Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Система автоматизированного составления полетного плана с учетом динамического изменения погодных условий

Студент ___ИУ7-85Б____	_____	____Гулая М. Д.____
(Группа)	(Подпись, дата)	(И.О.Фамилия)
Руководитель ВКР	_____	_Солодовников В. И.__
	(Подпись, дата)	(И.О.Фамилия)
Консультант	_____	Мухамеджанов Б. А._
	(Подпись, дата)	(И.О.Фамилия)
Нормоконтролер	_____	___Строганов Ю. В.__
	(Подпись, дата)	(И.О.Фамилия)

2020 г.

РЕФЕРАТ

Расчетно-пояснительная записка содержит 58 стр. , 21 рис., 16 источн., 1 прил.

Объект разработки: система автоматизированного составления полётного плана с учётом динамического изменения погодных условий.

Цель работы: разработать систему автоматизированного составления полётного плана, способную учитывать изменения погодных условий.

Задачи работы:

— анализ предметной области и существующих решений;

— проектирование архитектуры программного обеспечения;

— разработка соответствующего метода и необходимых программных модулей;

— исследование разработанного ПО.

Результаты работы: техническое задание было выполнено. По итогу выполнения данной работы была спроектирована архитектура, разработано и протестировано программное обеспечение, осуществляющее построение маршрута для воздушного судна, обеспечивающего безопасное достижение пункта назначения посредством избегания критических погодных явлений, с помощью обучения с подкреплением. Помимо вышесказанного, были проведены исследования, показывающие эффективность разработанного метода.

Обозначены следующие пути развития данной работы:

— добавление дополнительного количества входных параметров и учитываемых факторов в целях увеличения эффективности;

— возможность строить маршрут в 3D пространстве;

— возможность обучения нескольких агентов на основе единого окружения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы мы наблюдаем прогресс в исследованиях и общем внедрении автоматизации в различные области деятельности человека, будь то обычный быт каждого из нас или серьезные технические аспекты сложных систем.

Построение маршрутов - это одна из актуальных задач настоящего времени, которая имеет множество различных применений, таких как различные картографические и навигационные системы. Проблема маршрутизации различного рода транспорта является очень серьезной задачей, которая особенно остро встала перед людьми еще в XIX веке, если вспомнить задачу коммивояжёра.

Если говорить об автоматизации и ассоциировать ее с проблемой маршрутизации транспорта, то особый интерес представляет построение полетного плана, потому что прогнозирование траектории полета лежит в основе большей части функциональности систем управления воздушным движением, в использовании которых сегодня обычно применяются предопределенные правила и модели прогнозирования траектории из доступных входных данных.

1 Аналитический раздел

Данный раздел представляет собой анализ предметной области, выявление проблемы и исследование существующих ее решений, обзор технологий, методик и средств, а также конечную постановку задачи на основе представленной информации.

1.1 Проблема построения маршрутов

Задача построения оптимального маршрута являлась актуальной как в 18 веке во времена Задачи Коммивояжёра, так и в 21 в наши дни - разрабатываются новые методы решения задачи, реализуются программы, которые позволяют работать с количеством узлов близким к миллиону за приемлемое время.

Неугасаемый интерес к этой задаче обусловлен разнообразием применений ее на практике. Поиск оптимального пути широко используется во всех задачах транспортной логистики, на производстве и в области воздушного движения, которая и будет рассматриваться в данной дипломной работе.

Ниже представлены сущетсвующие задачи построения оптимальных путей и решения проблем маршрутизации в целом.

1.1.1 Задача Коммивояжёра

Задача Коммивояжёра - одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные точки, которые могут быть представлены как в виде городов, так и обычных координат. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый) и соответствующие матрицы расстояний, а также критерий стоимости.

Особенно важным результатом исследования данной задачи является доказательство ее принадлежности к NR-полным в работе Ричарда М. Карпа. Данное работа дает четкое научное объяснение вычислительной сложности нахождения оптимального пути.

Задачу Коммивояжёра можно представить в виде модели на графе $G = (N, A)$, где $N = \{1, ..., n\}$ вершины , соответствующие точкам (координатам), а ребра $A = \{(i, j)|i, j \in N\}$ между ними, являющиеся путями сообщений между координатами. Каждому ребру (i, j) можно сопоставить некоторый критерий выгодности маршрута $c_{i_j} > 0$.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа. Таким образом, решение задачи коммивояжёра — это нахождение гамильтонова цикла $(a_1, ..., a_n), a_i \in A$ минимального веса в полном взвешенном графе, при этом имеющего минимальную стоимость:

$$min \sum_{i=1}^{n} c_{a_i} \tag{1.1}$$

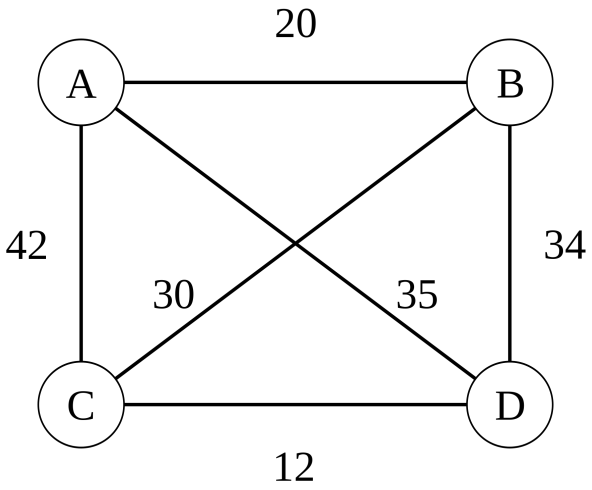

Рисунок 1.1 — Симметричная задача с четыремя точками.

1.1.2 VRP

В настоящее время одной из вариаций данной проблемы является проблема маршрутизации транспорта, более известная, как VRP - Vehicle Routing Problem.

На сегодняшний день, большинство исследований в области задач маршрутизации посвящено именно задаче маршрутизации транспорта, включая её различные модификации.

Задача маршрутизации транспорта является модификацией и расширением задачи коммивояжёра. [1]

Основными особенностями задачи являются:

а) определённая точка, являющаяся начальной точкой для всех транспортов;

б) возможность использования некоторого множества транспортных средств (соответственно и маршрутов).

Зачастую, в задаче определяют так же дополнительные ограничения на отдельный маршрут, таких как ограничение на количество посещённых точек или на длину маршрута.

Рассмотрим множество точек $N = \{1, ..., n\}$ и начальную точку $\{0\}$. Обозначим множество точек $V = \{0\} \cup N$ и матрицу стоимостей $C = \{c_{i_j}\}, i,j \in V$. Предположим, что мы имеем доступ к m транспортных средств. Введём переменные $x_{i_j} \in 0,1$, где 1 - какой-то из маршрутов проходит по пути (i, j), и 0 - иначе. Таким образом, задача будет выглядеть следующим образом:

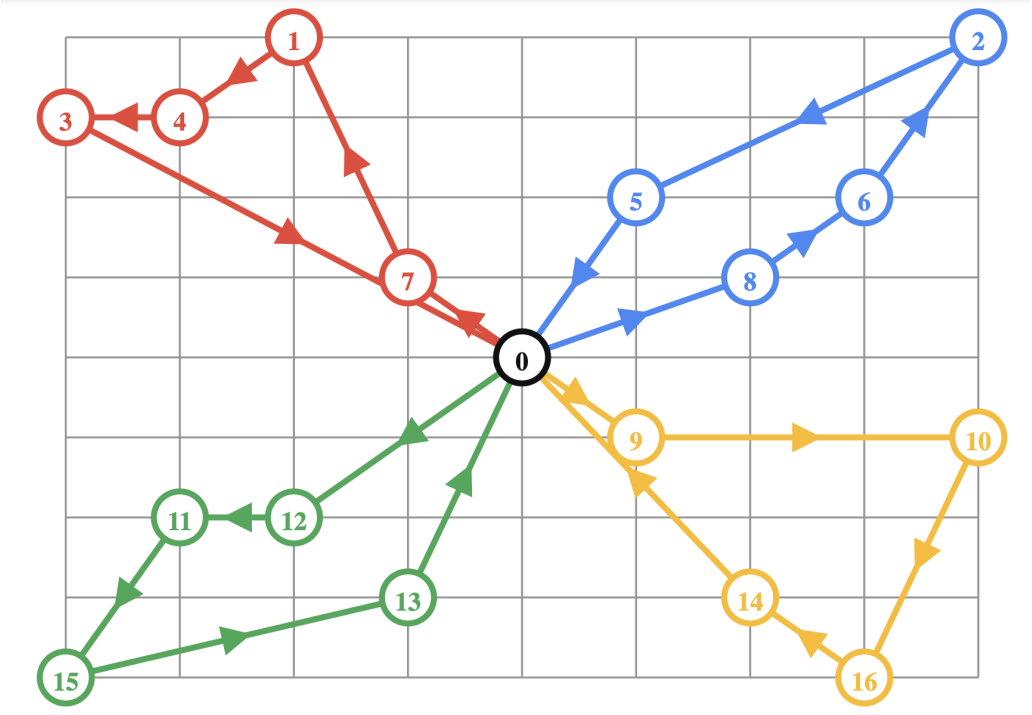

Рисунок 1.2 — Графическое изображение задачи

$$min \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i_j} x_{i_j}, j \neq i \tag{1.2}$$

$$\sum_{j=1}^{n} x_{i_j} = \sum_{j=1}^{n} x_{j_i} = 1, \forall i \in N \tag{1.3}$$

$$\sum_{j=1}^{n} x_{0_j} = \sum_{j=1}^{n} x_{j_0} \leqslant m \tag{1.4}$$

Функция (1.2) отличается от Задачи Коммивояжёра наличием дополнительной точки – начальной. Данные ограничения (1.3) аналогичны задаче коммивояжёра и гарантируют посещение каждой точки, и только один раз. Ограничение (1.4) накладывает рамки на количество покидающих и возвращающихся в начальную точку транспортных средств.

1.1.3 Задача обхода препятствий

Задача обнаруживать и избегать препятствия в режиме реального времени - это логичное развитие проблемы маршрутизации транспорта в классическом ее виде.

Способность обходить препятствия является важным требованием к проектированию при любом практическом применении автономных транспортных средств. Поэтому для этой проблемы было предложено значительное количество решений. К сожалению, большинство из этих решений требуют большой вычислительной нагрузки, что затрудняет, если не делает невозможным, их реализацию в большом количестве систем.

Данный вопрос можно рассматривать с разных точек зрения. В зависимости от конкретных целей и нужд препятствия могут представлять собой абсолютно разные объекты, в том числе погодные явления, что особенно важно учитывать при передвижении на самолетах.[2]

Чтобы описать данную задачу, можно предположить, что имеется некоторая точка S, обозначающая начало маршрута, и точка G, обозначающая

конец маршрута. Между данными точками находится некоторый блокирующий объект, который необходимо преодолеть некоторым обходным путем.

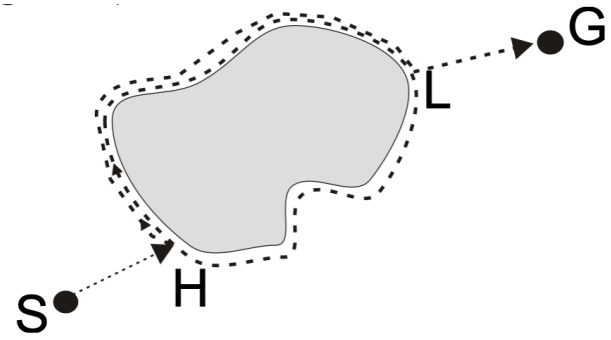

Рисунок 1.3 — Графическое представление задачи обхода препятствий.

1.1.4 Воздушные судна и погодные условия

Осведомленность о погодных условиях

В метеорологическом радаре используются технологии трехмерного объемного сканирования и сжатия импульсов, которые обеспечивают полный обзор погоды от 0 до 60000 футов в диапазоне обнаружения 320 нм.

Инструменты анализа погоды помогают пилотам лучше понимать погодные явления и рассчитать лучшие стратегические и тактические ответы. Уникальная возможность отображения объединяет данные как о погоде, так и о местности, чтобы обеспечить более интуитивный горизонтальный и виртуальный обзор динамики предстоящей погоды - уменьшая или устраняя ненужные отклонения от маршрута.

Пилоты могут выбирать отдельные кусочки воздушного пространства; включая индикаторы определенного диапазона, азимута или высоты, чтобы принимать более обоснованные решения о маршруте и маневрировании. Эти характеристики продемонстрировали 26-ти процентное улучшение обнаружения опасных погодных явлений по сравнению с обычными радиолокационными системами.

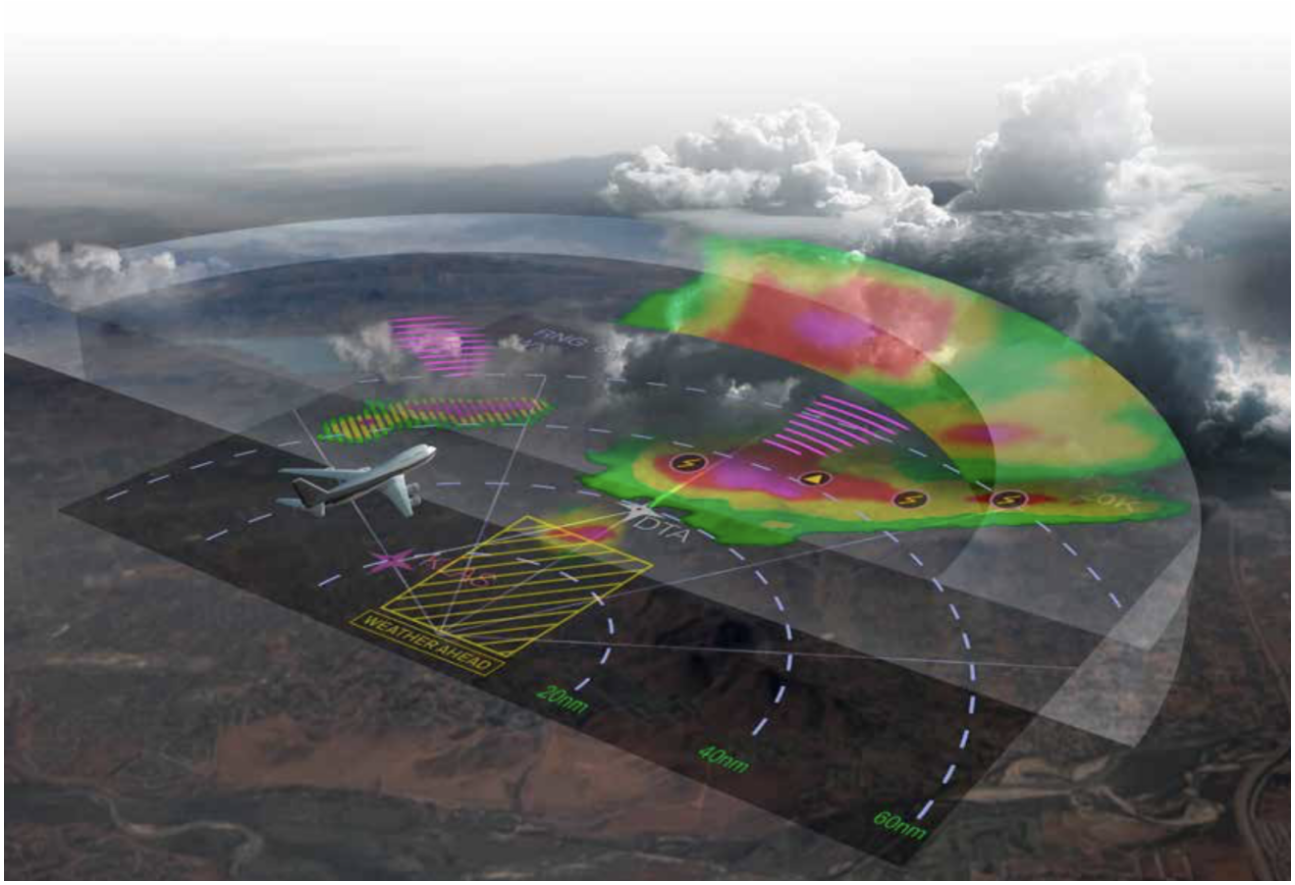

Рисунок 1.4 — Воздушные суда и погодные условия.

Расширенное обнаружение опасностей

Существуют системы (например система отслеживания погодных условий для авиадиспетчеров IntuVue) для улучшенного обнаружения турбулентности, пилоты будут лучше информированы о турбулентности воздуха и будут иметь возможность принимать более безопасные, более информированные решения о маршруте. Благодаря расширенному диапазону 60 нм, доступному при обновлении дисплея опасности V2.0, эта усовершенствованная защита продемонстрировала снижение инцидентов, связанных с турбулентностью, более чем на 45 процентов - повышение безопасности и комфорта пассажиров, а также снижение затрат, связанных с повреждениями.

Такие системы также обеспечивает встроенную защиту от перелета с агрессивным процессом наклона и трехмерного сканирования. Опасные ячейки ниже уровня полета самолета обнаруживаются раньше и отображаются дольше, чем традиционные радиолокационные системы, что повышает безопасность пассажиров. [3]

Вышеуказанный IntuVue является первым радаром, который предлагает прогнозирующее обнаружение града, молнии и ветра и оповещение для воздушных судов с небольшими антеннами, и он снижает ложные сигналы тревоги в опасных погодных условиях на 15 процентов, что приводит к более эффективным решениям по изменению маршрута.

1.2 Существующие решения проблемы маршрутизации

В данном разделе приведены существующие подходы, решающие задачу проблемы маршрутизации транспорта, а также описаны их основные принципы и идеи.

1.2.1 Алгоритм A*

В классической математике является алгоритмом поиска по первому найденному наилучшему совпадению в графе, находящий маршрут от начальной вершины до целевой с минимальной стоимостью.

Эвристической функция задает порядок обхожа вершин (расстояние и стоимость). Данная функция является суммой двух других:

— функции $g(x)$ - стоимости достижения вершины x из начальной;

— функции $h(x)$ - эвристической оценки расстояния от начальной вершины до целевой.

Функция $h(x)$ обязательно должна быть допустимой эвристической оценкой. То есть для задачи маршрутизации $h(x)$ может представлять расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

A* реализуется на графах. С помощью сетки каждый узел обозначают как отдельную точку, а каждое ребро как соединение с соседними точками ко всем сторонам света.

Путь через граф, как известно, — это некоторое количество узлов, которые соединены с помощью ребер. Соответственно A* итеративно просмат-

ривает все пути из некоторой стартовой вершины к целевой до тех пор, пока не найдёт минимальный.

$g(x)$, как уже описывалось выше — это стоимость достижения вершины из начальной. [4]

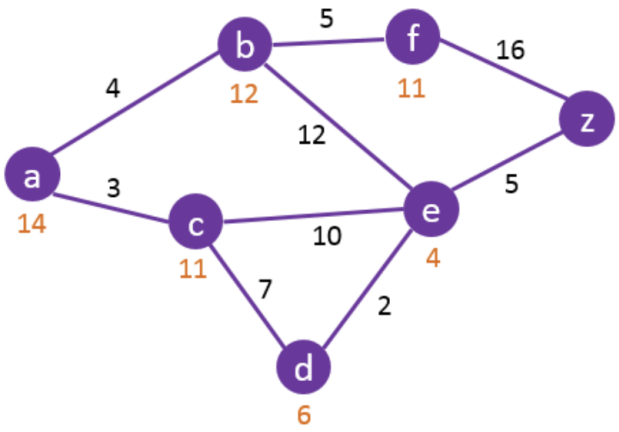

Рисунок 1.5 — Пример графа.

Изначально анализируются узлы, смежные с начальным. Выбирается узел, имеющий минимальное значение функции $f(x)$. На каждом этапе исполнения алгоритм имеет дело с набором путей из начальной точки до каждой еще не рассмотренной. Приоритет пути определяют следующим образом:

$$f_1(x_1) = g_1(x_1) + h_1(x_1) \tag{1.5}$$

Функция оценки в данном случае:

$$f_2(x_1) = g_2(x_1) + h_2(x_1) + h_2(x_2), \tag{1.6}$$

где x_2 - это родительская вершина текущей.

В итоге алгоритм продолжает работу, пока значение $f(x)$ целевой вершины не окажется минимальным из всех возможных решений поиска пути.

У данного алгоритма есть некоторые минусы в реальной практике - в худшем случае, число вершин, которые исследует алгоритм, растёт экспоненциально относительно длины оптимального пути. Также проблемы представляют

собой потребляемые алгоритмом память: в худшем случае придется хранить экспоненциальное количество узлов.

1.2.2 Муравьиный алгоритм

Один из достаточно эффективных алгоритмов, предназначенных для нахождения решений задач поиска маршрутов на графах - муравьиный алгоритм. Его суть заключается в анализе и применении модели поведения муравьёв, которые занимаются поиском пути от колонии к источнику питания.

Основная идея состоит в том, чтобы посылать муравьев в граф поиска и использовать эвристику, называемую феромонами, чтобы заставить их выбрать лучший маршрут. Каждый муравей представляет решение либо к полной проблеме, либо к ее части. Эвристика указывает на вероятность того, что муравей выберите конкретный маршрут. Вероятность зависит как от предполагаемой желательности выбора этого маршрута, так и от количества муравьев, которые выбрали этот конкретный маршрут ранее.

Рассмотрим алгоритм немного подробнее, нам необходимо:

— создать реализацию модели муравья.

Начальная точка, где располагается муравей, напрямую зависит от ограничений, которые накладывают условия задачи. Для каждой задачи способ размещения муравьёв может быть разным. Либо все они помещаются в одну конкретную точку, либо в разные. На данном этапе также задаётся начальное значение феромона (небольшое положительное число);

— найти решение.

Чтобы вычислить вероятность перехода из вершины i в вершину j, необходимо применить следующую формулу:

$$P_{i_j,k}(t) = \frac{[r_{i_j}(t)]^a * [n_{i_j}]^b}{\sum\limits_{l \in J_{i,k}} [r_{i_j}(t)]^a * [n_{i_j}]^b}, \tag{1.7}$$

где t - это значение феромона, n – эвристическое значение расстояния, a и b – константы;

— произвести обновление феромона.

Обновление происходит в соответствии с формулой:

$$r_{i_j}(t+1) = (1-p) * r_{i_j}(t) + \sum_{k \in \{used(i,j)\}} \frac{Q}{L_k(t)}, \tag{1.8}$$

где p – это интенсивность испраения феромона (исчезновение следа), $L_k(t)$ - стоимость конкретного решения для k-ого муравья, Q – значение цены оптимального решения, $\frac{Q}{L_k(t)}$ - феромон, откладываемый муравьём на ребре (i, j);

— выполнить алгоритм локального поиска, если это необходимо.

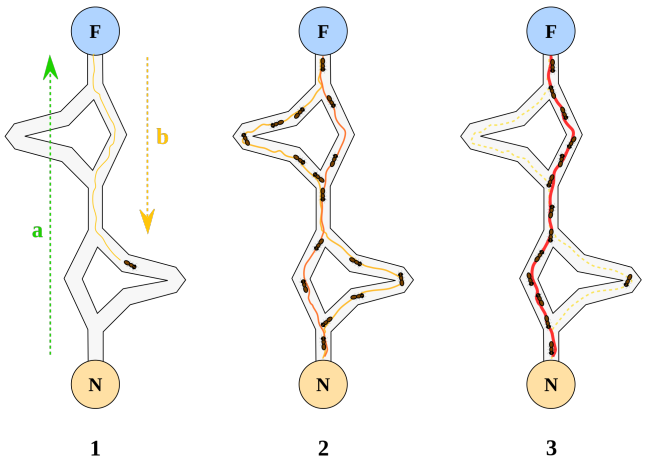

Рисунок 1.6 — Этапы муравьиного алгоритма.

1.3 Нейронные сети

Нейронная сеть — это система соединённых и взаимодействующих между собой простых процессоров (нейронов), соединенных между собой синапсами.

Нейрон является основополагающей частью системы. Это вычислительная единица, которая получает информацию и производит над ней некоторые вычисления. Они делятся на три основные типа:

а) входные нейроны;

б) скрытые нейроны;

в) выходные нейроны.

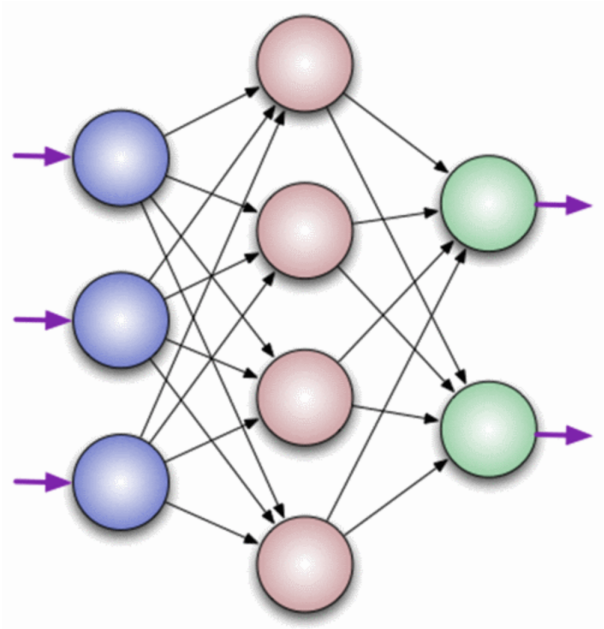

Рисунок 1.7 — Пример нейронных слоёв (входные, скрытые, выходные).

На рисунке выше также можно увидеть связи между нейронами, они и называются синапсами. Синапсы имеют параметр веса, благодаря которому входная информация изменяется при переходах от слоя к слою.

1.3.1 Архитектуры нейронных сетей

Существует множество видов архитектур нейронных сетей, однако стоит выделить три основных:

— полносвязные нейронные сети прямого распространения;

— сверточные нейроенные сети;

— рекуррентные нейронные сети.

Полносвязные нейронные сети прямого распространения

Данный вид архитектуры передает информацию от входа к выходу в прямом направлении. Нейронные сети описываются в виде набора слоёв, каждый из которых состоит из входных, скрытых или выходных нейронов. Нейроны конкретного слоя не связаны между собой, в то время как соседние слои обычно связаны тесно, полностью.

Неройнные сети прямого распространения обучаются по методу обратного распространения ошибки, в котором сеть получает множества входных и выходных данных. Данный процесс называется обучением с учителем. Ошибка является некоторым значением, характеризующим отличие получившегося результата работы сети от ожидаемого. [5] При наличии достаточного количества скрытых нейронов, сеть теоретически способна моделировать взаимодействие между входными и выходными данными.

Рисунок 1.8 — Нейросети прямого распространения.

Такие сети широко используются и успешно решают определенный класс задач: прогнозирование, кластеризация и распознавание.

Сверточные нейронные сети

Данная архитектура обычно используются для обработки изображений. Частым случаем применения сверточной нейронной сети является классификация изображений.

Такие сети обычно используют некоторый “сканер”, который не обрабатывает все данные за один раз. Данный "сканер"является ядром сверточной нейронной сети. Например, если имеется изображение 200x200, сеть будет считать квадрат размером 20x20, далее произойдет свдиг на пиксель и посчитает квадрат снова и т.д. Входные данные передаются через свёрточные слои, которые имеют свойство сжиматься с глубиной. Часто на практике к концу прикрепляют слои прямого распространения для дальнейшей обработки данных.

Размер ядра сверточной нейронной сети определяет количество признаков, которые будут объединены для получения нового признака на выходе.

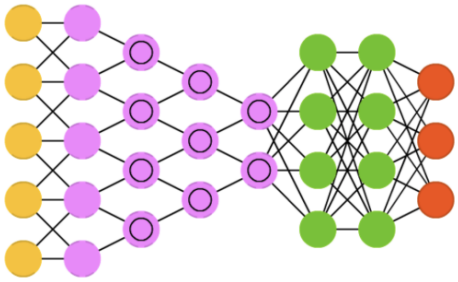

Рисунок 1.9 — Сверточная нейросеть.

Рекуррентные нейроные сети

Рекуррентные нейронные сети - сети с обратными или перекрестными связями между различными слоями нейронов. Они мгут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому данный вид сетей применим в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи.

В рекуррентных сетях присутствуют обратные рекурсивные связи - каждый узел получает данные не только от узлов предыдущего слоя, но и сохраняет некоторое значение предыдущей активации.

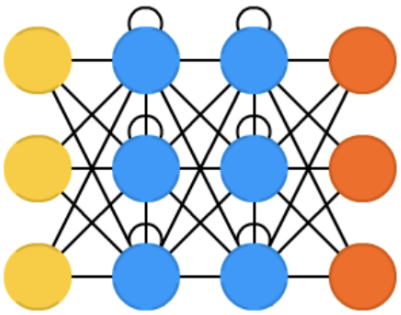

Рисунок 1.10 — Рекуррентная нейросеть.

В отличие от традиционных глубоких нейронных сетей, рекуррентные нейронные сети имеют одинаковые параметры на всех этапах, что отражает факт того, что выполняеся одна и та же задача с разными входами. Это значительно уменьшает общее количество параметров, которые нам нужно подобрать.

1.3.2 Обучение с подкреплением

Обучение с подкреплением (Reinforcement Learning, RL) является одним из способов машинного обучения, в ходе которого система (агент) обучается посредством взаимодействия с окружающей средой. В процессе обучения среда каким-то образом регарирует на действия агента, то есть создает отклик - некоторые операции подкрепления. Таким образом данный вид обучения является обучением с учителем, которым является построенная окружающая среда. Стоит отметить, что правила подкрепления строятся на взаимодействии нейронов, в связи с чем данный вид обучения также можно отнести к обучения без учителя.

Данная система имеет обратную связь: агент и среда взаимно воздействуют друг на друга. Таким образом ее можно рассматривать как одно целое, потому что разделение между средой и агентом фактически является условным. [6]

Модель в RL не имеет сведений о среде. Она имеет лишь возможность производить некий набор заданых действий внутри этой среды. Посредством выполнения действий среда переходит в некоторое обновленное состояние, после чего передает модели информацию о подкреплении или, проще говоря, каком-то вознаграждении за совершенное действие.

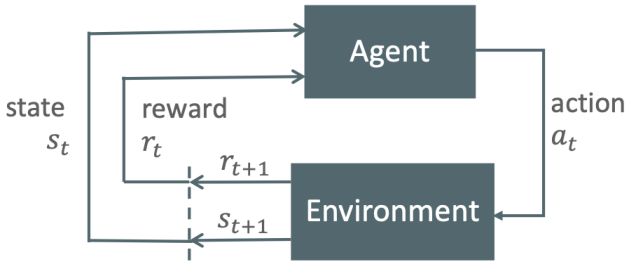

Рисунок 1.11 — Графическое представление работы обучения с подкреплением.

По фатку обучение с подкреплением ставит перед нами задачу, которую необходимо решить. Если попытаться формализовать ее кратко, то мы имеем некоторую сущность, модель, которую необходимо обучить, для чего мы

должны знать, как построить окружение для обучения данной модели и по какому принципу вознагрждать ее.

Q-обучение

Q-обучение (Q-Learning, QL) - это метод, который применяется при агентном подходе и относится к обучению с подкреплением.

Как описывалось выше - в процессе обучения среда каким-то образом вознаграждает агента. Суть QL заключается в том, чтобы на основе этого вознаграждения построить функцию полезности Q, что обеспечивает для обучающейся модели возможность подбирать дальнейшие действия, основываясь на опыте прошлого взаимодействия с окружающей средой. Данный вид обучения используется тогда, когда ситуацию можно представить в виде марковского процесса принятия решений.

$$Q(s,a) \tag{1.9}$$

Марковский процесс - это случайный процесс, эволюция которого в любой момент времени не зависит от эволюции, предшествовавшей этому моменту времени, при условии, что значения процесса фиксировано. [7]

В каждый момент времени процесс находится в некотором состоянии, и объект, принимающий решение, может выбрать любое из доступных в данном состоянии действий. Процесс переходит в новое состояние и дает соответствующее вознаграждение.

В основе вышеописанной функции полезности Q лежит уравнение Беллмана. Уравнение Беллмана - это достаточно известное уравнение динамического программирования. Оно является достаточным условием оптимальности и базируется на принципе оптимальности Беллмана. Данное уравнение яляется дифференциальным уравнением в частных производных с начальными условиями, заданными для последнего момента времени, для функции Беллмана, которая характеризует минимальное значение критерия оптимизации при условии эволюции системы из какого-то конкретного состояния в конечное. Это дает возможность перехода от решения исходной многоэтапной задачи

оптимизации к последовательному решению нескольких отдельных этапов задач оптимизации.

Q-обучение характеризует качество. Качество того, насколько полезно конкретное действие в целях получить награду.

Exploration vs. Exploitation

Компромисс между изучением и эксплуатацией - это хорошо известная проблема, возникающая в сценариях, когда система обучения должна неоднократно делать выбор с неопределенными последствиями. По сути, это дилемма для системы принятия решений, которая имеет лишь неполное знание мира, и состоит она в том, повторять ли решения, которые до сих пор работали хорошо (exploit), или принимать новые решения, надеясь получить еще большее вознаграждение (explore).

Данный компромисс необходимо найти, так как грамотное его выявление приведет к более эффективному обучению. Популярной практикой для не слишком сложных систем является добавление некоторой периодической случайности выбора действия с определенной вероятностью. В процессе обучение следует периодически давать агенту импровизиовать, то есть давать возможность заниматься исследованием (exploration) системы, с целью найти еще более выгодную стратегию для получения награды. [8]

В рамках данной работы предлагается использовать этот вид нейронных сетей для выявления возможного набора действий и определения маршрута для самолета. Этот подход позволяет производить эффективное обучение агента без его знаний об окружающей среде и адаптивно подстраиваться под нее.

1.4 Постановка задачи

В результате проведения анализа предметной области и возможных методов и подходов для реализации, постановка задачи, которую необходимо решить в рамках данной работы, может быть определена слудющим образом.

Требуется разработать программный модуль, использующий нейросеть, построенную на обучении с подкреплением. Данный модуль должен осуществлять поиск возможного маршрута, обеспечивающий самолету безопасное преодолении критических погодных условий на пути от точки А до точки Б. Входными данными являются координаты текущего положения самолета, его скорость, координаты его пункта назначения и координаты области с погодным условием, а выходным параметром является определенный нейросетью набор действией для успешного достижения самолетам его цели.

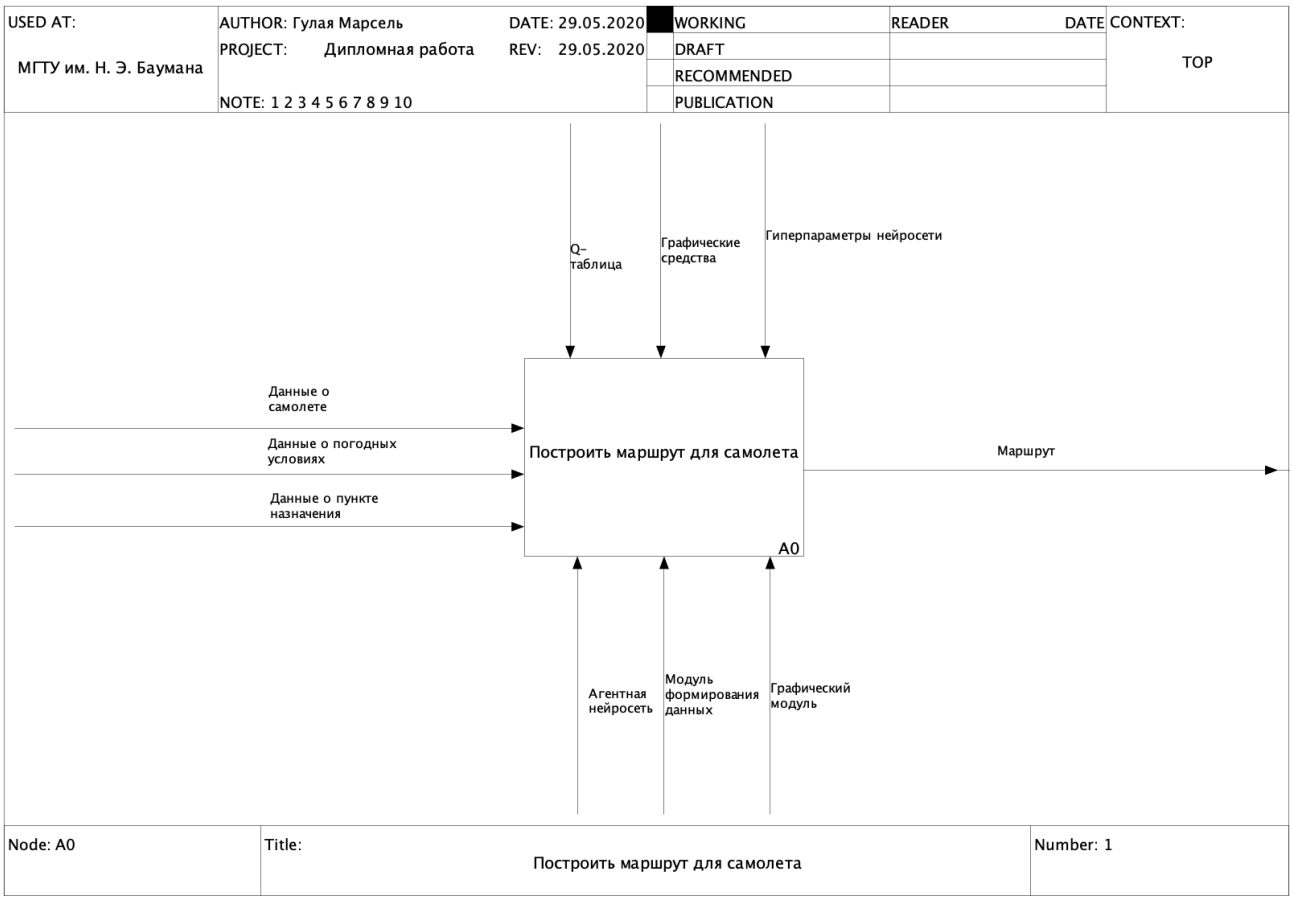

Рисунок 1.12 — IDEF0 диаграмма разрабатываемого модуля.

Для реализации данной задачи необходимо сформировать модель окружающей среды, описать агента и произвести обучение нейронной сети. [9] Также требуется спроектировать и разработать ПО, способное продемонстрировать работу конечного продукта. Данное ПО должно обеспечивать возможность задавать необходимые данные о местоположении самолета и его конечной цели, а также генерировать погодные условия и визуализировать все вышеперечисленное графически.

1.5 Выводы

В данном разделе были изучены основные понятие предметной области, проанализированы существующие методы построения маршрутов. В ходе анализа была формализована задача построения маршрута для самолета с помощью обучения с подкреплением, а также определены входные и выходные данные разрабатываемого ПО и необходимый функционал.

2 Конструкторский раздел

Данный раздел описывает все этапы реализации и необходимые вычисления с описанием методических инструментов, предлагаемых схем и разработанных программных модулей.

2.1 Описание метода построения маршрута с помощью обучения с подкреплением

Данный метод заключается в выполнении алгоритма, получающего на вход координаты текущего положения самолета, его скорость, координаты его пункта назначения и координаты области с погодным условием, а на выходе обеспечивающий графическое представление маршрута, построенного на основе некоторого набора действий, предоставленного нейросетью.

Данный алгоритм можно разделить на несколько основных этапов, которые представлены на IDEF0-диаграмме, изображенной на рисунке 2.1.

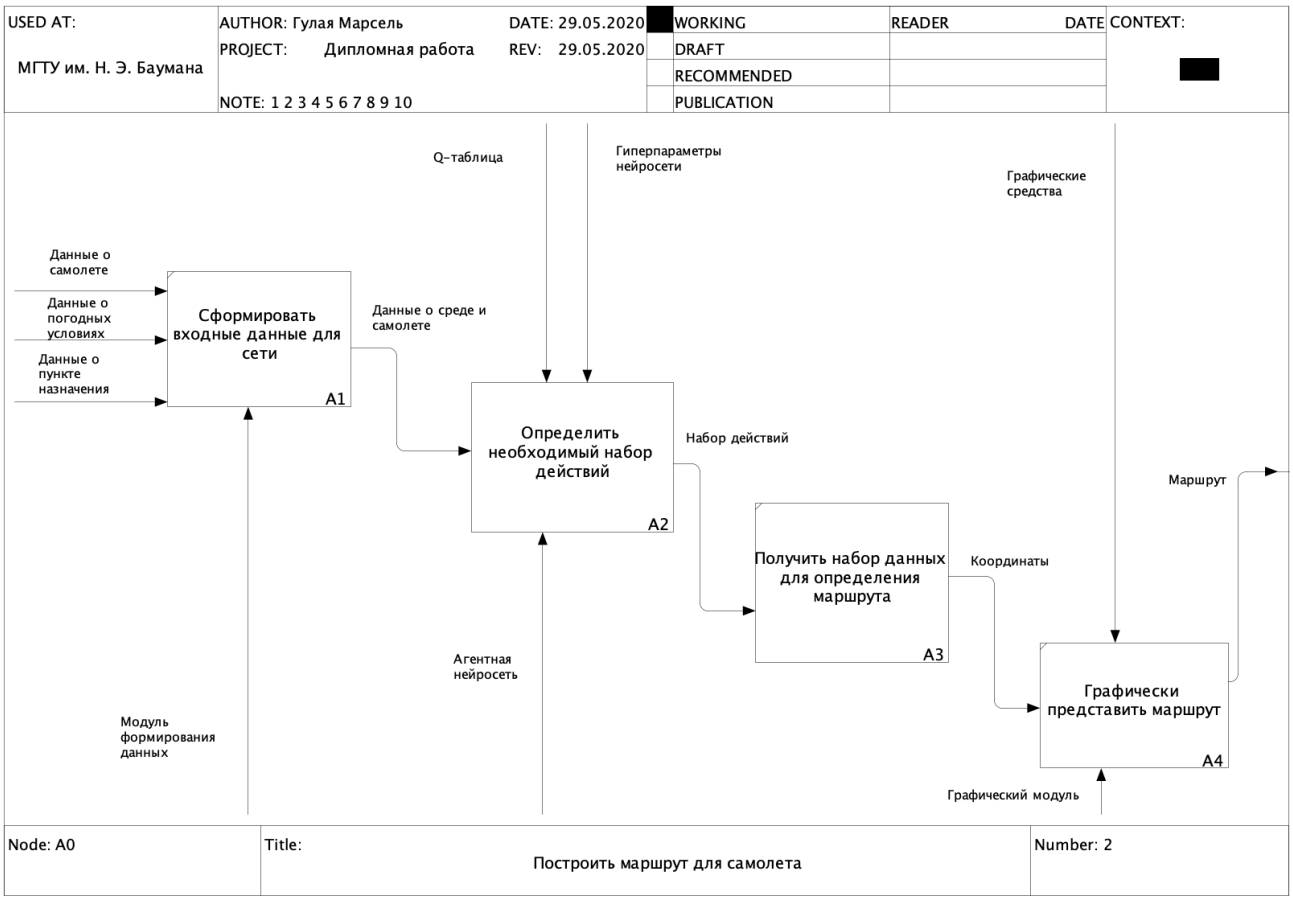

Рисунок 2.1 — IDEF0 диаграмма модуля определения маршрута.

— формирование и подготовка необходимых входных данных для агентной нейросети (рисунок 2.1 блок A1);

— определение требуемого набора действий для формирования маршрута с помощью нейросети (рисунок 2.1 блок A2);

— приведение набора действий к удобной для графического представления форме (рисунок 2.1 блок A3);

— графическая визуализация маршрута (рисунок 2.1 блок A4).

2.1.1 Построение собственной среды

Для построения среды обучения были выделены основные признаки, по которым составлялось описание окружения с точки зрения обучающегося агента. Чем больше количество параметров, тем точнее он будет отзываться на сигналы изменения окружения. [10] В то же время разме пространства состояния может сильно сказаться на скорости обучения, поэтому данный этап в обучении с подкреплением является особенно важным.

Были выделены следующие параметры окружающей среды:

— текущая скорость движения самолета;

— текущее местоположение самолета;

— геолокация пункта назначения самолета;

— координаты погодного условия (дождевые облака);

— направление движения дождевых облаков.

Определение необходимого набора действий

С построенной средой для обучения алгоритм выявления предположительных действий для агента выглядит следующим образом (см. рисунок 2.2).

— определение обзервации. Обзервация (obseravtion) - это некоторое наблюдение, построенное на основе окружающей среды и ее параметров. На основе построенной среды можно выявить следующие наблюдения - расстоя-

ние от текущего местоположения самолета до конечного пункта назначения, а также расстояние от текущего местоположения самолета до обнаруженного погодного явления;

— на основе данных наблюдений из Q-таблицы выявляются некоторые действия, за которые среда подсчитывает вознаграждение для агента;

— далее происходит уточнение весов в Q-таблице;

— набор действий формируется до тех пор, пока система не найдет решение.

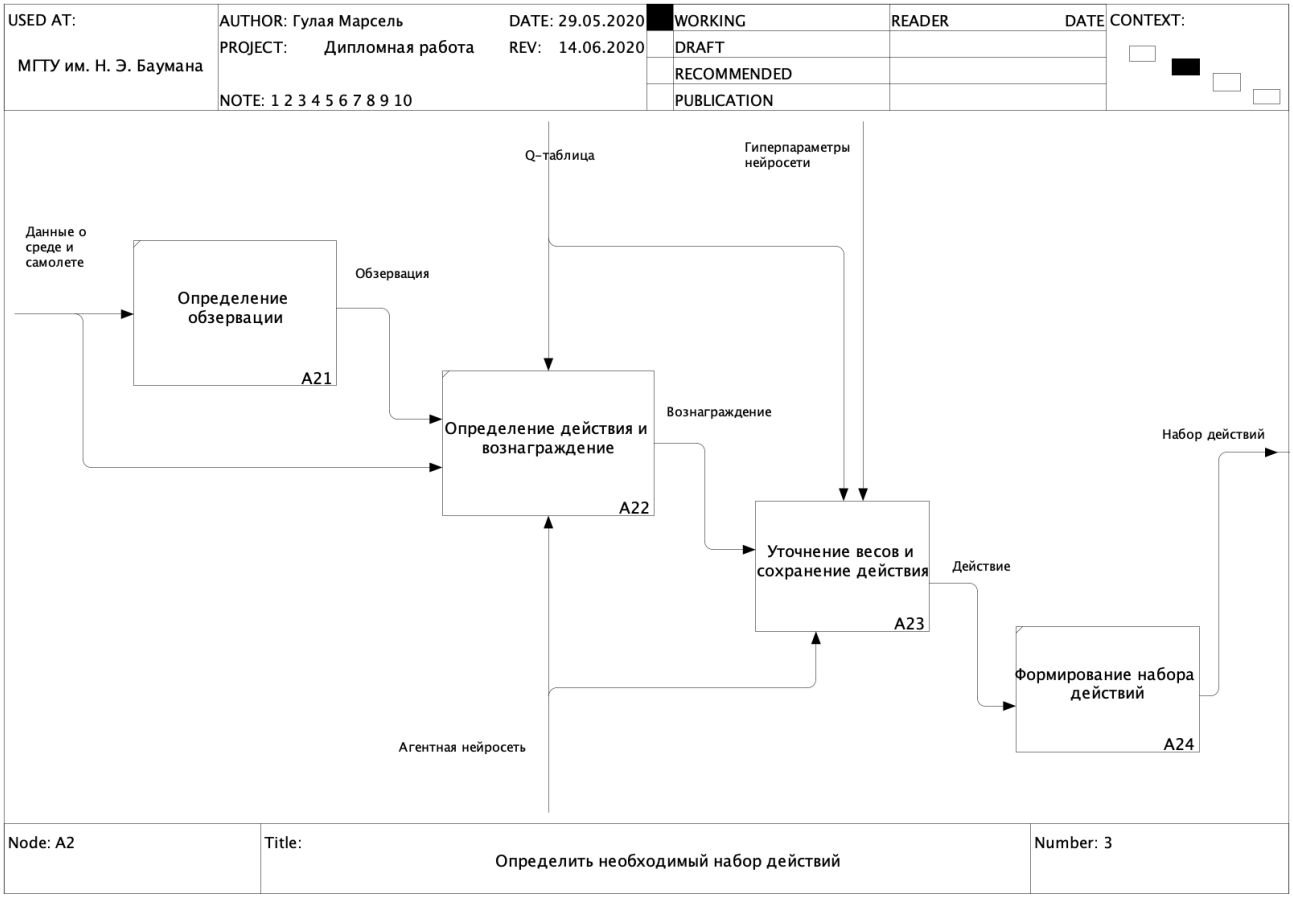

Рисунок 2.2 — IDEF0 диаграмма модуля выявления необходимых действий.

2.1.2 Формирование Q-таблицы

Итак, необходимо ввести функцию Q, которая характеризует ценность каждого возможного действия агента a для текущего состояния s.

$$Q(s,a) \tag{2.1}$$

Данная функция определяет оценку награды, которую получит агент при совершении того или иного действия. Также она определяет значение оценки, которую агент может получить в будущем. [11] В итоге процесс обучения представляет из себя уточнение значения функции Q. Величину награды, которую агент получит на текущей итерации - r_t.

Максимальная ожидаемая награда:

$$maxQ(s_{t+1},a) \tag{2.2}$$

Далее необходимо решить, какого рода награды имеют большее значение: сиюминутные или будущие. Для решения данной проблемы введем дополнительный коэффициент при составляющей оценки будущих наград. Таким образом предсказываемая агентом величина функции Q на конкретном этапе должна быть максимально приближена к значению:

$$r_t + \gamma * maxQ(s_{t+1},a) \tag{2.3}$$

Тогда ошибка предсказания может быть представлена как:

$$\Delta q = r_t + \gamma * maxQ(s_{t+1},a) - Q(s_t, a_t) \tag{2.4}$$

Введем коэффициент для регулирования скорости обучения агента (learning rate, α):

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha * \Delta q \tag{2.5}$$

Итоговая формула итерационного расчета Q-функции:

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha * (r_t + \gamma * maxQ(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.6}$$

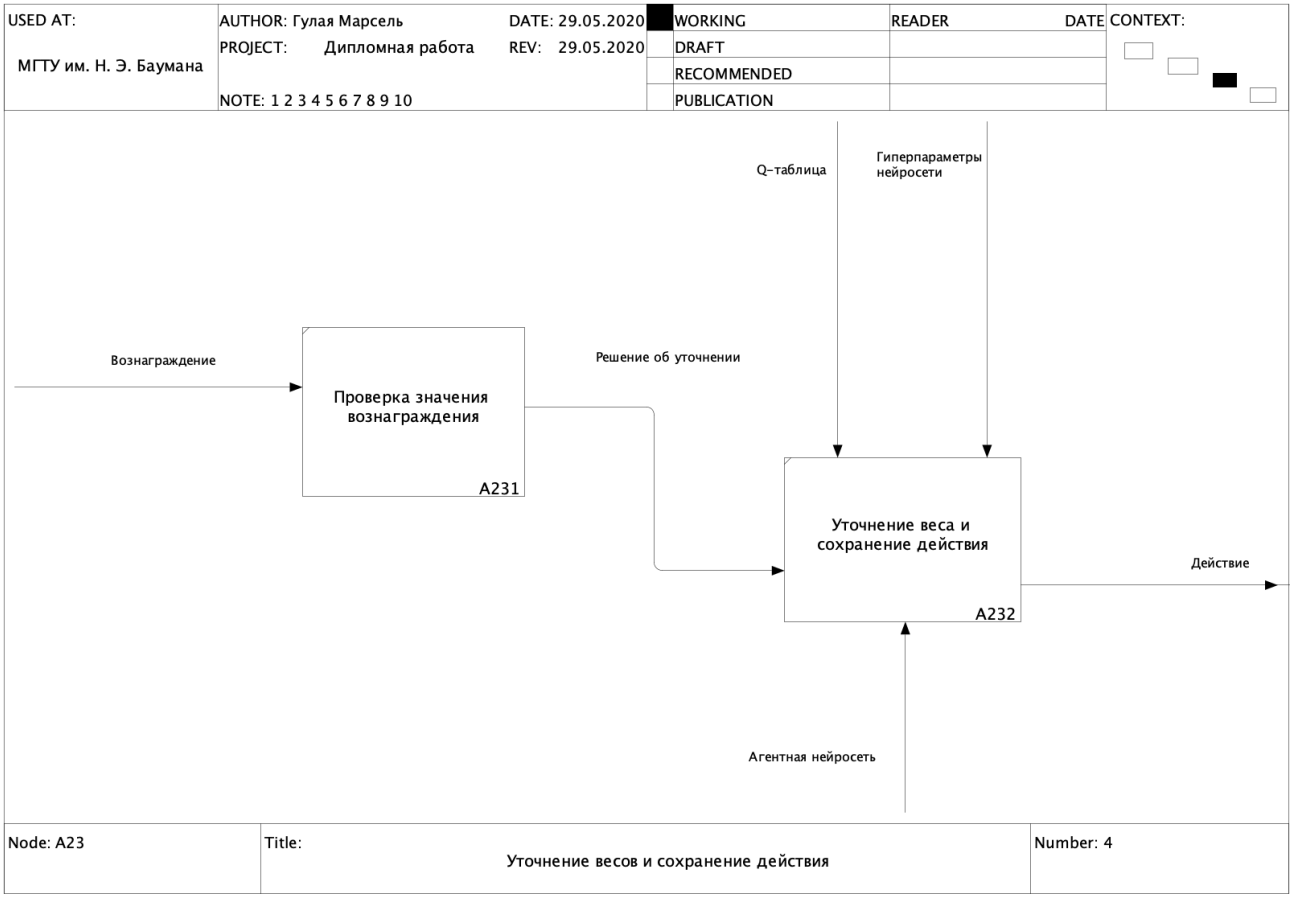

Рисунок 2.3 — IDEF0-диаграмма с описанным уточнением весов Q-таблицы.

— проверка значения вознаграждения для агента (рисунок 2.1 блок A1). На данном этапе анализируется значение полученной награды и на его основе определяется необходимость уточнения весов таблицы;

— определение требуемого набора действий для формирования маршрута с помощью нейросети (рисунок 2.1 блок A2);

— приведение набора действий к удобной для графического представления форме (рисунок 2.1 блок A3);

— графическая визуализация маршрута (рисунок 2.1 блок A4).

На основе данной формулы происходит формирование Q-таблицы итеративным подходым. В процессе обучения на каждый итерации производится уточнение значений, составляющих Q-таблицу, по своей сути характеризующих веса. Эти веса являются основополагающим фактором выбора действий агентом, регаирующего на сигналы среды. [12]

2.2 Архитектура ПО

Архитектура ПО использует внутри себя принципы архитектурного паттерна MVC (Model-View-Controller), который разделяет общую структуру кода на три отдельных компонента:

— модель - часть, ответственность которой заключается сугубо в предоставлении данных конкретным элементам системы;

— представление - каким-то образом реагирует на изменение данных в системе и обеспечивает их отображение пользователю;

— контроллер - является связующим звеном, некой точкой входа для доступа к модели и представлению. Обрабатывает действия пользователя, после чего отдает сигнал модели о необходимости каким-либо образом измениться.

При использовании MVC достигается возможность разделения компонентов работы с данными, пользовательским интерфейсом и логикой взаимодействия пользователя с приложением, благодаря чему модификация одного из этих компонентов оказывает минимальное воздействие на остальные или не оказывает его вовсе, что добавляет системе гибкости.

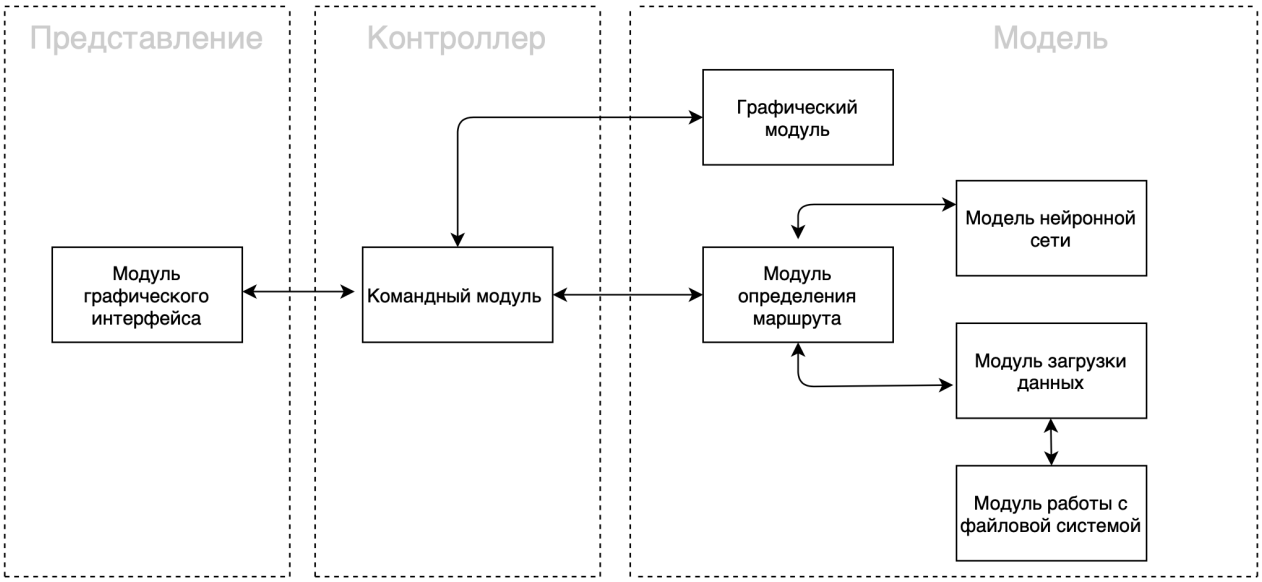

Рисунок 2.4 — Схематическое представление архитектуры программного обеспечения.

Особенностью данного архитектурного подхода является то, что он построен на принципах разделения данных так называемой чистой архитектуры. Любой компонент системы может быть очень легко и просто взаимозаменен другим. То же касается и используемых инструментов, технологий, фреймворков - система не завязывается на чем-то конкретном, в любой момент от этого можно отказаться и заменить чем-то другим.

Графический интерфейс должен предоставлять пользователю возможность взаимодействия с разработанным ПО. Под взаимодействием подразумевается запуск симуляции среды и отображение результата работы системы в диалоговом окне.

Доступный функционал:

— возможность расположения самолета на графической сцене и определение его пункта назначения;

— возможность определения координат погодного явления;

— запуск симуляции;

— графическая визуализация маршрута и анимированное представление результата.

Модули, изображенные на рисунке 2.4, содержат необходимые классы и методы, выполняющие основную логику приложения. Основные из них можно увидеть на схеме, представленной на рисунке 2.5.

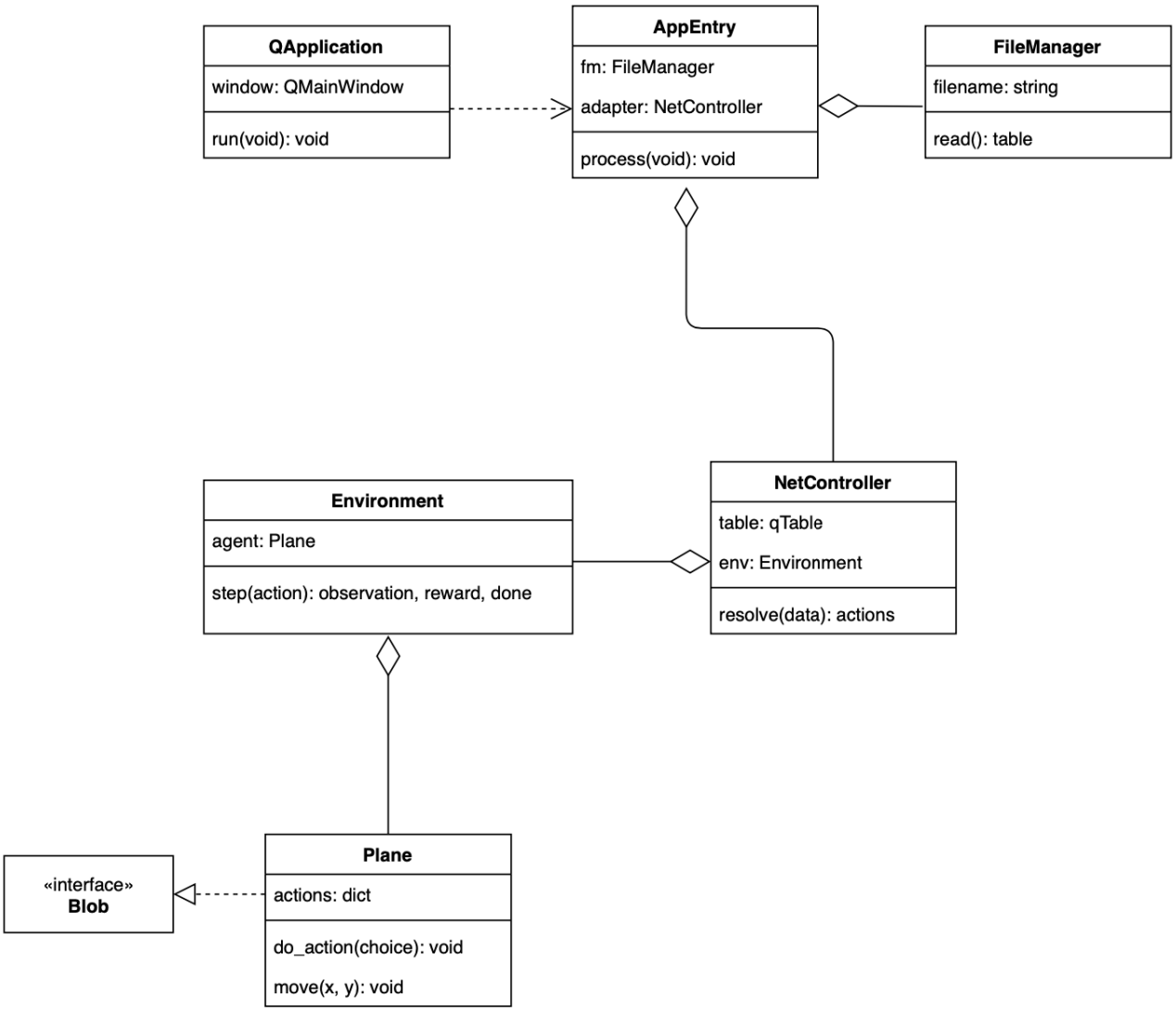

Рисунок 2.5 — Схематическое представление архитектуры программного обеспечения.

Класс AppEntry - точка входа в приложение. Он выполняет основную логику, относящуюся к обработке запросов пользователя к графическому интерфейсу. [13] Он производит взаимодействие с классом FileManager, ответственность которого заключается в загрузке необходимой Q-таблицы обученной нейронной сети. Далее происходит инициализация класса NetController данной таблицей и созданным окружением. [14] Окружение, описанное классом Environment, содержит внутри себя информацию об агенте и имеет основной метод - выполнить шаг, действие и наградить агента в соответствии с его решением.

Формальная схема работы метода

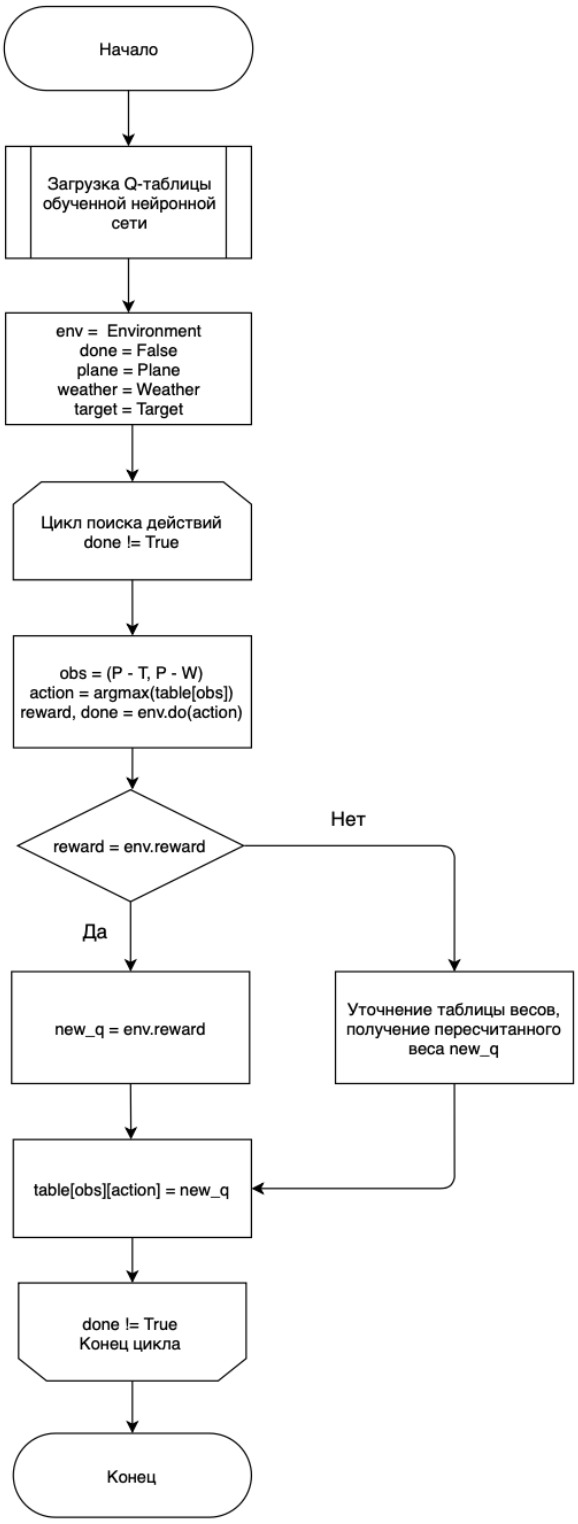

Рисунок 2.6 — Схема работы метода.

2.3 Выводы

В ходе написания данного раздела было спроектировано ПО для построение маршрута самолета, а также были предоставлены некоторые IDEF0 диаграммы и описаны его основные компоненты. Результатом проектирования являются:

— спроектированная архитектура ПО;

— алгоритм построения маршрута с использованием нейронной сети;

— построено окружение для обучения с подкреплением;

— разработана структура ПО.

3 Технологический раздел

В данном разделе представлены используемые в данной работе инструменты и решения, а также технологии, описание исходных данных и расчеты для обоснования целесообразности применения предлагаемых решений.

3.1 Выбранные средства реализации

Разработка ПО для данной работы подразумевает под собой обучение модели нейронной сети, а также реализацию графического пользовательского интерфейса.

В качестве языка программирования был Python.

Python - объектно-ориентированный интерпретируемый язык программирования высокого уровня. Данный язык имеет следующие интерпретаторы:

— Python utility;

— IPython;

— IDLE.

Благодаря продвинутым структурам (связным блокам, функциям, классам, модулям и пакетам) и простейшим в использовании объектам, а также объектно-ориентированного подхода, python позволяет писать чистый, понятный и логически структурированный код, что является огромным преимуществом при написании как маленьких, так и больших приложений.

Преимуществами данного языка также являются:

— Встроенные типы данных высокого уровня: строки, списки, словари и т.д.

— Объектно-ориентированность - питон предоставляет интуитивный интерфейс использования объектов. В питоне легко создавать новые типы объектов.

— Огромная база различных библиотек для прикладных задач.

— Широко используется в научных исследованиях т.к. предоставляет готовые решения для, используемых в научных задачах, алгоритмов и методов.

Одни из самых распространенных библиотек в питоне: SciPy, NumPy, tensorflow и другие. [15]

Вышеперечисленные библиотеки предоставляют удобный функционал для работы со сложными математическими вычислениями и достаточно крупными объемами данных.

Также данные библиотеки предоставляют функционал для решения различных прикладных задач. Среди них присутствует достаточно большое количество работ, связанные с нейронными сетями. Данный язык часто используется при обучении и использовании нейросетевых моделей по причинам, указанным выше.

NumPy - библиотека для python, которая добавляет поддержку для больших многомерных массивов и матриц, а так же огромное количество методов для манипуляций над этими массивами. Данная библиотека использует свои типы данных для своих структур и объектов матриц, массивов.

Matplotlib - библиотека для python, использующаяся для построения графиков и диаграмм. Она использует объекты NumPy и предоставляет объектно-ориентированный интерфейс для разметки, отображая получившиеся результаты с помощью встроенных графических библиотек.

PyQt5 - графическая библиотека для python, которая предоставляет возможность создавать графические интерфейсы для пользователя. Данная библиотека предоставляет объектно-ориентированные решения, которые включают в себя логическую иерархию между объектами, имеет понятную структуру наследования. Она является бесплатной, распространяется по лицензии GPL, LGPL.

3.2 Выбор языка программирования

Прогресс компьютерных технологий определил процесс появления новых разнообразных знаковых систем для записи алгоритмов, которые сегодня

известны как языки программирования. Значение появления такого языка – это образование оснащённого набора вычислительных формул дополнительной информации, что превращает данный набор в алгоритм. Язык программирования служит двум связанным между собой целям: он формирует концепции, которыми пользуется программист для достижения поставленной задачи, и он даёт программисту аппарат для задания действий, которые должны быть выполнены. Сформируем список наиболее популярных языков программирования и проведем обзор на установление их соответствия для решения поставленной задачи:

— C++;

— Python;

— Java;

— Ruby;

— C#;

— Perl;

3.2.1 C++

C++ - это универсальный язык программирования. За исключением второстепенных деталей C++ является надмножеством языка программирования C. Помимо возможностей, которые дает C, C++ предоставляет эффективные и гибкие средства определения новых типов. Используя определения новых типов, точно отвечающих концепциям приложения, программист может разделять разрабатываемую программу на легко поддающиеся контролю части. Такой метод построения программ часто называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определённых пользователем. Такие объекты просты и надёжны в использовании в тех ситуациях, когда их тип нельзя установить на стадии компиляции. [16]

Программирование с применением таких объектов часто называют объектно-ориентированным. При правильном использовании этот метод даёт

легче контролируемые программы, более короткие и проще понимаемые. C++ предлагает программисту полный набор операторов структурного программирования. Он также обладает очень большим набором операций. Многие операции C++ соответствуют машинным командам, и поэтому допускают прямую трансляцию в код ассемблера. Разнообразие операций позволяет выбирать их различные наборы для минимизации результирующего поля. C++ поддерживает указатели на переменные и функции. Указатель на объект программы соответствует машинному адресу данного объекта. Посредством разумного использования указателей можно создавать эффективные программы, которые выполняются быстро, так как указатели позволяют ссылаться на объекты тем же самым путём, как это делает машина.

C++ поддерживает алгебру указателей, и тем самым позволяет осуществлять прямой доступ и непосредственные манипуляции с адресами памяти. В своём составе C++ содержит препроцессор, который обрабатывает текстовые файлы перед компиляцией. Среди его полезных приложений при написании программ на C++ являются: определение программных констант, условная компиляция и замена вызова функций аналогичными, но более быстрыми макросами. Препроцессор не ограничен процессированием только исходных текстовых файлов C++, он может быть использован для любого текстового файла.

C++ - гибкий язык, позволяющий принимать в конкретных ситуациях самые разнообразные решения. В C++ полностью поддерживаются принципы объектноориентированного программирования, включая три кита, на которых оно стоит: наследование, инкапсуляцию, и полиморфизм. Язык C++ поддерживает наследование. Это значит, что можно объявить новый тип данных (класс), который является расширением существующего. Инкапсуляция в C++ поддерживается посредством создания пользовательских типов данных, называемых классами.

Хотя язык C++ справедливо называют продолжением C, и любая работоспособная программа на языке C будет поддерживаться компилятором C++, при переходе от C к C++ был сделан весьма существенный скачок. Язык C++ выигрывал от своего родства с языком C в течение многих лет,

поскольку сообщество программистов обнаружило, что для того, чтобы в полной мере воспользоваться особенностями и преимуществами языка C++, им нужно приобрести новые и отказаться от некоторых своих прежних знаний, а именно: изучить новый способ решения проблем программирования и концептуальности.

3.2.2 Python

Python – высокоуровневый язык программирования общего назначения, который ориентирован на повышение читаемости кода и производительности разработчика. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных в ходе разработки функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, императивное, аспектно-ориентированное и функциональное. Основными архитектурными чертами являются такие особенности как автоматическое управление памятью, динамическая типизация, механизм обработки исключений, полная интроспекция, гибкие высокоуровневые структуры данных и поддержка многопоточных вычислений. Код в Python организовывается в классы и функции, которые могут объединяться в модули, которые могут быть объединены в пакеты.

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, которая позволяет использовать его без ограничений в любых приложениях, включая проприетарные. Наиболее часто Python сравнивают с Ruby и Perl. Эти языки обладают примерно одинаковой скоростью выполнения программ и также являются интерпретируемыми. Как и Ruby, Python является хорошо продуманной системой для ООП. Сравнимо с Perl, Python также может успешно применяться для написания различных скриптов. В среде коммерческих приложений скорость выполнения программ на Python часто сравнивают с приложениями на языке программирования Java.

Данный язык позволяет писать кроссплатформенные приложения. Наряду со стандартной библиотекой существует большое количество сторонних библиотек, значительно уменьшающих время разработки и увеличивающих функциональность. Python и подавляющее большинство библиотек к нему поставляются в исходных кодах и являются бесплатными. Более того, в отличие от многих открытых систем, лицензия не налагает никаких обязательств кроме указания авторских прав и никак не ограничивает использование Python в коммерческих разработках.

В число недостатков входят низкое быстродействие (хотя реализованное на Python множество программ и библиотек для интеграции с другими языками программирования предоставляют возможность использовать другой язык для написания критических участков, потому для сохранения совместимости со многими языками программирования модуль обработки снимков реализован на C++), глобальная блокировка интерпретатора и невозможность модификации встроенных классов. Данные недостатки не являются серьезными для разработки и работы ПМ ВИЗ.

3.2.3 Java

Java – объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems, которая позже была приобретена компанией Oracle. Программы на Java транслируются в байтовый код, который выполняется виртуальной машиной Java (JVM) — программой, обрабатывающей байт-код и в качестве интерпретатора передающей инструкции оборудованию.

Преимуществом данного способа выполнения программ является полная независимость байт-кода от аппаратно-программного комплекса, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важным достоинством технологии Java является гибкая система безопасности, в границах которой выполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного соединения с другим компьютером или доступа к данным), вызывают немедленное прерывание.

Часто к недостаткам технологии виртуальной машины относят снижение производительности. Следующий ряд улучшений несколько увеличил скорость исполнения программ на Java:

— широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках;

— применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;

— аппаратные средства, обеспечивающие ускоренную обработку байтового кода (например, технология Jazelle, которая поддерживается некоторыми процессорами фирмы ARM).

Основные возможности Java:

— расширенные возможности обработки исключительных ситуаций;

— автоматическое управление памятью;

— набор стандартных коллекций: массив, список, стек и т. п.;

— богатый набор средств фильтрации ввода-вывода;

— наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;

— наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);

— унифицированный доступ к базам данных;

— на уровне отдельных SQL-запросов — на основе JDBC, SQLJ;

— на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects и Java Persistence API;

— встроенные в язык средства создания многопоточных приложений;

— параллельное выполнение программ;

— поддержка лямбд, обобщений, замыканий, встроенные возможности функционального программирования.

3.2.4 Ruby

Ruby – рефлективный, интерпретируемый и динамический высокоуровневый язык программирования, который замечательно подходит для удобного и быстрого объектноориентированного программирования. Язык обладает следующими возможностями:

— полностью объектно-ориентированный язык программирования, т.е. все данные в Ruby являются объектами в понимании Smalltalk. Также поддерживается добавление методов в класс и даже в конкретный экземпляр класса во время выполнения программы;

— не поддерживает множественное наследование, но вместо него может использоваться концепция «примесей», основанная в данном языке на механизме модулей;

— позволяет обрабатывать исключения в стиле Python и Java;

— содержит автоматический сборщик мусора, который работает для всех объектов Ruby, в том числе и для подключенных внешних библиотек;

— позволяет переопределять операторы, которые в действительности являются методами;

— целочисленные переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (более 32 разрядов) в зависимости от хранимого в них значения, что позволяет производить целочисленные математические расчёты со сколь угодно большой точностью;

— поддерживает замыкания с полной привязкой к переменным;

— в Ruby непосредственно в языке реализованы многие шаблоны проектирования;

— может динамически загружать расширения, если это позволяет сделать операционная система;

— создавать расширения для Ruby на C очень просто частично из-за сборщика мусора, частично из-за несложного и удобного API;

— имеет независимую от операционной системы поддержку невытесняющей многопоточности.

Кроссплатформенная реализация интерпретатора языка является полностью свободной, а большинство расширений могут быть использованы в любом проекте практически без ограничений, так как распространяются под свободными лицензиями (LGPL, лицензия Ruby).

3.2.5 C#

C# – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах в компании Microsoft в качестве языка разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ISO/IEC 23270 и ECMA-334. C# относится к семье языков с C-подобным синтаксисом, среди них его синтаксис наиболее близок к C++ и Java. Язык поддерживает полиморфизм, перегрузку операторов (в том числе операторов неявного и явного приведения типа), имеет статическую типизацию, атрибуты, делегаты, свойства, события, обобщённые методы и типы, анонимные функции с поддержкой замыканий, итераторы, LINQ-запросы, комментарии в формате XML и исключения.

Переняв многое от своих предшественников – языков Pascal, C++, Модула, Smalltalk и, в особенности, Java – C#, опираясь на практику их применения, исключает некоторые модели использования, зарекомендовавшие себя как проблематичные при разработке программных систем. Так, в отличие от C++, C# не поддерживает множественное наследование классов (но между тем допускается множественное наследование интерфейсов).

C# разрабатывался как язык программирования прикладного уровня для CLR и потому прежде всего зависит от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Отсутствие или присутствие каких-либо выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в со-

ответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно преобразился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (стоит заметить, что данная закономерность была нарушена с выходом C# версии 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

Взаимодействие между различными языками является ключевой особенностью .NET Framework. Поскольку код на промежуточном языке (IL), создаваемый компилятором C#, соответствует спецификации CTS, промежуточный код на основе C# может взаимодействовать с кодом, который создается версиями языков Visual Basic, Visual J#, Visual C++ платформы .NET Framework и еще другими более чем 20 CTS-совместимых языков. В одной сборке может быть несколько модулей, написанных на разных языках платформы .NET Framework, при этом типы могут ссылаться друг на друга как в том случае, если бы они были написаны на одном языке. Помимо служб времени выполнения, в .NET Framework также имеется богатая библиотека, состоящая из более чем 4000 классов, структурированных по пространствам имен, которые обеспечивают разнообразные полезные функции для любых действий, начиная от элементов управления Windows Forms до ввода и вывода файлов для управления строками для разбивки XML. В обычном приложении на языке C# библиотека классов .NET Framework интенсивно используется в ходе написания кода программы. C# – объектно-ориентированный язык, который является элегантным, типобезопасным и предназначенным для разработки разнообразных мощных и безопасных приложений, выполняемых в среде .NET Framework. С помощью языка C# можно создавать обычные приложения Windows, распределенные компоненты, XML-веб-службы, приложения «клиент-сервер», приложения баз данных и т.д.

3.2.6 Perl

Perl – динамический интерпретируемый высокоуровневый язык программирования общего назначения, который был первоначально предназначен для манипуляций с текстом, но на данный момент активно используется для выполнения широкого спектра задач, включая веб-разработку, системное администрирование, сетевое программирование, биоинформатику, игры и разработку графических пользовательских интерфейсов.

Главными достоинствами языка являются поддержка различных парадигм (процедурный, функциональный и объектно-ориентированный стили программирования), встроенная поддержка обработки текста, большая коллекция модулей сторонних разработчиков, а также контроль за памятью (без сборщика мусора, основанного на циклах). Perl унаследовал много свойств от языков C, AWK, скриптовых языков командных оболочек UNIX. Основной особенностью языка считаются его богатые возможности для работы с текстом, в том числе работа с регулярными выражениями, встроенная в синтаксис. Perl обладает множеством встроенных функций, которые обеспечивают набор инструментов, часто используемых для программирования оболочки, например, вызов системных служб или сортировку. Все версии Perl выполняют автоматический контроль над памятью и автоматическую типизацию данных. Интерпретатор знает запросы памяти и тип каждого объекта программы, он освобождает и распределяет память, производя подсчёт ссылок. Перевод одного типа данных в другой — например, числа в строку — происходит автоматически во время исполнения, невозможные для выполнения переводы типов данных приводят к фатальной ошибке.

В результате сравнения языков программирования оптимальным вариантом является сочетание Python и QML. Для написания модулей логики и взаимодействия будет применяться Python, для создания пользовательского интерфейса – QML, установление связи между модулями будет обеспечено с помощью механизма слотов и сигналов и библиотеки PyQt5.

3.3 Выбор среды разработки

В качестве среды разработки была выбрана среда разработки от JetBrains PyCharm. Данная среда разработки имеет доступный интерфейс, а также большое количество полезных функций и плагинов, которые упрощают процесс разработки.

Так как PyCharm нативно поддерживает Python, данная среда разработки выполняет большое количество рутинных действий за разработчика. Также изначально имеется отладчик, который необходим для разработки. Данная среда предоставляет возможность удобного рефакторинга кода, меняя все зависимости во всем проекте.

JetBrains PyCharm предоставляет возможность подключения расширений практически на всех уровнях. Таким образом, можно дополнительно установить расширение для системы контроля версий, что существенно упрощает процесс разработки.

Данная среда разработки имеет постоянную поддержку, частые обновления и большое сообщество пользователей, что упрощает процесс поиска информации.

3.4 Реализация

3.4.1 Окружение

На листинге 3.1 ниже представлен код, описывающий окружение.

Листинг 3.1 — Окружение

```
1     class MyEnv:
2         def __init__(self, size):
3             self.size = size
4             self.observation_space_values = (size, size, 3)
5             self.action_space_size = 9
6             self.episode_step = 0
7
8             self.plane = Plane(size)
9             self.weather_condition = WeatherCondition(size,
                  random.randint(0, 1000))
10            self.target = Target(size)
```

```
11
12              self.fly_penalty = 1
13              self.weather_penalty = 300
14              self.weather_penalty_coef = 0.5
15              self.target_reward = 50
16
17          def reset(self):
18              self.plane = Plane(self.size)
19
20              self.target = Target(self.size)
21              while self.target == self.plane:
22                  self.target = Target(self.size)
23
24              self.weather_condition = WeatherCondition(self.size,
                    random.randint(0, 1000))
25              while self.weather_condition == self.plane or
                    self.weather_condition == self.target:
26                  self.weather_condition = WeatherCondition(self.size,
                        random.randint(0, 1000))
27
28              self.episode_step = 0
29
30              if self.return_images:
31                  observation = np.array(self.get_image())
32              else:
33                  observation = (self.plane - self.target) + (self.plane -
                        self.weather_condition)
34
35              return observation
36
37          def step(self, action):
38              self.episode_step += 1
39              self.plane.do_action(action)
40
41              if self.return_images:
42                  new_observation = np.array(self.get_image())
43              else:
44                  new_observation = (self.plane - self.target) + (self.plane
                        - self.weather_condition)
45
46              if self.plane == self.weather_condition:
47                  reward = -self.weather_penalty
48              elif self.plane == self.target:
49                  reward = self.target_reward
50              else:
51                  reward = -self.fly_penalty
```

```
52
53              done = False
54              if reward == self.target_reward or reward ==
                    -self.weather_penalty or self.episode_step >= 200:
55                  done = True
56
57              return new_observation, reward, done
```

3.4.2 Описание базовых объектов

Ниже представлен листинг кода, реализующий описание базовых объектов.

Листинг 3.2 — Описание базовых объектов

```
1       @six.add_metaclass(ABCMeta)
2       class Blob(object):
3           def __init__(self, size):
4               self.size = size
5               self.x = np.random.randint(0, size)
6               self.y = np.random.randint(0, size)
7
8           def __str__(self):
9               return f"{self.x}, {self.y}"
10
11          def __sub__(self, other):
12              return self.x - other.x, self.y - other.y
13
14          @abstractmethod
15          def do_action(self, choice):
16              pass
17
18
19      class Plane(Blob):
20          def __init__(self, size):
21              super().__init__(size)
22              self.actions_dict = {0: {"x":  1, "y":  1},
23                                   1: {"x": -1, "y": -1},
24                                   2: {"x": -1, "y":  1},
25                                   3: {"x":  1, "y": -1},
26                                   4: {"x":  1, "y":  0},
27                                   5: {"x": -1, "y":  0},
28                                   6: {"x":  0, "y":  1},
29                                   7: {"x":  0, "y": -1},
30                                   8: {"x":  0, "y":  0}}
```

```
31
32          def do_action(self, choice):
33              if choice == 0:
34                  self.move(x=1, y=0)
35              elif choice == 1:
36                  self.move(x=-1, y=0)
37
38              elif choice == 2:
39                  self.move(x=0, y=1)
40              elif choice == 3:
41                  self.move(x=0, y=-1)
42
43              return
44
45
46          def move(self, x=0, y=0):
47              if not x:
48                  self.x += np.random.randint(-1, 2)
49              else:
50                  self.x += x
51
52              if not y:
53                  self.y += np.random.randint(-1, 2)
54              else:
55                  self.y += y
56
57              if self.x < 0:
58                  self.x = 0
59              elif self.x > self.size - 1:
60                  self.x = self.size - 1
61
62              if self.y < 0:
63                  self.y = 0
64              elif self.y > self.size - 1:
65                  self.y = self.size - 1
66
67
68      class WeatherCondition(Blob):
69          def __init__(self, size, risk_level, radius=0):
70
71              super().__init__(size)
72              self.risk_level = risk_level
73              self.radius = radius
74
75          def do_action(self, choice):
76              pass
```

```
77
78
79      class Target(Blob):
80          def __init__(self, size):
81              super().__init__(size)
82
83          def do_action(self, choice):
84              pass
```

3.5 Выводы

В результате выполнения технологического раздела, был разработан метод, реализуюший автоматизирование составление маршрутного плана с учетом динамического изменения погодных условий. Данный метод был реализован и протестирован, также был проведен анализ различных показателей работы данного метода, которые будет превдвен в эксперементальном разделе.

4 Исследовательский раздел

В данном разделе рассматриваются исследования, проведенные в результате реализации программного комплекса, с последующей оценкой получившихся результатов для построения окончательного вывода по проведенной работе.

4.1 Исследование времени работы метода

В данном разделе рассматриваются исследования, проведенные в результате реализации программного комплекса, с последующей оценкой получившихся результатов для построения окончательного вывода по проведенной работе.

Для того, чтобы оценить эффективность и применимость в реальной жизни разработанного алгоритма в первую очередь необходимо оценить временные характеристики его работы. Для этого был реализован программный модуль, выполняющий замеры времени с последующим построением графиков (рисунок 4.1).

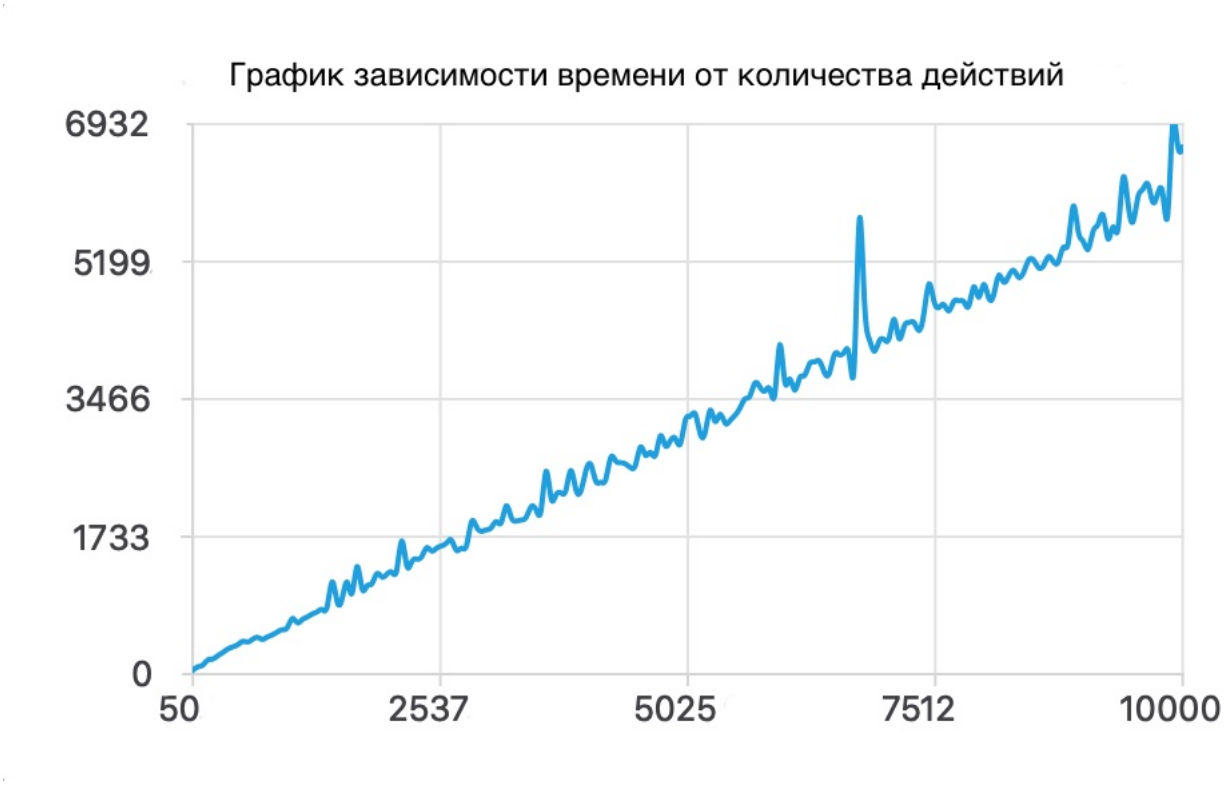

Рисунок 4.1 — график выполнения замеров времени

Для более объемной оценки замеры строятся на большом интервале количества опорных элементов в системе. Полученный график отражает зависимость времени от количества элементов.

В процессе проведения работы была оценена сложность работы алгоритма. Предполагаемая сложность - $O(k * log_2 n)$, где k - коэффициент времени выполнения одной итерации, а n - количество элементов в системе. В результате, исследуя результаты, полученные с помощью временного тестирования, можно заметить, что график ведет себя отлично от $O(log_2 n)$, однако не принимает линейный характер.

Таким образом, при исследовании ряда полученных результатов, можно сделать вывод о том, что коэффициент k не пренебрежимо велик и увеличивает время выполнения алгоритма. Однако итоговые результаты характеризуют этот алгоритм как вполне применимый и удовлетворяющий поставленным условиям в данной работе.

4.2 Исследование характеристик обучения модели

Для оценки этапа метода, на котором осуществляется обучение разработанной модели, необходмо выявить зависимость уровня вознаграждения от количетсва проводимых эпизодов обучения. Такая оценка позволит не только получить статистическую оценку эффективности проводимого обучения, но и выявить минимальное количетсво эпизодов для обучения модели с заданной точностью. Такое исследование ялвяется необходимым при оценке разрабатываемого решения на основе нейронных сетей.

Для этого был реализован вспомогательный программный модуль, формирующий статистику, необходимую для проведения исследования в виде графика зависимости награды от количества эпизодов, заданных при построении обучающей схемы сформированной модели.

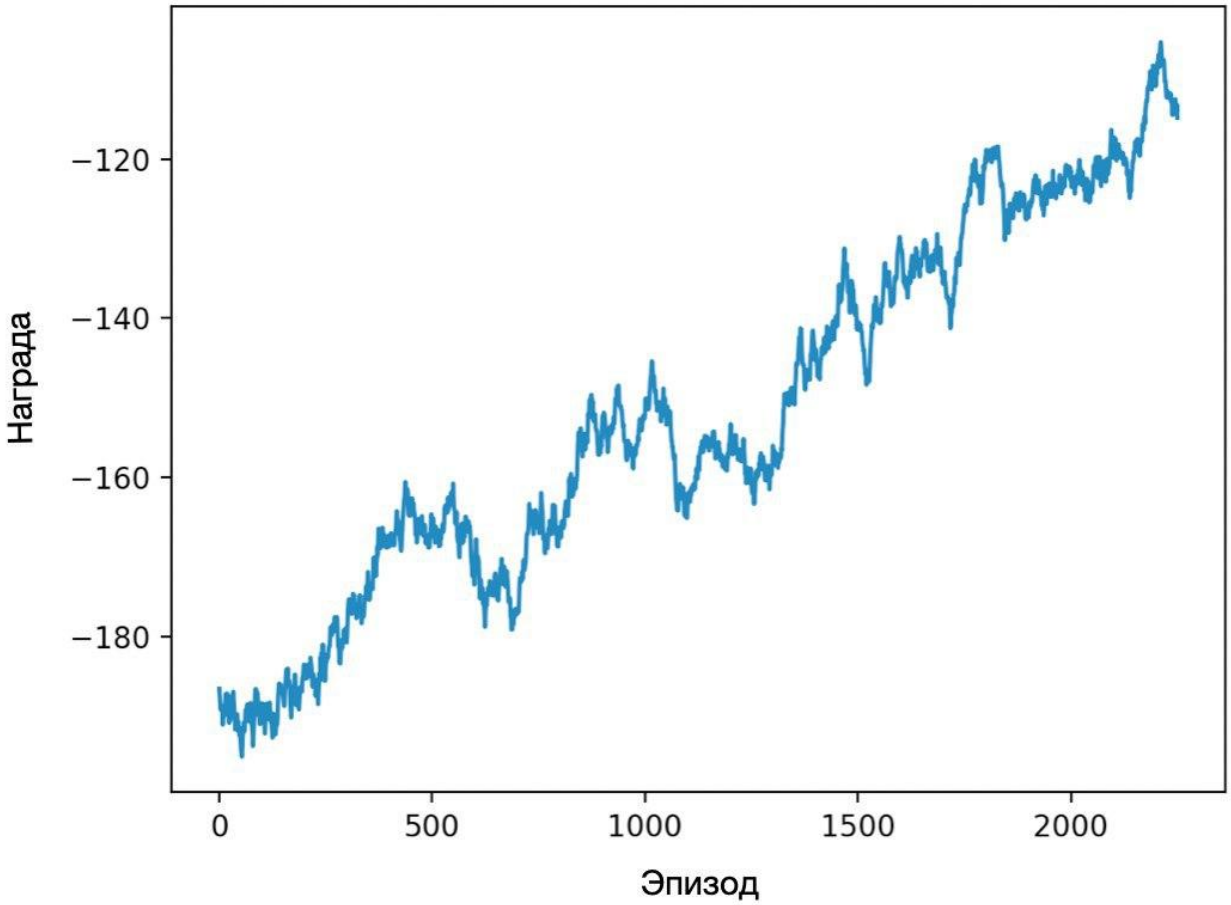

Рисунок 4.2 — график зависимости уровня награды от количества эпизодов

В результате исследования полученной статистической оценки работы этапа обучения сформированной модели, была выявлена линейная зависимость, характеризующая высокую эффективность работы рассматриваемого этапа.ё

4.3 Исследование характеристик работы метода

Помимо оценки итоговых результатов работы разработанного решения в виде обученной модели, построенной на основе нейронных сетей, необходимо оценить эффективность работы самой модели. Требуется установить границы наград на этапе работы самой системы. На этапе аналитичнского исследования предполагалось соблюдение требования отклонения награждения в виде единицы награждения, заданного в системе, а именно 10 условных единиц.

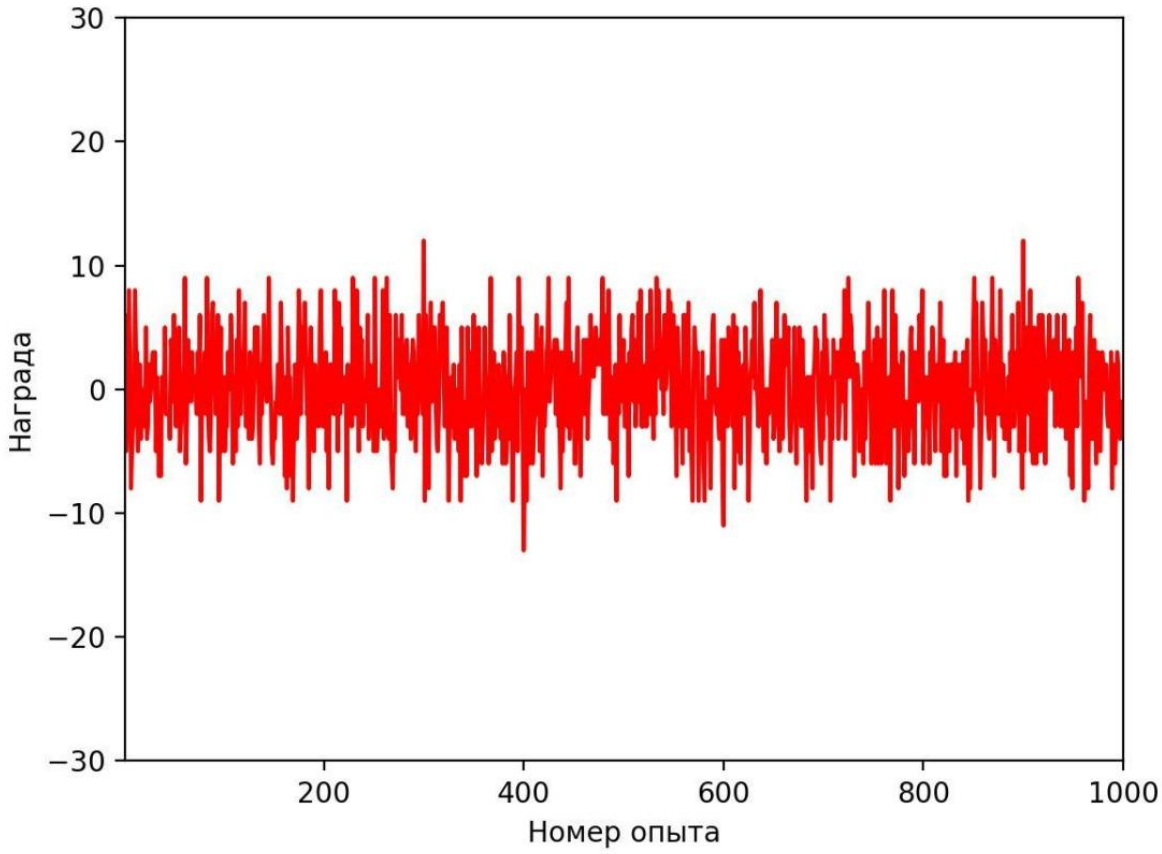

Рисунок 4.3 — график зависимости уровня награды от количества опытов

В результате построения графика зависимости награды от количества опытов с помощью программного комплекса было выявлено четкое соблюдение модели предлагаемых требований. Отклонение итогового награждения системой варьируется в интервале от -10 до 10 условных единиц, из чего следует соблюдение требуемой при постановке задачи адекватности работы модели.

4.4 Выводы

В результате проведения исследовательской работы над разработанным решением было рассмотрено и оценено соблюдение всех требований, предъявленных при постановке задачи, а также построены графические и статистические оценки рассматриваемых областей разработанного решения.

Таким образом, была выявлена эффективная работа разработанного решения, а также соблюдение всех необходимых требований.

ЗАКЛЮЧЕНИЕ

По итогу выполнения данной работы была спроектирована архитектура, разработано и протестировано программное обеспечение, осуществляющее построение маршрута для воздушного судна, обеспечивающего безопасное достижение пункта назначения посредством избегания критических погодных явлений, с помощью обучения с подкреплением. Помимо вышесказанного, были проведены исследования, показывающие эффективность разработанного метода.

В процессе выполнения работы:

— проанализирована предметная область и существующие решения;

— спроектирована архитектура программного обеспечения;

— разработаны соответствующий метод и необходимые программные модули;

— произведено исследование разработанного ПО.

Из достоинств разработанного метода можно выделить:

— достаточно высокая эффективность на общем фоне проведенных опытов;

— оптимальные временные затраты;

— низкое время обучения модели, построенное на основе лично разработанного окружения для обучения.

Можно обозначить следующие пути развития данной работы:

— добавление дополнительного количества входных параметров и учитываемых факторов в целях увеличения эффективности;

— возможность строить маршрут в 3D пространстве;

— возможность обучения нескольких агентов на основе единого окружения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. T. Lumelsky V. Skewis. Incorporating Range Sensing in the Robot Navigation Function. — IEEE Transactions on Systems, Man, and Cybernetics, 1990. — 1068 p.

2. O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. — Proceedings of the IEEE Symposium on Intelligent Control, 1988. — 384 p.

3. Koren Y. Borenstein J. High-Speed Obstacle Avoidance for Mobile Robotics. — IEEE Transactions on Systems, Man, and Cybernetics, 1985. — 505 p.

4. Y. Borenstein J. Koren. The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots. — Intelligent Agent Multi-Agent Systems, 1991. — 288 p.

5. D.Elakkiya A.Francy Golda S.Aridha. Algorithmic Agent for Effective Mobile Robot Navigation in an Unknown Environment. — IEEE Journal of Robotics and Automatio, 2009. — 145 p.

6. Noborio Takayuki Goto Takeshi Kosaka Hiroshi. On the Heuristics of A* or A Algorithm in ITS and RobotPath-Planning. — International Conference on Intelligent Robots and Systems, 2003. — 1166 p.

7. ´edira I. Alaya C. Solnon K. Gh. Ant colony optimization for multi-objective optimiza- tion problems. — IEEE International Conference on Tools with Artificial Intelligence, 2007. — 457 p.

8. Kochetov. D.A. Alexandrov Y.A. The behavior of the ant colony algorithm for the set covering problem. — Operations Research Proceedings, 1999. — 260 p.

9. ´atal W.J. Cook. D. Applegate R.E. Bixby V. Chv. The Traveling Salesman Problem: A Computational Study. — Princeton University Press, 2006. — 243 p.

10. Deneubourg R. Beckers J.-L., Goss S. Modulation of trail laying in the ant Lasius niger (hymenoptera: Formicidae) and its role in the collective selection of a food source. — Journal of Insect Behavior, 1993. — 759 p.

11. Goss R. Beckers J.-L. Deneubourg S. Modulation of trail laying in the ant Lasius niger (hymenoptera: Formicidae) and its role in the collective selection of a food source. — Journal of Insect Behavior, 1992. — 600 p.

12. Gaspero S. Benedettini A. Roli L. Di. Two-level ACO for haplotype inference under pure parsimony. — Journal on Computing, 2006. — 611 p.

13. Horiuchi Hiroshi Noborio Keiichi Fhjimura Yohei. A Comparative Study of Sensor-Based Path-Planning Algorithms in an Unknown Maze. — Intelligent Robots and Systems, 2000. — 909 p.

14. hui Cao wen Shi. A* algorithm Improvement and Its Application In Path Planning. — Geomatics Spatial Information Technology, 2009. — 208 p.

15. MCCULLOCH W. S. PITTS W. A logical calculus of the ideas immanent in nervous activity. — Butt. math. Biophysics, 1943. — 133 p.

16. W. Barto A. G. Sutton R. S. Anderson C. Neuronlike elements that can solve difficult learning control problems. — IEEE Transactions on Systems, Man, and Cybernetics, 1983. — 834 p.

ПРИЛОЖЕНИЕ А
РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ