт».

Виходить своєрідна циклічна посилання, коли, маючи програмний об'єкт, можна отримати покажчик на фізичне тіло і, навпаки, маючи тіло, отримати покажчик на програмний об'єкт.

Наприклад, в наведених нижче ситуаціях без використання такого покажчика не обійтися:

− пошкодження об'єкта після зіткнення;

− виконання якої-небудь події при попаданні персонажа в обмежуючий бокс;

− повідомлення ігрових об'єктів про розрив з'єднання між фізичними об'єктами.

Слід пам'ятати, що покажчик на дані можна використовувати на свій розсуд і зберігати в ньому все що завгодно, але при його використанні слід бути обережним. Наприклад, якщо в цьому покажчику зберігається посилання на ігровий об'єкт, то і в інших об'єктах повинна зберігатися посилання того ж типу, інакше, може статися виняткова ситуація при спробі приведення типу покажчика.

3.4.5 Використання мови програмування C ++ в Box2D

C ++ мова з класичною інкапсуляцією даних і поліморфізмом, але не настільки добра для побудови API (інтерфейсів програмування додатків).

У випадку з фабриками, з одного боку, API виглядає простіше, але в кінцевому підсумку доводиться ставати на шлях ефективної розробки та налагодження. При використанні концепції захищених (private) даних і «дружніх» функцій, можливо з часом кількість «друзів» стане величезною.

Для Box2D обраний шлях мінімального опору. Якщо для деяких випадків класи з їх пристроєм і функціональністю підходять краще, то використовуються відкриті функції-інтерфейси і захищені дані. В інших випадках використовуються класи або структури, всі дані яких відкриті для доступу ззовні.

Завдяки такому підходу вдалося досягти високої швидкості розробки, легкості при налагодженні і мінімальної турботи про збереження непроникною для стороннього погляду інкапсуляції даних. Зворотною стороною став той факт, що в Box2D немає чистого і простого API.

3.5 Світ в Box2D

3.5.1 Створення і знищення світу в Box2D

Клас b2World містить тіла і з'єднання. Він керує всіма аспектами моделювання і дозволяє обробляти асинхронні черги (як черги AABB).

Значна частина взаємодії з Box2D відбувається через об'єкт b2World. Створити світ нескладно. Для цього необхідно надати обмежує бокс світу (AABB) і вектор гравітації. AABB повинен містити весь світ. Можна поліпшити продуктивність, зробивши бокс трохи більше ніж світ, наприклад, раз на 2 для надійності. Якщо виходить багато тіл, що потрапляють за межі світу, то ці тіла необхідно знаходити і видаляти. Це підвищить продуктивність і запобіжить можливому переповнення при обробці чисел з плаваючою точкою. Для створення і видалення світу слід використовувати відповідно оператори new і delete.

AABB світу завжди повинен бути більше регіону, де розташовуються тіла. Якщо тіло (можливо в процесі моделювання) перетинає кордон AABB світу, то воно заморожується і виключається з процесу моделювання.

3.5.2 Використання світу в Box2D

Клас світу містить функції-фабрики, призначені для створення і знищення тіл і з'єднань.

Клас світу використовується для запуску та управління процесом

моделювання. Для цього потрібно задати тривалість кроку і кількість ітерацій на крок. Після виконання розрахунку кроку фізики можна обробити всі тіла і з'єднання. Швидше за все, доведеться перевіряти позиції тел, щоб мати можливість оновити позиції об'єктів в грі і промальовувати їх на екрані.

Можна виконувати розрахунок кроку фізики в будь-якому місці ігрового циклу, але слід бути обережним з порядком виконання операцій. Наприклад, якщо необхідно в поточному кадрі створити нове тіло і відразу перевірити те об'єктів з ним, то потрібно спочатку створити тіло, розташувати його в потрібному місці світу, запустити розрахунок кроку фізики, і тільки потім перевіряти чи відбулося зіткнення.

Крім того, в Box2D слід використовувати фіксовану тривалість кроку. Використання великих кроків підвищить продуктивність в додатках з низькою частотою кадрів. Але не варто використовувати крок довший 1/30 секунди. Зазвичай 1/60 секунди досить для високоякісного моделювання.

Кількість ітерацій визначає, скільки разів буде проведено розрахунки над усіма тілами і з'єднаннями в світі. Більше число ітерацій завжди забезпечує кращу якість моделювання. Але не варто поєднувати маленький крок з великою кількістю ітерацій. Тобто 60Гц і 10 ітерацій набагато краще 30Гц і 20 ітерацій.

Справа тут в тому, що при розподілі дуже маленького числа (довжини кроку) на дуже велику (кількість ітерацій) крім усього іншого можна взагалі отримати нульове значення.

Як згадувалося раніше, світ є «контейнером» для всіх об'єктів і з'єднань. Тому, зі світу можна отримати список всіх тіл і з'єднань світу, а потім зробити що-небудь с кожним з них.

Іноді потрібно визначити всі фігури знаходяться в певному регіоні. Клас b2World містить таку функцію і швидкість її роботи висока. Функція приймає в якості параметрів AABB в світових координатах і повертає

масив всіх фігур, які перетинаються з AABB (або знаходяться повністю в ньому). Слово «перетинаються» не зовсім правильно, тому що перевірка на перетин відбувається між AABB регіону і AABB самих фігур.

3.6 Фізичні тіла в Box2D

3.6.1 Визначення фізичних тіл в Box2D

У фізичних тіл є положення і швидкість. До них можна застосовувати сили, моменти і імпульси. Тіла можуть бути статичними або динамічними, статичні тіла ніколи не переміщаються і не взаємодіють з іншими статичними тілами.

Тіла – основа всіх програмних форм. Вони складаються з фігур і переміщаються разом з ними. Тіла, перш за все, є «твердими». Це означає, що дві фігури, приєднані до одного тіла, завжди будуть залишатися нерухомими друг для друга.

В «ігрових» двигунах зазвичай зберігаються покажчики на всі створені фізичні тіла. Тому, при необхідності, можна отримати положення тіла, наприклад, для поновлення графічного об'єкта. Покажчики на тіла також необхідно зберегти, якщо їх буде потрібно видалити.

Перед створенням тіла необхідно заповнити визначення тіла (b2BodyDef). b2BodyDef можна задати як в якості локальної змінної, так і в якості частини структури ігрових даних.

Box2D копіює ухвалу тіла і не зберігає покажчик на отримане визначення. Тому одне визначення можна використовувати для створення декількох тел.

Далі будуть описані основні частини структури b2BodyDef.

3.6.1.1 Маса

У більшості додатків, що використовують Box2D, має сенс обчислювати масу, враховуючи щільність фігур. Це дозволяє перевірити, чи мають тіла реалістичну масу. Однак іноді може знадобитися конкретне значення маси. Наприклад, в механізмі зразок ваг, який вимагає точні значення мас. Є кілька способів завдання маси тіла:

− задати значення маси безпосередньо у визначенні тіла;

− задати значення маси вже після створення тіла;

− обчислити масу на основі щільності приєднаних фігур.

3.6.1.2 Позиція і кут

Визначення тіла дозволяє вказати початкову позицію тіла. Це швидше, ніж створювати тіло, а потім переміщати його. Тіло має два важливі параметри. Перший – позиція тіла. Фігури і з'єднання створюються в координатах щодо позиції тіла. Другий – центр мас.

Позиція центру мас визначається виходячи з щільності фігур і їх форми або безпосередньо параметром massData визначення тіла.

Більшість внутрішніх обчислень Box2D відбувається з використанням центру мас. Наприклад, b2Body зберігає лінійну швидкість саме центру маси. При заповненні визначення тіла може бути невідомо, де знаходиться центр мас, тому вказується позиція тіла.

Також можна вказати кут повороту тіла в радіанах, який не залежить від позиції центру мас. Зміна позиції центру мас не впливає на позицію тіла і приєднані фігури і з'єднання.

3.6.1.3 Гальмування

Використовується, щоб зменшувати швидкість тел. Воно

відрізняється від тертя тим, що тертя зменшує швидкість тільки при торканні, а гальмування діє постійно. Таким чином, гальмування не замінює тертя і вони повинні використовуватися разом.

Параметр «гальмування» повинен мати значення між нулем і нескінченністю, де 0 – відсутність гальмування, а нескінченність – повне

гальмування. Найчастіше використовується «гальмування» з діапазону 0 – 0.1.

Лінійне гальмування зазвичай не використовується, так як воно робить тіла схожими на що знаходяться під водою.

При маленькому значенні гальмування ефект здебільшого не залежить від величини тимчасового кроку, але при великих значеннях буде помітно залежати. Це не стане проблемою при використанні фіксованої довжини кроку (як рекомендується).

3.6.1.4 Параметри сну

Моделювання тіл – досить коштовна задача, і чим менше об'єктів бере участь в моделюванні, тим краще. Коли тіло переходить в стан спокою, необхідно припинити його моделювання.

Коли тіло (або група тіл) переходить в стан спокою, Box2D переводить це тіло в стан сну, яке дуже слабо завантажує процесор. Якщо бодрствующее тіло стосується сплячого, то спляче тіло прокидається. Тіла можуть також прокидатися, якщо руйнується з'єднання або контакт, приєднаний до нього. Також тіло можна розбудити вручну. Визначення тіла дозволяє вам точно визначити, чи може тіло засипати і створюється тіло вже сплячим.

3.6.1.5 Снаряди

Програмне моделювання зазвичай генерує послідовність зображень-

кадрів, які показуються з певною частотою (FPS). В таких умовах тверді тіла можуть пересуватися на великі відстані за один крок розрахунків.

Якщо фізичний движок не призначений для розрахунків великих пересувань, то можна побачити, як одні об'єкти проходять крізь інші. Цей ефект отримав назву тунельного.

За замовчуванням, Box2D використовує визначення зіткнень на протязі шляху (CCD) щоб оберегти динамічні об'єкти від проскаківання крізь статичні об'єкти, тобто від тунельного ефекту.

Працює це дуже просто: фігури простягаються від початкової до кінцевої позиції. Движок стежить, чи не відбувається зіткнень в процесі переміщення тіла між позиціями і визначає час зіткнення (TOI). Тіла переміщаються в точки відповідні їх першим TOI і моделювання триває до кінця часового відрізку. Цей процес повторюється стільки раз, скільки знадобиться.

Зазвичай CCD не використовується для розрахунків зіткнень між динамічним тілами, це зроблено щоб зберегти розумну швидкість роботи. У деяких ситуаціях може знадобитися використовувати CCD для динамічних тел. Наприклад, якщо потрібно вистрілити високошвидкісний кулею в тонку стіну. Без CCD, куля, швидше за все, пролетіла б крізь стіну.

Швидкого руху об'єктів в Box2D іменуються «снарядами». Установка прапора «снаряда» дієва тільки для динамічних тел. CCD–це дорога операція, так що краще не відзначати всі рухомі тіла як «снаряди». Так в Box2D за замовчуванням CCD використовується тільки між парами динамічний тіло – статичне тіло. Цей підхід ефективний, якщо необхідно не дати тіл покинути ваш ігровий світ. Однак можливі ситуації, коли слід мати кілька об'єктів, що рухаються, які зажадають постійного використання CCD.

3.6.2 Видалення фізичних тіл в Box2D

Видалення об'єктів відбувається за допомогою фабрики тіл. Фабрика тіл дозволяє створювати і знищувати тіла з використанням методів-фабрик об'єкта b2World. Це дає можливість світу створити тіло з використанням ефективного аллокатора і додати його в світ.

Тіла можуть бути динамічними або статичними в залежності від маси. Обидва типи тел створюються і знищуються за допомогою одних і тих же методів.

При цьому вкрай не рекомендується використовувати new або malloc для створення тел. Світ не знатиме про існування цього тіла, і воно не буде правильно налаштований. Статичні тіла не переміщаються під впливом інших тіл. Можна вручну переміщати статичні тіла, але слід бути обережним, щоб не роздавити динамічні тіла, що знаходяться між двома або кількома статичними.

Тертя не працює при переміщенні статичного тіла, а також статичні тіла ніколи не взаємодіють один з одним. При створенні статичної геометрії ігрового рівня, швидше приєднати кілька фігур до одного статичному тілу, ніж створювати кілька статичних тел.

Box2D встановлює масу і зворотний масу статичних тел рівними нулю, тому для внутрішніх алгоритмів моделювання немає різниці між статичними і динамічними тілами.

Box2D не зберігає посилання на визначення тіла або будь-які знаходяться в ньому дані (за винятком покажчика на призначені для користувача дані). Тому можна оголошувати визначення як локальні змінні і використовувати один і той же визначення для створення декількох тел.

Box2D позбавляє від необхідності видаляти кожне тіло вручну, так як при видаленні світу об'єкт сам виконує всю роботу по збірці сміття. Однак, після цього необхідно обнулити всі покажчики на фізичні тіла в ігровому движку.

При видаленні тіла, все приєднані до нього фігури і з'єднання видаляються автоматично. Ця деталь має велике значення при використанні покажчиків на фігури і з'єднання.

У випадках, коли необхідно з'єднати динамічне тіло з нерухомою точкою, слід з'єднати динамічний тіло зі статичним тілом. Якщо статичного тіла немає, то можна використовувати загальний бокс землі.

3.6.3 Використання фізичних тіл в Box2D

Після створення тіла над ним можна робити безліч операцій, включаючи настройку параметрів маси, отримання позиції і швидкості тіла, додаток до нього сил, перетворення точок та векторів між локальними і світовими координатами.

Можна задавати масу прямо під час виконання програми. Зазвичай це робиться, коли створюється або знищується фігура тіла. Може знадобитися задати масу тіла виходячи з даних про приєднаних до нього фігурах.

Також буває необхідно встановити масу для тіла вручну. Наприклад, якщо необхідно змінити фігури, але при цьому використовувати свої власні формули для розрахунку маси.

Зазвичай необхідно дізнатися позицію і кут повороту фізичного тіла, наприклад для відображення відповідного графічного об'єкта. Можна вручну встановити позицію тіла, але так роблять рідко, зазвичай для моделювання переміщень використовується Box2D.

Можна дізнатися світові координати центру мас тіла. У більшості внутрішніх алгоритмів Box2D для моделювання використовується саме центр мас. Але зазвичай знати координати центру мас не потрібно, частіше потрібно знати позицію тіла.

До тіла можна застосовувати сили, моменти і імпульси. При застосуванні сили або імпульсу вказується світова координата їх

застосування. Часто при цьому виникає момент обертання навколо центру мас.

Застосування сили, моменту або імпульсу виводить тіло зі стану сну. Іноді потрібно застосувати силу тільки для стережуть тел.

Клас b2Body має допоміжні функції для перетворення точок та векторів між локальними і світовими координатами. Ці функції не ресурсомісткі, так що ними можна регулярно користуватися.

Також при бажанні можна отримати доступ до всіх фігур заданого тіла. Що дуже зручно, якщо необхідно отримати призначені для користувача дані для кожної фігури, приєднаної до певного тілу.

3.7 Фігури

3.7.1 Визначення фігури в Box2D

Фігури – це геометрія, яка приєднується до тіл і використовується для перевірки зіткнень між ними. Фігури також використовуються для визначення маси тіла. Це дозволяє визначити масу тіла, лише поставивши щільність, Box2D зробить все обчислення сам.

Фігури мають коефіцієнти тертя і пружності. У фігурах міститься інформація про фільтрах зіткнень, що дозволяє не перевіряти зіткнення між певними об'єктами

Фігури завжди належать певному тілу. До одного тіла можна приєднати кілька фігур. Фігури є абстрактними класами, тому в Box2D можуть бути реалізовані різні їх типи. Тому при бажанні можна створити свій власний тип фігур і відповідні алгоритми розрахунку зіткнень між ними.

Визначення фігур використовуються для їх створення. У визначенні b2ShapeDef присутній як загальна для всіх інформація, так і специфічна інформація для окремих типів фігур.

3.7.1.1 Коефіцієнти тертя і пружності

Тертя використовується для створення реалістичного ковзання одних об'єктів по іншим. Box2D підтримує статичну і динамічну тертя, але використовує один і той же параметр для обох. Тертя в Box2D моделюється безпомилково, і сила тертя пропорційно залежить від величини нормальної сили (тертя Колумба).

Параметр тертя має значення в межах від 0 до 1. Нуль відповідає відсутності тертя, а одиниця –максимальному тертю. При обчисленні тертя між двома тілами Box2D повинен використовувати коефіцієнти тертя обох тел.

Коефіцієнт пружності використовується при зіткненні тіл. Значення коефіцієнта може бути з діапазону від 0 до 1. Наприклад, при падінні кулі на стіл, значення 0 означає, що куля не відскакує – це називається неупругое зіткнення, при одиниці же куля відскочить з тією ж швидкістю – це абсолютно пружне зіткнення.

Якщо тіла мають кілька спільних позицій, то відновлення моделюється наближено. Це відбувається через те, що Box2D використовує ітеративний обробник зіткнень.

Box2D також використовує нееластичні зіткнення, якщо швидкість зіткнення мала. Це необхідно для запобігання тремтіння тіл.

3.7.1.2 Щільність

При необхідності Box2D може обчислити масу і момент інерції тіл, використовуючи щільність приєднаних фігур. Вказівка маси тіла вручну часто призводить до поганого моделювання, тому, кращим шляхом завдання маси тіла буде спочатку вказівку щільності всіх фігур тіла, а потім виклик функції SetMassFromShape.

3.7.1.3 Фільтрація зіткнень

Фільтрація зіткнень дозволяє запобігти зіткненням між певними фігурами. Наприклад, при створенні персонажа, який їде на велосипеді, якщо необхідно, щоб велосипед і персонаж взаємодіяли з ландшафтом, але при цьому велосипед і персонаж не взаємодіяли між собою (так як вони перетинаються). Для таких випадків Box2D підтримує фільтрацію зіткнень за допомогою категорій і груп.

Box2D підтримує 16 категорій взаємодії. Для кожної фігури можна вказати категорію, якій вона належить. Можна також вказати, з якими іншими категоріями може взаємодіяти ця фігура.

Наприклад, в багатокористувацької грі необхідно вказати, що всі гравці не взаємодіють один з одним, і що монстри не взаємодіють один з одним, але при цьому необхідно, щоб гравці і монстри взаємодіяли один з одним. Це можна здійснити за допомогою маски бітів maskBits.

Групи взаємодії дозволяють вказати індекс групи. Всі фігури з однаковим індексом групи можуть завжди взаємодіяти (якщо індекс позитивний) або ніколи не взаємодіяти (якщо індекс негативний). Індекси груп зазвичай використовуються для речей, які як-небудь пов'язані, подібно частинам велосипеда.

Зіткнення між фігурами різних груп фільтруються відповідно до категорій і «маскують» битами. Іншими словами, групи мають більш високий пріоритет у порівнянні з категоріями. Слід зазначити, що в Box2D відбувається додаткова фільтрація зіткнень:

− фігури статичних тіл ніколи не взаємодіють один з одним;

− фігури одного і того ж тіла ніколи не взаємодіють один з одним;

− при бажанні можна вказати чи будуть стикатися фігури з'єднаних тіл.

Іноді необхідно змінити фільтрацію зіткнень після того, як фігура була створена. Можна отримати і встановити структуру b2FilterData

наявної фігури, використовуючи методи GetFilterData і SetFilterData. Box2D кешируєт результати фільтрування, тому, після зміни фільтрації, потрібно викликати Refilter.

3.7.1.4 Сенсори

Іноді потрібно дізнатися, коли перетнулися дві фігури, і при цьому не потрібно обробляти зіткнення між ними. Це завдання вирішується за допомогою сенсорів.

Сенсор – це фігура, яка лише виявляє зіткнення з іншими тілами і при цьому не впливає на них. Можна помітити будь-яку фігуру (статичну і динамічну) як сенсор. При цьому якщо тіло має кілька фігур, то будь-які з них можуть бути сенсорами.

3.7.1.5 Геометричні фігури

Структура b2CircleDef – це розширення b2ShapeDef шляхом додавання полів радіусу і локальної позиції.

Структура b2PolyDef використовується для створення опуклих полігонів. При цьому максимальне число вершин визначається константою b2_maxPolygonVertices, за замовчуванням рівний 8. Якщо необхідно використовувати більшу кількість вершин, то слід змінити константу b2_maxPolygonVertices.

При побудові полігону необхідно вказати число вершин, яке буде використовуватися. Координати вершин повинні бути вказані в порядку проти годинникової стрілки щодо осі z правобічної системи координат. Але на екрані це може виглядати як за годинниковою стрілкою в залежності від екранних координат.

Полігони повинні бути опуклими і межі полігону не повинні перетинатися. Іншими словами, кожна вершина повинна виглядати

назовні. Box2D автоматично замкне контур полігону.

Вершини задаються в системі координат тіла. Якщо необхідно змістити полігон щодо тіла, то треба тільки змістити позиції вершин. Для зручності є функції ініціалізації полігонів у вигляді боксів. Можна створити як AABB з центром в позиції тіла, так і повернений бокс зі зміщенням щодо позиції тіла.

3.7.2 Фабрики фігур в Box2D

Для того, щоб створити фігуру, потрібно заповнити визначення фігури і передати його методу-фабриці тіла. Покажчик на фігуру необов'язково десь зберігати, так як фігура автоматично знищиться при знищенні тіла. Після приєднання фігур до тіла, може знадобитися перерахунок маси тіла.

Певна фігура тіла видаляється просто. Це можна використовувати для моделювання руйнуються об'єктів. Потрібно просто попросити тіло видалити приєднану фігуру. Після знищення фігур тіла бажано викликати метод SetMassFromShapes знову.

Так само можна дізнатися тип фігури і якого тіла вона належить і перевірити, чи знаходиться задана точка в фігурі.

3.8 З'єднання

3.8.1 Визначення сполук в Box2D

З'єднання використовуються для приєднання тіла до світу або інших тіл.

З'єднання можуть поєднуватися безліччю способів, що дозволяє отримати безліч цікавих рухів. Деякі сполуки надають обмеження, за допомогою яких ви можете контролювати діапазон можливий рухів цього

з'єднання. Деякі надають двигуни-м'язи, які можна використовувати для приведення з'єднання в рух із заданою швидкістю і силою / обертає.

З'єднання-м'язи можуть використовуватися безліччю способів. Ставлячи з'єднанню швидкість, пропорційну відстані від поточної точки до бажаної, можна рухати предмети.

Також можна використовувати «м'язи» для симуляції амортизує, що труться з'єднання. Для цього слід задати з'єднанню нульову швидкість, і в якості максимально можливої сили / крутного моменту з'єднання встановити мале значення. Тоді двигун буде намагатися зберегти з'єднання нерухомим, поки навантаження не стане занадто високою.

Всі визначення сполук успадковані від b2JointDef. Всі з'єднання створюються між двома різними тілами. Одне з них може бути статичним. Втім, якщо потрібно витрачати пам'ять, то краще з'єднувати також і статичні тіла.

Крім того, можна задати призначені для користувача дані для будь-якого з'єднання, а також відключити перевірку зіткнень між з'єднаними тілами. Насправді, це поведінка за умовчанням, і якщо потрібно перевіряти зіткнення з'єднаних тел, то потрібно вручну виставити прапор collideConnected в true.

Багато визначення сполук вимагають, щоб ви надали деякі геометричні дані. Часто з'єднання будуть визначатися через точки-якорі. Це точки, які зафіксовані на приєднаних тілах. Box2D вимагає, щоб ці точки визначалися в локальних координатах тел. У цьому випадку з'єднання може бути задано, навіть якщо трансформації тіла насильно змінили з'єднання, наприклад, коли додаток зберігається і перезавантажується.

Додатково, деякі визначення тел повинні знати початковий відносний кут між тілами (болтове з'єднання). Це необхідно, зокрема для коректного обмеження обертання, при використанні обмежень або фіксованого кута між тілами.

Так як ініціалізація геометричних даних може бути досить складним заняттям, багато типів з'єднань мають функції ініціалізації, які використовують поточний стан тіла, щоб зменшити кількість необхідної роботи від розробника.

Проте, ці функції ініціалізації зазвичай слід використовувати тільки для прототипування. В кінцевому коді краще визначати геометрію вручну. Це зробить поведінку сполук більш адекватним.

Вміст визначення тіла залежить від типу з'єднання.

3.8.1.1 З'єднання «жорсткий відрізок»

Найпростішим типом з'єднання є з'єднання «жорсткий відрізок», яке означає, що відстань між двома точками двох тіл завжди має бути постійним. При створенні з'єднання «жорсткий відрізок» тіла вже повинні знаходитися в потрібному положенні.

Потім вказуються якірні точки в світових координатах. Перша якірна точка приєднується до першого тіла, а друга відповідно до другого. Відстань між точками на увазі довжину жорсткого відрізка.

3.8.1.2 Болтове з'єднання

Болтове з'єднання можна уявити, як ніби в певній точці просвердлили отвір через обидва тіла і вставили в нього болт, навколо якого вони можуть обертатися. Цю точку часто називають шарнірної точкою.

Болтове з'єднання має одну ступінь свободи: кут повороту другого тіла відносно першого. Цей кут називається кут з'єднання.

Для створення болтового з'єднання необхідно надати два тіла і вказати якірну крапку в світових координатах. Функція ініціалізації передбачає, що тіла вже знаходяться в правильному положенні. Кут

болтового з'єднання позитивний, коли body2 повертається проти годинникової стрілки щодо body1.

Подібно до всіх інших кутах в Box2D, кут болтового з'єднання вимірюється в радіанах. За замовчуванням, кут болтового з'єднання після створення дорівнює нулю, незалежно від поточних кутів повороту тел.

Іноді потрібно контролювати кут з'єднання. Для цього, болтове з'єднання може моделювати обмеження і / або м'язів. Обмеження з'єднання змушує кут болтового з'єднання перебувати в певних межах. При виході кута болтового з'єднання за межі до тіл застосовується момент для повернення кута в правильне положення. Тому верхню чи нижню межу кута з'єднання повинен включати нуль, інакше при початку моделювання буде помітний різкий ривок тел.

Мускул дозволяє вказати швидкість з'єднання (похідну кута за часом). Швидкість може бути позитивною або негативною. Мускул може мати нескінченну силу, але це не бажано. Ви можете встановити максимальний крутний момент для м'яз. Мускул буде підтримувати задану швидкість, поки необхідний для цього момент, що обертає не перевищить певного максимуму. При перевищенні максимуму, обертання з'єднання буде призупинено і може продовжитися в іншому напрямку. Можна використовувати м'язів для моделювання тертя в з'єднанні. Треба тільки встановити швидкість з'єднання в нуль, а максимальний обертовий момент зробити невеликий ненульовий величиною. Мускул намагатиметься запобігти обертання з'єднання, але не зможе протистояти значному навантаженні.

3.8.1.3 Призматичне з'єднання

Призматичне з'єднання дозволяє тіл переміщатися лише щодо певної осі (заданої щодо тел) і не допускає обертання цих тіл. Тому, призматичне з'єднання має одну ступінь свободи: переміщення другого тіла відносно

першого.

Визначення призматичного з'єднання подібно опису болтового з'єднання; тільки замінює кут переміщенням, а момент силою.

Болтове з'єднання має неявно задається вісь (яка перпендикулярна екрану, тому не може бути відображена) навколо якої тіла можуть обертатися.

Для призматичного з'єднання необхідно явно вказати вісь, що лежить в площині екрану. Ця вісь зберігає своє положення щодо тіл і переміщається разом з ними.

Подібно болтових з'єднань, переміщення призматичного з'єднання дорівнює нулю при створенні з'єднання. Тому необхідно щоб верхню чи нижню межу дорівнював нулю.

3.8.1.4 Талеві з'єднання

Талеві з'єднання використовується для створення ідеальних талів. Талі з'єднують два тіла з землею і один з одним. Загальна довжина двох сегментів талей фіксована.

Також можна використовувати коефіцієнт, щоб змоделювати «блоки і талі». Це може бути потрібно, якщо ви хочете, щоб одна зі сторін талі видовжувалася швидше, ніж інша. Крім того іноді обмежує сила на одній стороні слабкіше ніж на інший.

Наприклад, якщо коефіцієнт сторін дорівнює 2, то lenght1 буде змінюватися пропорційно подвоєною lenght2. Також сила на відрізку, приєднаному до body1, буде дорівнює половині стримуючої сили на відрізку, приєднаному до body2.

Коли одна сторона повністю подовжилася, можуть виникнути проблеми. Відрізок на іншій стороні вийде нульовим. У цьому випадку рівняння обмеження матиме єдине рішення (причому погане). Тому Талеві з'єднання обмежує максимальну довжину, яку будь-який з відрізків може

досягти. Хоча, може знадобитися контролювати максимальну довжину з причин обраної предметної області.

Завдання максимально можливої довжини для відрізків підвищує стабільність і дає більше контролю над ситуацією.

3.8.1.5 Передаточне з'єднання

При створенні складних, хитромудрих механізмів не обійтися без використання зубчастих з'єднань. В принципі, можна створити шестерні, об'єднуючи кілька фігур для моделювання зубів, але це дуже неефективно і досить утомливо. Також доведеться налаштувати шестерні, щоб зуби знаходилися в зачепленні. Тому Box2D має більш простий спосіб – передавальне з'єднання.

Для передавального з'єднання необхідно, щоб два тіла мали болтове або призматичне з'єднання з статичним тілом (або з землею). Подібно коефіцієнту сторін в талевого з'єднанні, передавальне з'єднання має коефіцієнт передачі. Однак в даному випадку коефіцієнт передачі може бути негативним.

Також слід пам'ятати, що, якщо одне з'єднання болтове (обертання), а інше призматичне (переміщення), то коефіцієнт передачі буде мати розмірність довжини або одиниці, поділеній на розмірність довжини.

Важливо пам'ятати, що передавальне з'єднання залежить від двох інших з'єднань, що може привести до небезпечної ситуації. Що станеться, якщо ці з'єднання будуть видалені? Необхідно спочатку видаляти передавальне з'єднання перед видаленням болтових / призматичних. Інакше виникне виняткова ситуація через неправильні покажчиків в передавальному з'єднанні. Також необхідно видаляти передавальне з'єднання перед видаленням з'єднаних тел.

3.8.2 Фабрика з'єднань в Box2D

З'єднання створюються і знищуються за допомогою методів-фабрик світу. Тому потрібно пам'ятати, що не слід намагатися створювати тіла або з'єднання на стеку або на хіпі, використовуючи new або malloc. Необхідно створювати і видаляти тіла і з'єднання тільки за допомогою методів створення і знищення тіл і з'єднань класу b2World.

Вважається хорошим тоном обнуляти покажчики після їх знищення. Це стане причиною гарантованого падіння програми в разі використання покажчика після його знищення.

Час життя з'єднання не завжди визначається просто. Слід запам'ятати, що після знищення тіла, відразу знищуються з'єднання, прикріплені до нього. Проте така обережність не є строгим правилом.

3.8.3 Використання сполук в Box2D

Часто з'єднання створюються і не рушають до їх знищення. Однак в з'єднаннях зберігається багато корисної інформації, яку можна використовувати для створення більш різноманітного моделювання, тобто можна отримати тіла, які з'єднують якірну крапку і призначені для користувача дані.

Всі з'єднання мають силу і крутний момент реакції. Це ті сили, які застосовуються до другого тіла в якірній крапці. Ці сили можна використовувати для розриву з'єднання або для перемикання іншої події. Ці функції можуть виконувати різні обчислення, тому їх не варто викликати занадто часто.

Мускул з'єднання має цікаві можливості. Можна оновлювати швидкість з'єднання на кожному часовому кроці, що дозволяє змусити з'єднання виконувати зворотно поступальні рухи або за іншим законом, відповідно до вказаної функцією. Також можна використовувати м'язів

з'єднання для підтримки певного кута з'єднання. Крім того, зазвичай параметр підсилення не задається занадто великим, інакше з'єднання може стати нестабільним.

3.9 Контакти в Box2D

3.9.1 Опис контактів в Box2D

Контакти –це об'єкти, створювані Box2D для управління зіткненнями фігур. Існують різні види контактів, які успадковуються від класу b2Contact, для управління контактами між різними типами фігур. Наприклад, існує клас контактів для зіткнення полігон-полігон і клас для зіткнення окружність-коло.

Існує деяка термінологія, яка використовується для опису контактів. Ця термінологія в основному призначена для Box2D:

− точка контакту – це точка зіткнення двох фігур. В реальності об'єкти мають пляма контакту, при зіткненні поверхонь. Box2D спрощує це, використовуючи лише невелике число точок контакту;

− нормаль контакту – це одиничний вектор, спрямований від shape1 до shape2;

− поділ контакту – це протилежність глибини проникнення. Поділ негативно, коли фігури перекриваються. Можливо, в майбутніх версіях, Box2D з'явиться можливість створювати точки контакту з позитивним поділом і можна буде перевірити знак поділу при повідомленні про контактах;

− нормальна сила – Box2D використовує ітеративний метод знаходження зіткнень і зберігає результати обчислень в точках контакту. Можна безболісно використовувати нормальну силу для визначення сили удару. Наприклад, можна використовувати її для створення руйнуються предметів або для програвання звуку при ударі;

− дотична сила – використовується для моделювання сили тертя;

− ідентифікатори контактів – Box2D, по можливості, використовує обчислені на попередньому кроці сили, для контактів наступного кроку в якості початкових умов. Для розрізнення контактів протягом кроків моделювання використовуються ідентифікатори контактів. Ідентифікатори містять індекси геометричних параметрів, що використовуються для розрізнення однієї точки контакту від іншої.

Контакти створюються при перетині AABB двох фігур. Іноді фільтрація зіткнень запобігає створення контактів. Для Box2D іноді потрібно створити контакт, незважаючи на фільтрацію зіткнень. В такому випадку використовується b2NullContact, який не схильний до фільтрації. Контакти знищуються, коли AABB фігур перестають перетинатися.

Дивно, але контакти можуть створюватися для фігур, які не перетинаються (перетинаються лише їх AABB). Можна видалити контакт відразу після того як фігури перестануть перетинатися або просто почекати поки перестануть перетинатися AABB фігур. Box2D використовує останній підхід.

3.9.2 Оброблювач контактів в Box2D

Можна отримувати інформацію про контакти, реалізувавши методи класу b2ContactListener (обробник контактів). При появі, існуванні протягом декількох ітерацій або знищення контакту буде викликатися відповідний метод обробника контактів. Слід пам'ятати, що дві фігури можуть мати кілька точок контакту.

При цьому потрібно пам'ятати, що не слід зберігати посилання на контакт, який передається в b2ContactListener. Замість цього необхідно робити глибоке копіювання даних контакту в свій власний буфер. Дозвіл контактів відбуватися за декілька ітерацій, тому контакт може бути доданий або вилучений протягом одного кроку моделювання. Тому код

повинен вміти обробляти таку ситуацію.

Про контактах повідомляється відразу ж, як тільки вони створюються, існують або видаляються. Це відбувається перед тим, як буде викликаний оброблювач зіткнень, тому об'єкт b2ContactPoint не містить обчисленого імпульсу. Однак містить відносну швидкість тел в точці контакту, тому можна приблизно оцінити імпульс.

Якщо викликати метод Result, то буде отриманий об'єкт b2ContactResult для точок контакту після виклику обробника зіткнень. Цей об'єкт буде містити імпульси, обчислені для поточної ітерації.

Аналогічно, метод Result може викликатися кілька разів протягом одного кроку моделювання. Хотілося б реалізувати логіку, яка змінює світ при обробці контактів. Наприклад, можна отримати зіткнення, яке застосовує пошкодження і намагається знищити відповідний об'єкт і фізичне тіло. Однак Box2D не дозволяє змінювати фізичний світ при обробці контактів, так як можна знищити об'єкти, які Box2D обробляє в даний момент, що призведе до звернення до недійсним вказівниками.

Правильним методом обробки контактів вважається збереження потрібних контактів в буфер і робота з ними вже після кроку моделювання. Необхідно обробляти контакти відразу ж після кроку моделювання, інакше інший код може змінити фізичний світ, що призведе до недостовірних даних в буфері контактів. При обробці буфера контактів можна змінювати фізичний світ, але це слід робити обережно, щоб покажчики, що зберігаються в буфері контактів, не стали недійсними.

3.9.3 Фільтрація контактів в Box2D

Часто потрібно, щоб деякі тіла не стикалися. Наприклад, може знадобитися створити двері, через яку можуть проходити тільки певні тіла. Це називається фільтрацією контактів, так як деякі зіткнення фільтруються.

Box2D дозволяє домогтися строго певної фільтрації контактів, для цього необхідно реалізувати клас b2ContactFilter. В цьому класі необхідно реалізувати метод ShouldCollide, який приймає два покажчика на фігури. Цей метод повертає істину, якщо фігури повинні взаємодіяти.

3.10 Знищення тел і кордони світу в Box2D

Можна реалізувати клас b2BoundaryListener, який дозволить світу інформувати вас про тілах, що вилетіли за межі обмежує боксу світу. При отриманні повідомлення не можна намагатися видаляти тіло, замість цього необхідно позначити об'єкт для видалення або помилки. І видалити тіло після кроку моделювання фізики.

Потім необхідно підключити даний обробник до об'єкта світу. Це необхідно зробити при ініціалізації світу.

Box2D не використовує підрахунок посилань. Тому якщо відбувається знищення тіла, то воно дійсно знищується. Спроба завантажити вказівником на знищене тіло може мати невизначений поведінка.

Для вирішення таких проблем, відладочна версія менеджера пам'яті заповнює знищені об'єкти значеннями FDFDFDFD. Це допомагає швидше знайти проблеми в деяких випадках. При видаленні об'єкта Box2D, в першу чергу необхідно переконатися, що також були видалені або обнулені всі посилання на віддалений об'єкт. Це просто, якщо є одне посилання на об'єкт. Якщо є кілька посилань, то можна задуматися над реалізацією обгортки над посиланням на об'єкт Box2D.

Зазвичай при використанні Box2D, буде створюватися безліч тіл, фігур і з'єднань. Робота з цими об'єктами частково автоматизована в Box2D. Якщо видаляється тіло, то і всі пов'язані з ним фігури і з'єднання видаляються автоматично. Це називається – неявне знищення. При цьому звільняється тіло, що знаходиться на іншому кінці з'єднання.

Цей процес зазвичай зручний, але слід пам'ятати про критичний момент. При видаленні тіла, всі фігури і з'єднання пов'язані з тілом автоматично видаляються. Необхідно обнуляти всі наявні покажчики на ці фігури і з'єднання. Інакше, пізніше програма впаде при спробі доступу або знищення цих фігур або з'єднань.

Для допомоги в обнулення покажчиків на з'єднання Box2D надає клас b2WorldListener, який можна реалізувати і зареєструвати в об'єкті світу. Після цього, об'єкт світу буде повідомляти перед неявним видаленням з'єднання.

Неявне знищення в більшості випадків економить багато часу. Але також може привести до падіння програми. При цьому можна зберігати покажчики на фігури і з'єднання десь в коді. Ці покажчики стають недійсними, коли видаляється відповідне тіло. Ситуація погіршується, коли з'єднання створюються частиною коду, що не має відношення до обробки відповідного тіла. Наприклад, тестовий стенд створює b2MouseJoint для маніпуляцій над тілами на екрані.

Box2D надає механізм оповіщення про спробу неявного знищення. Це дозволяє додатку обнулити всі недійсні покажчики. Для цього необхідно реалізувати клас b2DestructionListener, який дозволяє світу інформувати про неявному знищення фігури або з'єднання, якщо знищено відповідне тіло. Це допомагає запобігти доступ до недійсним вказівниками.

Клас обробника неявних знищень потрібно зареєструвати в об'єкті світу. Це необхідно зробити при ініціалізації світу.

3.11 Налаштування Box2D

Є два вихідних файли, що містяться в Box2D спеціально для налаштувань користувача. Це b2Settings.h і b2Settings.cpp. Box2D використовує числа з плаваючою точкою, тому необхідно використовувати

деякі відхилення для правильної роботи Box2D. Є безліч відхилень, що залежать від використовуваних одиниць виміру.

Вся робота Box2D з пам'яттю здійснюється через b2Alloc і b2Free за винятком таких випадків:

− об'єкт b2World може бути створений як локальна змінна або будь-яким іншим способом на розсуд розробника;

− будь-який інший клас Box2D створюваний без використання фабричного методу.

Це включає такі об'єкти як обробники і буфери контактів. Спосіб виділення пам'яті може бути змінений в b2Settings.cpp.

Так само можна реалізувати клас b2DebugDraw (файл b2WorldCallbacks.h), щоб отримати можливість малювати фізичні сутності у всіх деталях:

− контури фігур;

− з'єднання;

− фігури ядра;

− обмежувальні бокси, включаючи AABB світу;

− орієнтовані бокси;

− пари потенційних зіткнень (потенційні контакти);

− центри маси.

Такий метод малювання фізичних об'єктів переважніший, ніж прямий доступ. Причина цього в тому, що потрібна інформація є внутрішньою і постійно змінюється. Тестовий стенд бібліотеки малює фізичні об'єкти, використовуючи оцінювальний рендер і обробник контактів.

4. РОЗРОБКА СИСТЕМИ КООРДИНАЦІЇ ВЗАЄМОДІЯ РОБОТІВ НА ОСНОВІ АЛГОРИТМУ BOIDS

4.1 Аналіз існуючих алгоритмів ройового інтелекту

В ході виконання магістерської роботи розглянуті наступні алгоритми:

− алгоритм Boids;

− алгоритм рою частинок;

− мурашиний алгоритм;

− бджолиний алгоритм.

Оцінивши такі параметри як ефективність, універсальність, адаптація до нових ситуацій, а також простота реалізації і швидкість роботи алгоритму, прийнято рішення, що найбільш повно цим критеріям відповідає алгоритм Boids.

Незважаючи на те, що даний алгоритм послужив прототипом до створення алгоритму PSO, в цілому Boids не відноситься до алгоритмів рою частинок. Це пов'язано з тим, що на відміну від PSO і його модифікацій, алгоритм Boids вимагає знаходження центру мас (градієнта) і підбору відповідних значень регульованих параметрів алгоритму.

Проте, в рамках поставлених завдань магістерської роботи дані відмінності є рекомендуємими, так як дозволяють більш точно налаштовувати координацію руху роботів. Знаходження центру мас дозволяє організувати точний колективний рух, а ручний підбір параметрів алгоритму дозволяє налаштувати пріоритети поведінки роботів.

Мурашиний і бджолиний алгоритми також розглянуті, але з огляду на те, що дані методи більше підходять для завдань оптимізації, пошуку і прийняття рішення, встановлено, що вони не є задовільними для вирішення завдань магістерської роботи.

В цілому алгоритм Boids відповідає всім критеріям. У зв'язку з тим,

що це один з перших алгоритмів в області ройового інтелекту, він досить простий у вивченні і реалізації. Незважаючи на його простоту, він ефективний і має задовільну продуктивність роботи. Також даний алгоритм відмінно підходить для завдання координації і взаємодії роботів, так як в першу чергу розроблявся з метою візуалізації руху зграй птахів.

Слід зауважити, що в ході виконання магістерської роботи алгоритм Boids адаптований до взаємодії з фізичної бібліотекою Box2D. Дане рішення привело до того, що замість вектора швидкості алгоритм знаходить імпульс руху робота. Обмеження по середній швидкості групи замінено на обмеження по максимально можливому значенню швидкості. Крім того, додані такі моделі поведінки, як пошук джерел енергії, а також захист від можливих ворогів. В цілому дані зміни як і раніше здійснені в рамках алгоритму Boids, відповідно до пропозицій автора алгоритму – К. Рейнольдса.

4.2 Реалізація алгоритму Boids

Безпосередньо робота алгоритму Boids полягає в установці певного набору правил поведінки частинок.

Також для кожного боіда слід зберігати координати і швидкість. Крім того потрібно знати координати цілі і перешкод. Під час перерахунку нової ітерації слід визначити нову координату і швидкість для всіх боідов. Для цього необхідно визначити нову швидкість частинки і пересунути її.

Кожне з правил дозволяє отримати частину швидкості. Загальна швидкість боіда дорівнює сумі швидкостей, отриманих після застосування правил і попередньої швидкості самого юніта.

Щоб група боідів рухалася більш цілісно – швидкість рекомендується обмежувати. Також на практиці швидкості, видані кожним з правил, необхідно зменшувати шляхом ділення на якийсь коефіцієнт. Обмеження і коефіцієнти підбираються експериментальним шляхом для

досягнення найкращого результату.

Правила поведінки частинок можна визначити наступним чином:

− згуртованість – кожен боід намагається триматися якомога ближче один до одного. Сплочность знаходиться шляхом розрахунку центру мас. На рисунку 4.1 частинки під номерами від 1 до 5 утворюють центр мас в точці під номером 6;

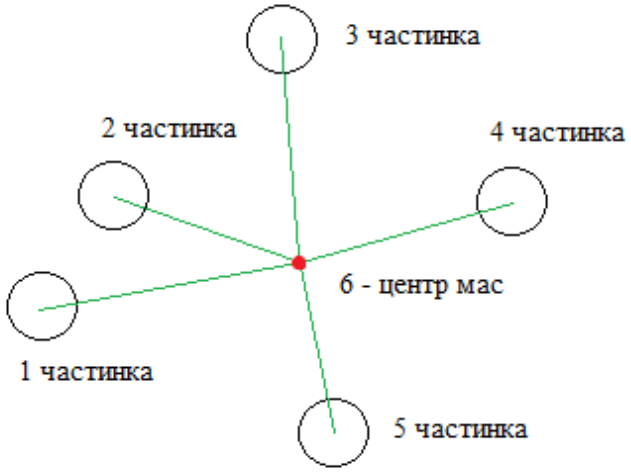

Рисунок 4.1 – Рух до центру мас

− поділ – ті боіди, які виявилися досить близько до сусідів, прагнуть розійтися з метою триматися один від одного на деякій відстані. Поділ розраховується шляхом знаходження всіх координат найближчих сусідів і розрахунку вектора швидкості спрямованого в зворотну сторону від боіда. На рисунку 4.2 можна побачити, що частка під номера 2 виявилася в зоні видимості частки 1, і для частки 1 був розрахований вектор поділу (3 – зона видимості, 4 – вектор поділу);

− рух до мети – боіди рухаються до заданої мети. Мета генерується випадковим чином або задається за умови виявлення цільової точки. На рисунку 4.3 частинки під номерами від 1 до 4 рухаються в цільову точку під номером 5;

Рисунок 4.2 – Вектор руху частки від найближчих сусідів

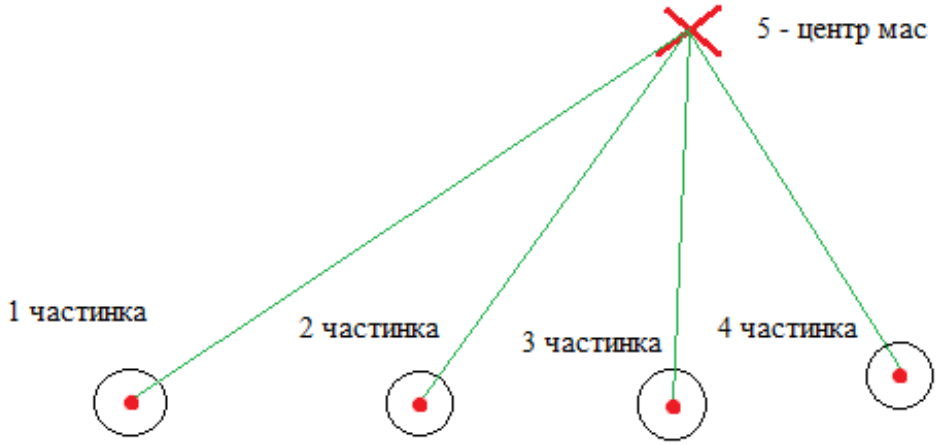

Рисунок 4.3 – Рух до мети

– уникнення перешкод – боіди прагнуть рухатися в протилежну сторону від перешкод. При наявності декількох перешкод розраховується середній вектор швидкості від перешкод. Також як і в разі руху від найближчих сусідів вектор уникнення перешкод розраховується на перетині певної зони видимості перешкод. На рисунку 4.4 можна бачити синій вектор під номером 1 – вектор підсумкової швидкості і червону дугу під номером 2 – область видимості боіда (3, 4 – вектора спрямовані від перешкод);

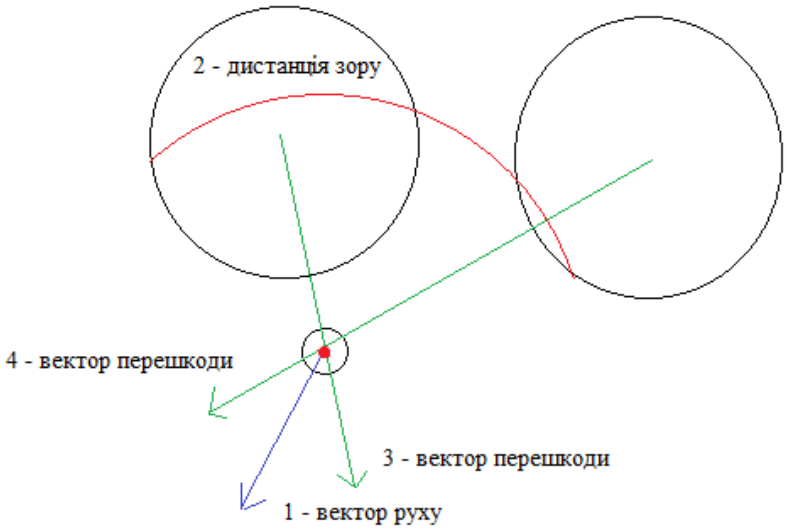

Рисунок 4.4 – Рух від перешкоди (синім кольором позначено підсумковий вектор швидкості)

– вирівнювання швидкостей – полягає в тому, що боіди з однієї групи прагнуть рухатися з однаковою швидкістю. Вирівнювання швидкостей потрібно з тієї причини, що при накопиченні швидкості частинки можуть розганятися занадто сильно. Крім того, таке обмеження створює більш плавний рух.

В ході реалізації алгоритму Boids принцип роботи даних правил і всього алгоритму був модифікований. Розрахунок координат і швидкостей був замінений на знаходження імпульсу тіла для фізичної бібліотеки Box2D.

Також Іпульс згуртованості збільшується в міру віддалення боіда від центру мас. Імпульс поділу також збільшується в міру наближення сусідніх частинок.

Крім того, вектор спрямований від перешкод складається з напрямком до мети. Після чого вектор нормується і множиться на фіксовану швидкість боіда. Це дозволяє уникати перешкод не зменшуючи швидкості частинок.

Також вирівнювання швидкостей було замінено на обмеження

верхньої межі швидкості. Це пов'язано з тим, що частинки часто стикаються з перешкодами і через це сильно обмежується загальна швидкість групи боідов.

Таким чином модифікований алгоритм Boids можна описати наступними пунктами:

− установка випадкових початкових координат для кожного боіда;

− знаходження вектора згуртованості;

− знаходження вектора поділу;

− знаходження вектора спрямованого в зворотну сторону від перешкод;

− знаходження вектора спрямованого до цільової точці;

− знаходження сумарного вектора напрямку і застосування імпульсу до тіла;

− обмеження швидкості за максимальним значенням.

Блок-схему модифікованого алгоритму Boids можна представити таким чином (рисунок 4.5):

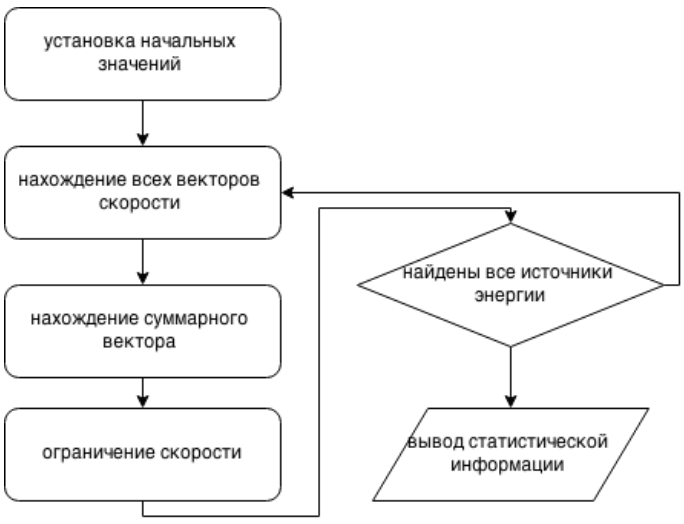

Рисунок 4.5 – Блок-схема модифікованого алгоритму Boids

4.3 Опис роботи системи і оцінка результатів

Для написання програмного додатка використовувалася фізична бібліотека Box2D і візуальне середовище розробки Visual Studio.

Обрані технології дозволили реалізувати алгоритм ройового інтелекту Boids, а також середовище для взаємодії і координації роботів.

Середовище взаємодії боідов виглядає у вигляді обмеженого простору, яке містить в собі точки енергії і перешкоди (рисунок 4.6).

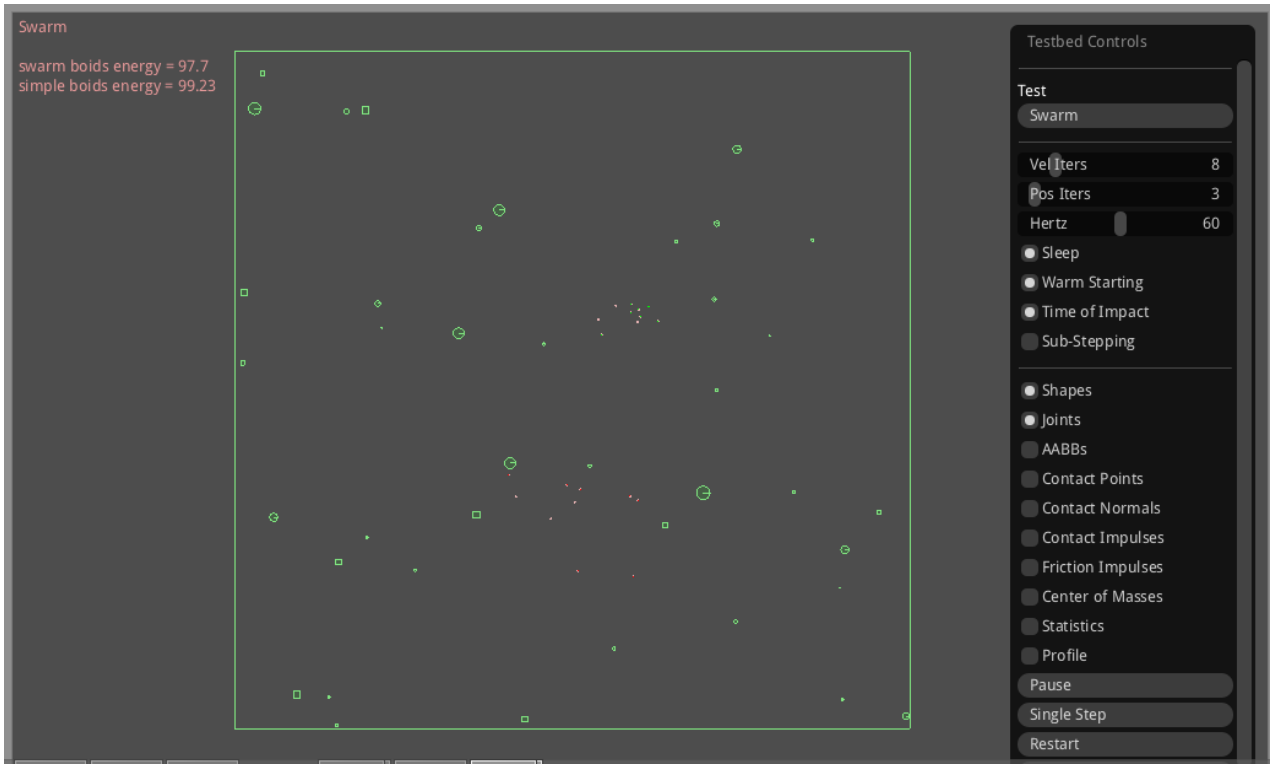

Рисунок 4.6 – Середовище взаємодії роботів, побудована за допомогою бібліотеки Box2D

На початковому етапі симуляції дві групи боідов з'являються у випадковому місці (рис. 4.7).

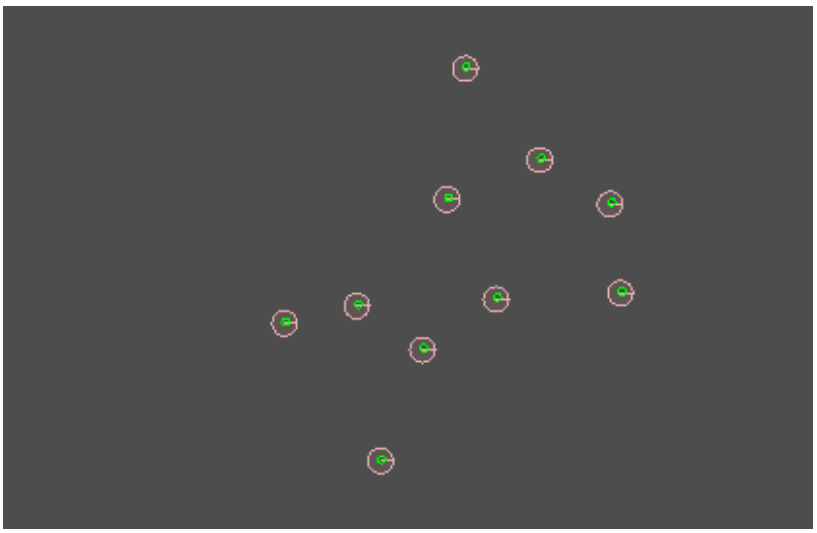

Рисунок 4.7 – Початкова ініціалізація положення боідов

При виявленні енергетичної точки група боідов наближається до неї і збирає енергію до тих пір, поки енергія не закінчиться (рисунок 4.8).

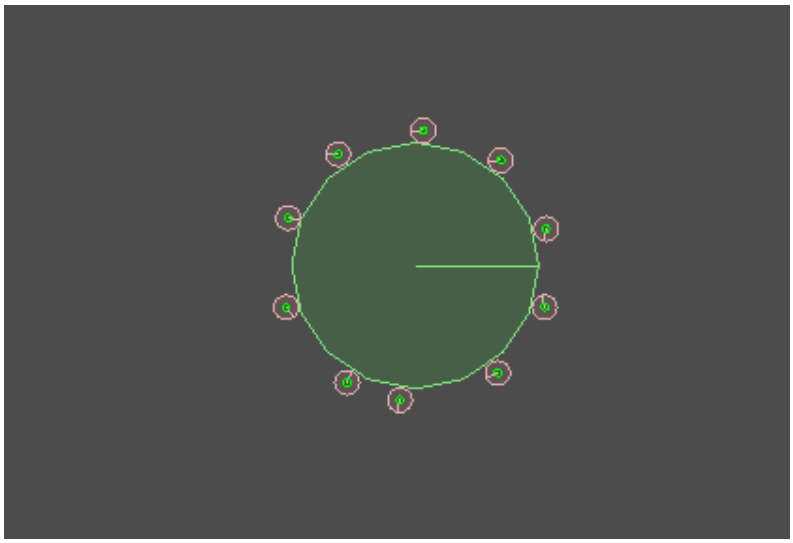

Рисунок 4.8 – Виявлення та збір енергії

При зустрічі з перешкодами боіди огинають його по контуру і після знову з'єднуються в групу. При цьому вектор, спрямований від перешкоди, підсумовується з вектором спрямованим на цільову точку (рисунок 4.9).

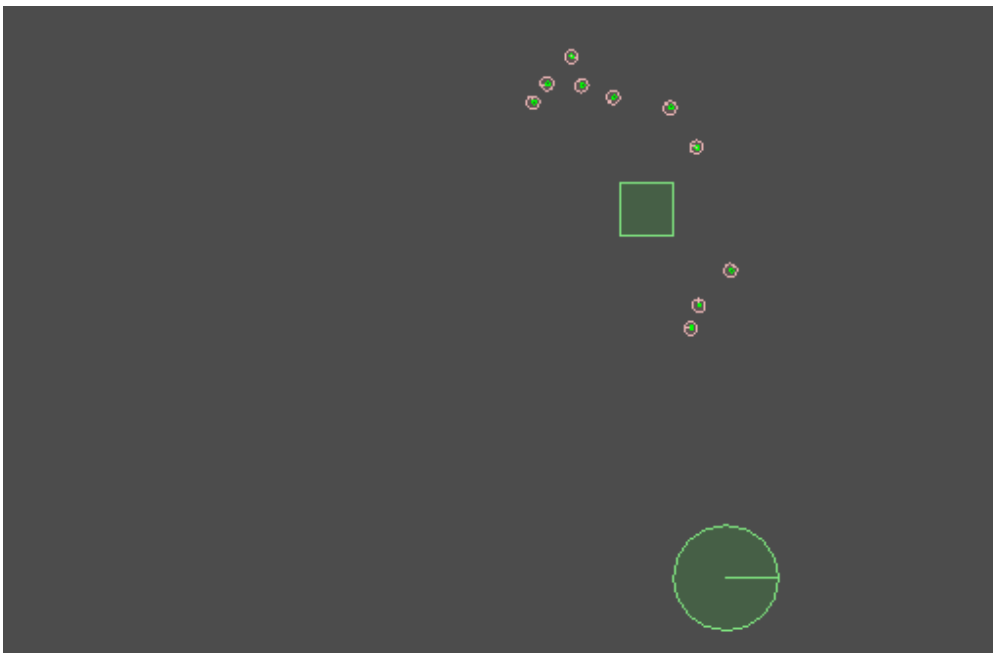

Рисунок 4.9 – Уникнення перешкоди

При зустрічі з противником група боідов атакує ворога, відбираючи в нього енергію. Робот вважається відключеним, коли у нього повністю закінчується енергія.Коли у робота закінчується енергія, він перестає рухатися, збирати енергію і оборонятися (рисунок 4.10).

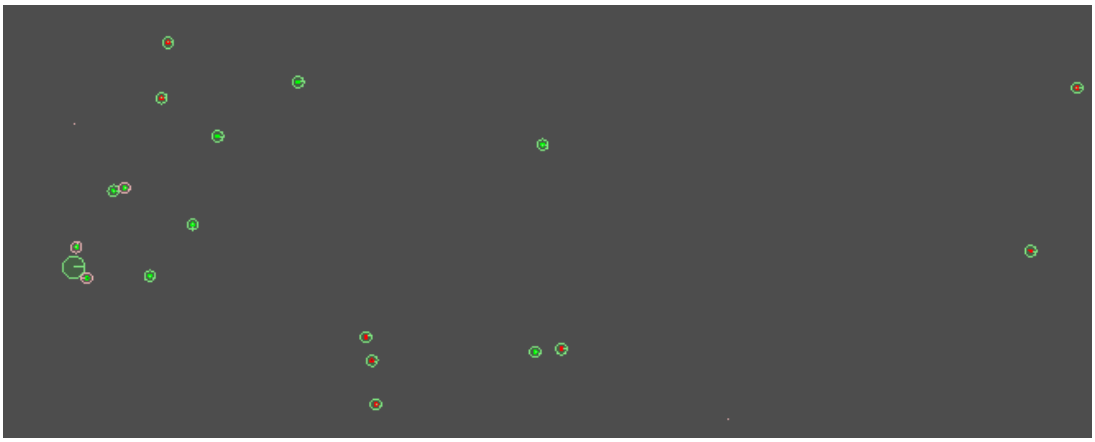

Рисунок 4.10 – Зіткнення двох груп боідів

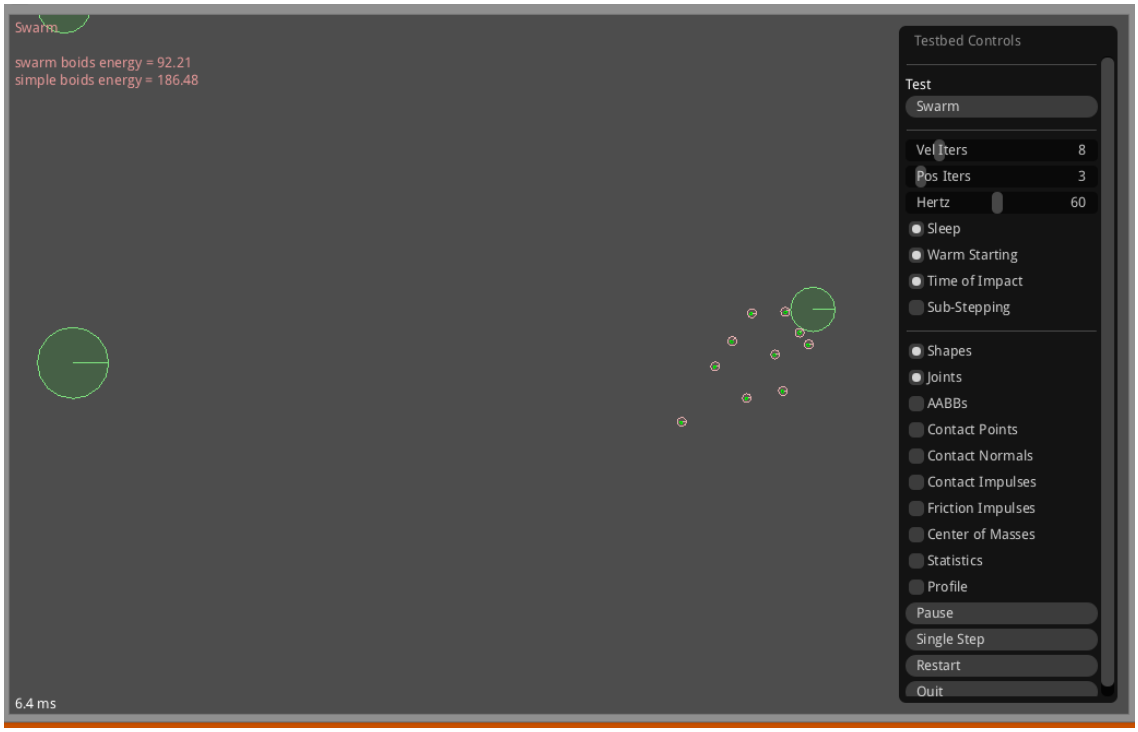

Рисунок 4.11 – Скріншот середовища (виявлення енергії групою боідов)

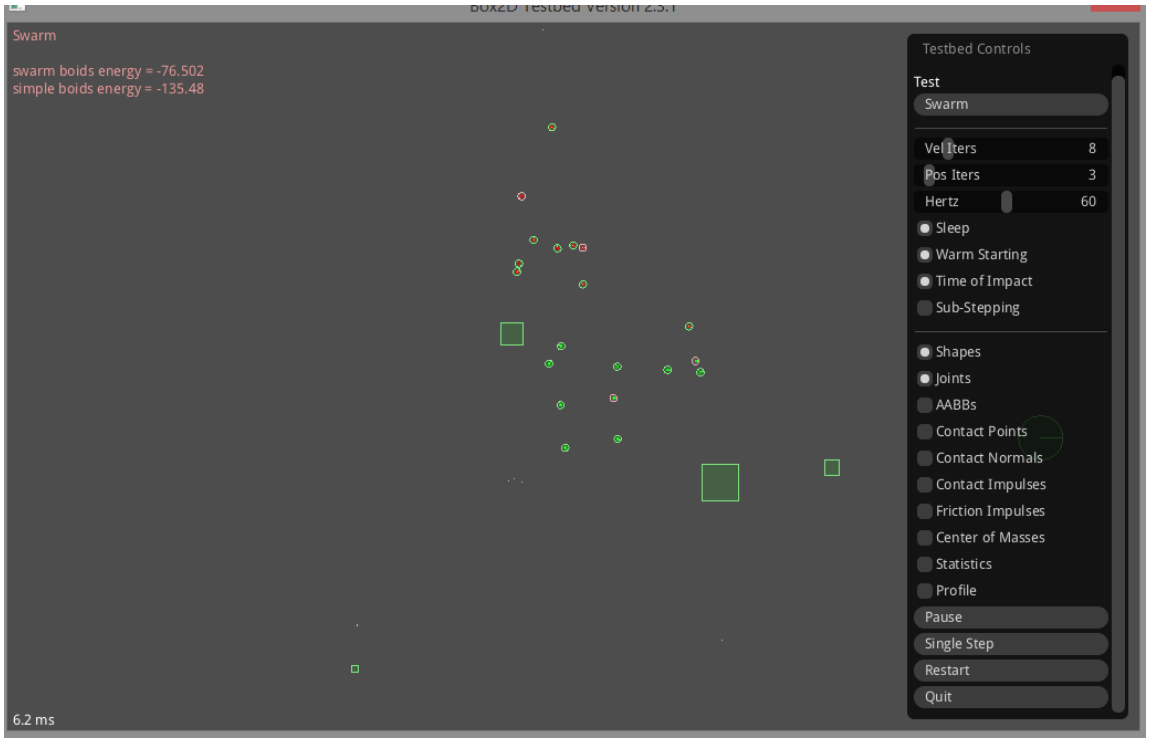

Рисунок 4.12 – Скріншот середовища (зіткнення двох груп боідов при русі до точки енергії)

Для оцінки ефективності роботи ройового інтелекту і колективної взаємодії друга група працювала без розрахунку векторів згуртованості і поділу. Це дозволило групі діяти індівіндуально і розділятися при знаходженні спірних цільових точок. Крім того, дана група роботів не враховує можливість випадкового ураження союзників.

В якості оціночних параметрів були обрані кількість перемог і сумарне значення зібраної енергії.

При проведенні симуляцій помічено, що завдяки розділенню друга група боідів, яка працювала без колективної взаємодії, на початковому етапі швидше збирала енергію. Але при зустрічах з першою групою програвала в зіткненнях і після чого втрачала або всю групу, або не могла перегнати групу, що діяла як рій, за кількістю зібраної енергії.

З проведених 20 симуляцій перша група знищила другу 14 раз, тобто відсоток перемог становить – 70%. Крім того відношення зібраної енергії становить 6496 до 860, тобто – 88%. Статистичні результати роботи програми представлені в таблиці 4.1.

Таблиця 4.1 – Статистичні результати роботи програми

№	Група 1		Група 2	
	результат	енергія	результат	енергія
1	2	3	4	5
1	перемога	218	поразка	-217
2	перемога	784	поразку	-89
3	поразка	-56	перемога	279
4	перемога	516	поразка	-76
5	поразка	-24	перемога	890
6	перемога	329	поразка	-113
7	поразка	-17	перемога	456
8	перемога	333	поразка	-88
9	перемога	279	поразка	-160
10	перемога	672	поразка	-13
11	перемога	611	поразка	-198
12	поразка	-21	перемога	234
13	перемога	998	поразка	-71
14	перемога	247	поразка	-142

Продовження таблиці 4.1

1	2	3	4	5
15	поразка	-110	перемога	345
16	перемога	399	поразка	-132
17	поразка	-46	перемога	64
18	перемога	536	поразка	-11
19	перемога	712	поразка	-37
20	перемога	136	поразка	-61
сума	14 перемог	6496 енергії	6 перемог	860 енергії

За результатами роботи програми можна прийти до висновку, що колективна взаємодія має більше переваг, ніж недоліків в порівнянні з індівіндуальними діями. Також повністю підтверджується вибір алгоритму Boids.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи проведено дослідження методів використання ройового інтелекту в робототехніці для вирішення завдань координації та комунікації роботів. Також розроблено програмний додаток, що реалізує один з алгоритмів ройового інтелекту.

Для наочності створено середовище, що симулює взаємодію роботів в різноманітних ситуаціях. Перед кожним роботом ставиться завдання пошуку енергії, її транспортування, а також захист від зовнішніх впливів у вигляді груп інших боідів.

В роботі проведена оцінка ефективності колективної взаємодії в порівнянні з індивідуальними діями. В якості оцінюваних параметрів обрані кількість знайдених джерел енергії і кількість роботів в кожній групі.

Для виконання поставленого завдання проведено дослідження існуючих алгоритмів ройового інтелекту, таких як алгоритм рою частинок, мурашиний алгоритм, бджолиний алгоритм і багато інших.

Оцінивши такі параметри як ефективність, універсальність, адаптація до нових ситуацій, а також простота реалізації і швидкість роботи алгоритму, прийнято рішення, що найбільш повно цим критеріям відповідає алгоритм Boids.

Незважаючи на те, що даний алгоритм один з перших алгоритмів в області ройового інтелекту, він досить простий у вивченні і реалізації. Алгоритм Boids ефективний і має задовільну швидкість роботи. Також даний алгоритм відмінно підходить для завдання координації і взаємодії роботів, так як в першу чергу розроблявся з метою візуалізації руху зграй птахів.

Для розробки середовища взаємодії боідів вирішено використовувати бібліотеку Box2D, яка дозволяє реалізувати фізичну і графічну складові симулятора. Для написання безпосередньо програмного

додатка використовувалася фізична бібліотека Box2D і візуальне середовище розробки Visual Studio.

Для оцінки ефективності роботи ройового інтелекту і колективної взаємодії роботи були розділені на 2 групи, друга група працювала без розрахунку векторів спільного руху. Це дозволило групі діяти індівіндуально і розділятися при знаходженні спірних цільових точок. Крім того, дана група роботів не враховувала можливість випадкового ураження союзників.

При проведенні симуляцій помічено, що завдяки розділенню друга група боідів, що працює без колективної взаємодії, на початковому етапі швидше збирала енергію. Але при зустрічах з першою групою програвала в зіткненнях і не могла перегнати боідів, що діють як рій, за кількістю зібраної енергії.

З проведених 20 симуляцій перша група знищила другу 14 раз, тобто відсоток перемог становить – 70%. Крім того, відношення зібраної енергії становить 6496 до 860, тобто – 88%.

За результатами роботи програми можна прийти до висновку, що колективна взаємодія має більше переваг, ніж недоліків в порівнянні з індівіндуальними діями. Також повністю підтверджується вибір алгоритму Boids.

За підсумками виконаної магістерської кваліфікаційної роботи встановлено, що такі характеристики ройових алгоритмів, як можливість самоорганізації, відсутність централізованої системи управління і порівняльна простота реалізації, роблять їх незамінними при вирішенні завдань оптимізації та візуалізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Суботін С.А., Олійник А.А., Олійник О.А. Неітеративному, еволюційні та мультиагентні методи синтезу нечеткологічних і нейромережевих моделей: монографія. Запоріжжя: ЗНТУ, 2009. 375 с.

2. Уолднер Ж.Б. Нанокомп'ютер і ройовий інтелект. Лондон: ISTE, 2007. С. 242–248.

3. Кажаров А.А., Курейчик В.М. Про деякі модифікаціях мурашиного алгоритму. *Известия Південного федерального університету*. 2008. С. 7–12.

4. Кажаров А.А., Курейчик В.М. Використання шаблонних рішень в мурашиних алгоритмах. *Известия Південного федерального університету*. 2008. С. 17–22.

5. Міллер П.Д. Ройовий інтелект: Мурахи, бджоли і птахи здатні багато чому нас навчити. М .: Фактор, 2007. С. 88–107.

6. Reynolds C.W. Flocks, Herds, and Schools: A Distributed Behavioral Model. London, 1987. P. 25–34.

7. Kennedy J.K., Eberhart R.C. Particle swarm optimization. IEEE International Conference on Neural Networks. 1995. P. 1942–1948.

8. Kennedy J.K., Eberhart R.C. A new optimizer using particle swarm theory. Sixth International Symposium. 1995. P. 39–43.

9. Heppner F.B., Grenander U.A. A stochastic nonlinear model for coordinated bird flocks. The Ubiquity of Chaos. 1990. P. 233–238.

10. Shi Y.N., Eberhart R.C. A modified particle swarm optimizer. IEEE International Conference. 1998. P. 69–73.

11. Shi Y.N., Eberhart R.C. Empirical study of particle swarm optimization. Proceedings of the 1999 IEEE Congress. 1999. P. 1945–1950.

12. Dorigo M.T. Ant Algorithms for Discrete Optimization. Artificial Life. 1999. P. 137–172.

13. Bonabeau E.M., Dorigo M.T. Swarm Intelligence: From Natural to

Artificial Systems. Oxford University Press. 1999. ISBN 0-19-513159-2

14. Clerc M.K., Kennedy J.K. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions. 2002. P. 58–73.

15. Mendes R.R., Kennedy J.K. The fully informed particle swarm: Simpler, maybe better. IEEE Transactions. 2004. P. 204–210.

16 Dorigo M.T., Bonabeau E.M. Optimization, Learning and Natural Algorithms. Politecnico di Milano. Italy, 1992. 348 p.

17. Farley BF, Clark W.K. Simulation of Self-Organizing Systems by Digital Computer. IRE Transactions on Information Theory. 2003. P. 76–84.

18. Ясницький Л.Н. Введення в штучний інтелект. М.: Изд. Академія, 2005. 29 с.

19. Уотермен Д.К. Посібник з експертним системам : Пер. з англ. під ред. В. Л. Стефанюка. М.: Мир, 1989. 388 с.

20. Марк Р.Р. Нанотехнології: просте пояснення черговий геніальної ідеї. М.: Вільямс, 2006. 240 с.

21. Штовба С.Д. Мурашині алгоритми. Exponenta Pro. 2004. С. 73–92.

22. Кажаров А.А., Курейчик В.М. Мурашині алгоритми для вирішення транспортних завдань. Известия Російської академії наук. 2010. С. 32–45.