Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

Кафедра прикладної математики

Курсова робота на тему:

“Мультиагентний застосунок”

Студента 4-ого курсу групи ПМП-42

напрямку підготовки прикладна математика

Демчишина М.В.

Керівник: кандидат фізико-математичних наук,

доцент Ящук Ю.О.

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_\_\_\_\_\_\_\_ Оцінка ECTS:\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Львів 2021

Зміст

Вступ………………………………………………………………………...3

Розділ 1.Постановка задачі та проблеми………………………………….4

Постановка задачі

Проблеми

Основна концепція та зміст

Розділ 2. Реалізація проекту………………………………………………18

Опис

Порівняння топологічної оптимізації реалізованої на Matlab і на Python

Список використаної літератури………………………………………….24

**Вступ**

Багатоагентна система — це система, утворена декількома взаємодіючими інтелектуальними агентами. Багатоагентні системи можуть бути використані для розв'язання таких проблем, які складно або неможливо вирішити за допомогою одного агента або монолітної системи. Прикладами таких завдань є онлайн-торгівля, ліквідація надзвичайних ситуацій, і моделювання соціальних структур, самостійне переміщення транспортних засобів.

Головна перевага мультиагентних систем — це гнучкість. Багатоагентна система може бути доповнена й модифікована без переписування значної частини програми. Також ці системи мають здатність до самовідновлення й мають стійкість до збоїв, завдяки достатньому запасу компонентів і самоорганізації.

## Застосування мультиагентних систем

## Багатоагентні системи застосовуються в нашому житті в графічних застосунках, наприклад, в [комп'ютерних іграх](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0). Агентні системи також були використані у фільмах. Теорія мультиагентних систем використовується в складених системах оборони. Також мультиагентні системи застосовуються в транспорті, логістиці, графіці, [геоінформаційних системах](https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) і багатьох інших. Багатоагентні системи добре зарекомендували себе в сфері мережних і мобільних технологій, для забезпечення автоматичного й динамічного балансу завантаженості, розширюваності й здатності до самовідновлення.

## На сьогодні багатоагентні системи є одним з найважливіших напрямків досліджень та розробок в області інформаційних технологій та штучного інтелекту. Багатоагентна система складається з декількох взаємодіючих програмних компонентів – агентів, які здатні співпрацювати між собою для вирішення проблем, які не залежать від можливостей будь-якого окремого агента. Багатоагентні системи важливі, перш за все, тому що вони, як виявилося, мають дуже широке застосування в різних сферах, таких як контроль промислового процесу, електронна комерція, управління ресурсами, диспетчеризація, біотехнології та медицина, робототехніка, багатоагентна соціологія та багато інших. Вивчення даної дисципліни починається з введення в поняття агента і має привести до розуміння студентом того, що таке агенти, як вони можуть бути побудовані, як можна створити агентів, які ефективно співпрацюють один з одним для вирішення певної проблеми, а також підходи до автономного прийняття рішень агентами в мультиагентному контексті.

З початку винайдення парового двигуна, а згодом і електроенергії, життя людей почало стрімко змінюватись у бік автоматизації праці та зменшення кількості робочих годин. «Автоматичний» означає не лише те, що робот має батарею, хорошу силу обчислювання та гарні «поведінкові» звички, а набагато більше. автономність – це не просто поєднання змістовних правил, це можливість створювати свої власні правила. Пересуватися, прокласти шлях або поприбирати – це усе дуже просто для нас, людей, та багатьох тварин. Різниця між роботами та тваринами полягає в тому, що тварини використовують мозок для того, щоб вирішити задачу, і ми починаємо розуміти складність цих «простих» завдань, коли аналізуємо складність мозку . За останні кілька десятків років дуже активно почали застосовуватись різні підходи зі сфери комп’ютерних наук, особливо це актуально для сфери робототехніки. Ця сфера потребує розумних рішень, для того, щоб адаптуватись до зовнішніх умов, за рахунок форми, сенсорів, датчиків та ефективних алгоритмів. розробка робота є багатоетапним процесом, який потребує застосування знань з багатьох сфер діяльності, зокрема навігації, сканування середовища, алгоритмів пересування, інженерної частини тощо.

Зараз є активними дослідження в галузі автоматичного керування автомобілем, без допомоги людини. Тому я вирішив реалізувати симуляцію руху автомобілів на дорозі. Де кожен автомобіль – це незалежний агент, який ніким не керується, і виконує своє завдання так, щоб не завадити іншому агенту. Тобто, переміщуватись по дорозі без створення аварійних ситуації.

**Постановка задачі**

У багатоагентній системі агенти мають кілька важливих характеристик:

* Автономність: агенти, хоча б частково, [незалежні](https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%BD%D0%B8%D0%B9_%D0%B0%D0%B3%D0%B5%D0%BD%D1%82)
* Обмеженість уявлення: у жодного з агентів немає уявлення про всю систему, або система занадто складна, щоб знання про неї мало практичне застосування для агента.
* [Децентралізація](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%86%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F): немає агентів, що керують усією системою

Завданням цієї курсової роботи є:

* Створити симульоване 3D середовище з правильним розрахунком взаємодій між агентами.

• Створити модель дороги та автомобілів, що будуть рухатись по дорозі.

• Під час їзди, автомобілі мають прокладати оптимальний маршрут до довільної точки на місцевості.

• Створення мінімального графічного інтерфейсу.

**Проблеми**

1. знання, бажання й наміри ([BDI](https://uk.wikipedia.org/w/index.php?title=BDI_software_agent&action=edit&redlink=1)),
2. кооперація й координація,
3. організація,
4. комунікація,
5. узгодження,
6. розподілене рішення,
7. [Кооперативне розподілене розв'язання задач](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B5_%D1%80%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D0%B5_%D1%80%D0%BE%D0%B7%D0%B2%27%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87),
8. мультиагентне навчання
9. надійність і стійкість до збоїв

**Мета**

Метою цієї курсової роботи є дослідження та впровадження ефективних алгоритмів пересування віртуальних транспортних засобі, їх навігації в реальному часі у симульованому середовищі. Завданням цієї курсової роботи є створення симуляції пересування, навігації, виявлення перешкод та поведінки моделей у 3D середовищі з обмеженнями, які діють у реальному світі, для отримання максимально достовірних результатів.

Використане програмне забезпечення для виконання поставлених завдань було прийнято рішення використовувати ігровий рушій Unity3D, який підходить не тільки для задач, пов’язаних зі створенням комп’ютерних ігор, а також широко використовується для створення 3D симуляторів в абсолютно різних галузях, починаючи з симулювання робота для подальшого створення реальних моделей на основі отриманих даних, закінчуючи створенням симуляторів самокерованого автомобіля. Скриптова система рушія зроблена на Mono — вільний відкритий проект з реалізації .NET Framework. Для написання логіки було обрано мову C# - це стандартна мова для розробки проектів у середовищі Unity3D, в якості IDE (Integrated Development Environment) – visual studio.

## 

**Дослідження мультиагентних систем**

Відмінною особливістю сучасних мультиагентних систем є перехід від локалізованого до розподіленого типу. Загальноприйнятого визначення «агента» ще не існує, а найбільш визнаним є визначення агента як програмної системи, яка має такі особливості: автономність, взаємодія, мобільність, реактивність, активність, індивідуальність бачення «світу», комунікабельність і кооперативность, інтелектуальність поведінки. Кожен агент - це процес, який володіє достатньою частиною знань про об'єкт і можливістю обмінюватися цими знаннями з іншими агентами. З точки зору об'єктно-орієнтованого підходу агент можна розглядати як комплекс функцій в сукупності з інтерфейсом, який здатний посилати відповіді і отримувати питання. Також можна визначити агента як комп'ютерну програму, яка виконується асинхронно відповідно до поведінкою, закладеним в неї конкретною особистістю або організацією. На сьогодні існує два основних типи агентів: стаціонарні та мобільні.

Основою архітектури агента є контекст, або серверна середовище, в якому він виконується. Один і той же сервер може підтримувати декілька контекстів, кожен з яких має власний набір захисних обмежень. Кожен агент має постійний ідентифікатор - ім'я. Агенти можуть працювати спільно або обмінюватися інформацією в агентних системах, місцях і регіонах, надсилаючи та приймаючи повідомлення.

Для передачі повідомлень між агентами або між агентом і користувачем існують посередники. Вони приховують місцезнаходження агента для попередження доступу до його коду.

Базовим інструментом розробки інтелектуальних багатоагентних систем, що дозволяє створювати, знищувати, інтерпретувати, запускати і переміщати агентів є агентна платформа. Агентна платформа − це програмне середовище, що реалізує основні механізми, які забезпечують роботу багатоагентних систем і полегшує розробку агентних систем та агентів взагалі. Основними функціями агентних платформ є:

– організація взаємодії агентів;

– передача повідомлень між агентами всередині платформи (на різних рівнях: на рівні мережевих пакетів, повідомлень будь-якої мови реалізації, протоколів передачі) і між різними платформами;

– підтримка онтологій;

– управління агентами, їх життєвими циклами;

– пошук агентів і даних про них всередині системи;

– забезпечення безпеки агентів.

**Найбільш відомі такі агентні платформи:**

– JADE (Java Agent Development Framework);

– Coguaar (Cognitive Agent Architecture) – найпотужніший проект, підтримується DARPA (Defense Advanced Research Projects Agency – агентство передових оборонних дослідницьких проектів);

– Aglobe – володіє недостатньою підтримкою FIPA (Foundation for Intelligent Physical Agents) – організації стандарту IEEE, яка сприяє розвитку технології на основі агентів і взаємодії їх з іншими технологіями;

– Jason (multi-agent systems development platform) – для розробки багатоагентних систем, має багато функцій, що можна налаштовувати;

– Jack − одна з небагатьох платформ, де використовуються модель логіки агента, що заснована на принципах переконання-бажання-наміри (Beliefdesire-intention software model – BDI) і має вбудовані формально-логічні планування роботи агента.

Крім агентних платформ для програмування агентів можуть застосовуватися:

– універсальні мови (Java, C++, Visual Basic, C#);

– мови представлення знань (SL, KIF);

– мови комунікацій та обміну знаннями (KQML, AgentSpeak, April);

– мови сценаріїв (TCL/TK, Python, Perl 5);

– спеціалізовані мови (TeleScript, COOL, Agent0, AgentK);

– символьні мови та мови логічного програмування (Oz, ConGolog, IMPACT, Dylog, Concurrent METATEM);

– а також інші мови і засоби розробки агентів.

## Для виконання даної курсової роботи я обрав мову програмування C#. Оскільки дана мова програмування активно розвивається, є повністю об’єкто-орієнтованою. Це дозволило зменшити розміри програми порівняно з іншими аналогами і об’єднати функції, які необхідно для роботи агентів, у відповідні класи, розділені по спеціалізації. А також зробити програму зрозумілішою і легшою для майбутньої розробки і підтримки. C# надає нам більші можливості для написання застосунку порівняно із іншими агентно-орієнтованими мовами програмування. Завдяки цьому ми можемо створити кращий дизайн програми, ніж він міг би бути у агентно-орієнтованих мовах.

## Обґрунтування використання імітаційної моделі

Імітаційна модель – це математична бізнес-модель, що поєднує в собі як математичні, так і логічні поняття, які виконують симуляцію системи реального життя за допомогою програмного забезпечення.

Імітаційне моделювання покликане вирішувати проблеми реального світу, використовуючи при цьому мінімум затрат, роблячи це швидко і безпечно, оскільки воно не потребує фізичного втілення рішень на практиці. Це зручний, наглядний, простий для розуміння та перевірки інструмент для аналізу. Імітаційне моделювання допомагає знаходити оптимальні рішення у різних галузях бізнесу та науки і дає чітке розуміння системи в загальному та рівня її складності.

Реальні системи піддаються впливу різних випадкових подій, тому поводження системи краще та ефективніше досліджувати саме із використанням імітаційних моделей.

Умови, при яких потрібно використовувати імітаційні моделі:

* якщо математичні процедури аналітичних методів складні і ресурсномісткі;
* якщо протягом деякого часу необхідно спостерігати змінами деяких компонентів системи;
* якщо необхідно контролювати процеси в системі, сповільнюючи чи пришвидшуючи їх;
* якщо йде процес дослідження об’єкта моделювання;
* якщо особливе місце має послідовність подій;
* якщо імітаційне моделювання є єдиним способом дослідження системи через неможливість проведення реальних експериментів.

Однією з основних переваг використання імітаційних моделей є те, що вони базуються на достовірних даних, отриманих від користувачів при розробці реальних систем. Це дозволяє інженерам визначати правильність та ефективність конструкції, перш ніж система фактично буде побудована. Таким чином, інженер може дослідити переваги альтернативних конструкцій без фізичної побудови системи та те, як конкретні інженерні рішення впливають на систему в цілому. Даний підхід дозволяє виявляти неефективні рішення на етапі проектування, а не на етапі будівництва, що дозволяє зменшити витрати на побудову системи.

Таким чином, імітаційна модель має наступні переваги:

* представляє моделюючу систему більш адекватно, ніж інші види моделей;
* забезпечує гнучкість структури та параметрів системи;
* дозволяє досліджувати складні та трудомісткі практичні задачі, коли їх неможливо вирішити за допомогою математичних методів;
* зменшує час тривалості досліджень у порівнянні з реальними дослідженнями;
* дає можливість досліджувати особливості функціонування реальної системи у різних умовах з урахуванням випадкових подій;
* є практично реалізованим методом для дослідження складних систем;
* допомагає економити ресурси та фінанси;
* дозволяє включати до моделювання результати досліджень реальних систем;
* допомагає виявити непередбачувану поведінку системи.

## Огляд моделей імітації транспортного руху

Модель передбачає, що водій може знаходитися у одному із 4 станів:

а) *Вільний рух*. Кожний водій має бажану швидкість, яку він намагається досягнути і дотримуватися впродовж усього руху. В реальних умовах водій не може притримуватися постійної швидкості.

б) *Наближення*. Процес зближення транспортного засобу із транспортним засобом, що йде попереду. Водій автомобіля починає гальмування для того, щоб уникнути зіткнення із автомобілем попереду. Різниця швидкостей двох автомобілів буде дорівнювати нулю, коли водій наблизиться до автомобіля попереду на безпечну для нього відстань.

в) *Слідування*. Водій слідує за транспортним засобом, що рухається попереду нього. При цьому він не використовує прискорення чи гальмування, а притримується безпечної для себе відстані. Різниця швидкостей автомобілів коливається в межах нуля, але не дорівнює нулю через недосконалість органів керування.

г) *Гальмування*. Коли дистанція між автомобілями стає меншою, ніж безпечна відстань, водій застосовує середнє або сильне гальмування для збереження дистанції

Водій переходить із одного стану в інший у той момент, коли він досягає деякого бар’єру, який може бути описаний як комбінація різниць швидкості та відстані.

Алгоритм імітації тривимірного транспортного руху представлений наступними кроками:

***Крок 1.***Визначаємо чи є попереду водія транспортний засіб.

***Крок 1.1.*** Якщо, зменшуємо швидкість транспортного засобу, щоб уникнути зіткнення.

***Крок 1.2.*** Якщоце зустрічний транспортний засіб то повертаємось назад, щоб дати змогу проїхати.

***Крок 2.*** Якщо транспортний засіб знаходиться у зоні дії світлофору, то переходимо до наступного кроку 2.1. В іншому випадку переходимо на крок 3.

***Крок 2.1.*** Якщо сигнал світлофору забороняє рух, зменшуємо швидкість до повної зупинки авто. При сигналі, який дозволяє рух, переходимо до кроку 2.3.

***Крок 2.2.*** Чекаємо на зміну регулюючого сигналу, який дозволить продовжити рух.

***Крок 2.3.*** Коли сигнал дозволяє рух, продовжуємо рух та збільшуємо швидкість.

***Крок 3.*** Визначаємо наявність перехрестя попереду транспортного засобу.

***Крок 3.1.*** Якщо попереду знаходиться перехрестя та є необхідність надати перевагу, то зупиняємо транспортний засіб.

***Крок 3.2.*** Після завершення маневру продовжуємо рух.

**Опис технологій**

Unity - це кросплатформовий ігровий рушій. Програма-редактор Unity працює на [Windows](https://uk.wikipedia.org/wiki/Microsoft_Windows), [macOS](https://uk.wikipedia.org/wiki/MacOS" \o "MacOS) і [Linux](https://uk.wikipedia.org/wiki/Linux" \o "Linux), а сам рушій може запускатися на 25 платормах.

Редактор Unity має інтерфейс, що складається з різних вікон, які можна розташувати на свій розсуд. Завдяки цьому можна проводити налагодження застосунка прямо в редакторі. Головні вікна — це оглядач ресурсів проєкту, інспектор поточного об'єкта, вікно попереднього перегляду, оглядач сцени та оглядач ієрархії ресурсів.

Проєкт в Unity поділяється на сцени (рівні) — окремі файли, що містять свої набори об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як об'єкти-моделі (ландшафт, персонажі, предмети довкілля тощо), так і порожні об'єкти — ті, що не мають моделі, проте задають поведінку інших об'єктів (тригери подій, точки збереження прогресу тощо). Їх дозволяється розташовувати, обертати, масштабувати, застосовувати до них скрипти. В них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково наявний компонент Transform — він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за умовчанням присутній компонент Mesh Renderer, що робить модель видимою. Різні моделі можуть об'єднуватися в набори для швидкого доступу до них. Наприклад, моделі споруд на спільну тему.

Unity підтримує фізику твердих тіл і тканини. У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

У 3D іграх Unity переважно використовує спрайти. Unity здебільшого використовує тривимірні моделі, на які накладаються текстури (зумовлюють вигляд поверхні об'єктів), матеріали (зумовлюють як поверхня реагуватиме на різні фактори) та шейдери (невеликі скрипти, за яким вираховується зміна кольору кожного пікселя згідно заданих параметрів, як-от розсіяння відбитого світла). В обох видах застосовуються системи часток для відображення субстанцій, таких як рідини чи дим.

Unity підтримує стиснення текстур, міпмапінг і різні налаштування роздільності екрана для кожної платформи; забезпечує бамп-мапінг, мапінг відображень, паралакс-мапінг, затінення навколишнього світла у екранному просторі, динамічні тіні за картами тіней, рендер у текстуру та повноекранні ефекти обробки зображення, такі як зернистість, глибина чіткості, розмиття в русі, відблиски віртуальних лінз або ореол навколо джерел світла.

Рендеринг зображення відбувається через віртуальну камеру огляду. В робочій області редактора ігрова сцена може розміщуватися як завгодно, а при рендерингу — так, як її видно з камери. В сцені може бути декілька камер, які рухаються за персонажем чи за вказаною траєкторією. Вигляд з камери подається в двовимірно чи тривимірно (в перспективі або ортографічно). Фон сцени, видимий через камеру, типово зображає небо, утворене скайбоксом, але може презентувати й інше довкілля.

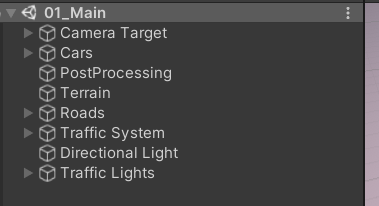
Скриптова система ігрового рушія зроблена на [Mono](https://uk.wikipedia.org/wiki/Mono" \o "Mono) — вільний відкритий проєкт з реалізації [.NET Framework](https://uk.wikipedia.org/wiki/.NET_Framework).

**Mono** — багато-платформове [вільне](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F" \o "Вільне програмне забезпечення) [відкрите](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F" \o "Відкрите програмне забезпечення) втілення системи [.NET](https://uk.wikipedia.org/wiki/.NET), яке відповідає стандартам [ECMA](https://uk.wikipedia.org/wiki/ECMA), включаючи серед іншого і [компілятор](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80" \o "Компілятор) [C#](https://uk.wikipedia.org/wiki/C_Sharp), і [Common Language Runtime](https://uk.wikipedia.org/wiki/Common_Language_Runtime).

Mono включає [компілятор](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) мови [C#](https://uk.wikipedia.org/wiki/C_Sharp) — mcs, середовище виконання [.NET](https://uk.wikipedia.org/wiki/.NET) — mono (із підтримкою [JIT-компіляції](https://uk.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%86%D1%96%D1%8F)) і mint (без підтримки JIT), [зневаджувач](https://uk.wikipedia.org/wiki/%D0%97%D0%BD%D0%B5%D0%B2%D0%B0%D0%B4%D0%B6%D1%83%D0%B2%D0%B0%D1%87" \o "Зневаджувач), і ряд [бібліотек](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Бібліотека (програмування)), включаючи реалізацію [ADO.NET](https://uk.wikipedia.org/wiki/ADO.NET) і [ASP.NET](https://uk.wikipedia.org/wiki/ASP.NET).

Середовище виконання mono може виконувати модулі, написані мовами [C#](https://uk.wikipedia.org/wiki/C_Sharp), [F#](https://uk.wikipedia.org/wiki/F_Sharp), [VisualBasic,.NET](https://uk.wikipedia.org/wiki/Visual_Basic_.NET), [Java](https://uk.wikipedia.org/wiki/Java), [Boo](https://uk.wikipedia.org/wiki/Boo), [Nemerle](https://uk.wikipedia.org/wiki/Nemerle), [Python](https://uk.wikipedia.org/wiki/Python), [Forth](https://uk.wikipedia.org/wiki/Forth), [JavaScript](https://uk.wikipedia.org/wiki/JavaScript),  [PHP](https://uk.wikipedia.org/wiki/PHP) і [Object Pascal](https://uk.wikipedia.org/wiki/Object_Pascal) (за наявності компілятора у середовище .Net/Mono).

**Структура проєкту Unity**

****

**Camera Target:** відповідає за розташування камери і огляд робочої ділянки програми.

**Cars:** описує всі транспортні засоби і їх поведінку у різних передбачених ситуаціях.

**Roads:** описує всі транспортні шляхи і рух по них.

**Traffic Lights:** є представленням фази світлофору. Він встановлює потрібну фазу (зелену чи червону) для світлофору. При потраплянні машини у зону дії світлофору автомодбілі виконують відповідні діх.

**Traffic System:** Ядро логіки системи моделювання тривимірного транспортного руху. Даний клас відповідає за створення вузлів транспортної системи, управління ними та побудову зв’язків між вибраними вузлами мережі.

Вся поведінка об’єктів у різних ситуаціях описана за допомогою скриптів на мові C#.

Усі класи, наслідують базовий клас Unity – MonoBehaviour. MonoBehaviour реалізує метод Start, який використовується для ініціалізації початкових параметрів та об’єктів системи.

**Структура скриптів**

|  |  |  |
| --- | --- | --- |
| Клас | Метод | Опис |
| Wheel | OnEnable | Метод робить початкове налаштування автомобіля |
| Move | Метод описує рух і поведінку автомобіля |
| GetSpeed | Метод повертає швидкість автомобіля в даний момент |
| Waypoint | Refresh | Метод оновлює вигляд точки |
| RemoveCollider | Метод видаляє об’єкт з точки |
| GetVisualPos | Метод повертає поточну позицію |
| VehicleAI | Start | Метод виконує початкове створення об’єкта |
| Update | Метод виконує оновлення об’єкта |
| WaypointChecker | Метод перевіряє и транспортний засіб дойшов до заданої точки маршруту і якщо потрібно додає наступну ціль |
| MoveVehicle | Метод відповідає за рух транспортного засобу і всі його дії |
| GetDetectedObstacles | Метод визначає перешкоди |
| SetWaypointVehicleIsOn | Метод встановлює точки маршруту |
| GetSegmentVehicleIsIn | Метод отримує транспортний засіб на сегменті |
| TrafficSystem | SaveTrafficSystem | Метод виконує збереження системи |
| ResumeTrafficSystem | Метод продовжує виконання системи |
| Segment | IsOnSegment | Метод визначає чиє елемент на сегменті |
| Intersection | Start | Метод створює перехрестя |
| SwitchLights | Метод змінює фазу світлофору |
| OnTriggerEnter | Метод виконується при потраплянні транспортного засобу на перехрестя |
| OnTriggerExit | Метод виконується при покидані транспортним засобом перехрестя |
| TriggerStop | Метод виконується при зупинці транспортного засобу на перехресті |
| TriggerLight | Метод виконується при змінні фази світлофору |
| IsRedLightSegment | Метод перевіряє фазу світлофору |
| MoveVehiclesQueue | Метод рухає транспортні засоби у черзі |

Всі скрипти прив’язані до відповідних об’єктів. Вони спрацьовують при виникненні необхідності. Всі агенти є незалежними і виконують завдання окремо від інших, але так щоб не завадити виконанню завдань інших агентів.

Саме незалежність агентів дозволяє нам точніше імітувати рух транспортних засобів, адже у реальному житті рух кожний транспортний засіб переміщується куди захоче, намагаючись не завадити іншим учасникам руху.

Тому це дозволяє нам створити максимально наближену до реального життя модель, яка дозволить нам передбачити різні ситуації.

//TODO: описати конкретні методи, формули руху, визначення швидкості і тд