МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

Имитационная модель перекрестка со светофором

Курсовая работа

Таборова Льва Алексеевича

студента 2 курса,
специальность 1-31 03 09
Компьютерная математика
и системный анализ

Научный руководитель:
кандидат физ.-мат. наук,
доцент О.А.Лаврова

Минск, 2021

ОГЛАВЛЕНИЕ

Введение 3
1. Обучающийся агент в виртуальной среде 4
1.1. Имитационное моделирование 4
1.2. Знакомство с AnyLogic 5
1.3. Об искусственном интеллекте 6
2. Построение модели и обучение агента 8
2.1. Построение дорожного движения 8
2.2. Обучение с подкреплением 10
2.3. Сравнение интеллектуального агента и оптимизации 13
Заключение 16
Список литературы 17
Приложение А 18

ВВЕДЕНИЕ

Искуственный интеллект на сегодняшний день является одной из самых интересных и востребованных направлений информационных технологий. Специалисты научились обучать и интегрировать системы, основанные на искуственном интеллекте, в различные области повседневной жизни, начиная от предложений наиболее актуальных приложений на смартфоне до автоматизированной поездке на автомобиле, когда бортовой компьютер может сам «видеть» дорогу и управлять транспортным средством.

Однако создание и особенно обучение нейронных сетей на реальных системах очень дорогое, неэффективное по временным затратам, а порой и опасное, когда от системы регулировки движения зависят жизни людей. Для того, чтобы обойти данные ограничения при обучении, применяется имитационное моделирование.

В настоящей работе было изучено одно из возможных реальных применений комбинации ИИ и имитационное модели — оптимизация продолжительности фаз светофора на автомобильном перекрестке. Для этого были изучены такие программные продукты как AnyLogic и Pathmind.

1. ОБУЧАЮЩИЙСЯ АГЕНТ В ВИРТУАЛЬНОЙ СРЕДЕ

1.1. Имитационное моделирование

Часто бывает невозможно использовать настоящие системы для экспериментирования и/или тестирования разработанных решений. В этих случаях прибегают к имитационному моделированию. **Имитационная модель** на достаточном уровне повторяет настоящую моделируемую систему. Имитационная модель позволяет проводить любые эксперименты в любых количествах, а также изучать последствия тех или иных примененных решений. Одним из направлений имитационного моделирования является **агентное моделирование**. Агентное моделирование позволяет изучать сложное взаимодействие автономных агентов в моделируемой среде, чье совокупное поведение и формирует поведение системы. Для моделирования перекрестка была использована программа AnyLogic.

AnyLogic — коммерческий программный продукт, созданный для моделирования процессов, для последующего использования результатов в бизнесе[3]. AnyLogic позволяет работать с такими видами моделирования как дискретно-событийным моделированием, арендным моделированием, а так же позволяет изучать системную динамику. Для построения моделей различных процессов в программе присутствуют различные отраслевые библиотеки. Для моделирования перекрестка мы будем использовать библиотеку дорожного движения — Road Traffic Libriary.

При условии взаимодействия с ИИ для имитационной модели обычно определяют следующие роли:

- Среда для генерации случайных данных
- Среда для обучения агента
- Среда для тестирования агента

Мы реализуем 2й подход — построенная модель будет виртуальной площадкой для обучающегося агента, где он будет обучаться.

1.2. Знакомство с AnyLogic

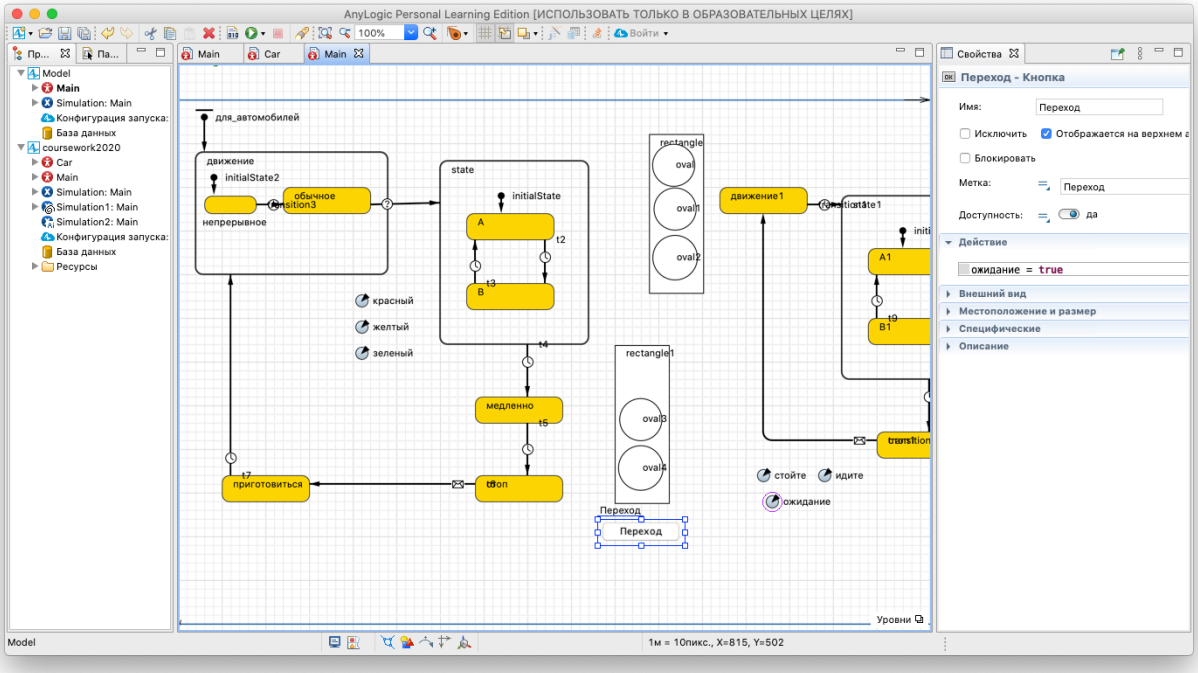

Рисунок 1.1 Интерфейс AnyLogic

В качестве ознакомительного примера для знакомства со средой моделирования, была реализована стейт-чарт модель системы светофоров[5]. Как видно на рисунке 1.1, интерфейс среды моделирования достаточно прост и понятен: имеются рабочая область, окно свойств, окно обзора проектов и окно с палитрой элементов. Для построение Стейт-чарта были использованы элементы палитры из категории диаграмм состояний.

1.3. Об искусственном интеллекте

Для обучения ИИ применяются 3 основных метода:

- Обучение с учителем
- Обучение без учителя
- Обучение с подкреплением

Обучение с учителем справляется с аппроксимацией целевой функции, когда система имеет доступ к «правильным» результатам, и основываясь на них может учиться правильно обрабатывать новую информацию. Обучение с учителем подходит для задач компьютерного зрения, например для распознавания объектов на изображениях. В качестве обучающего набора используются размеченные наборы данных.

Обучение без учителя решает такие задачи как кластеризация объектов по некому критерию. «Правильного» решения здесь нет (набор данных неразмечен), алгоритм сам учиться объединять объекты в группы согласно какой-то логике.

Широк спектр задач, с которыми может справиться обучение с подкреплением. Данный метод позволяет выработать политику решений для системы, основываясь на неком отклике системы. Данный отклик является функцией вознаграждения(просто «вознаграждением»), которая определяется архитектором при построении. Вознаграждение вычисляется после действия системы и определяет «полезность» совершенного действия. Обычно вознаграждение варьируется в промежутке [-1; 1], но всё зависит от задачи. В зависимости от вознаграждения система корректирует своё поведение для дальнейших итераций.

Для обучения агента был использован продукт PathMind который представляет собой web-сервис, использующий модель AnyLogic в качестве виртуальной среды для тестирования вырабатываемой политики. В PathMind

используется Population Based Training[1] для скорейшего нахождения гиперпараметров нейросети и, следовательно, искомой политики.

Для обучения агента в PathMind необходимо определить следующее:

- **Observations** — те данные, которые будут определять состояние системы.
- **Metrics** — данные, определяющие полезность действий агента (в дальнейшем будут входить в функцию полезности).
- **Actions** — действия, которые могут быть совершены агентом.
- **Reward** — функция полезности, или «вознаграждение», которое получает обучаемый агент как отклик на его действие.

Для обучение, модель AnyLogic экспортируется в PathMind, после чего указывается функция полезности и начинается обучение. По завершению обученную политику можно импортировать обратно в модель, после чего управление светофором будет основываться на выработанной политике.

2. ПОСТРОЕНИЕ МОДЕЛИ И ОБУЧЕНИЕ АГЕНТА

2.1. Построение дорожного движения

Для изучения среды AnyLogic и сервиса PathMind был реализован пример, продемонстрированный на презентации компании[2]. Для наглядности, рассматриваемый пример был выбран максимально элементарным.

Дорожная сеть состоит из 4х дорог (рисунок 2.1): северной, восточной, южной и западной. Каждая дорога разбита на 2 зоны согласно направлению движения. Дороги соединены непосредственно перекрестком. Таким образом у нас образуются 9 зон. Их мы будет использовать для оценки состояния системы.

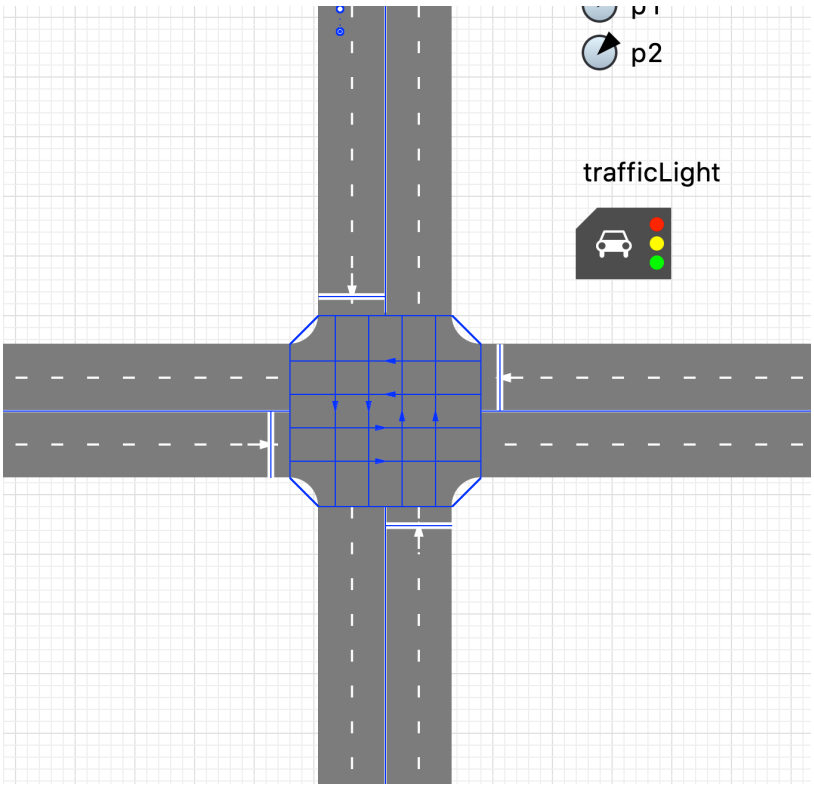

Рисунок 2.1 Модель перекрестка

Для контроля эффективности регулировки движения был пользовательский тип агентов *Car* (рисунок 2.2). Агенты данного типа при проезде перекрестка добавляют свое время проезда к набору данных *timeInModel*. Значения данного набора динамически отображаются на гистограмме (рисунок 2.3).

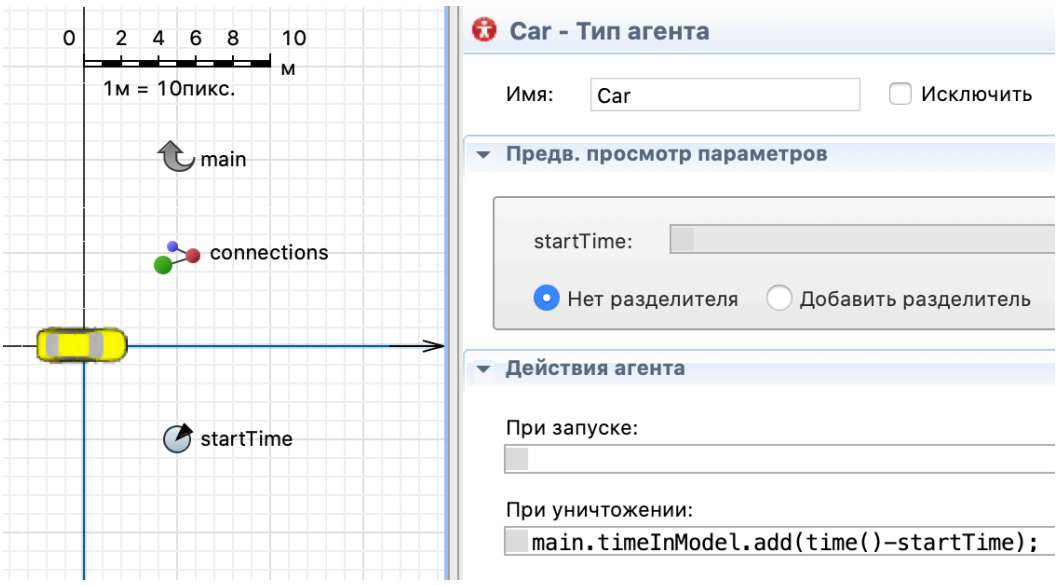

Рисунок 2.2 Пользовательский агент

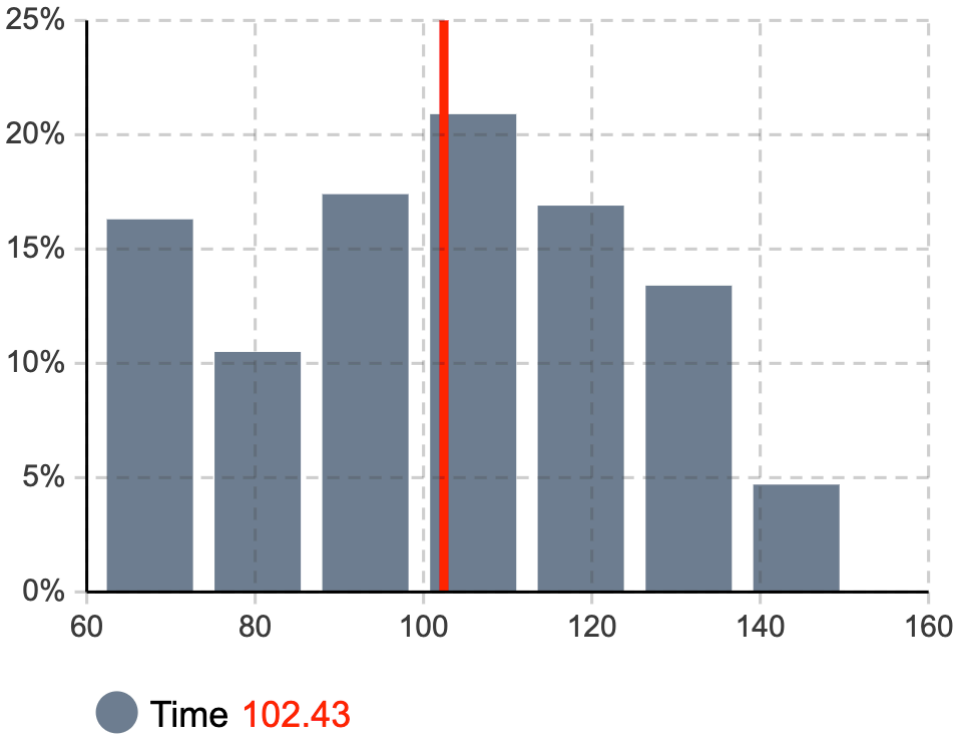

Рисунок 2.3 Временная гистограмма

Для управления движением транспорта была построена схема, представленная на рисунке 2.4. Блоки *carSource* создают новых агентов типа *Car*, и затем блоки *carMoveTo* помещают агентов на дороги 4х направлений.

Для управления движением был добавлен блок *trafficLight* ассоциированный с элементом-перекрестком. У данного блока есть метод,

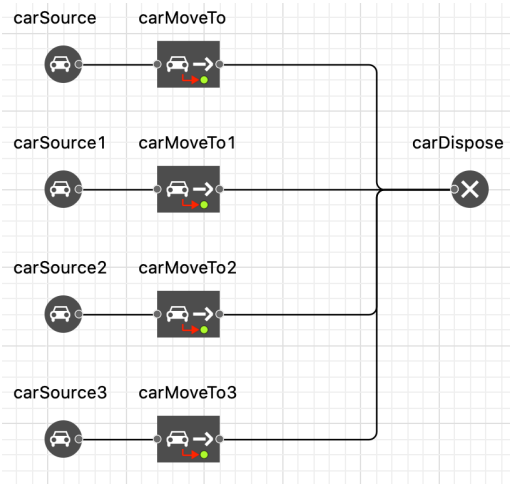

Рисунок2.4 Схема движения транспорта

срабатывающий при переключении фазы, для мониторинга продолжительности зеленой фазы. Данные графически представлены на графике (рисунок 2.5).

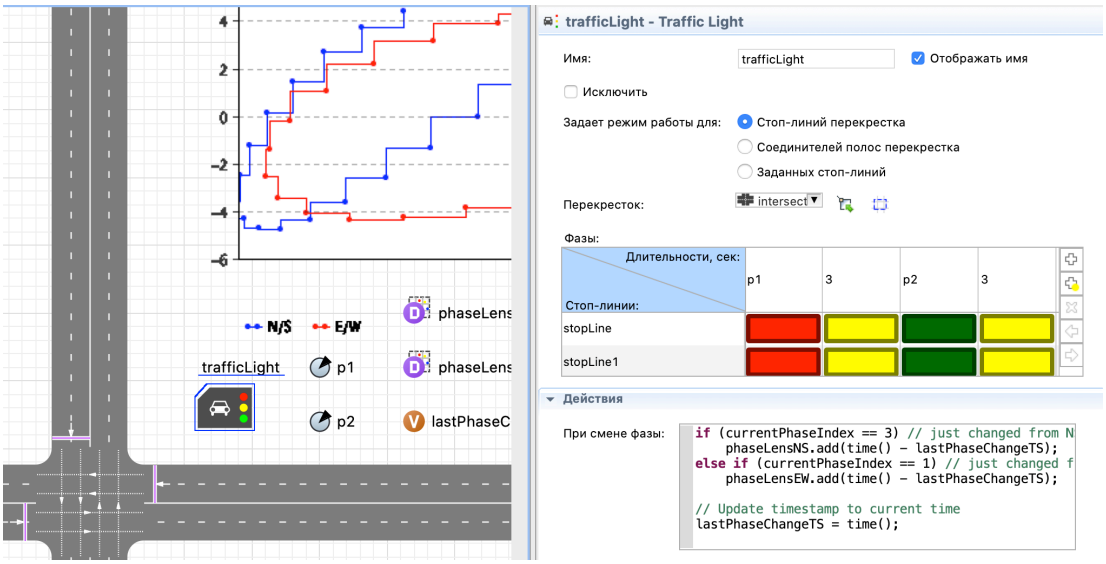

Рисунок 2.5 Светофор

2.2. Обучение с подкреплением

Для использования обучения с подкреплением в модели AnyLogic, был добавлен блок *pathmindHelper*(рисунок 2.6). В предыдущей главе перечислялись необходимые для обучения в сервисе Pathind блоки. Данные блоки программно представлены в виде Java-классов Рассмотрим каждый класс подробнее:

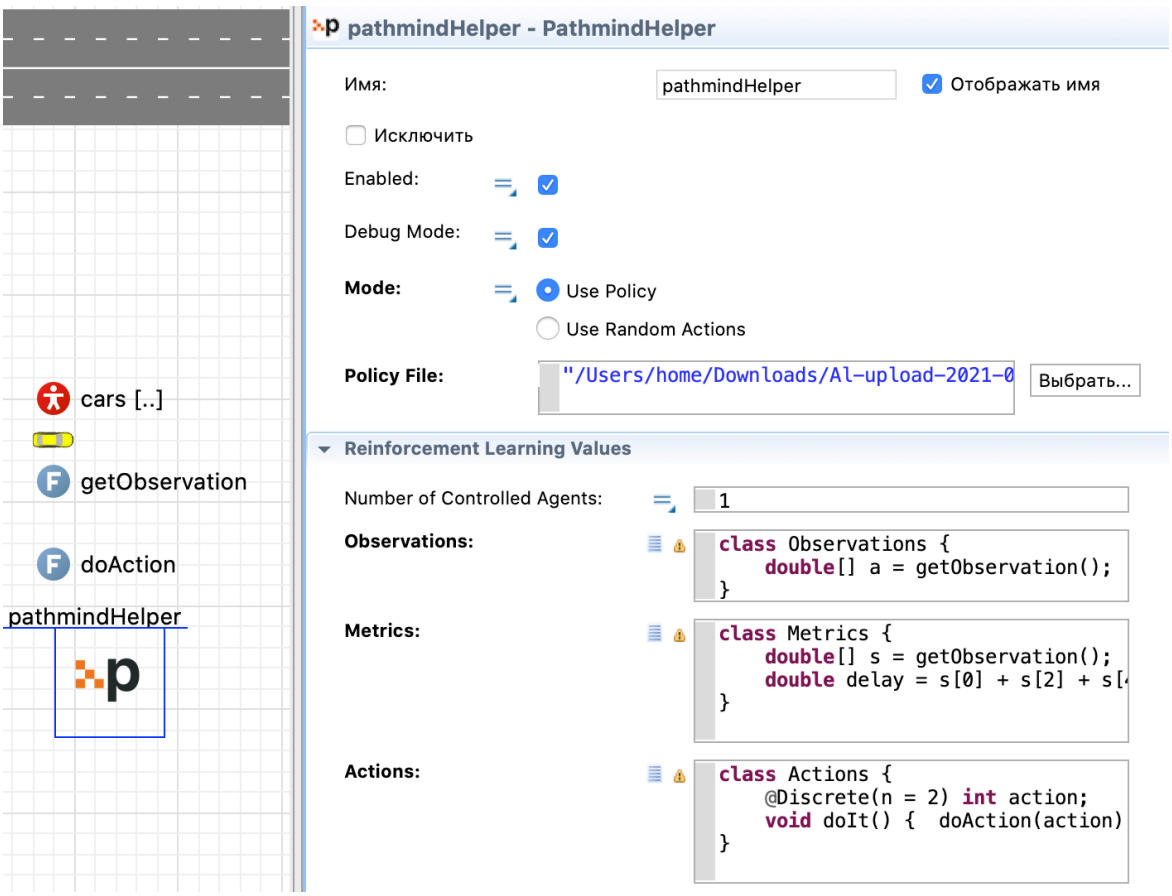

Рисунок 2.6 Блок pathmindHelper

Observations

В качестве параметров, определяющих состояние системы, будем рассматривать вектор, состоящий из значений времени, которое агенты провели на определенных участках дорожной сети. В рамках модели время измеряется в секундах. Первые 8 элементов вектора состояния — это времена, потраченные агентами на преодоление участков, начинающихся за 150 метров до перекрестка, и заканчивающихся на 150 метрах после перекрестка. 9й элемент — это время, проведенное непосредственно на перекрестке. 10й элемент — это индекс текущей фазы светофора(1, 2, 3, 4).

Для получения данного вектора была создана функция *getObservation* (приложение А), которая вызывается блоком Pathmind. Данные предварительно нормализуются.

Metrics

Наиболее важными показателями системы являются времена, проведенные агентами на участках по направлению к перекрестку, ведь чем меньше автомобиль проводит в ожидании зеленого сигнала светофора, тем эффективнее работает система регулировки движения. Также в качестве метрики берется время агентов на элементе-перекрестке. Вся данная информация суммируется в переменную *delay*, она то и является ключевой для вычисления функции вознаграждения.

Reward

В качестве вознаграждения будем использовать время, проведенное агентами вблизи светофора, а точнее разность 2х таких времен, до и после совершения действия агентом.

```
// reward
double[] s = getObservation();
double delay = s[0] + s[2] + s[4] + s[6] + s[8];

reward += before.delay – after.delay;
```

Actions

В нашем примере агент может совершить на выбор 2 действия: никакого, или же переключить сигнал светофора. Кроме того, агент может переключить фазу светофора только если горит не желтый свет. Переключение происходит модельной функцией doAction, которая вызывается блоком Pathmind. В классе Actions явно указывается дискретность пространства возможных действий, а также их количество.

```
// doAction
// Дополнительное условие, не позволяющее переключать сигнал
// светофора при активной желтой фазе
if (action > 0 && trafficLight.getCurrentPhaseIndex()%2==0) {
    trafficLight.switchToNextPhase();
}
```

Оценка состояния системы и действие агента будут производиться раз в 10 секунд.

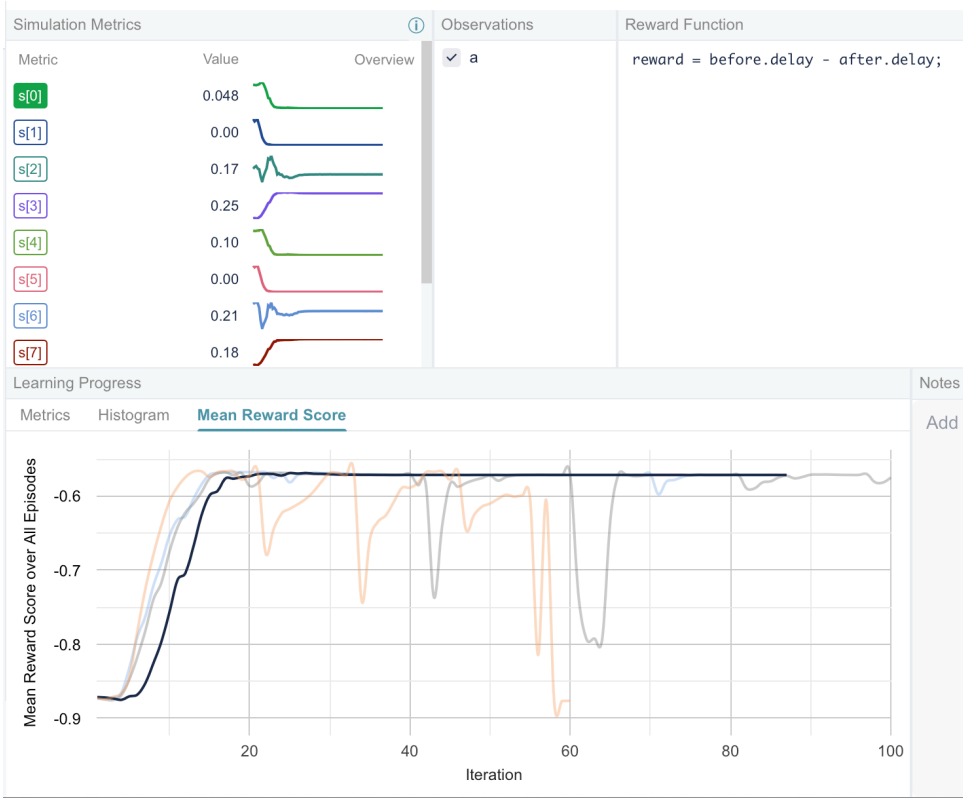

Рисунок 2.7 Интерфейс Pathmind

После экспорта модели в Pathmind начинается обучение нейросети (рисунок 2.7). На вход принимается текущее состояние системы, и в зависимости от значения вознаграждения, система самокорректируется. После окончания обучения готовую политику поведения агента можно скачать в виде zip-архива и с помощью блока Pathmind используется в системе.

2.3. Сравнение интеллектуального агента и оптимизации

При линейных зависимостях величин в системе применение искусственного интеллекта может быть необоснованно. Однако при сложных зависимостях переменных польза нейросетей неоспорима. В работе было смоделировано изменение автомобильного потока с течением времени

(рисунок 2.8) и было произведено сравнение эффективности оптимизации и применения обучаемого агента.

Начало	Конец	Значение
1	2	500.0
2	3	500.0
3	4	500.0
4	5	500.0
5	6	5000.0
6	7	5000.0
7	8	5000.0
8	9	5000.0
9	10	5000.0

Рисунок 2.8 (а) Северное направление

Начало	Конец	Значение
0	1	1000.0

Рисунок 2.8 Р(б) Западное направление

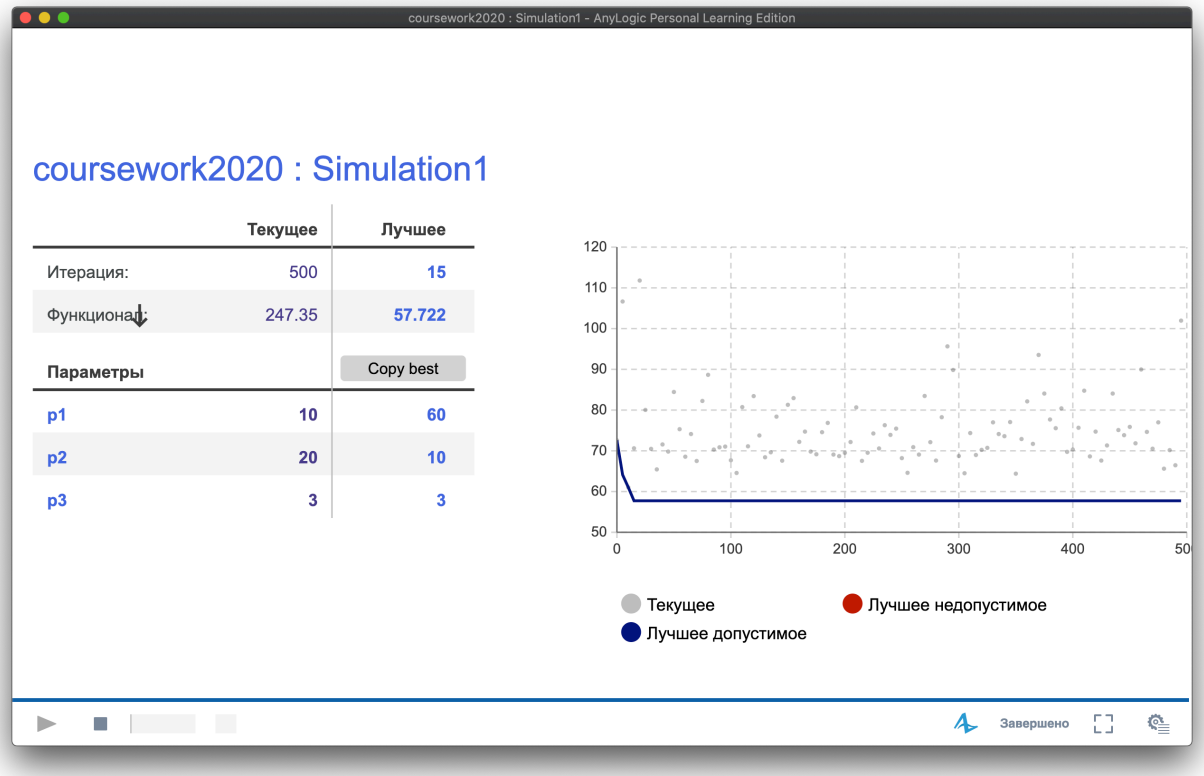

Рисунок 2.9 Оптимизационный эксперимент в AnyLogic

Было проведено 2 симуляции, используя значения, полученные в оптимизационном эксперименте(рисунок 2.9), и используя обученную политику поведения интеллектуального агента (рисунки 2.10 и 2.11).

Уже по количеству автомобилей на дороге можно судить об эффективности обученного агента. Кроме того, гистограмма показывает, что время в системе, управляемой интеллектуальным агентом, не превышает 300с, в то время как при статичных параметрах всё у большего числа автомобилей время в системе близко к 2000 секундам.

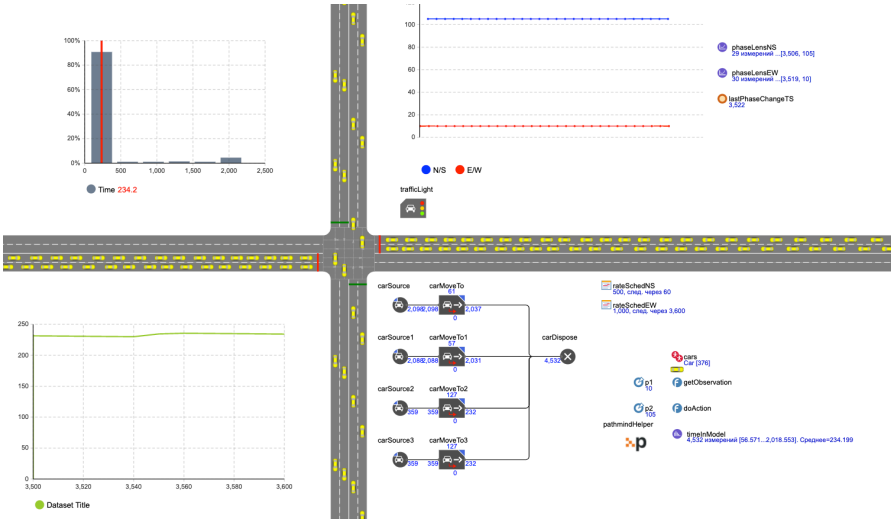

Рисунок 2.10 Использование статичных параметров

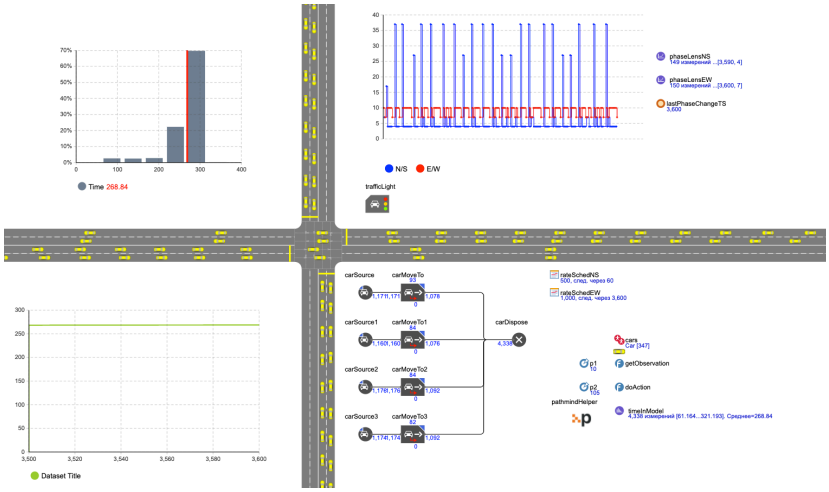

Рисунок 2.11 Использование обученной политики

ЗАКЛЮЧЕНИЕ

Комбинация моделирования сложных реальных систем и техноголий искуственного интеллекта дает невероятные возможности по изучению систем, проведению различных экспериментов и тестированию внедрения новых решений. Наиболее подходящий метод обучения интеллектуальных агентов в среде — это обучение с подкреплением, чей принцип действия основывается на вознаграждении от обучающей среды. При написании работы были изучены различные материалы как по AnyLogic так и по Pathmind, был изучен и реализован пример имитационной модели, а так же было произведено сравнение результатов, полученных с помощью алгоритмов оптимизации и с помощью обучения с подкреплением. Результаты показывают, что даже на таком простом примере, как рассмотренный, интеллектуальный агент показывает намного лучший результат.

СПИСОК ЛИТЕРАТУРЫ

1. https://deepmind.com/blog/article/population-based-training-neural-networks
2. https://www.anylogic.ru/resources/articles/imitatsionnye-modeli-kak-virtualnaya-sreda-dlya-obucheniya-i-testirovaniya-iskusstvennogo-intellekta/
3. https://www.anylogic.ru
4. Agent-based modeling and reinforcement learning for optimizing energy systems operation and maintenance: the Pathmind solution[Электронный ресурс] - Режим доступа: https://www.rpsonline.com.sg/proceedings/esrel2020/pdf/5863.pdf
5. Мезенцев К.Н Практикум «Моделирование систем в среде AnyLogic 6.4.1»

ПРИЛОЖЕНИЕ А

```
// getObservation
// Создаем вектор состояния и считываем
// текущее время в системе
double[] state = new double[10];
double time = time();

// Для выбора агентов на нужных участках, присваиваем переменным
длины дорог
double lenN = scale.toLengthUnits(roadN.getSegment(0).length(),
LengthUnits.METER);
double lenS = scale.toLengthUnits(roadS.getSegment(0).length(),
LengthUnits.METER);
double lenE = scale.toLengthUnits(roadE.getSegment(0).length(),
LengthUnits.METER);
double lenW = scale.toLengthUnits(roadW.getSegment(0).length(),
LengthUnits.METER);

// getCars(true)  -- автомобили, двигающиеся по направлению к
перекрестку
// getCars(false) -- автомобили, двигающиеся от перекрестка
// Рассматриваем только наиболее близкие к перекрестку автомобили
(не дальше чем 150м)
state[0] = roadN.getCars(true).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[1] = roadN.getCars(false).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[2] = roadE.getCars(true).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenE-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
```

```
state[3] = roadE.getCars(false).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[4] = roadS.getCars(true).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[5] = roadS.getCars(false).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[6] = roadW.getCars(true).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();
state[7] = roadW.getCars(false).stream()
                .filter(c ->
((Car)c).getRoadOffset(LengthUnits.METER) >= lenN-150)
                .mapToDouble(c -> time -
((Car)c).startTime).sum();

// Время агентов на перекрестке
state[8] = intersection.getCars().stream().mapToDouble(c -> time -
((Car)c).startTime).sum();

// Индекс текущей фазы светофора
state[9] = trafficLight.getCurrentPhaseIndex();

// Нормализация данных
double sum = Arrays.stream(state, 0, 9).sum();
        for (int i = 0; i < 9; i++) {
            state[i] = Double.isNaN(state[i]/sum)?0:state[i]/sum;
        }
return state;
```