

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет (институт)
Вычислительной техники
(наименование)

Кафедра
«Вычислительная техника»
(наименование)

Направление подготовки
(специальность)

09.04.01 Информатика и вычислительная техника
(код и наименование)

Профиль / Магистерская программа

Прикладной искусственный интеллект
(наименование)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

СИСТЕМА ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ
ПО РАЗВЁРТЫВАНИЮ МОБИЛЬНЫХ ПУНКТОВ
ВРЕМЕННОГО РАЗМЕЩЕНИЯ ПРИ
ЧРЕЗВЫЧАЙНЫХ СИТУАЦИЯХ

Студент

Абдраманов 06.06.2025
(подпись, дата)

Абдраманов Анатолий Талгатович
(ФИО полностью)

Руководитель

Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

главк. спец. асс. А.О. КИПР. РУБИН
(должность, место работы)

Васильков В.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 28.05.2025 № 16)

Заведующий кафедрой

Митрохин М.А.
(подпись, дата) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 25.06.2025 № 11)

Секретарь ГЭК

Гурин Е.И.
(подпись, дата) (фамилия, инициалы)

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 29 » 09 20 23 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Абдраманов Анатолий Талгатович гр. 23ВВИм1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Система поддержки принятия решений по развёртыванию мобильных пунктов временного размещения в чрезвычайных ситуациях

Тема утверждена приказом ПГУ № 986/0 от " 29 " 09 20 23 г.

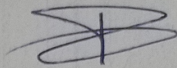
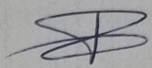
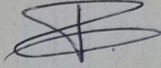
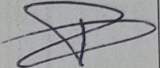
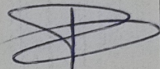
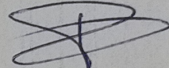
3. Руководитель магистерской диссертации к.т.н, доцент Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

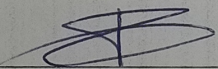
назначением разработки является ускорение принятия решений по развёртыванию пунктов временного размещения (ПВР) путём частичной автоматизации процесса анализа местности на карте с помощью нейросетей и картографических онлайн сервисов

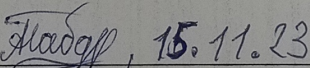
5. Перечень подлежащих разработке вопросов Подбор и изучение материала по теме работы; Анализ предметной области; Построение моделей и алгоритмов; Реализация моделей и алгоритмов.

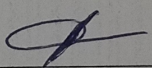
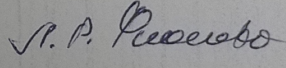
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Выбор темы и разработка индивидуального плана проведения работ	100 ч	с 15.10.2023 по 15.11.2023	
2	Подбор и изучение материала по теме работы	100 ч	с 15.11.2023 по 15.12.2023	
4	Анализ предметной области	360 ч	с 10.2.2024 по 31.05.2024	
5	Построение моделей и алгоритмов	350 ч	с 10.09.2024 по 25.12.2024	
6	Реализация моделей и алгоритмов	250 ч	с 10.02.2025 по 30.04.2025	
7	Оформление отчёта	80 ч	с 30.04.2025 по 25.05.2025	

Дата выдачи задания " 15 " ноября 2023г.

Руководитель магистерской диссертации 
(подпись, дата)

Задание к исполнению принял студент 
(подпись, дата)

Работу к защите допустить
Декан факультета  10.06.2025 
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 92 листов, 28 рисунков, 24 источника и 3 приложения

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, НЕЙРОННЫЕ СЕТИ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ГЛУБОКОЕ ОБУЧЕНИЕ, СИСТЕМЫ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ

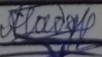
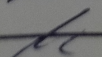
Целью проекта является разработка системы поддержки принятия решений (СППР) по развёртыванию пунктов временного размещения (ПВР) в чрезвычайных ситуациях (ЧС).

Объектом исследования является процесс обучения нейронных сетей.

Предметом исследования является система поддержки принятия решений по развёртыванию пунктов временного размещения, в основу которой положена модель YOLOv11ms.

Технологии разработки: C#, ASP.NET, ООП, YOLOv11m.

Результат работы: спроектированное и разработанное приложение, позволяющее распознавать объекты на карте с помощью нейросетей и находить ближайший населённый пункт и объекты жизнеобеспечения.

					ПГУ 1.090401.01.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Абдраманов А.Т.		06.06.25	СППР по развёртыванию ПВР в ЧС Пояснительная записка	Лит.	Лист	Листов
Провер.		Такташкин Д.В.		06.06.25			2	92
И. Контр		Бычков А.С.		06.06.25		ФВТ, каф. ВТ		
Итверд.		Митрохин М.А.		06.06.25		гр. 23ВВим1		

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Анализ предметной области	8
1.1 Предметная область.....	8
1.1.1 Чрезвычайная ситуация	8
1.1.2 Пункт временного размещения	9
1.1.2.1 Основные задачи ПВР.....	10
1.1.2.2 Классификация ПВР.....	11
1.1.3 Системы поддержки принятия решений	11
1.2 Примеры систем поддержки принятия решений в ЧС и их анализ	13
1.2.1 Система RODOS	13
1.2.2 Система Sahana	14
1.2.3 Анализ примеров СППР для ЧС	16
1.3 Анализ картографических сервисов	17
1.4 Анализ моделей глубокого обучения	19
1.5 Анализ функциональных требований	26
1.6 Анализ нефункциональных требований	27
1.7 Требования к интерфейсу	28
1.7.1 Технологический стек	30
2 Построение моделей и алгоритмов.....	34
2.1 Графическое представление алгоритма обнаружения объектов на карте с помощью нейросети.....	34
2.2 Графическое представление алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения	35
3 Реализация моделей и алгоритмов.....	38
3.1 Обучение модели YOLOv11.....	38
3.2 Реализация алгоритма вычисления географических координат для ограничивающих рамок, показывающих объекты, обнаруженные моделью YOLO	44
3.3 Программная реализация модели и алгоритмов	50

ВВЕДЕНИЕ

На протяжении всего своего существования человечество имело дело с различными чрезвычайными ситуациями (далее ЧС). В древние времена к таким явлениям в основном относились: природные бедствия, к которым можно отнести землетрясения, пожары, ураганы, наводнения и т. д.; эпидемии и пандемии различных болезней (например, Юстинианова чума, бушевавшая с 541 года по середину 8 века); социальные и политические конфликты, к которым можно отнести войны, террористические акты, массовые беспорядки и протесты. На данный момент к списку ЧС добавились техногенные катастрофы, примером которых можно назвать аварию на АЭС Фукусима-1 11 марта 2011 года в Японии после землетрясения и цунами.

В России главным государственным органом, ответственным за управление и координацию различных сил в случае чрезвычайной ситуации, является министерство Российской Федерации по делам гражданской обороны, чрезвычайным ситуациям и ликвидации последствий стихийных бедствий (МЧС).

Для того, чтобы разместить людей, пострадавших во время ЧС, устраивают пункты временного размещения (далее ПВР). Эти пункты могут быть размещены как в открытом поле, так и в городских зданиях. Они предоставляют условия для первичного жизнеобеспечения людей, пострадавших в ЧС.

В рамках разрабатываемого приложения будут использоваться: нейросеть для распознавания изображений и картографический сервис, предоставляющий доступ к картам местности.

Нейронные сети на данный момент стали широко применяться в самых разных областях деятельности. К задачам, где используются нейросети, можно отнести:

- автоматизация проверки качества продукции в промышленности (например, выявление визуальных дефектов на продукции, таких как трещины),
- диагностика заболеваний пациентов в медицине,

- перевод текстов на разные языки (например, с помощью веб-сервиса Google Translate),
- общение с пользователями с помощью чат-ботов (например, с помощью веб-сервиса ChatGPT),
- оценка данных о заемщике для принятия решения о выдаче кредита.

Также, нейросети используются в некоторых устройствах, к примерам которых можно отнести: инсулиновые помпы, протезы рук, экзоскелеты.

Картографические сервисы, как и нейросети, на данный момент встречаются повсеместно. К примерам использования картографических сервисов можно отнести следующие:

- оптимизация маршрутов для курьерских служб,
- определение ближайших водителей и расчет маршрута в сервисах такси,
- исследование рельефа, плотности застройки и использования земель для принятия решения о постройке здания,
- построение маршрутов для пожарных, скорой помощи и полиции.

Помимо всего вышеперечисленного многие из нас лично используют такие сервисы, как Яндекс карты или Google maps, для личной навигации в своем или чужом городе.

Выполненные исследования опираются на результаты работ в областях принятия решений и искусственного интеллекта таких экспертов как: Ларичев О.И., Петровский А.Б., Гришунин С.И., Van de Walle B., Turoff M., Gutierrez G., Llerena J.P., Usero L., Patricio M.A, Jegham N., Khanam R., Hussain M.

Объект исследования – система поддержки принятия решений по развёртыванию пунктов временного размещения в чрезвычайных ситуациях.

Предмет исследования – модель глубокого обучения для распознавания объектов на изображениях, а также алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм поиска ближайших объектов жизнеобеспечения и построения маршрутов до них.

Целью данной работы является ускорение принятия решений по развёртыванию пунктов временного размещения (ПВР) путём частичной автоматизации процесса анализа местности на карте с помощью нейросетей и картографических онлайн сервисов.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области, картографических онлайн сервисов и моделей глубокого обучения,
- разработать графические модели для алгоритма обнаружения объектов на карте с помощью нейросети и алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения,
- разработать алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм поиска ближайшего населённого пункта и объектов жизнеобеспечения.

Решение поставленных задач позволит ускорить принятие решений по развёртыванию пунктов временного размещения в чрезвычайных ситуациях.

Основные результаты работы получены и обоснованы при помощи методов анализа и синтеза, системного анализа, теории глубокого обучения и технологии объектно-ориентированного программирования.

Достоверность научных результатов подтверждена теоретическими вкладами и результатами тестирования, а также сравнением полученных результатов с результатами, приведенными в научной литературе.

Научная новизна работы заключается в следующем:

- разработана графическая модель для алгоритма обнаружения объектов на карте с помощью нейросети,
- разработан алгоритм обнаружения объектов на карте с помощью нейросети,
- разработан алгоритм расчёта географических координат для ограничивающих рамок объектов, обнаруженных нейросетью на изображениях с карты.

Основные положения, выносимые на защиту:

- графическое представление алгоритма обнаружения объектов на карте с помощью нейросети в нотации UML,
- графическое представление алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения в нотации UML,
- программное обеспечение, реализующее алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм поиска ближайшего населённого пункта и объектов жизнеобеспечения.

Использование результатов исследования позволяет ускорить принятие решений по развёртыванию ПВР в чрезвычайных ситуациях.

Структура и объём работы. Работа состоит из введения, 3 тематических глав, заключения, списка литературы и приложений. Общий объём основного текста 70 страниц, включая 28 рисунков и 6 таблиц. Список литературы изложен на 3 страницах и содержит 24 наименования.

1 Анализ предметной области

В целом можно рассказать про предметную область. Однако хороший вопрос насколько это вообще надо делать. Это скорее введение получается. Хотя можно посмотреть, как я это делал в своей дипломной бакалаврской работе. Если там я в разделе анализа предметной области описывал предметную область, то это можно попробовать и здесь сделать. Пока пропустим это дело.

1.1 Предметная область

1.1.1 Чрезвычайная ситуация

Чрезвычайная ситуация (ЧС) – совокупность условий и обстоятельств, создающих опасную для жизнедеятельности человека обстановку на конкретном объекте, территории (акватории), возникших в результате совершившейся аварии или катастрофы, опасного природного явления.

Среди стихийных бедствий и техногенных аварий к наиболее опасным для населения относят следующие [1]:

- катастрофические землетрясения,
- крупные аварии на химически опасных объектах,
- аварии на радиационно-опасных объектах с выбросом радиоактивных веществ в атмосферу,
- наводнения.

В процессе борьбы с ЧС участвуют государственные структуры и организации, негосударственные организации и даже простые люди в роли волонтеров. По большей части задействуются силы и средства, находящиеся в регионе, где возникла ЧС.

Борьба с ЧС имеет много форм. В общем можно все меры борьбы разделить на следующие: предотвращение ЧС; подготовка к ЧС; реагирование на случившуюся ЧС; восстановление после ЧС.

Для сравнения можно посмотреть на направления по обеспечению национальной готовности к ЧС на сайте федерального агентства США по

управлению в чрезвычайных ситуациях (The Federal Emergency Management Agency, сокращенно FEMA) [2]. Данные направления включают следующие:

- предотвращение (англ. Prevention) ЧС,
- защита (англ. Protection) людей и имущества на территории США,
- смягчение (англ. Mitigation) – сокращение людских и материальных потерь за счет заблаговременной подготовки к возможным ЧС,
- реагирование (англ. Response) – быстрое реагирование на ЧС и удовлетворение основных человеческих потребностей после катастрофического инцидента,
- восстановление (англ. Recovery) инфраструктуры и домов, экономики, а также медицинской, социальной, культурной, исторической и экологической структуры сообществ, пострадавших в результате катастрофического инцидента.

Очевидно, что все направления по обеспечению национальной готовности к ЧС в США, за исключением защиты людей и имущества на территории США, соотносятся с формами борьбы с ЧС, которые были выделены мной выше (см. абзац 3 с. 7).

1.1.2 Пункт временного размещения

Пункт временного размещения населения, пострадавшего в ЧС (ПВР) – комплекс технических и материальных средств, оборудования и изделий, предназначенных для первоочередного жизнеобеспечения населения, вынужденных переселенцев (беженцев) и лиц, эвакуируемых из зоны ЧС.

ПВР могут быть развернуты на основе палаток, мобильных зданий и сооружений, общественных зданий или сооружений (капитальных строений). В качестве капитальных строений могут использовать находящиеся в регионе санатории, дома отдыха, гостиницы, пансионаты и так далее.

Рекомендуется создавать ПВР на основе стационарных зданий. Связано это с тем, что разворачивать мобильные ПВР стоит гораздо дороже, чем стационарные [1].

Органы исполнительной власти субъекта РФ должны заблаговременно определять какие здания можно использовать для размещения населения в случае ЧС, а также определять задачи органов местного самоуправления по подготовке ПВР. Рекомендуется ежеквартально обновлять перечни ПВР.

Мобильные ПВР рекомендуется разворачивать в случаях, когда много пострадавших, для которых не могут быть созданы необходимые условия проживания в зонах ЧС и местах эвакуации, а также в случаях, когда вместимость стационарных ПВР субъекта РФ, на территории которого возникла ЧС, не позволяет разместить всех пострадавших даже по минимальным нормам обеспечения жильем, продукцией и услугами жизнеобеспечения в условиях ЧС.

Также стоит отметить, что в мобильных ПВР можно размещать только тех людей, которым не требуется специализированная медицинская помощь.

Разрабатываемое приложение будет искать места для развертывания именно мобильных ПВР.

1.1.2.1 Основные задачи ПВР

В ПВР первоочередное жизнеобеспечение пострадавшего населения должно осуществляться по следующим видам [3]:

- обеспечение жильем,
- обеспечение водой,
- обеспечение питанием (продуктами питания),
- обеспечение коммунально-бытовыми услугами и предметами первой необходимости,
- медицинское обеспечение,
- санитарно-эпидемиологическое обеспечение,
- транспортное обеспечение,
- информационное обеспечение,
- психологическое обеспечение.

Пункты временного размещения играют важную роль в жизнеобеспечении населения в случае ЧС. Они дают людям возможность пережить наиболее сложное время в приемлемых условиях.

1.1.2.2 Классификация ПВР

Согласно ГОСТР 22.3.18–2021 [3] ПВР можно классифицировать: по способу развертывания; по вместимости; по исполнению. Рассмотрим данную классификацию подробнее:

- по способу развертывания ПВР бывают:
 - 1) мобильные;
 - 2) стационарные;
- по вместимости ПВР бывают:
 - 1) малой вместимости (до 125 человек);
 - 2) средней вместимости (от 125 до 250 человек);
 - 3) большой вместимости (более 250 человек);
- по исполнению ПВР бывают:
 - 4) стандартные (умеренный климат);
 - 5) северные (холодный климат);
 - 6) арктические (арктический климат);
 - 7) тропические (субтропический, тропический климат).

Как было написано ранее (см. абзац 3 в разделе 1.2 «Пункт временного размещения» на с. 8), рекомендуется использовать стационарные ПВР (на основе городских капитальных строений). Мобильные же ПВР обычно используются лишь в крайнем случае.

Также можно заметить, что мобильные ПВР обычно размещают во время непосредственного реагирования на ЧС, так как в остальных случаях ПВР либо совсем не нужны (например, если ЧС была предотвращена) или выгоднее разместить людей в стационарных ПВР (во время восстановления после ЧС).

1.1.3 Системы поддержки принятия решений

Системы поддержки принятия решений (СППР) нужны для того, чтобы помогать людям принимать эффективные решения. Такие системы как правило

автоматизируют рутинные процессы анализа имеющихся данных и предлагают пользователю возможные варианты решения проблемы.

Системы поддержки принятия решений используются для решения слабоструктурированных проблем. К слабоструктурированным проблемам относятся проблемы, которые содержат как количественные, так и качественные переменные, причем качественные аспекты проблемы имеют тенденцию доминировать [4].

Все проблемы можно поделить на структурированные и неструктурированные в зависимости от соотношения качественных и количественных переменных, которые они содержат. Чем меньше содержание количественных переменных в проблеме по отношению к качественным, тем менее она структурирована и наоборот. Примерами хорошо структурированных проблем являются дела, относящиеся к бытовой рутине. Например, к таким делам можно отнести стирку и глажку одежды. Примерами плохо структурированных задач можно считать дела, относящиеся к творческой деятельности, такие, как написание стихов или сочинение музыки.

Слабоструктурированные проблемы обладают следующими особенностями: решения этих проблем зависят от текущей неполноты информации; имеется широкий диапазон альтернатив; решения этих проблем содержат элементы риска. В таких проблемах недостаток объективной информации принципиально неустраним на момент принятия решения [5].

Таким образом полностью автоматизировать процесс принятия решений в случае слабоструктурированных проблем невозможно, поэтому решения в итоге принимает человек, а СППР только помогает ему в этом деле.

В сфере организационного управления принято выделять высший, средний и низший уровни иерархической структуры. СППР предназначены для представителей высшего и среднего уровня данной иерархии. Однако руководители из высшего уровня обычно принимают решения на основе альтернатив, подготовленных другими сотрудниками, поэтому они в меньшей степени непосредственно работают с СППР. Таким образом конечными

пользователями СППР являются люди, которые заинтересованы в анализе альтернативных решений и хотят влиять на решение. К таким людям относятся руководители, аналитики-посредники, эксперты-специалисты в различных областях и сотрудники административного аппарата [4].

1.2 Примеры систем поддержки принятия решений в ЧС и их анализ

К примерам СППР, использующихся в ЧС, можно отнести такие системы, как RODOS и Sahana.

1.2.1 Система RODOS

После аварии на Чернобыльской АЭС 1986 года в Европейском союзе было принято решение разработать СППР для ЧС (аварий) на радиационно-опасном объекте с выбросом радиоактивных веществ в атмосферу, вследствие чего была создана система RODOS.

RODOS (акроним от real-time online decision support system) – система поддержки принятия решений в режиме реального времени для управления ядерными чрезвычайными ситуациями, разработанная Евросоюзом, с помощью СССР. Разработка данной системы была осуществлена в рамках третьей европейской рамочной программы. Первый прототип данной системы был разработан в 1992 году, а окончательная версия в 1999 году [6].

СППР RODOS состоит из трех отдельных подсистем, каждая из которых содержит множество модулей. Эти три подсистемы включают в себя подсистему анализа, подсистему контрмер и подсистему оценки [7].

Модули подсистемы анализа обрабатывают входящие данные и прогнозируют местоположение и количество загрязнения, включая временные изменения. Эти модули содержат метеорологические модели, модели атмосферного рассеяния, модели гидрологического рассеяния, модели осаждения и поглощения, модели воздействия на здоровье и другие модели. Данные модули предсказывают развитие ситуации на основе научного понимания процессов, которые связаны с ситуацией.

Модули подсистемы контрмер предлагают возможные контрмеры, проверяют их на осуществимость и рассчитывают ожидаемую выгоду с точки зрения ряда критериев.

Модули подсистемы оценки ранжируют стратегии контрмер в соответствии с их потенциальной выгодой и предпочтениями, предоставленными лицами, принимающими решения.

RODOS – это работающая в режиме реального времени онлайн-система, подключенная к сетям метеорологических и радиологических данных; таким образом, включающая несколько модулей связи. Ее форматы баз данных определяют основу для обмена данными в европейском масштабе.

Все данные, необходимые модулям для обработки информации, хранятся в базах данных, из которых в RODOS есть три основные категории [7]:

- база данных, хранящая программные данные, которые включают входные и выходные данные, требуемые или производимые различными модулями, промежуточные и конечные результаты, временные данные и т. д.,
- база данных, обновляющаяся в режиме реального времени и содержащая информацию, поступающую из региональных или национальных радиологических и метеорологических сетей,
- географическая база данных, содержащая географическую и статистическую информацию для всей Европы.

RODOS используется различными европейскими организациями, ответственными за ядерную безопасность. Она помогает координировать усилия различных служб для эффективного реагирования на ядерные инциденты.

1.2.2 Система Sahana

В 2005 году некоммерческой компанией Lanka Software Foundation (LSF) была разработана система Sahana для организации борьбы с последствиями землетрясения и цунами в Индийском океане в 2004 году [8].

Sahana – это веб-инструмент для совместной работы, который решает общие проблемы координации во время катастрофы между

правительственными группами, гражданским обществом (представлено в виде неправительственных организаций – НПО) и самими жертвами в рамках таких задач, как поиск пропавших людей, управление помощью, управление волонтерами, отслеживание мест переселения и т. д. Sahana – это интегрированный набор подключаемых веб-приложений, использующихся для организации деятельности по борьбе с катастрофами и предоставляющих решения для крупномасштабных гуманитарных проблем после катастрофы.

Система Sahana состоит из следующих компонентов [7]:

- реестр пропавших без вести лиц – помощь в снижении травматизма путем повышения эффективности поиска пропавших без вести лиц,
- реестр организаций – координация и балансировка распределения организаций по оказанию помощи в пострадавших районах и объединение групп по оказанию помощи, что позволяет им работать как единое целое,
- система управления запросами – регистрация и отслеживание всех входящих запросов на поддержку и помощь (имеется в виду материальная гуманитарная помощь) вплоть до момента их выполнения и помощь донорам (поставщикам гуманитарной помощи) в том, чтобы быстрее и проще подключаться к удовлетворению потребностей в помощи,
- реестр лагерей – отслеживание местоположения и количества жертв в различных лагерях и временных убежищах, созданных по всей пострадавшей территории.

В 2006 году система Sahana была превращена в проект с открытым исходным кодом, созданный по большей части на основе PHP.

В 2009 году по запросу LSF в США была создана некоммерческая организация Sahana Software Foundation (SSF), которая занялась разработкой программного обеспечения (ПО) с открытым исходным кодом для гуманитарных целей. Основным проектом SSF стала система Sahana EDEN, которая является идейным продолжением системы Sahana, выпущенной в 2005 году.

Система Sahana и система Eden применялись для борьбы со следующими катастрофами [9]:

- 2004 год – землетрясение и цунами в Индийском океане,
- 2005 год – оползни в Филиппинах, землетрясение в Кашмире,
- 2006 год – землетрясение в Джокаякарте,
- 2007 год – землетрясение в Ике, циклон Сидр,
- 2008 год – землетрясение в провинции Сычуань, циклон Наргис (Мьянма), наводнение в Бихаре,
- 2010 год – наводнение в Венесуэле, ураган в Веракрусе, наводнение в Пакистане, землетрясение в Чили, землетрясение на Гаити,
- 2011 год – торнадо Джоплин, землетрясение и цунами в Тохоку, землетрясение в Крайстчерче, наводнения в Колумбии,
- 2012 год – ураган Сэнди, лесные пожары в Чили,
- 2013 год – тайфун Хайян (Иоланда).

Системы Sahana и Eden до сих пор используются многими странами и международными организациями для координации действий по борьбе с последствиями катастроф.

1.2.3 Анализ примеров СППР для ЧС

Разрабатываемое приложение нацелено на анализ местности и выбор мест для размещения мобильных ПВР. Главной целью приложения станет автоматизация поиска мест для размещения мобильных ПВР, которая ускорит принятие решения о развертке данных пунктов, что в конечном счете позволит пострадавшим от ЧС людям быстрее получить доступ к временной крыше над головой.

В качестве примеров СППР для ЧС были рассмотрены две системы – RODOS и Sahana. Это позволяет поверхностно представить, как выглядят СППР для ЧС.

В рамках, рассмотренных выше СППР (Sahana и RODOS), изначально не предполагалось использование нейросетей. Тем не менее на данный момент

данные системы могут гипотетически использоваться в сочетании с нейросетями на практике.

В нашем случае, разрабатываемое приложение изначально будет нацелено на использование модели глубокого обучения для распознавания объектов на изображении и использование картографического сервиса для доступа к картам местности, которые будет анализировать приложение.

1.3 Анализ картографических сервисов

На таблице 1 показано сравнение Javascript библиотек для веб-картографии [10].

Таблица 1 – Сравнение Javascript библиотек для веб-картографии

№	Критерий	OpenLayers	Leaflet	Google Maps	Яндекс карты
1	Размер библиотеки	170 Кбайт (v. 3.13.1)	123 Кбайт (v.0.7.7)	21.9 Кбайт (остальное загружается динамически)	44.7 Кбайт (v. 2.1, остальное загружается динамически)
2	Функциональность				
2.1	Кластеризация	+	+	+	+
2.2	Тепловые карты	+	+	+	+
2.3	Маршрутизация и навигация	-	-	+	+
2.4	Отслеживание пробок на дорогах	-	-	+	+
2.5	Каталог организаций	-	-	+	+
2.6	Геокодинг	-	-	+	+
3	Доступность базовых карт	MapQuest, OpenStreetMap, Bing Maps	OpenStreetMap, Bing Maps, Esri Maps	Проприетарные – схематичные, спутниковые и смешанные	Проприетарные – схематичные, спутниковые и смешанные
4	Поддержка внешних карт				
4.1	Tile Layer, TMS	+	+	+	+
4.2	Web Map Service	+	+	-	-

Продолжение таблицы 1

№	Критерий	OpenLayers	Leaflet	Google Maps	Яндекс карты
4.3	Web Feature Service	+	+	-	-
4.4	KML (GML)	+	+	+	+
4.5	GeoJSON	+	+	+	-
4.6	GPX	+	+	-	+
4.7	GeoCSV	-	+	-	-

4.8	Shapefile	-	+	-	-
5	Встроенные инструменты				
5.1	Приближение (Zooming)	+	+	+	+
5.2	Смена активных слоёв	+/-	+	+	+
5.3	Отображение координат курсора	+	+	-	-
5.4	Шкала масштаба	+	+	+	+
5.5	Полноэкранный режим	+	+	+	+
5.6	Overview map	+	+	+	+
5.7	Возможность крутить карту	+	-	-	+
5.8	Печать карты	-	+	+	+
5.9	Измерение длины	+/-	+	+	+
5.10	Измерение площади	+/-	+	-	-
6	Поддержка репроекции картографических данных	+	+	+	+
7	Расширяемость	+	+	+/-	+/-
8	Документация	+	+	+	+
9	Поддержка	+	+	+	+

На основе сравнения вышеприведённых Javascript библиотек можно сделать следующие выводы.

Библиотеки Yandex maps и Google maps представляют собой уже готовые решения с закрытым исходным кодом, которые относительно легко можно использовать в своих проектах. Также данные библиотеки предоставляют доступ к таким недоступным в OpenLayers и Leaflet функциям, как: построение маршрутов; отслеживание пробок на дорогах; доступ к каталогу организаций; геокодинг.

С другой стороны библиотеки OpenLayers и Leaflet имеют открытый исходный код и как следствие большое количество плагинов, за счёт которых можно значительно расширить функционал данных библиотек в своём проекте.

В своей статье Хитрин [10] отмечает, что OpenLayers и Leaflet имеют более широкий функционал, чем Yandex maps и Google maps, но при этом данные библиотеки выдвигают более высокие требования к разработчику.

Также в данной статье отмечено, что одним из недостатков Yandex maps и Google maps является отсутствие поддержки Web Map Service (WMS) и Web Feature Service (WFS).

Web Map Service и Web Feature Service – это стандарты, разработанные Open Geospatial Consortium (OGS) и описывающие протоколы обмена данными между клиентом (например, браузером) и сервером, предоставляющим географические данные.

Web Map Service – описывает простой HTTP интерфейс для запрашивания гео-зарегистрированных изображений карт из одной или нескольких распределенных геопространственных баз данных [11]

Web Feature Service – это стандарт, который описывает интерфейс для запрашивания детальных геопространственных данных о картографических объектах с сервера [12].

Разница между WMS и WFS в том, что первый в качестве результата выдаёт пользователю статичную картинку с изображением географической местности и объектов на ней, а второй – детальную пространственную информацию об объектах карты, которую пользователь может редактировать под свои нужды.

Google maps и Yandex maps не поддерживают WMS и WFS, но компенсируют это за счёт функционала своих API (Application Programming Interface). Например, Yandex maps позволяет загружать статичные изображения карты с помощью Static API (что компенсирует отсутствие поддержки WMS) и отображать свои данные на карте (в виде маркеров, полигонов и линий) с помощью JavaScript API (который частично компенсирует отсутствие поддержки WFS).

1.4 Анализ моделей глубокого обучения

В рамках разрабатываемого ПО запланировано использование модели глубокого обучения для распознавания объектов на изображениях.

Такие модели относятся к области компьютерного зрения (Computer Vision). Данная область достаточно быстро развивается на данный момент и с каждым годом архитектуры нейронных сетей здесь становятся совершенней. Среди наиболее известных и производительных нейронных сетей в данном направлении можно выделить семейство моделей глубокого обучения YOLO.

Модели YOLO обладают хорошей производительностью и высокой скоростью обработки входных изображений, что позволяет их применять для распознавания объектов на видео в режиме реального времени.

Чтобы понять, насколько актуальна YOLO надо сравнить её с другими актуальными моделями глубокого обучения. Модели глубокого обучения обычно сравнивают в рамках какой-нибудь отдельно взятой задачи. Например, на таблицах 2 – 3 представлены данные из сравнительного исследования архитектур для обнаружения дронов на тепловизионных изображениях [13].

На таблице 2 представлены данные о производительности моделей глубокого обучения с точки зрения времени.

Таблица 2 – Производительность моделей с точки зрения времени

Модель	Количество эпох	FPS	Время обработки	Время обучения
YOLO	100	134	7.5 мс	48 часов 5 минут
GELAN	75	95	10.5 мс	50 часов 32 минуты
DETR	55	38	26 мс	93 часа 6 минут
ViTDet	55	36	28 мс	97 часов 48 минут

Модели на таблице 1 сравниваются по следующим критериям: количество эпох; FPS (frames per second); время обработки; время обучения.

Количество эпох показывает сколько раз в ходе обучения нейросеть осуществила проход по всей обучающей выборке данных. Если эпох для обучения будет слишком мало, то модель будет недообученной и будет плохо распознавать объекты. Если эпох будет слишком много, то модель станет переобученной и тоже будет плохо распознавать объекты из новых наборов данных, отличных от обучающей выборки.

Количество кадров в секунду (frames per second – FPS) показывает сколько изображений в секунду может обрабатывать модель глубокого обучения.

Время обработки (inference time) показывает сколько времени тратит модель глубокого обучения на обработку одного изображения.

Время обучения показывает сколько времени было потрачено на обучение модели.

Таблица 2 показывает нам, что модель YOLO является самой быстрой в плане обработки изображений со скоростью обработки одного изображения равной 7,5 мс (таким образом, в секунду модель способна обрабатывать 134 изображения). Также можно отметить, что относительно других сравниваемых моделей модель YOLO потребовала наименьшее время на обучение – 48 часов и 5 минут.

На таблице 3 показаны данные о точности некоторых моделей глубокого обучения.

Таблица 3 – Точность моделей

Модель	AP@50	mAP@50:95
YOLO	74.7%	72%
GELAN	81%	77%
DETR	78.3%	75.7%
ViTDet	65%	62%
UTTracker	68.8%	–
Baseline	69.7%	–

Модели в таблице 3 сравниваются по значениям следующих метрик: AP@50; mAP@50:95.

AP@50 показывает среднюю точность модели. При этом правильно определённым объектом (также такие объекты называют true positives) считается тот, для которого модель верно определила класс и для которого значение IoU будет больше или равно 0.5 (для AP@60 нижний порог IoU был бы равен 0.6). В противном случае объект считается неверно определённым.

IoU (Intersection over Union) – это значение вычисляемое по следующей формуле:

$$IoU = \frac{\text{ППР}}{\text{ПОР}},$$

где ППР – площадь пересечения рамок;

ПОР – площадь объединения рамок.

В данном случае имеются ввиду две рамки. Первая рамка – это та рамка, которой выделила объект модель глубокого обучения. Вторая рамка («истинная») – это рамка, которой был выделен объект в обучающем наборе данных изначально.

Чем выше значение IoU , тем более точно модель глубокого обучения определила местонахождение объекта на изображении.

$mAP@50:95$ – метрика, которая подсчитывается по следующей формуле:

$$mAP@50:95 = \frac{AP@50 + AP@55 + \dots + AP@90 + AP@95}{10} = \frac{\sum_{x=0.5}^{0.95} AP@x}{10}.$$

$mAP@50:95$ также, как и $AP@50$, показывает среднюю точность модели, но при этом данная метрика более чувствительна к тому, насколько точно модель глубокого обучения позиционирует рамки для обнаруженных объектов.

Из данных таблицы 3 видно, что в плане точности лидирующей моделью является GELAN. Модель Yolo показывает среднюю точность относительно других сравниваемых моделей.

На основе вышеприведённых данных можно сделать вывод о том, что архитектура YOLO приемлема для использования в ПО, которое разрабатывается в рамках данной работы.

Архитектура моделей семейства YOLO включает в себя три основных абстрактных компонента: backbone, neck, head [14].

Backbone – это компонент, отвечающий за извлечение признаков из входного изображения. Таким образом формируются карты признаков (feature maps).

Neck проводит промежуточную обработку полученных карт признаков и таким образом формирует улучшенные карты признаков, которые впоследствии передаются в Head. Данная промежуточная обработка обычно включает в себя повышение разрешения (upsampling) и конкатенацию карт признаков из разных уровней.

Head получает данные от Neck и на основе них определяет, какие объекты находятся на изображении. Здесь для каждого найденного на изображении объекта определяется класс и ограничивающая рамка, показывающая местоположение объекта.

Таким образом можно сделать вывод, что процесс определения объектов на изображении включает две стадии: извлечение признаков; обнаружение объектов. Backbone и Neck отвечают за извлечение и улучшение признаков, а Head – за обнаружение объектов, на основе выявленных признаков.

На рисунке 1 показана схема архитектуры YOLOv11 [15].

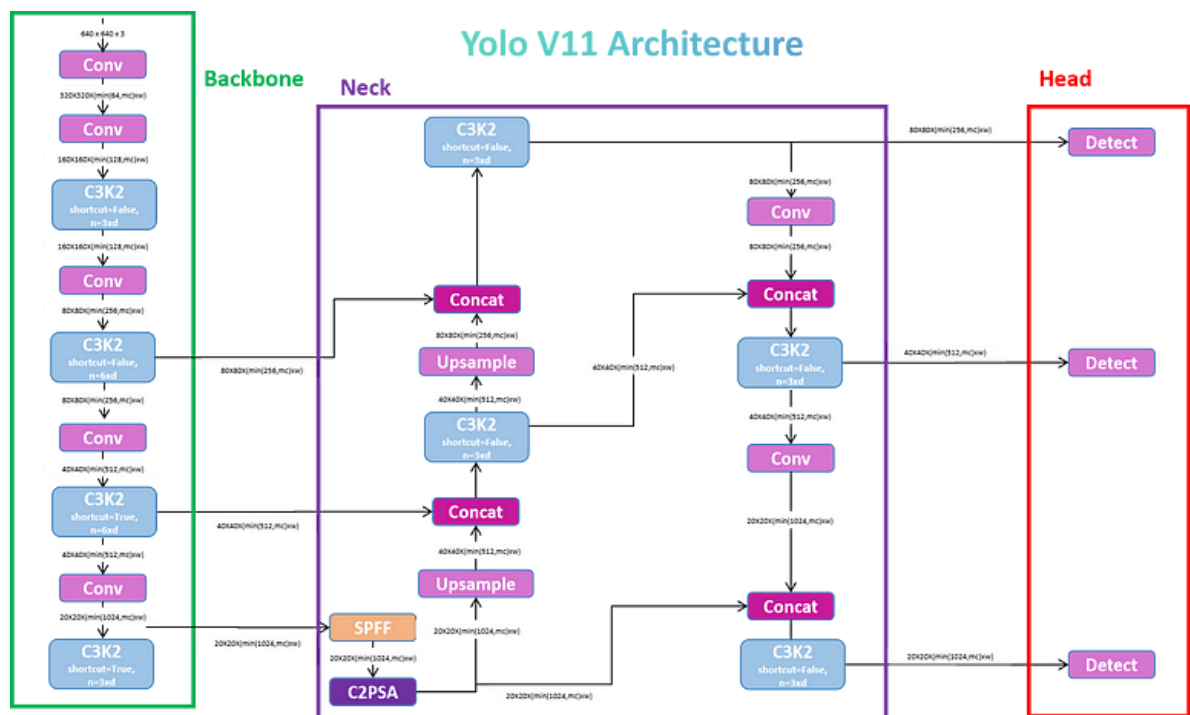


Рисунок 1 – Схема архитектуры YOLOv11

В рамках Backbone в YOLOv11 используются свёрточные слои (отмеченные на рисунке 1, как Conv), которые уменьшают разрешение исходного изображения и одновременно повышают количество каналов, которые хранят информацию о признаках.

Признаки (features) – это структурные элементы изображения, такие как: вертикальные линии; горизонтальные линии; круги и так далее. Чем глубже нейронная сеть, тем более сложные признаки она может распознавать.

В рамках YOLOv11 также был добавлен C3k2 блок, пришедший на смену C2f блоку, используемому ранее. Он также, как и свёрточные слои, предназначен для извлечения признаков.

SPPF (Spatial Pyramid Pooling Fast) – это модуль, который помогает захватывать признаки на разных масштабах, используя последовательные MaxPool-операции, объединяя их с исходным тензором. Это позволяет извлекать многоуровневые пространственные признаки из карт признаков, не изменяя пространственные размеры выходного тензора.

C2PSA (Cross Stage Partial with Spatial Attention) – это расположенный сразу после SPPF модуля блок, который вводит механизмы внимания, улучшающие фокусировку модели на важных областях изображения, таких как мелкие или частично закрытые объекты.

В рамках семейства моделей YOLO есть множество версий моделей. Среди этих моделей надо выбрать, какую именно использовать в данном случае.

YOLOv11 имеет следующие версии: nano; small; medium; large; extra-large. Данные версии отличаются друг от друга точностью определения объектов, скоростью обработки изображений и требованиями к вычислительным мощностям. Самой «лёгкой» и быстрой, а также наименее точной, является версия nano. Самой точной, но при этом наиболее медленной и требовательной, является версия extra-large.

На таблице 4 представлено ранжирование версий YOLO по таким характеристикам, как: точность; скорость; количество GFLOPs для обучения модели; размер обученной модели [16].

Таблица 4 – Ранжирование моделей YOLO

Версия	Ранг точности	Ранг скорости	Ранг GFLOPs	Ранг размера
YOLOv3u-tiny	28	1	6	11
YOLOv3u	20	24	28	28
YOLOv5un	27	6	2	4
YOLOv5us	24	7	8	9
YOLOv5um	17	15	13	18
YOLOv5ul	14	19	21	23
YOLOv5ux	17	27	26	27
YOLOv8n	26	5	4	5
YOLOv8s	23	9	11	10
YOLOv8m	15	17	16	20
YOLOv8l	13	22	22	22
YOLOv8x	8	26	27	26
YOLOv9t	20	12	3	1
YOLOv9s	7	15	10	6
YOLOv9m	4	21	15	15
YOLOv9c	9	25	19	19
YOLOv9e	12	28	24	25

Таблица 4 – Продолжение таблицы 4

YOLOv10n	25	2	5	3
YOLOv10s	19	3	9	7
YOLOv10m	5	10	12	12
YOLOv10b	9	12	18	14
YOLOv10l	11	17	20	17
YOLOv11n	22	3	1	2
YOLOv11s	16	8	7	8
YOLOv11m	1	11	14	13
YOLOv11l	3	14	17	16
YOLOv11x	5	19	25	24

Данные из таблицы 4 показывают нам, что наибольшей точностью среди всех моделей семейства обладает YOLO11m.

На основе вышеприведённых данных можно сделать вывод о том, что YOLO11m является наиболее сбалансированной и универсальной моделью в своём семействе и при этом обеспечивает хорошую точность распознавания изображений. Поэтому в качестве модели глубокого обучения для разрабатываемого в рамках данной работы ПО выбрана модель YOLO11m.

1.5 Анализ функциональных требований

Разрабатываемая программа должна выполнять следующие функции:

- выбрать точку на карте,
- задать размер области для обработки,
- обнаружить объекты на карте и в заданной области,
- найти ближайший населённый пункт и объекты жизнеобеспечения,
- очистить карту от маршрутов и обнаруженных объектов.

Для того чтобы наглядно показать общее устройство системы, была составлена диаграмма вариантов использования, представленная на рисунке 2.

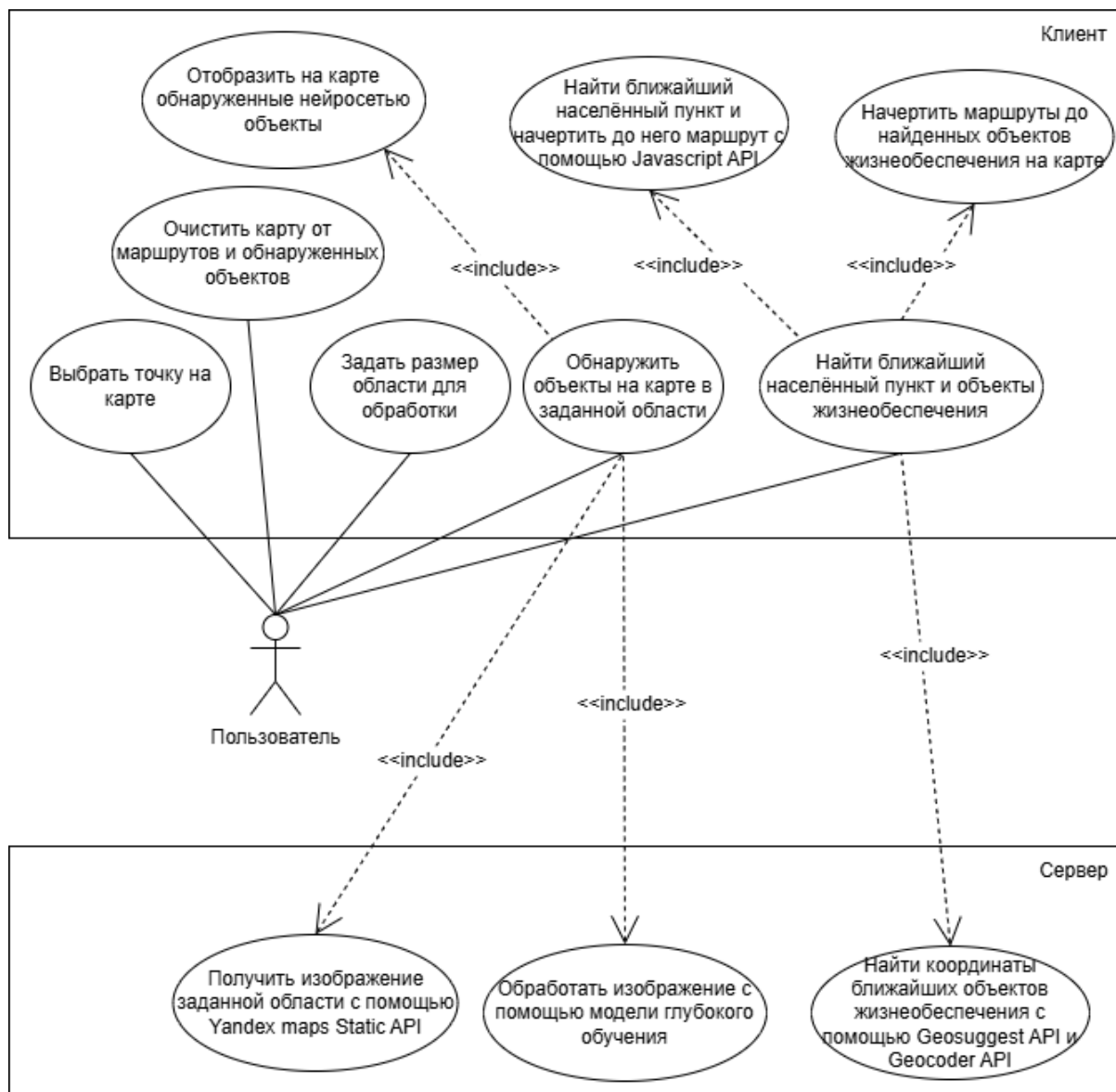


Рисунок 2 – Диаграмма вариантов использования

Для показанных на рисунке 2 вариантов использования были составлены сценарии, приведённые в приложении А.

1.6 Анализ нефункциональных требований

На данном этапе были составлены нефункциональные требования к разрабатываемому приложению. Данные требования включают в себя следующие:

- время реакции системы на действия пользователя должно быть не больше 1 секунды,
- размер готового ПО должен быть не больше 300 Мб,

- серверная часть программы должна запускаться на локальном сервере машины,
- клиентская часть программы должна запускаться с обычного браузера (например, с браузера Google Chrome) на том же компьютере, что и серверная часть,
- клиентская часть программы должна иметь простой и удобный интерфейс на русском языке,
- серверная часть программы должна запускаться на ОС Windows 11 x64.

Таким образом выглядят основные нефункциональные к разрабатываемому ПО.

1.7 Требования к интерфейсу

Интерфейс приложения, с которым будет взаимодействовать пользователь, должен быть доступен в виде веб-страницы, открываемой через браузер (например, Google Chrome или Microsoft Edge). Схема данного интерфейса представлена на рисунке 3.

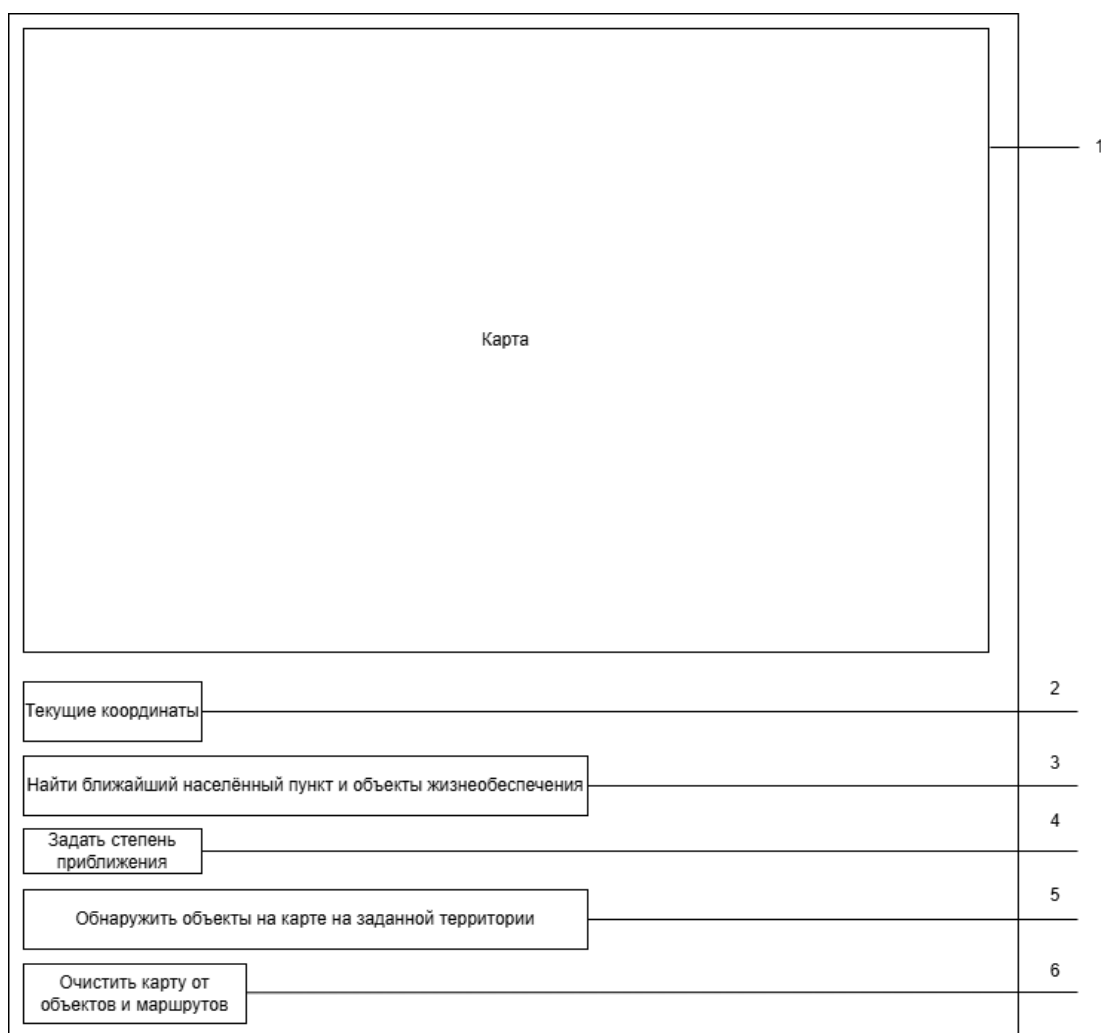


Рисунок 3 – Схема интерфейса

На рисунке 3 цифрами выделены следующие объекты:

- карта местности (обозначена цифрой 1),
- поле «Текущие координаты» (обозначено цифрой 2),
- кнопка «Найти ближайший населённый пункт и объекты жизнеобеспечения» (обозначена цифрой 3),
- поле для ввода «Задать степень приближения» (обозначено цифрой 4),
- кнопка «Обнаружить объекты на карте на заданной территории» (обозначена цифрой 5),
- кнопка «Очистить карту от объектов и маршрутов» (обозначена цифрой 6).

Поле для ввода «Задать степень приближения» принимает целочисленное значение, заданное пользователем в ограниченном разработчиком диапазоне.

Карта местности показывает непосредственно карту местности, на которой кликом можно выделить точку. Координаты точки, выделенной на карте, отображаются в поле «Текущие координаты».

При нажатии на кнопку «Обнаружить объекты на карте на заданной территории» запускается алгоритм обнаружения объектов на заданной территории (территория задаётся точкой на карте и степенью приближения) с помощью модели глубокого обучения. Объекты, обнаруженные алгоритмом, выделяются на карте местности.

При нажатии на кнопку «Найти ближайший населённый пункт и объекты жизнеобеспечения» запускается алгоритм поиска ближайших (относительно выделенной пользователем точки на карте) объектов жизнеобеспечения и ближайшего населённого пункта. Маршруты до найденных объектов отображаются на карте местности, а также на карте местности отображаются названия найденных объектов и расстояния до них.

При нажатии на кнопку «Очистить карту от объектов и маршрутов» с карты стираются все объекты, нарисованные на карте в ходе: обнаружения объектов на карте; поиска ближайшего населённого пункта и объектов жизнеобеспечения. Таким образом карта возвращается в исходное состояние.

1.7.1 Технологический стек

Для разработки приложения будут использованы следующие технологии:

- объектно-ориентированный подход (ООП),
- архитектура Model-View-Controller (MVC),
- модель глубокого обучения для распознавания объектов на изображениях,
- картографический онлайн-сервис.

ООП является очень распространённым подходом в разработке ПО на данный момент. Данный подход включает в себя три основных принципа: инкапсуляция; наследование; полиморфизм. Данные принципы обеспечивают

чёткую структурированность, модульность и расширяемость кода, что способствует легкой поддержке и адаптации программных систем к изменяющимся требованиям [17].

Архитектура MVC это широко используемый шаблон проектирования ПО, который особенно часто используется в разработке web-приложений. Данная архитектура предлагает выделить в проекте три сущности: модель данных (model); вид (view); контроллер (controller). Данная архитектура обеспечивает следующие преимущества проекту [18]:

- более гибкий код,
- возможность повторного использования каждой из трёх составных частей MVC,
- безболезненная замена и изменение каждого из элементов системы (другие алгоритмы расчета, способы хранения данных, визуальные представления и т. д.),
- скорость разработки,
- легкость поддержки кода.

Модели глубокого обучения являются мощным и разносторонним инструментом в области компьютерного зрения. Существует довольно много различных задач в данной области, как например: классификация изображений; обнаружение объектов; сегментация изображений; анализ позы. В большинстве случаев для каждой из вышеупомянутых задач найдётся эффективная архитектура нейросети, выдающая приемлемый результат.

Свёрточные нейронные сети (СНС) являются одним из наиболее известных инструментов в области компьютерного зрения. Они используются для обработки изображений и видео. Главным преимуществом СНС является способность автоматически выявлять сложные зависимости в изображениях в процессе обучения на больших объёмах данных [19].

К недостаткам данного инструмента можно отнести следующие проблемы [19]:

- для обучения моделей СНС требуются большие объёмы размеченных данных,
- сложные архитектуры могут требовать наличия мощных вычислительных ресурсов,
- существует проблема интерпретируемости моделей, что делает сложным понимание того, как принимаются решения на основе анализа изображений.

Картографический онлайн сервис предоставляет различные географические данные, необходимые для анализа местности при выборе места развёртывания мобильных ПВР.

Далее будут перечислены технические средства, с помощью которых будет реализовано, разрабатываемое в рамках данной работы ПО.

В качестве среды разработки выбрана «Microsoft Visual Studio». Это удобная и универсальная среда разработки, поддерживающая большое количество языков программирования и технологий (к которым можно отнести и различные нейросети).

Приложение создано на основе платформы «.NET» от Microsoft на языке программирования «C#». Платформа «.NET» является кроссплатформенной и предназначена для разработки самых разных типов приложений.

Также используется веб-фреймворк «ASP.NET», который является частью платформы «.NET». Данный фреймворк предназначен для разработки веб-приложений (в частности приложений на архитектуре MVC), а также он хорошо интегрирован с другими технологиями от Microsoft.

Для создания веб страниц в рамках «ASP.NET» используется движок представлений Razor, созданный для работы с файлами в формате «.cshtml». Одной из особенностей Razor является возможность использования вставок кода на «C#» при создании HTML-страниц.

Для получения картографических данных используется сервис «Яндекс карты» (см пункт 1.3).

Для обнаружения объектов на изображениях местности используется модель глубокого обучения на архитектуре YOLO11m (см. пункт 1.4).

Для управления версиями кода выбрана платформа GitHub.

Таким образом, можно сделать следующие выводы.

Среди найденных примеров систем поддержки принятия решений, использующихся в чрезвычайных ситуациях, не было упоминаний об использовании нейросетей для обработки данных. Разработанная же модель рассчитана на использование модели глубокого обучения для обнаружения объектов на изображении.

Для реализации проекта в ходе анализа моделей глубокого обучения и картографических веб технологий было принято решение использовать модель глубокого обучения YOLOv11m и картографический онлайн сервис Яндекс карты.

Разработанный проект основывается на готовых решениях для снижения времени разработки и стоимости.

2 Построение моделей и алгоритмов

2.1 Графическое представление алгоритма обнаружения объектов на карте с помощью нейросети

Графическое представление алгоритма обнаружения объектов на карте с помощью нейросети представлено на рисунке 4.

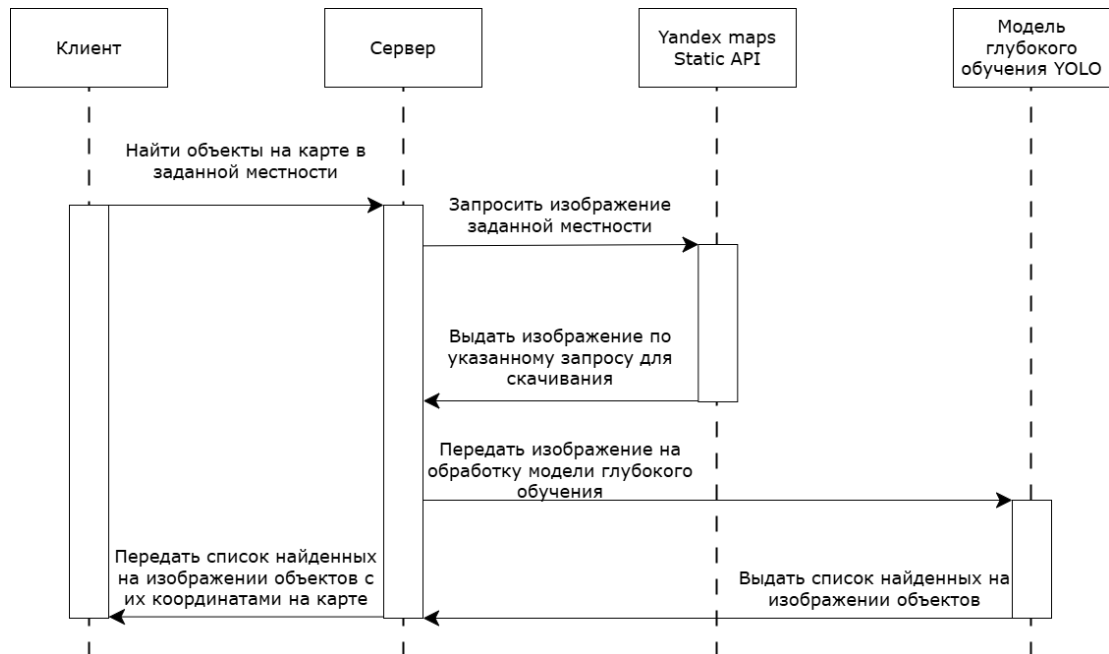


Рисунок 4 – Графическое представление алгоритма обнаружения объектов на карте с помощью нейросети

Теперь подробнее опишем алгоритм, представленный на рисунке 4. Сначала пользователь в клиентской части приложения задаёт географические координаты местности и приближение (оно влияет на площадь территории, которую будет просматривать модель глубокого обучения). После этого пользователь нажимает на кнопку, запускающую алгоритм обнаружения объектов на карте с помощью нейросети.

Клиент формирует запрос, включающий в себя заданные пользователем координаты территории и приближение, и отправляет его на локальный сервер.

Сервер формирует запрос к Yandex maps Static API, чтобы получить изображение заданной пользователем местности и отправляет его. Static API возвращает запрошенное изображение и сервер загружает его.

Далее скачанное изображение передаётся на обработку модели глубокого обучения YOLO. На выходе получается список обнаруженных объектов. Каждый элемент в списке включает название класса объекта и координаты прямоугольника, показывающего расположение объекта на изображении.

На основе полученных данных сервер формирует список обнаруженных объектов и вычисляет географические координаты объектов, обнаруженных нейросетью.

Список объектов с их географическими координатами передаётся клиенту приложения в виде JSON файла.

Клиент получает JSON файл и на основе данных, содержащихся в нём, рисует на карте, отображаемой перед пользователем, прямоугольники, выделяющие обнаруженные объекты.

Таким образом осуществляется алгоритм обнаружения объектов на карте с помощью нейросети.

2.2 Графическое представление алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения

Для того чтобы подобрать наилучшее место для развёртывания мобильного ПВР важно знать, насколько близко и где находятся ближайший населённый пункт и объекты жизнеобеспечения. К числу важных объектов жизнеобеспечения можно, например, отнести больницы и пожарные части. Для того чтобы определять, где находятся вышеперечисленные объекты и каков маршрут до них, разработан специальный алгоритм.

Графическое представление алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения представлено на рисунке 5.

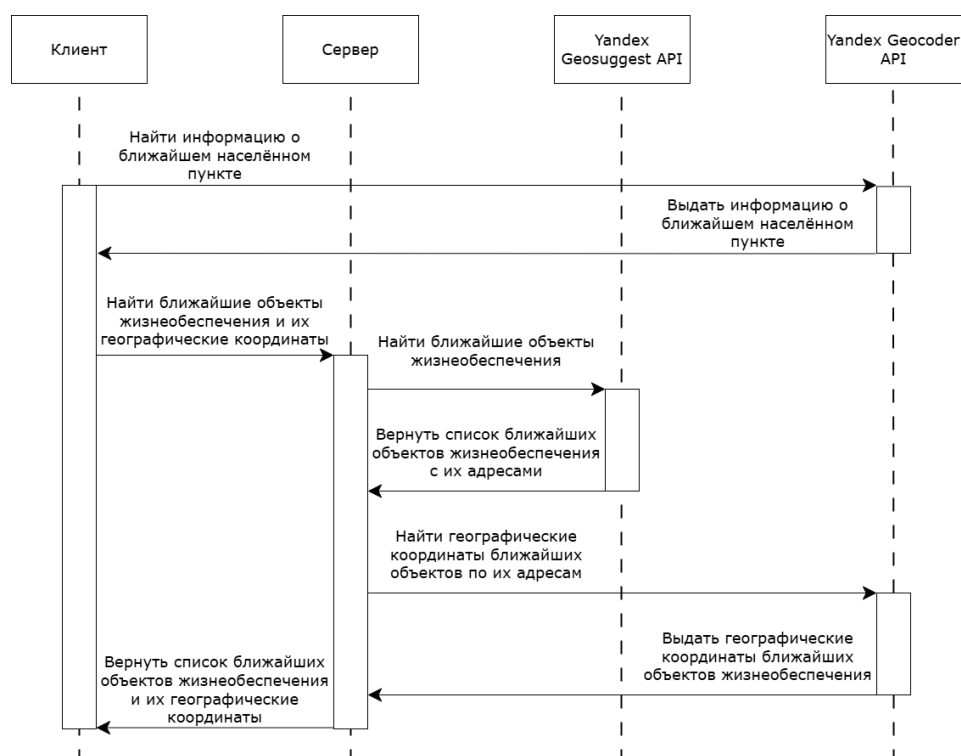


Рисунок 5 – Графическое представление алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения

Теперь подробнее опишем алгоритм, представленный на рисунке 5. Пользователь в клиенте задаёт координаты точки на карте. Далее клиент с помощью Yandex Geocoder находит координаты и название ближайшего населённого пункта.

Потом клиент формирует запрос к серверу, чтобы найти ближайшие объекты жизнеобеспечения и их координаты.

Сервер отправляет запросы Yandex Geosuggest API, чтобы найти ближайшие объекты жизнеобеспечения каждого типа. Geosuggest API возвращает список объектов жизнеобеспечения каждого типа и их адреса, а также прочую информацию о них (например, название объекта и расстояние до него). Для каждого типа объектов жизнеобеспечения Geosuggest API возвращает до 10 вариантов организаций. Например, если задать запрос на 10 больниц находящихся в пределах определённой области, то Geosuggest API выдаст список из 10 или менее больниц в заданной области с различной информацией о данных больницах (включая название организации, адрес организации и расстояние до неё).

Geosuggest API не выдаёт географические координаты объектов, поэтому сервер формирует запрос для Yandex Geocoder API, чтобы найти координаты объектов по их адресам и отправляет его. Geocoder API возвращает серверу информацию по запрошенным объектам, включая географические координаты объектов.

Сервер формирует ответ для клиента, содержащий список объектов жизнеобеспечения и их названия, адрес, координаты и т. д. Данный ответ отправляется клиенту, который отображает найденные объекты жизнеобеспечения и населённый пункт. Также клиент отображает маршруты до найденных объектов и расстояния до них.

На этом завершается выполнение алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения.

Таким образом, можно сделать следующие выводы.

Разработаны алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм поиска ближайшего населённого пункта и объектов жизнеобеспечения. Для демонстрации разработанных алгоритмов составлены графические схемы.

3 Реализация моделей и алгоритмов

3.1 Обучение модели YOLOv11

Для того, чтобы обучить модель YOLO был создан набор данных (далее датасет), включающий в себя аннотированные изображения разных мест на карте. Данные изображения были получены с помощью сервиса Yandex maps Static API. Каждое изображение имеет разрешение 450x450. Аннотации для изображений были созданы с помощью программы Labelme.

На рисунке 6 показано одно из изображений созданного датасета.



Рисунок 6 – Изображение из датасета

На рисунке 7, продемонстрировано аннотированная версия изображения, представленного на рисунке 6.

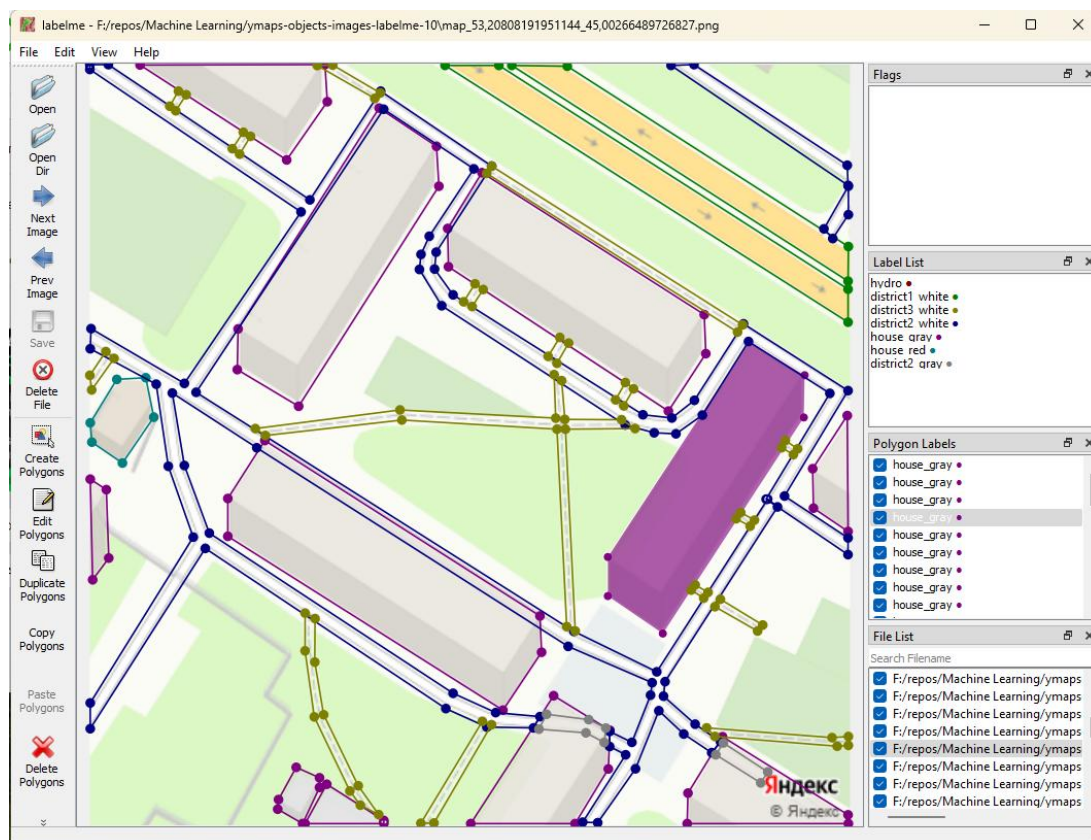


Рисунок 7 – Аннотированное изображение из датасета

Для каждого аннотированного изображения составляется JSON файл, в котором для каждого объекта, отмеченного на изображении, записан класс и координаты точек, формирующих многоугольник, которым отмечен объект. Класс и многоугольник в данном случае формируют метку объекта.

На рисунке 8 представлен фрагмент JSON файла с аннотацией изображения, где описана метка одного из объектов с классом `house_gray` (даным классом отмечаются жилые дома на карте).

```

{
  "label": "house_gray",
  "points": [
    [
      29.716981132075468,
      0.23584905660377353
    ],
    [
      116.50943396226415,
      56.367924528301884
    ],
    [
      140.56603773584905,
      21.933962264150942
    ],
    [
      139.1659975913288,
      0.0
    ]
  ],
  "group_id": null,
  "description": "",
  "shape_type": "polygon",
  "flags": {}
},

```

Рисунок 8 – Метка объекта

В созданном наборе данных были выделены следующие классы объектов:

- hydro,
- district1_white,
- district2_white,
- district3_white,
- district2_gray,
- district3_gray,
- house_gray,
- house_red.

Класс hydro обозначает водоёмы. Классы district1_white, district2_white, district3_white, district2_gray, district3_gray обозначают различные виды улиц. Классы house_gray, house_red обозначают различные виды домов.

На рисунке 9 показано, как выглядят на картах объекты классов district1_white, district2_white, district3_white, district2_gray, district3_gray, house_gray, house_red.

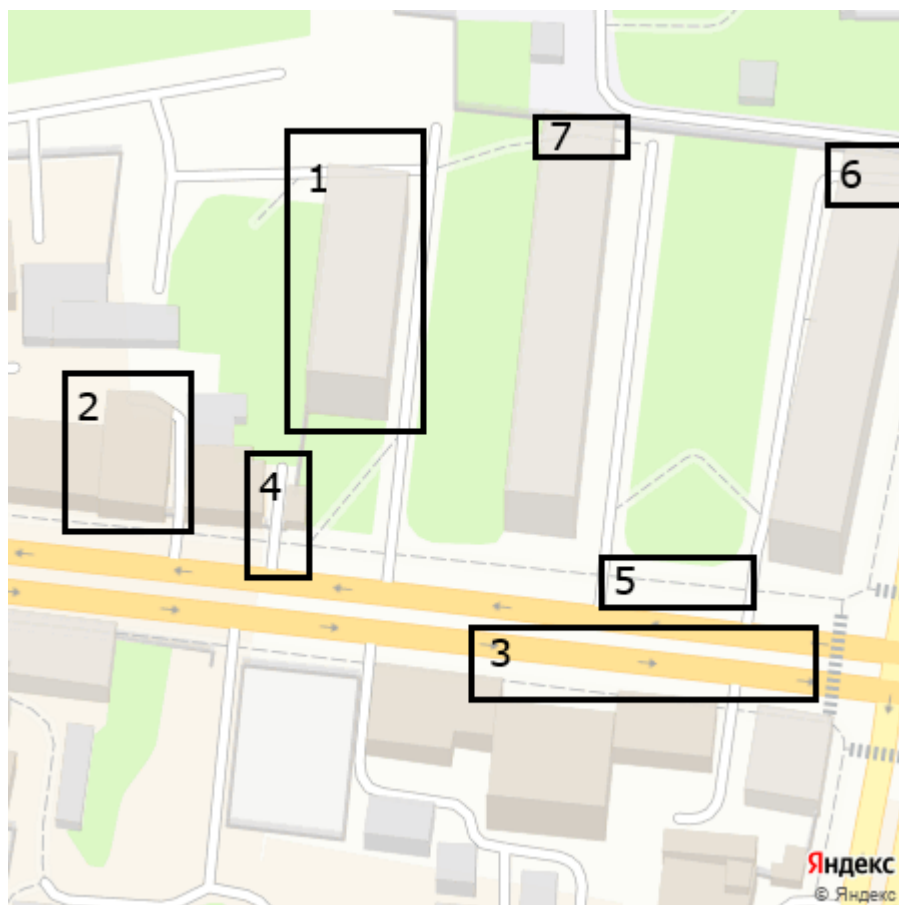


Рисунок 9 – Примеры объектов различных классов на карте

Теперь укажем какие классы объектов под какими номерами показаны на рисунке 9, а также подробнее их опишем.

Под номером 1 показан объект класса `house_gray`. Класс `house_gray` обозначает дома, которые на карте закрашены серым цветом.

Под номером 2 показан объект класса `house_red`. Класс `house_red` обозначает дома, которые на карте закрашены красным цветом.

Под номером 3 показан объект класса `district1_white`. Данный класс обозначает большие дороги (как правило четырёхполосные или даже более широкие), которые на Яндекс картах закрашены разными оттенками жёлтого цвета.

Под номером 4 показан объект класса `district2_white`. Данный класс обозначает небольшие дороги (двухполосные или более узкие), по которым как правило могут ездить машины.

Под номером 5 показан объект класса `district3_white`. Данный класс обозначает небольшие дороги, которые обычно являются пешеходными.

Под номером 6 показан объект класса `district2_gray`. Данный класс обозначает те же дороги, что и класс `district2_white`. Разница между этими классами в том, что `district2_gray` отмечает дороги, перекрытые на изображении карты домами. Дома на карте полупрозрачные и поэтому дороги просвечивают через них.

Под номером 7 на карте показан объект класса `district3_gray`. Данный класс по своей логике аналогичен классу `district2_white`.

На рисунке 11 выделен чёрной рамкой объект класса `hydro`.

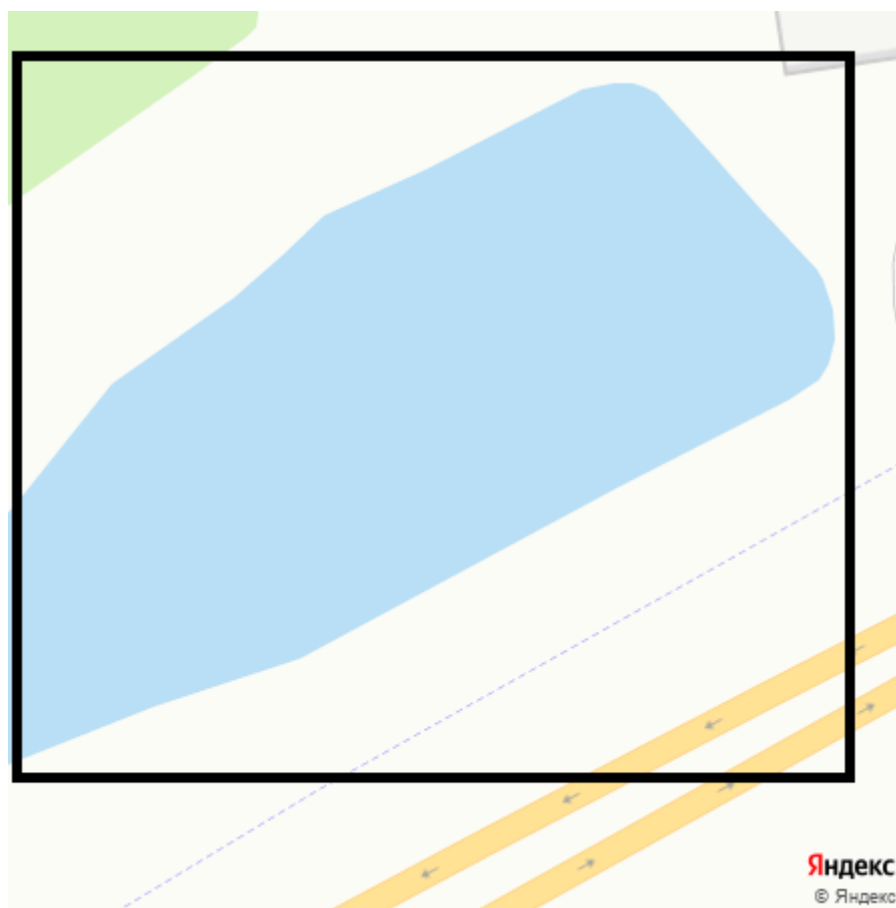


Рисунок 11 – Пример объекта класса `hydro`

Класс `hydro` обозначает водоёмы, моря, озёра и так далее.

Таким образом был создан датасет для обучения модели YOLO. Модель была обучена с помощью сервиса Google Colab. На листинге 1 показан код, с помощью которого в Google Colab была обучена модель YOLO.

Листинг 1 – Код для обучения модели YOLO в Google Colab

```
!pip install ultralytics
from ultralytics import YOLO
import os
import shutil

from google.colab import drive

drive.mount('/content/drive/')

#Выбор начальной модели для обучения
model = YOLO("yolo11m.pt")

import yaml

# Указываем путь к уже существующему YAML файлу
yaml_path =
'/content/drive/MyDrive/MachineLearning/Datasets/yolov11-with-
labelme/data.yaml'

# Читаем YAML-файл
with open(yaml_path, 'r') as f:
    data_config = yaml.safe_load(f)

# Запускаем процесс обучения
model.train(data='/content/drive/MyDrive/MachineLearning/Datasets/
ymaps_labelme/data.yaml', epochs=100, imgsz=640, batch=8,
workers=4)

%load_ext tensorboard
%tensorboard --logdir runs/train

# Сохраняем обученную модель
model.save('/content/drive/MyDrive/MachineLearning/Datasets/ymaps_
labelme/yolov11m_trained.pt')
```

Таким образом была обучена модель YOLO. На рисунке 12 продемонстрировано изображение, на котором обученная нейросеть обнаружила объекты.

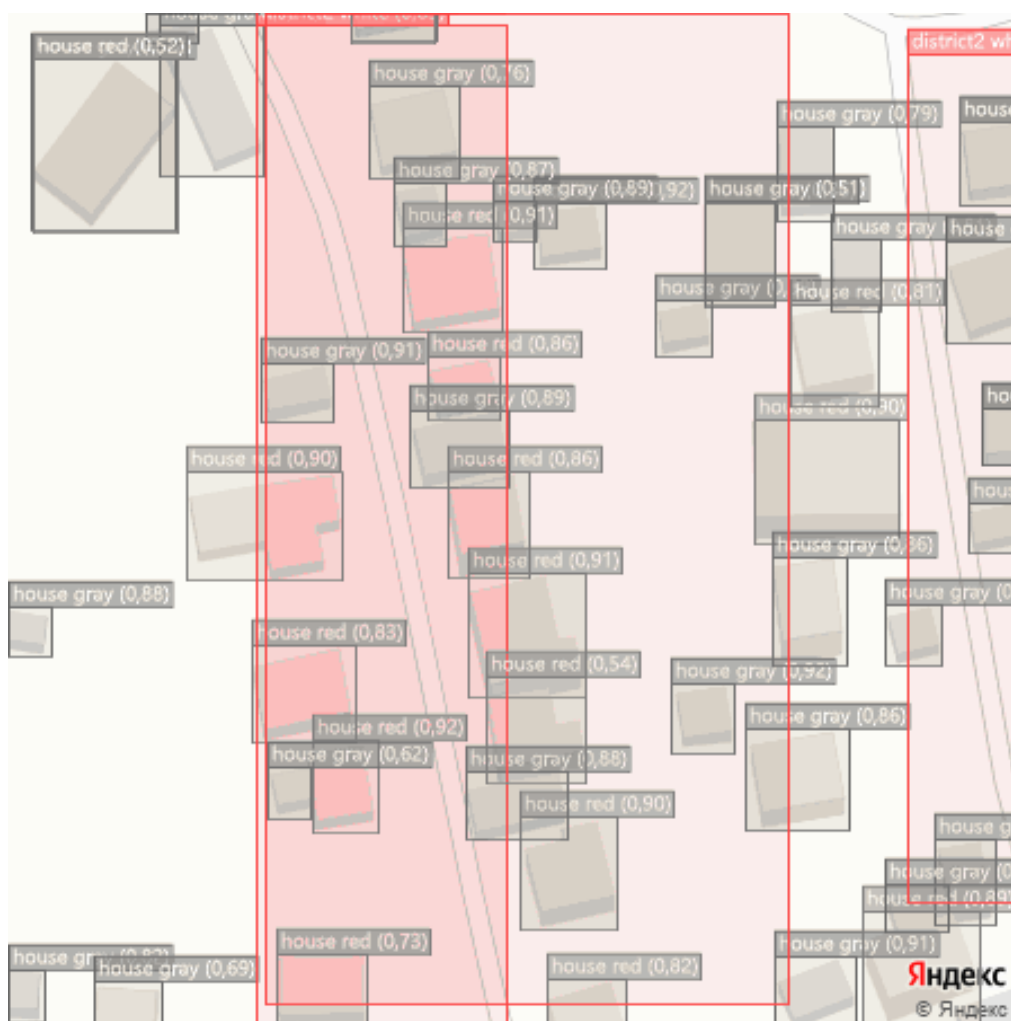


Рисунок 12 – Демонстрация работы обученной модели

Таким образом был составлен датасет для обучения и была обучена модель YOLO для детекции объектов на изображении.

3.2 Реализация алгоритма вычисления географических координат для ограничивающих рамок, показывающих объекты, обнаруженные моделью YOLO

Ранее (см. раздел 2.1) в рамках описания алгоритма обнаружения объектов на карте с помощью нейросети было упомянуто, что для объектов, которые обнаруживаются нейросетью надо высчитывать географические

координаты. Теперь рассмотрим детальнее, как именно в программе вычисляются географические координаты обнаруженных объектов.

Итак, имеется изображение разрешением 450x450. Известны географические координаты точки в центре данного изображения, а также известны пиксельные координаты, задающие ограничивающие рамки для объектов, обнаруженных моделью глубокого обучения. Если получится вычислить координаты одного любого пикселя на имеющемся изображении, то будет несложно вычислить координаты ограничивающих рамок для всех обнаруженных объектов.

В сервисе Яндекс карты используется проекция Меркатора. Опишем подробнее основные особенности данной проекции.

Меридианы в проекции Меркатора представляются параллельными равноотстоящими линиями. Параллели же представляют собой параллельные линии, расстояние между которыми вблизи экватора равно расстоянию между меридианами и быстро увеличивается при приближении к полюсам. Обычно карту в данной проекции ограничивают областями $80 - 85^\circ$ северной и южной широты [20].

Yandex Static API выдаёт изображения в проекции Меркатора. Если взять изображение в любой точке на карте с зумом равным 18 и разрешением 450x450 (далее будем называть это изображение базовым), то окажется, что данное изображение в ширину всегда имеет примерно 0,00243 градуса долготы, а в высоту количество градусов широты будет зависеть от широты местности.

Например, если взять изображение на экваторе, то базовое изображение будет иметь в высоту 0,00243 градуса широты. В данном случае получается, что в высоту и ширину базовое изображение вмещает в себя одинаковое количество градусов широты и долготы.

Однако, чем дальше от экватора мы будем замерять данные значения, тем меньше градусов широты будет уместиться по высоте в базовое изображение. Например, если взять изображение из города Лабытнанги, который находится на широте 66.65° северной широты, то данное изображение будет вмещать по

высоте 0.00097° широты и по ширине $0,00243^\circ$ долготы. Данное явление обуславливается проекцией Меркатора.

Дело в том, что градус широты всегда равен примерно 111 км на поверхности Земли, а градус долготы равен примерно 111 км на экваторе и становится всё меньше и меньше по мере приближения к одному из полюсов нашей планеты. Данное явление видно на рисунке 13.

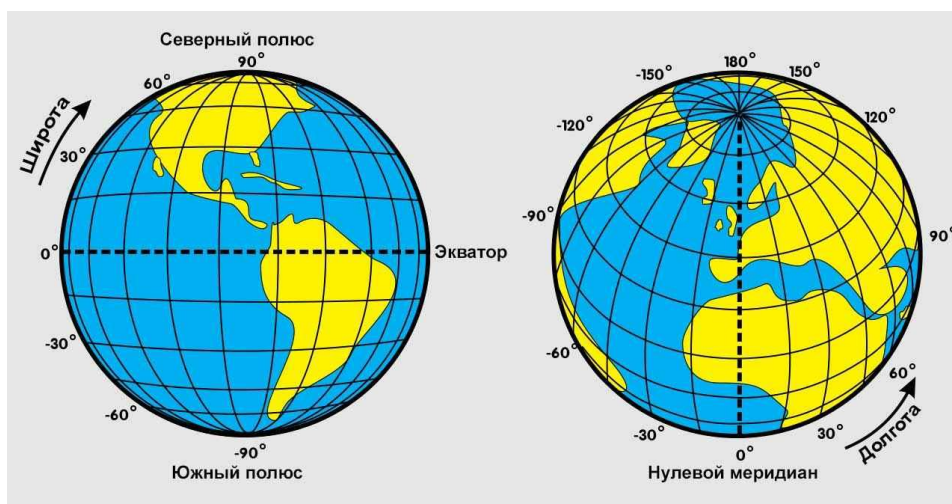


Рисунок 13 – Модель Земли, на которой изображены долгота и широта

На рисунке 14 продемонстрирована проекция Меркатора.

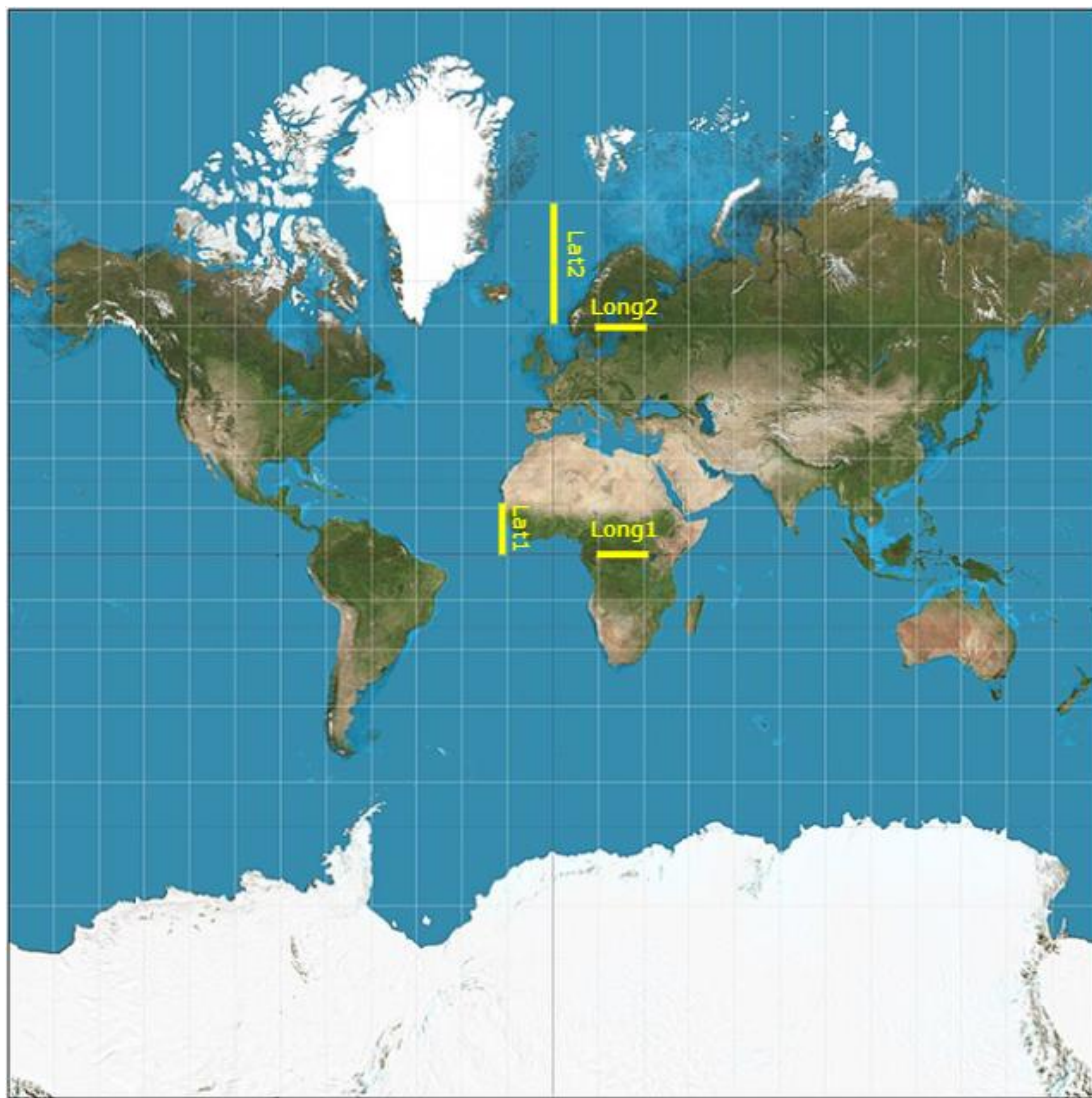


Рисунок 14 – Проекция Меркатора

На рисунке 14 жёлтым цветом выделены четыре отрезка Long1, Long2, Lat1, Lat2.

Отрезки Long1 и Long2 размещены на двух разных параллелях и имеют одинаковую длину на изображении. Фактически данные отрезки также вмещают в себя два одинаковых количества градусов долготы. Однако если считать длины данных отрезков в километрах, то окажется, что Long1 будет длиннее Long2.

Отрезки Lat1 и Lat размещены на двух разных меридианах и имеют разную длину на изображении. С другой стороны, данные два отрезка равны друг другу по количеству градусов широты, которые они в себя вмещают, а также они равны одинаковому количеству километров.

Также стоит отметить, что все четыре отрезка имеют длину в 15° и пары отрезков Lat1, Long1 и Lat2, Long2 формируют на карте прямоугольные секторы размерами 15° на 15°.

Видно, что участок, который формируется отрезками Lat1 и Long1 на рисунке занимает меньшую площадь, чем участок, формируемый отрезками Lat2, Long2. Однако на самом деле первый участок занимает большую площадь, если считать её в квадратных километрах на поверхности Земли.

Также видно, что чем дальше от экватора находится местность, тем визуально длиннее становится широта на проекции Меркатора. Соотношение длин широты и долготы на карте в пикселях равно соотношению реальных длин широты и долготы на поверхности Земли и описать его можно следующей формулой:

$$\frac{LatLengthPx}{LongLengthPx} = \frac{LatLength}{LongLength}, \quad (1)$$

где $LatLengthPx$ – длина градуса широты в пикселях;

$LongLengthPx$ – длина градуса долготы в пикселях;

$LatLength$ – реальная длина градуса широты на поверхности Земли в единицах СИ;

$LongLength$ – реальная длина градуса долготы на поверхности Земли в единицах СИ.

Для вычисления высоты базового изображения в градусах широты была составлена следующая формула:

$$LatHeight = \frac{LongWidth}{LengthLat} * \cos\left(\frac{Lat * \pi}{180}\right) * \frac{\pi * 6378137 * 0.001}{180}, \quad (1)$$

где $LongWidth$ – это количество градусов долготы, которое вмещает базовое изображение по ширине;

LengthLat – это длина в метрах 0.001 градуса широты;

Lat – это широта, соответствующая точке в центре базового изображения.

Для того чтобы нарисовать объекту ограничивающую рамку на карте достаточно знать координаты двух точек. В данном случае мы будем рассчитывать координаты левого верхнего угла рамки и правого нижнего угла рамки. Обозначим долготу и широту левого верхнего угла рамки, как *X1* и *Y1* соответственно. Долготу и широту правого нижнего угла рамки обозначим, как *X2* и *Y2* соответственно.

После обработки изображения моделью YOLO пиксельные координаты ограничивающей рамки каждого объекта задаются следующими параметрами: координата *X* левого верхнего угла рамки; координата *Y* левого верхнего угла рамки; ширина рамки; высота рамки.

Долгота левого нижнего угла рамки считается по следующей формуле:

$$X1 = longitude + LongWidth * \left(\frac{ImageX}{450} - \frac{1}{2} \right), \quad (2)$$

где *longitude* – долгота точки в центре базового изображения;

LongWidth – та же самая величина, что в формуле 1;

ImageX – пиксельная координата левого верхнего угла рамки.

Широта левого нижнего угла рамки считается по следующей формуле:

$$Y1 = latitude + LatHeight * \left(\frac{450 - ImageY}{450} - \frac{1}{2} \right), \quad (3)$$

где *latitude* – широта точки в центре базового изображения;

LatHeight – это количество градусов широты, которое вмещает в себя базовое изображение по высоте;

ImageY – пиксельная координата левого верхнего угла рамки.

Долгота правого верхнего угла рамки считается по следующей формуле:

$$X2 = X1 + LongWidth * \left(\frac{ImageWidth}{450} \right), \quad (4)$$

где $LongWidth$ – та же самая величина, что в формуле 1;

$X1$ – долгота левого нижнего угла рамки;

$ImageWidth$ – ширина ограничивающей рамки в пикселях.

Широта правого верхнего угла рамки считается по следующей формуле:

$$Y2 = Y1 - LatHeight * \left(\frac{ImageHeight}{450} \right), \quad (5)$$

где $LatHeight$ – та же самая величина, что в формуле 3;

$Y1$ – широта левого нижнего угла рамки;

$ImageHeight$ – высота ограничивающей рамки в пикселях.

Таким образом для ограничивающей рамки каждого обнаруженного нейросетью объекта высчитываются географические координаты.

3.3 Программная реализация модели и алгоритмов

Для разработанного приложения была создана диаграмма компонентов, представленная на рисунке 15.

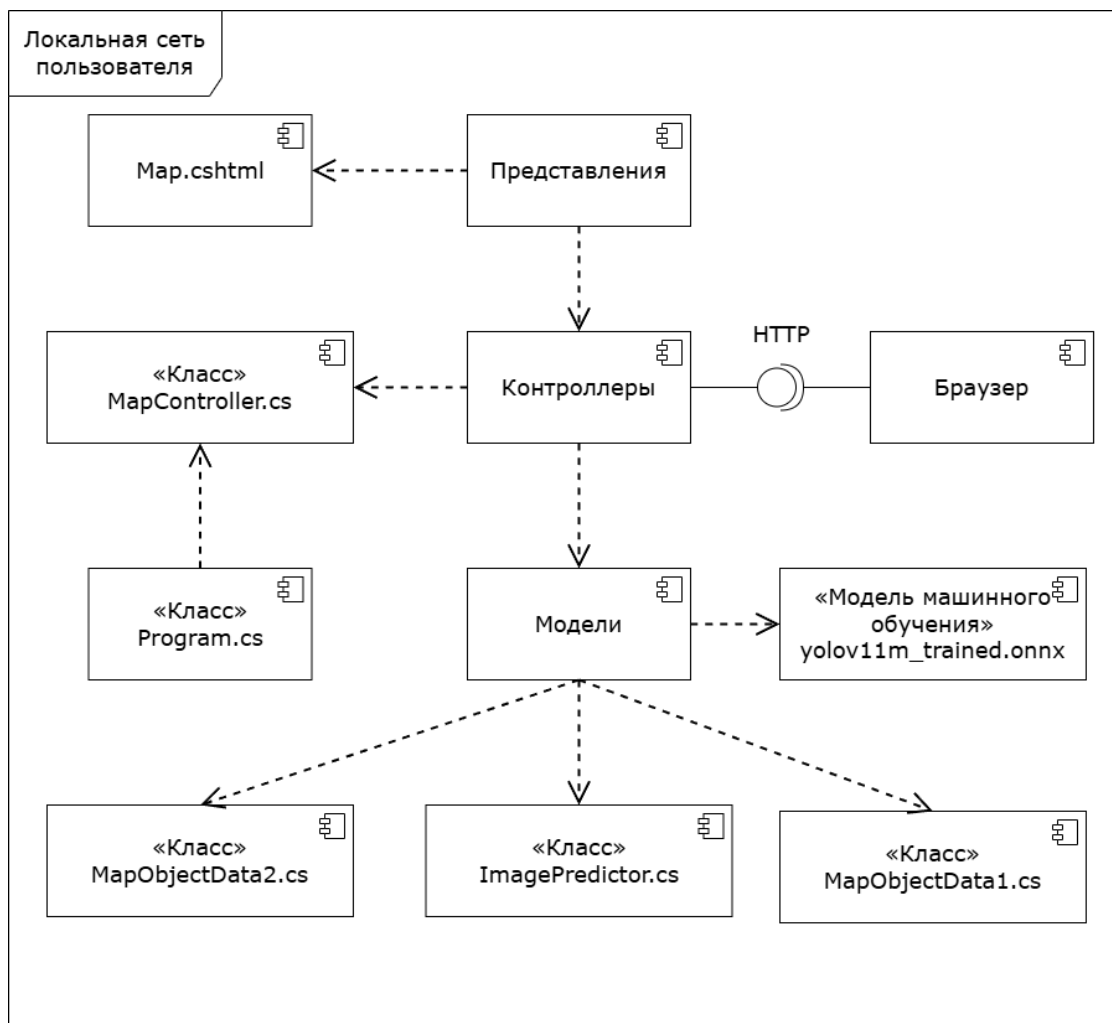


Рисунок 15 – Диаграмма компонентов

Разработанное приложение разворачивается на локальном сервере, доступ к которому можно получить с помощью веб-браузера.

Так как приложение разработано на основе MVC архитектуры, то оно включает в себя три абстрактных компонента: модели; представления; контроллеры.

Браузер взаимодействует с контроллером приложения посредством http-запросов и в качестве ответа получает информацию на html-страницах.

На рисунке 16 представлена диаграмма классов, составленная для разработанного приложения.

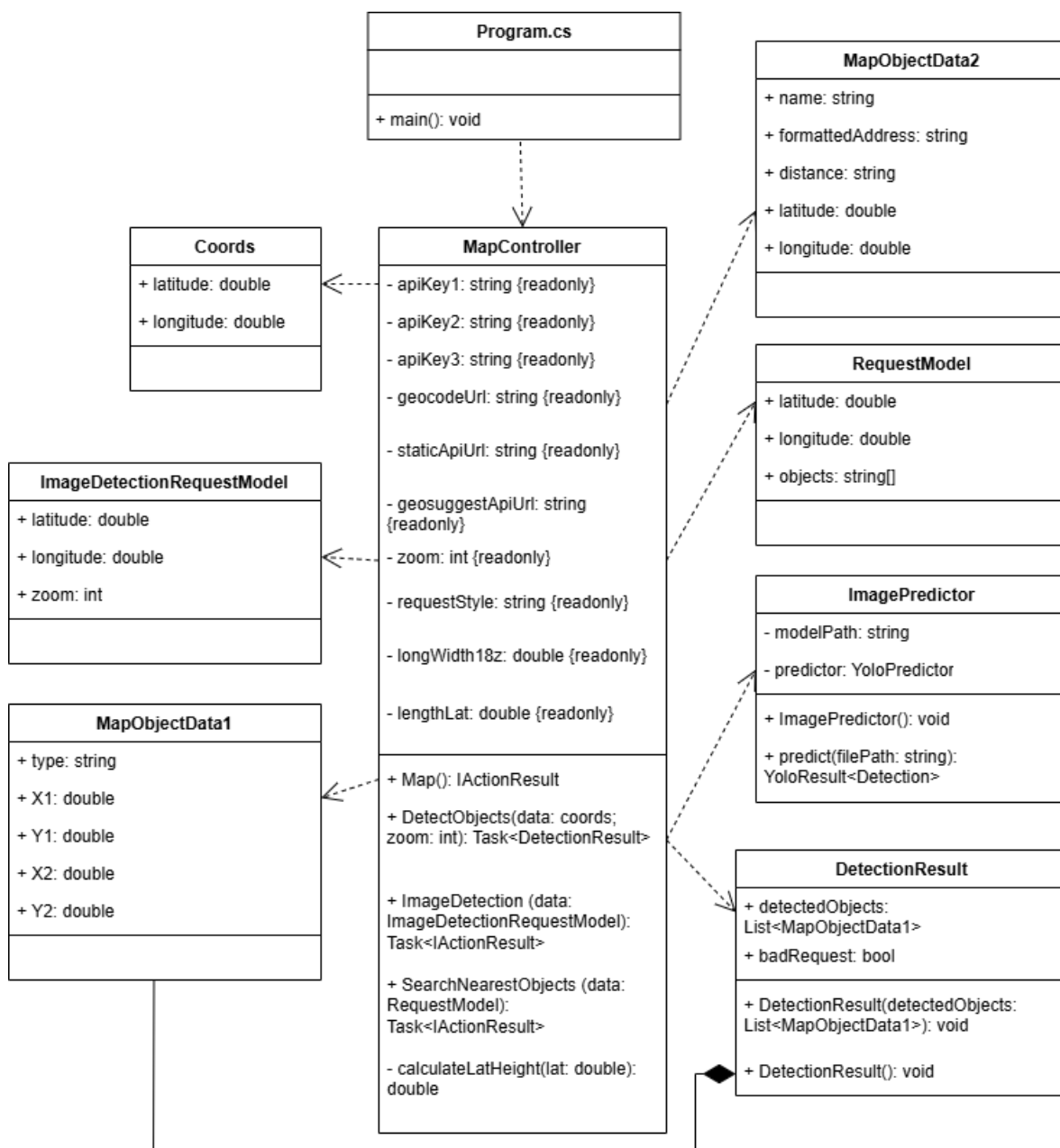


Рисунок 16 – Диаграмма классов

Класс «Program.cs» отвечает за инициализацию приложения на сервере с помощью «WebApplicationBuilder», который в свою очередь является частью «ASP.NET».

Класс «MapController.cs» описывает единственный контроллер приложения. В данном классе описаны основные алгоритмы приложения, такие как SearchNearestObjects и ImageDetection (которые отвечают за поиск объектов жизнеобеспечения и детекцию объектов на изображениях соответственно).

Классы «ImageDetectionRequestModel» и «RequestModel» описывают модели запроса к методам «ImageDetection» и «SearchNearestObjects» из контроллера «MapController» соответственно. Модели запроса описывают, какие данные надо передать контроллеру при обращении к конкретному методу. Рассмотрим это на примере класса «ImageDetectionRequestModel», код которого продемонстрирован в листинге 2.

Листинг 2 – Код класса ImageDetectionRequestModel

```
public class ImageDetectionRequestModel
{
    public double latitude { get; set; }
    public double longitude { get; set; }
    public int zoom { get; set; }
}
```

Класс «ImageDetectionRequestModel» содержит информацию о долготе и широте места, где надо обнаружить объекты. Также данный класс содержит поле, где должна храниться информация о приближении, которое должно быть задано при извлечении изображения из Yandex Static API, чтобы таким образом косвенно задать масштабы территории для обнаружения объектов. Таким образом, когда мы обращаемся к методу «ImageDetection», то мы должны передать ему долготу, широту и приближение.

Класс «Coords» описывает структуру, хранящую географические координаты. Данная структура используется в методах класса «MapController».

Класс «MapObjectData1» описывает структуру, которая хранит информацию об одном обнаруженном нейросетью объекте в рамках метода «ImageDetection» из класса «MapController». С помощью данной структуры метод «ImageDetection» формирует целый список обнаруженных объектов в формате JSON в результате своей работы. Код класса «MapObjectData1» представлен на Листинге 3.

Листинг 3 – Код класса MapObjectData1

```
public class MapObjectData1
{
```

```

    public string type { get; set; }
    public double X1 { get; set; }
    public double Y1 { get; set; }
    public double X2 { get; set; }
    public double Y2 { get; set; }
}

```

По информации из листинга 3 можно сделать вывод о том, что метод «ImageDetection» записывает о каждом найденном объекте следующую информацию: тип объекта; координаты X и Y левого нижнего угла ограничивающей рамки объекта; координаты X и Y правого верхнего угла ограничивающей рамки объекта.

Класс «MapObjectData2» описывает структуру, которая хранит информацию об одном найденном объекте жизнеобеспечения в методе «SearchNearestObjects» из класса «MapController». Метод «SearchNearestObjects» аналогично методу «ImageDetection», описанному выше, составляет список ближайших объектов жизнеобеспечения разных типов с помощью структуры, описанной в классе «MapObjectData2». Данная структура хранит информацию о следующих характеристиках объектов жизнеобеспечения: название организации; адрес организации; расстояние до организации относительно указанной пользователем точки на карте; широта; долгота.

Класс «ImagePredictor» описывает взаимодействие с моделью глубокого обучения. Данный класс вызывается в методе «ImageDetection» из класса «MapController» для того, чтобы обработать изображения и обнаружить на них объекты.

Файл «Map.cshtml» описывает HTML-страницу, с которой в браузере будет взаимодействовать пользователь. Данная страница показана на рисунке 17.

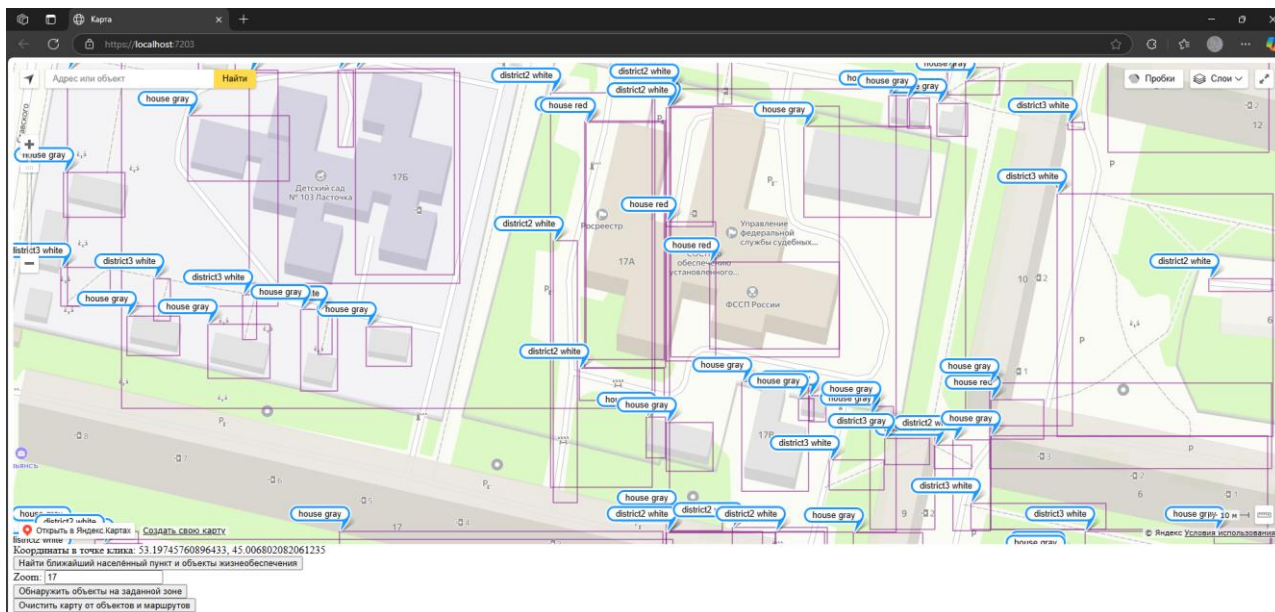


Рисунок 17 – Главная HTML-страница приложения

Полный исходный код программы можно посмотреть в репозитории github [21].

3.4 Тестирование реализации

3.4.1 Расчёт метрик кода

Одним из методов для оценки качества написанного кода является подсчёт метрик. Microsoft Visual Studio предоставляет возможность считать некоторые метрики автоматически. Данные метрики включают в себя следующие [22, 23]:

- индекс удобства поддержки (maintainability index – MI),
- сложность организации циклов (cyclomatic complexity – CyC),
- глубина наследования (depth of inheritance – DoI),
- взаимозависимость классов (class coupling – ClCo),
- строки исходного кода (lines of source code – LoS),
- строки исполняемого кода (lines of executable code – LoE).

Индекс удобства поддержки может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода. Значения метрики 0 – 9 показывают, что код сложно поддерживать. Значения метрики 10 – 19 показывают, что код умеренно удобно поддерживать. Значения выше 19

показывают, что код удобно поддерживать. Таким образом получается, что чем выше значение индекса удобства поддержки, тем лучше.

Сложность организации циклов показывает, насколько много управляющих конструкций (таких как if, switch, for) используется в коде. Чем выше значение данной метрики, тем сложнее поддерживать код. Рекомендуется поддерживать значение метрики в пределах 10. Однако это правило не строгое и можно устанавливать предел по своему усмотрению.

Глубина наследования показывает для каждого класса, какой он по счету в цепочке наследования. Чем выше значение данной метрики, тем больше вероятность ошибок в коде. С другой стороны, чем выше глубина наследования, тем больший потенциал для повторного использования кода через наследование. Для данной метрики рекомендуется поддерживать значение не больше 5.

Взаимозависимость классов показывает для каждого блока кода сколько уникальных классов в нём используется. Чем выше это число, тем больше зависимостей в коде и тем его сложнее поддерживать. Рекомендуется поддерживать значение данной метрики в пределе 9.

Строки исходного кода показывают точное количество строк исходного кода, присутствующих в исходном файле, включая пустые строки.

Строки исполняемого кода показывают приблизительное количество строк исполняемого кода или операций. В расчёт в данном случае не берутся пустые строки, комментарии, объявления пространств имён и так далее. Таким образом данная метрика показывает примерное число операций в коде.

Строки исходного кода и исполняемого кода показывают размер программы. Чем больше значение строк, тем сложнее программа или отдельный блок программы, для которого эта метрика считалась. Если число строк кода очень велико, то иногда имеет смысл разбить программу на меньшие блоки, которые будет легче поддерживать в будущем.

На таблице 5 показаны метрики кода, рассчитанные для разработанного ПО.

Таблица 5 – Метрики исходного кода

Класс или метод	MI	CyC	DoI	ClCo	LoS	LoE
Program	63	2	1	8	10	10
ImageDetectionRequestModel	100	6	1	0	10	0
RequestModel	100	6	1	0	7	0
MapObjectData1	100	10	1	0	17	0
MapObjectData2	100	10	1	0	17	0
Coords	96	6	1	0	14	2
DetectionResult	92	2	1	2	16	3
ImagePredictor	83	2	1	10	31	8
ImagePredictor.ImagePredictor()	100	1	-	0	1	0
ImagePredictor.predict()	69	1	-	10	22	6
MapController	53	21	3	37	441	105
MapController.Map()	100	1	-	3	5	1
MapController.detectObjects()	48	3	-	20	46	23
MapController.ImageDetection()	45	8	-	10	222	30
MapController.SearchNearestObjects()	39	8	-	19	138	47
MapController.CalculateLatHeight()	80	1	-	1	5	2

По данным из таблицы 5 видно, что все файлы программы имеют приемлемый индекс удобства поддержки. Минимальное значение данной метрики имеет класс «MapController» равное 53. Таким образом на основании данной метрики можно сделать вывод, что код удобно поддерживать.

Значение метрики сложности организации циклов в большинстве файлов не превышает 10, что является хорошим результатом. Однако у класса «MapController» данное значение равно 21, что объясняется тем, что в данном классе реализовано наибольшее количество операций и методов по отношению к остальным классам проекта. Если посмотреть на отдельные методы класса «MapController», то можно убедиться, что значения данной метрики для них не превышает 10.

Значения метрики глубины наследования для всех файлов проекта низкие и не превышают рекомендуемых.

Значения метрики взаимосвязанности классов не превышают рекомендуемых значений в большинстве файлов. Исключением здесь являются классы «MapController» и «ImagePredictor». Методы «predict», «detectObjects», «imageDetection» и «searchNearestObjects» данных классов имеют значение метрики взаимосвязанности классов, превышающее 9. Это говорит о том, что вышеперечисленные методы сложно поддерживать ввиду большого количества зависимостей в них.

Количество строк исходного кода и исполняемого кода показывает, что наиболее крупным и сложным классом в программе является «MapController». Это объясняется тем, что данный класс основную логику работы программы.

3.4.2 Функциональное тестирование

Для разработанного приложения будет проведено тестирование методом «чёрного ящика», так как оно в достаточной мере позволяет протестировать разработанное приложение и не требует больших затрат для реализации.

При тестировании «чёрного ящика» программа рассматривается как объект, внутренняя структура которого неизвестна. Тестирующий вводит данные и анализирует результат, но, как именно работает программа, он не знает [24].

Тестирование методом «чёрного ящика» удобно использовать в данном случае благодаря тому, что данный подход позволяет рассматривать программу, как абстрактный «чёрный ящик», который на выходе выдаёт результат, соответствующий поступившим на входе действиям пользователя.

В рамках выбранного вида тестирования для разработанного приложения составлены тесты. Данные тесты и их результаты представлены на таблице 6

Таблица 6 – Тесты для приложения

Номер теста	Описание теста	Входные данные	Ожидаемый результат	Фактический результат
1	Проверка алгоритма обнаружения объектов на карте с помощью нейросети для классов house_gray, district2_white.	Выбрать местность на карте, где присутствуют объекты классов: house_gray, district2_white. Задать зум равный 18 и нажать на кнопку «Обнаружить объекты на заданной зоне».	На карте в заданной местности будут обнаружены объекты классов house_gray, district2_white. Вокруг данных объектов будут нарисованы подписанные прямоугольные рамки.	Соответствует ожидаемому (см. рисунок 18)
2	Проверка алгоритма обнаружения объектов на карте с помощью нейросети для классов house_red, district1_white.	Выбрать местность на карте, где присутствуют объекты классов: house_red, district1_white. Задать зум равный 17 и нажать на кнопку «Обнаружить объекты на заданной зоне».	На карте в заданной местности будут обнаружены объекты классов house_red, district1_white. Вокруг данных объектов будут нарисованы подписанные прямоугольные рамки.	Соответствует ожидаемому (см. рисунок 19)
3	Проверка алгоритма обнаружения объектов на карте с помощью нейросети для класса hydro.	Выбрать местность на карте, где присутствуют объекты класса: hydro. Задать зум равный 18 и нажать на кнопку «Обнаружить объекты на заданной зоне».	На карте в заданной местности будут обнаружены объекты класса hydro. Вокруг данных объектов будут нарисованы подписанные прямоугольные рамки.	Соответствует ожидаемому (см. рисунок 20)
4	Проверка алгоритма обнаружения объектов на карте с помощью нейросети для класса ditrict2_gray.	Выбрать местность на карте, где присутствуют объекты класса: ditrict2_gray. Задать зум равный 19 и нажать на кнопку «Обнаружить объекты на заданной зоне».	На карте в заданной местности будут обнаружены объекты класса ditrict2_gray. Вокруг данных объектов будут нарисованы подписанные прямоугольные рамки.	Соответствует ожидаемому (см. рисунок 21)
5	Проверка алгоритма обнаружения объектов на карте с помощью нейросети для класса district3_gray.	Выбрать местность на карте, где присутствуют объекты класса: district3_gray. Задать зум равный 19 и нажать на кнопку «Обнаружить объекты на заданной зоне».	На карте в заданной местности будут обнаружены объекты класса district3_gray. Вокруг данных объектов будут нарисованы подписанные прямоугольные рамки.	Соответствует ожидаемому (см. рисунок 22)

Продолжение таблицы 6

Номер теста	Описание теста	Входные данные	Ожидаемый результат	Фактический результат
6	Проверка алгоритма поиска ближайших объектов жизнеобеспечения и ближайшего населённого пункта.	Выбрать точку на карте и нажать кнопку «Найти ближайший населённый пункт и объекты жизнеобеспечения».	От заданной точки будут построены маршруты до: населенного пункта; поликлиники; пожарной части.	Соответствует ожидаемому (см. рисунки 23 – 26)
7	Проверка работы кнопки «Очистить карту от объектов и маршрутов»	Выбрать точку на карте и задать приближение равное 17. Нажать кнопки из предыдущих тестов. После всего этого нажать на кнопку «Очистить карту от объектов и маршрутов».	Все объекты, созданные на карте, будут стёрты и таким образом карта вернётся в исходное состояние.	Соответствует ожидаемому (см. рисунки 27, 28).

На рисунке 18 показан результат теста №1.

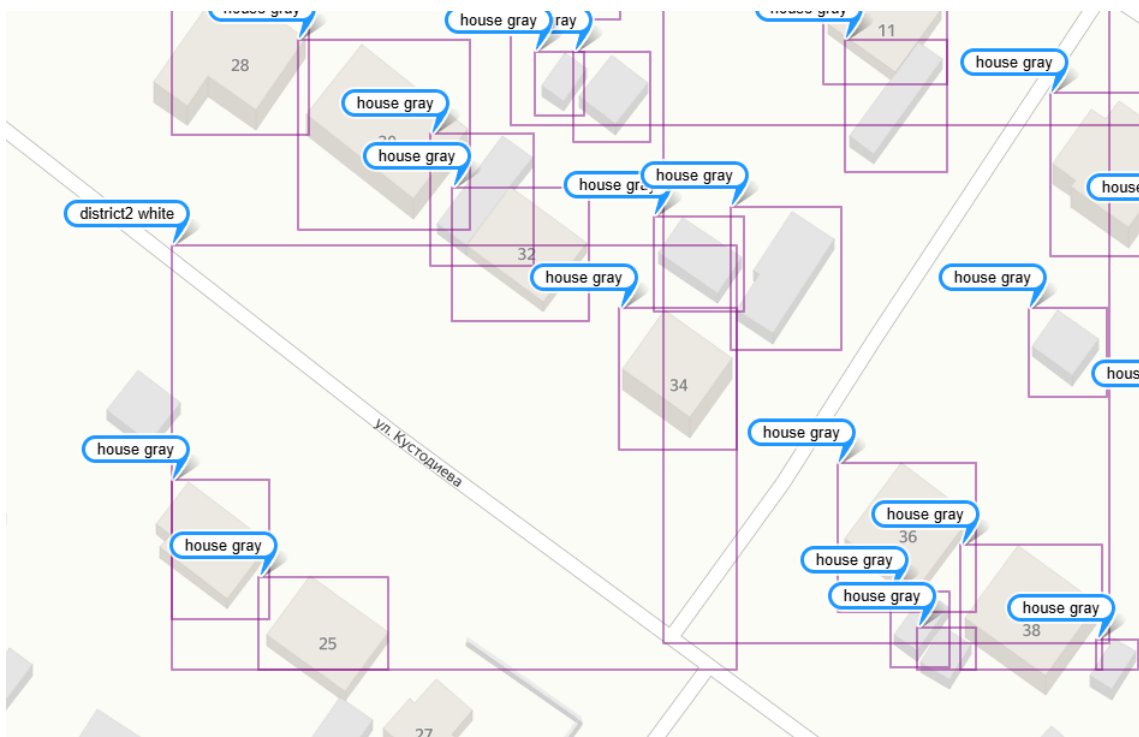


Рисунок 18 – Результат теста №1

Тест №1 показал, что программа корректно обнаруживает объекты классов house_gray и district2_white.

На рисунке 19 показан результат теста №2.

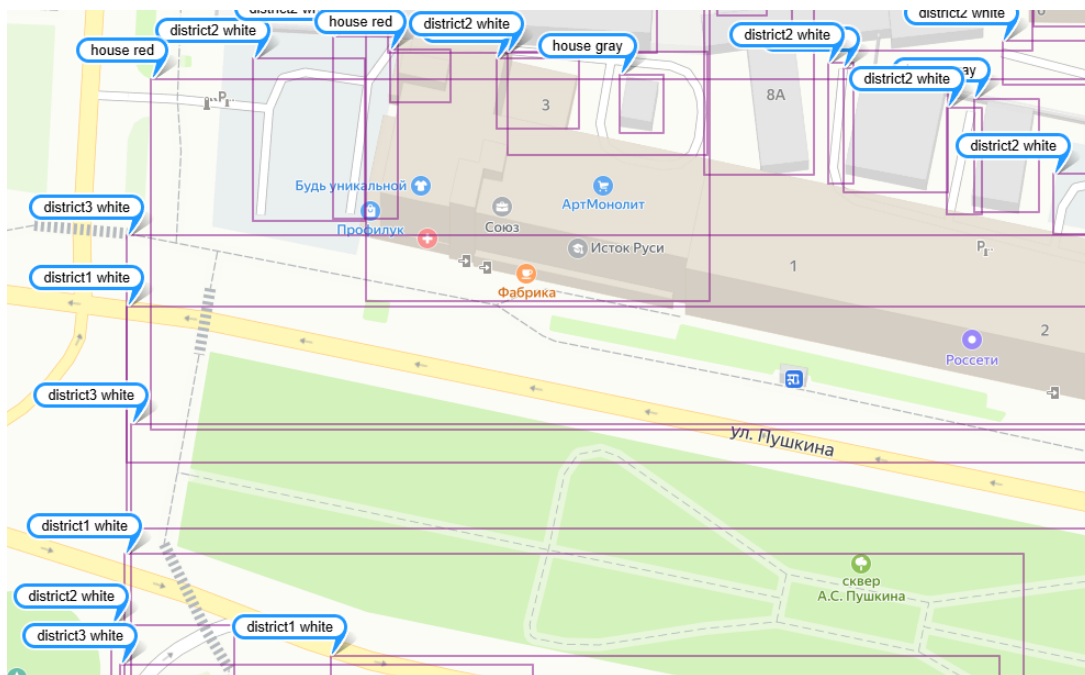


Рисунок 19 – Результат теста №2

На рисунке 19 видно, что программа корректно обнаруживает объекты классов house_red, district1_white, district3_white.

На рисунке 20 показаны результаты теста №3.

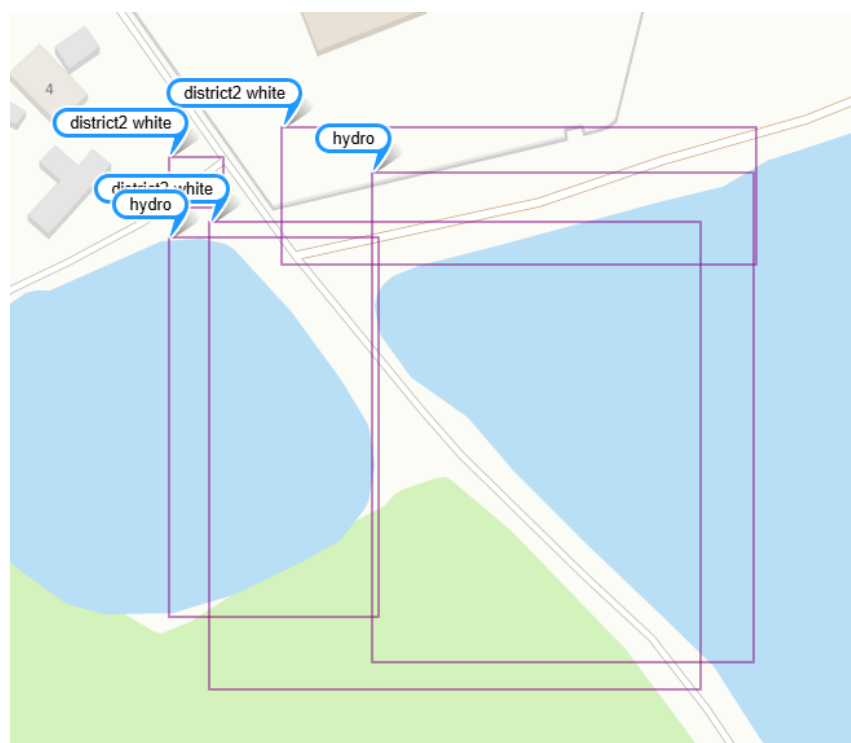


Рисунок 20 – Результат теста №3

На рисунке 20 видно, что программа корректно обнаруживает объекты класса hydro.

На рисунке 21 показаны результаты теста №4.

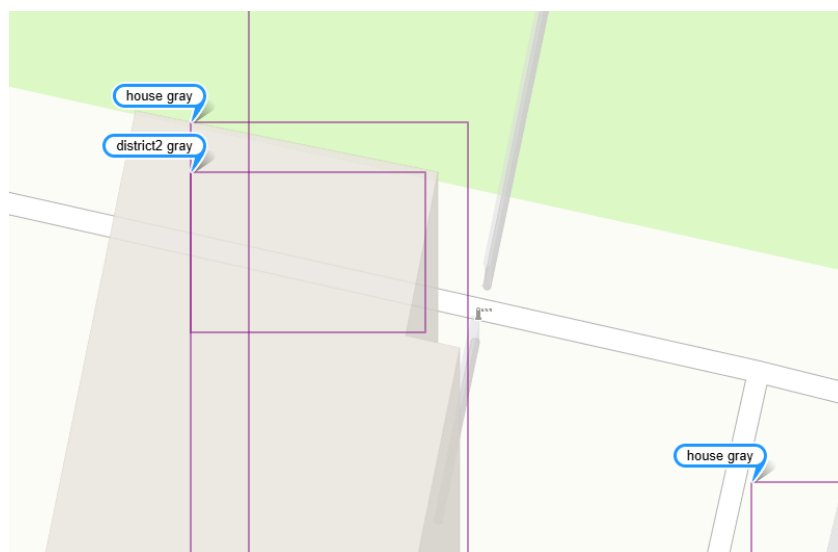


Рисунок 21 – Результат теста №4

На рисунке 21 видно, что программа может обнаруживать объекты класса `district2_gray`. Однако, стоит заметить, что объекты данного класса обнаруживаются достаточно редко. Для повышения способности модели глубокого обучения распознавать объекты данного класса надо продолжать работу над её обучением.

На рисунке 22 показаны результаты теста №5.

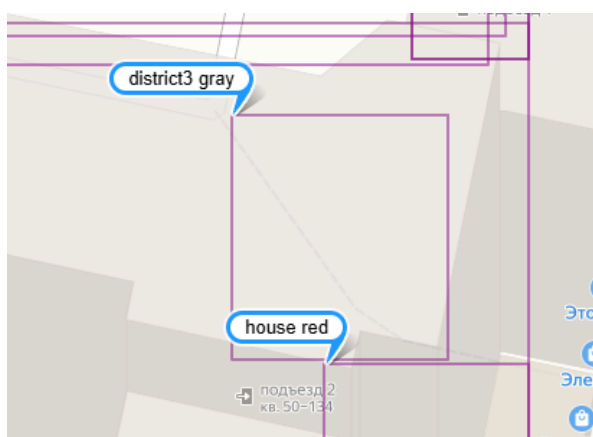


Рисунок 22 – Результат теста №5

В случае теста №5 результаты похожи на результаты теста №4. Программа может распознавать объекты класса `district3_gray`, но довольно часто упускает их на карте.

На рисунках 23 – 26 показаны результаты теста №6.

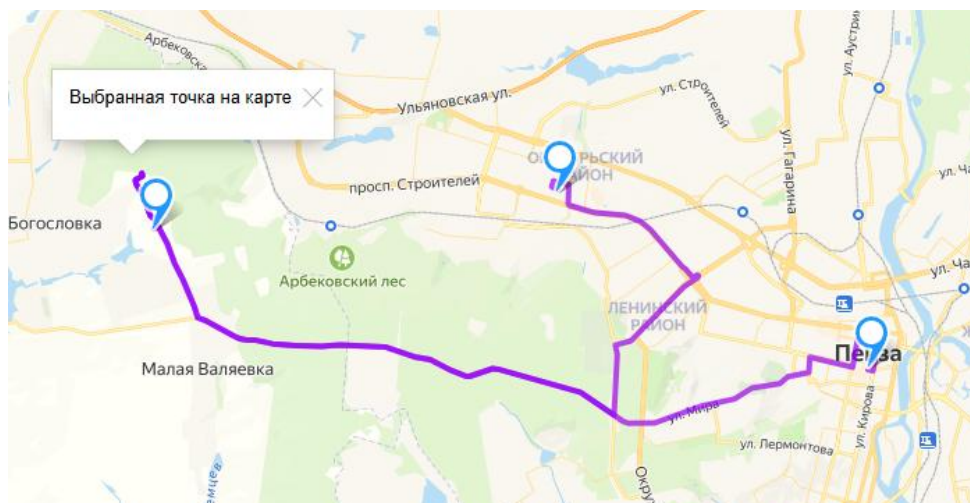


Рисунок 23 – Результат теста №6 (показана информация об исходной выбранной точке на карте)

В ходе выполнения алгоритма поиска ближайших объектов жизнеобеспечения и ближайшего населённого пункта, а также построения маршрутов до них, были определены четыре места на карте, отмеченные специальной меткой. Если щёлкнуть на данную метку, то всплывает текст, который содержит название отмеченного объекта и расстояние до него. На рисунке 23 показана всплывающая подсказка для точки, которую пользователь изначально выбрал на карте перед запуском алгоритма.

На рисунке 24 показана всплывающая подсказка для точки, отмечающей ближайший населённый пункт.

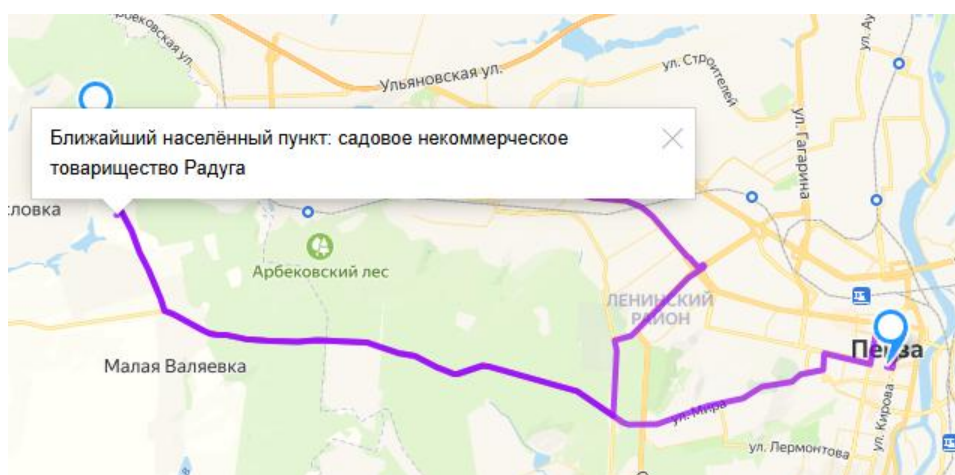


Рисунок 24 – Результат теста №6 (показана информация о ближайшем населённом пункте)

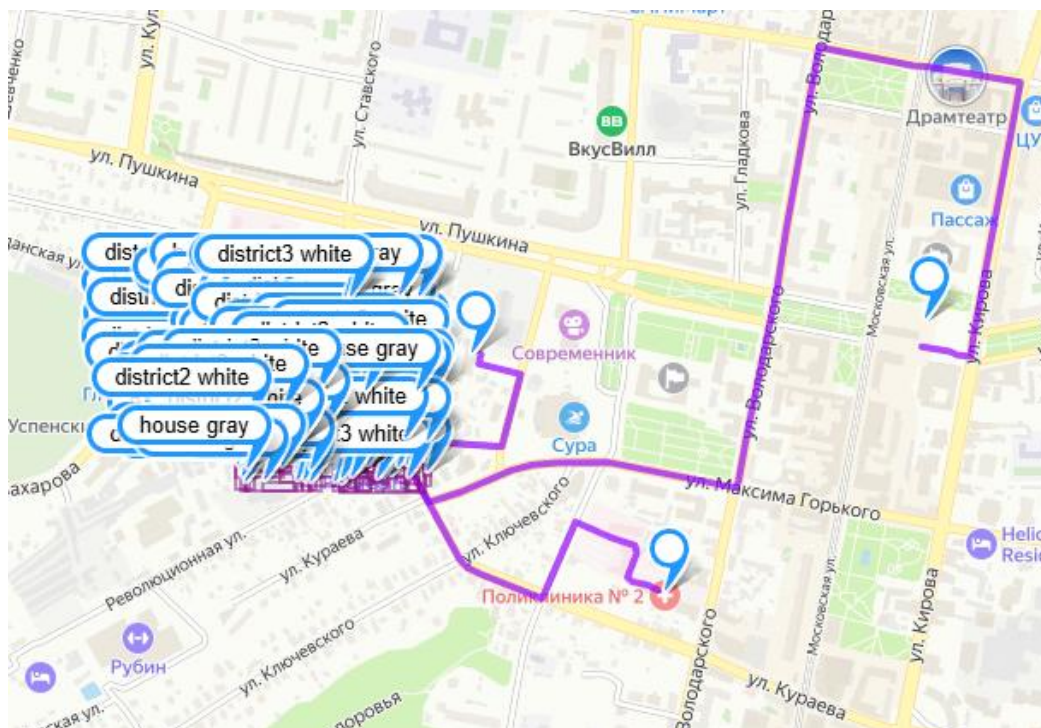


Рисунок 27 – Тест №7 (Карта заполненная объектами)

После нажатия на кнопку «Очистить карту от объектов и маршрутов» карта вернулась к исходному состоянию, что показано на рисунке 28.

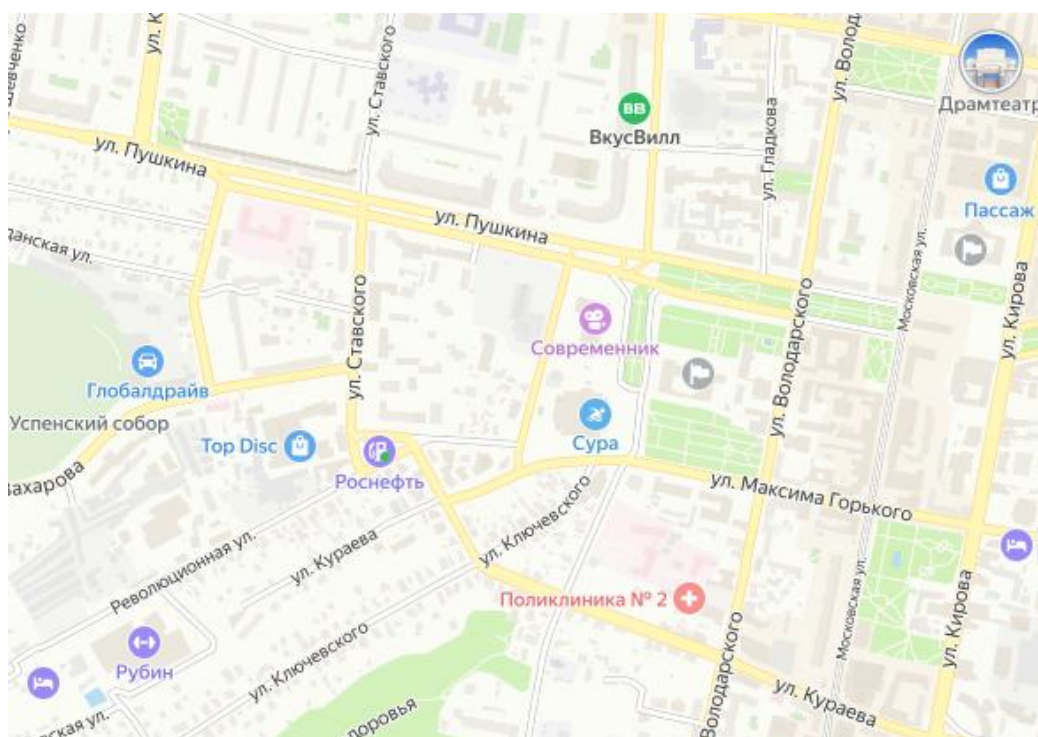


Рисунок 28 – Тест №7 (Карта возвращена в исходное состояние)

Таким образом можно сделать вывод о том, что кнопка «Очистить карту от объектов и маршрутов» работает корректно.

Таким образом, можно сделать следующие выводы.

Разработанные алгоритмы были реализованы с помощью выбранного технологического стека. Реализованный проект прошёл проверку ручным тестированием методом «чёрного ящика». Выявленные ошибки исправлены.

ЗАКЛЮЧЕНИЕ

В ходе магистерской диссертации были решены следующие задачи:

- проведён анализ предметной области, картографических онлайн сервисов и моделей глубокого обучения,
- разработаны графические модели для алгоритма обнаружения объектов на карте с помощью нейросети и алгоритма поиска ближайшего населённого пункта и объектов жизнеобеспечения,
- разработаны алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм поиска ближайшего населённого пункта и объектов жизнеобеспечения,
- на основании алгоритмов разработано программное обеспечение.

На этапе анализа предметной области, картографических онлайн сервисов и моделей глубокого обучения были:

- изучены основные элементы предметной области,
- изучены особенности различных картографических сервисов, среди которых для использования в разработанном ПО был выбран сервис Яндекс карты,
- изучены основные архитектуры моделей глубокого обучения в области компьютерного зрения и среди них была выбрана архитектура YOLO для использования в разработанном ПО.

В ходе выполнения научно-исследовательской работы были разработаны алгоритм обнаружения объектов на карте с помощью нейросети и алгоритм расчёта географических координат для ограничивающих рамок объектов, обнаруженных нейросетью на изображениях с карты.

На основе разработанных моделей и алгоритмов создана, протестирована и отлажена программная реализация системы поддержки принятия решений по развёртыванию ПВР в чрезвычайных ситуациях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические рекомендации по организации первоочередного жизнеобеспечения населения в чрезвычайных ситуациях и работы пунктов временного размещения пострадавшего населения: метод. рек. / утв. МЧС России 20.08.2020 № 2 – 4 – 71 – 18 – 11.
2. Mission Areas and Core Capabilities // FEMA.gov [Электронный ресурс] URL: <https://www.fema.gov/emergency-managers/national-preparedness/mission-core-capabilities> (дата обращения: 03.02.2025).
3. ГОСТ 22.3.18–2021. Безопасность в чрезвычайных ситуациях.
4. Ларичев О.И., Петровский А.Б. Системы поддержки принятия решений: современное состояние и перспективы развития // Итоги науки и техники. М.: ВИНТИ, 1987. Т. 21, с. 131 – 164.
5. Гришунин С.И., Интуитивные компоненты и субъективные модели в принятии решений / С.И. Гришунин // Ценности и смыслы. – 2012. – № 2(18). – С. 64 – 70.
6. Real-time on-line decision support system for off-site emergency management in Europe // cordis.europa.eu [Электронный ресурс] URL: <https://cordis.europa.eu/project/id/FI4P950007> (дата обращения: 03.02.2025).
7. Van de Walle, B., Turoff, M. Decision Support for Emergency Situations // Information Systems and e-Business Management. – 2008. – Т. 6, № 3. – С.295 – 316.
8. About // Sahana Foundation [Электронный ресурс] URL: <https://sahanafoundation.org/about/> (дата обращения: 03.02.2025).
9. About Us // Sahana Foundation [Электронный ресурс] URL: <https://sahanafoundation.org/about-us/> (дата обращения: 03.02.2025).
10. Khitrin, M.O. Comparison of JavaScript Libraries for Web-Cartography / M.O. Khitrin // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника». – 2017. – Т. 17, № 3. – С. 147–152.

11. Web Map Service (WMS) Стандарт // Публикации OGC [Электронный ресурс] URL: <https://www.ogc.org/ru/standards/wms/> (дата обращения: 31.05.2025).
12. Web Feature Service (WFS) Стандарт // Публикации OGC [Электронный ресурс] URL: <https://www.ogc.org/ru/standards/wfs/> (дата обращения: 31.05.2025).
13. Gutierrez G., Llerena J.P., Usero L., Patricio M.A. A comparative study of convolutional neural network and transformer architectures for drone detection in thermal images // Applied Sciences. – 2025. – Т. 15. – № 109.
14. Khanam R., Hussain M. YOLOv11: An overview of the key architectural enhancements //arXiv preprint arXiv:2410.17725. – 2024.
15. YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy // Medium.com [Электронный ресурс] URL: <https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71> (дата обращения: 31.05.2025).
16. Jegham N. et al. Evaluating the evolution of yolo (you only look once) models: A comprehensive benchmark study of yolo11 and its predecessors //arXiv preprint arXiv:2411.00201. – 2024.
17. Репетий Е.О., Хохлова Е.И. Принципы объектно-ориентированного программирования // НАУЧНЫЙ АСПЕКТ № 4 2024. – Самара: Изд-во ООО «Аспект», 2024 . – Т44 . – 5810 – 5815 с.
18. Беликова Наталья Валентиновна, Галич Марк Геннадьевич Использование паттерна проектирования Model-ViewController при разработке web-приложений // Объектные системы. 2011. №1 (3). URL: <https://cyberleninka.ru/article/n/ispolzovanie-patterna-proektirovaniya-model-viewcontroller-pri-razrabotke-web-prilozheniy> (дата обращения: 31.05.2025).
19. Атамухаммедов, Р. Применение сверточных нейронных сетей в компьютерном зрении / Р. Атамухаммедов, И. Атамырадов, Б. Батыров // Вестник науки. – 2024. – Т. 1, № 10(79). – С. 383 – 386.

20. Проекция Меркатора // ru.wikipedia.org [Электронный ресурс]
URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%D0%9C%D0%B5%D1%80%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B0 (дата обращения: 31.05.2025).

21. Абдраманов, А.Т. SPPR_PVR_CHS // GitHub [Электронный ресурс]
URL: https://github.com/tendrbeik/SPPR_PVR_CHS (дата обращения: 31.05.2025).

22. Средство оценки качества кода в MS Visual Studio // Habr.com [Электронный ресурс] URL: <https://habr.com/ru/articles/111524/> (дата обращения: 31.05.2025).

23. How code metrics help identify risks – Visual Studio (Windows) // Microsoft Learn [Электронный ресурс] URL: <https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 31.05.2025).

24. Канер Сэм и др. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: Пер. с англ./Сэм Канер, Джек Фолк, Енг Кек Нгуен. – К.: Издательство «ДиаСофт», 2001. – 544 с.

ПРИЛОЖЕНИЕ А

(Обязательное)

Система поддержки принятия решений по развёртыванию мобильных пунктов
временного размещения при чрезвычайных ситуациях

Подробное описание прецедентов

Таблица А.1 – Прецедент «Выбрать точку на карте»

Прецедент: Выбрать точку на карте
ID: 1
Краткое описание: Выбирает точку на карте
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: Нет.
Основной поток: 1. Пользователь устанавливает точку на карте, кликая по ней мышью 2. Программа записывает географические координаты точки на карте в переменную и выводит их на экран пользователя
Постусловия: У пользователя на экране отмечены координаты текущей выбранной точки
Альтернативные потоки: Нет.

Таблица А.2 – Прецедент «Задать размер области для обработки»

Прецедент: Задать размер области для обработки
ID: 2
Краткое описание: Задаёт размер области для обработки
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: Нет.
Основной поток: 1. Пользователь вводит в специальное поле значение приближения, которое косвенно задаёт область для сканирования местности
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица А.3 – Прецедент «Обнаружить объекты на карте в заданной области»

Прецедент: Обнаружить объекты на карте в заданной области
ID: 3
<p>Краткое описание:</p> <p>Обнаруживает объекты на заданной области с помощью нейросети</p>
<p>Главные актёры:</p> <p>Пользователь</p>
<p>Второстепенные актёры:</p> <p>Нет.</p>
Предусловия: Пользователь должен выбрать точку на карте и задать приближение
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку «Найти объекты на заданной области» 2. Клиент формирует запрос к локальному серверу и отправляет его 3. Сервер загружает с помощью Yandex maps Static API изображения карты 4. Сервер с помощью нейронной сети обнаруживает объекты на изображениях карты 5. Сервер формирует JSON файл, в который загружается информация об обнаруженных объектах, и отправляет его клиенту 6. Клиент принимает JSON файл и отображает на карте обнаруженные сервером объекты
Постусловия: У пользователя на экране отмечены обнаруженные нейросетью объекты
Альтернативные потоки: Нет.

Таблица А.4 – Прецедент «Очистить карту от маршрутов и обнаруженных объектов»

Прецедент: Очистить карту от маршрутов и обнаруженных объектов
ID: 4
<p>Краткое описание:</p> <p>Убирает с карты нарисованные в ходе выполнения других прецедентов объекты и возвращает её к исходному состоянию</p>
<p>Главные актёры:</p> <p>Пользователь</p>
<p>Второстепенные актёры:</p> <p>Нет.</p>
Предусловия: Пользователь должен выбрать точку на карте и задать приближение
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку «Очистить карту от маршрутов и обнаруженных объектов» 2. Клиент удаляет все нарисованные объекты с карты и возвращает её к исходному состоянию
Постусловия: Карта перед пользователем возвращена в исходное состояние
Альтернативные потоки: Нет.

Таблица А.5 – Прецедент «Найти ближайший населённый пункт и объекты жизнеобеспечения»

Прецедент: Найти ближайший населённый пункт и объекты жизнеобеспечения
ID: 5
<p>Краткое описание:</p> <p>Обнаруживает ближайший населённый пункт и объекты жизнеобеспечения, а также чертит на карте маршруты до них</p>
<p>Главные актёры:</p> <p>Пользователь</p>
<p>Второстепенные актёры:</p> <p>Нет.</p>
Предусловия: Пользователь должен выбрать точку на карте
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку «Найти ближайший населённый пункт и объекты жизнеобеспечения» 2 Клиент с помощью Javascript API находит ближайший населённый пункт и чертит маршрут до него на карте, а также ставит метку на карте, которая будет содержать информацию о названии населённого пункта и расстоянии до него 2. Клиент формирует запрос к локальному серверу и отправляет его 3. Сервер находит ближайшие объекты жизнеобеспечения с помощью Geosuggest API и Geocoder API 5. Сервер формирует JSON файл, в который загружается информация об обнаруженных объектах жизнеобеспечения, и отправляет его клиенту 6. Клиент принимает JSON файл и рисует на карте маршруты до обнаруженных объектов жизнеобеспечения, а также ставит метки на карте, которые будут содержать информацию о названиях объектов жизнеобеспечения и расстояниях до них
Постусловия: У пользователя на экране отмечены маршруты до ближайшего населённого пункта и объектов жизнеобеспечения, а также на карте поставлены метки, содержащие названия отмеченных объектов и расстояния до них
Альтернативные потоки: Нет.

ПРИЛОЖЕНИЕ Б

(Обязательное)

Система поддержки принятия решений по развёртыванию мобильных пунктов
временного размещения при чрезвычайных ситуациях

Алгоритм обнаружения объектов на карте с помощью модели глубокого
обучения

Листинг Б.1 – Функция drawRectanglesWithZoom() из представления
«Map.cshtml»

```
function drawRectanglesWithZoom(){

    const zoomValue = document.getElementById('zoom').value;

    //const zoomNumber = parseInt(zoomValue, 10);

    fetch('/Map/ImageDetection', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json' // ВАЖНО!
        },
        body: JSON.stringify({ latitude: coords[0], longitude:
coords[1], zoom: zoomValue})
    })
    .then(response => response.json())
    .then(data => {
        data.forEach(obj => {
            let myRectangle = new ymaps.Rectangle([
                // Задаем координаты диагональных углов
прямоугольника.
                [obj.y1, obj.x1],
                [obj.y2, obj.x2]
            ], {
                //Свойства
                hintContent: obj.type,
                //balloonContent: 'Мой первый прямоугольник на
карте'
            }, {
                // Опции.
                // Цвет и прозрачность заливки.
                fillColor: '#F700F7',
                // Дополнительная прозрачность заливки..
```

```

        // Итоговая прозрачность будет не #33(0.2), а
0.1(0.2*0.5).

        fillOpacity: 0,
        // Цвет обводки.
        strokeColor: '#860D86',
        // Прозрачность обводки.
        strokeOpacity: 0.5,
        // Ширина линии.
        strokeWidth: 2,
        interactivityModel: 'default#transparent'
    });

    myMap.geoObjects.add(myRectangle);

    //Теперь добавим текстовую метку для каждого объекта
    var upperLeftCorner = [
        obj.y1,
        obj.x1
    ];

    var label = new ymaps.Placemark(upperLeftCorner, {
        iconContent: obj.type
    }, {
        preset: 'islands#blueStretchyIcon',
        draggable: false
    });

    myMap.geoObjects.add(label)
    });
    })
    .catch(error => console.error('Ошибка:', error));
}

```

Листинг Б.2 – Код метода MapController.ImageDetection()

```

public async Task<IActionResult> ImageDetection([FromBody]
ImageDetectionRequestModel data)

```

```

{
    if (data.zoom >= 18 && data.zoom <= 20)
    {
        //Проводим детекцию объектов по заданным координатам
        Coords coords = new Coords(data.latitude, data.longitude);
        var result = detectObjects(coords, zoom);
        if (result.Result.badRequest)
            return BadRequest("Image load error!");
        else
            return Json(result.Result.detectedObjects);
    }
    else if (data.zoom == 17)
    {
        double latHeight18z = calculateLatHeight(data.latitude);
        //Высчитываем координаты 4-точек
        List<Coords> coords = new List<Coords>();
        Coords c = new Coords();
        c.latitude = data.latitude + 0.5 * latHeight18z;
        c.longitude = data.longitude - 0.5 * longWidth18z;
        coords.Add(c);
        c = new Coords();
        c.latitude = data.latitude + 0.5 * latHeight18z;
        c.longitude = data.longitude + 0.5 * longWidth18z;
        coords.Add(c);
        c = new Coords();
        c.latitude = data.latitude - 0.5 * latHeight18z;
        c.longitude = data.longitude + 0.5 * longWidth18z;
        coords.Add(c);
        c = new Coords();
        c.latitude = data.latitude - 0.5 * latHeight18z;
        c.longitude = data.longitude - 0.5 * longWidth18z;
        coords.Add(c);

        //Сюда будем записывать найденные объекты.
        var resultObjects = new List<MapObjectData1>();
    }
}

```

```

        foreach (var value in coords)
        {
            var result = detectObjects(value, zoom);
            if (result.Result.badRequest)
                return BadRequest("Image load error!");
            else
                foreach (var obj in result.Result.detectedObjects)
                {
                    resultObjects.Add(obj);
                }
        }
        return Json(resultObjects);
    }
    else
    {
        return BadRequest("Invalid zoom value!");
    }
}

```

Листинг Б.3 – Код метода MapController.detectObjects()

```

private async Task<DetectionResult> detectObjects(Coords data, int
zoom)
{
    string requestURL =
$"{{staticApiUrl}}?ll={{data.longitude.ToString(CultureInfo.InvariantCulture
Culture)}},{{data.latitude.ToString(CultureInfo.InvariantCultureCulture)}}&l
ang=ru_RU&size=450,450&z={{zoom}}&style={{requestStyle}}&apikey={{apiKe
y2}}";

    var resultObjects = new List<MapObjectData1>();

    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage response = await
client.GetAsync(requestURL);
    }
}

```

```

        if (!response.IsSuccessStatusCode)
        {
            //Возвращаем BadRequest.
            return new DetectionResult();
        }

        byte[] imageBytes = await
response.Content.ReadAsByteArrayAsync();

        string filename =
$"map_{data.latitude}_{data.longitude}.png";

        string filePath =
Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images",
filename);

        //Сохранить изображение в папку проекта.
        await System.IO.File.WriteAllBytesAsync(filePath,
imageBytes);

        string fileUrl = $"/images/{filename}";
        ImagePredictor predictor = new ImagePredictor();
        var result = predictor.predict(filePath);
        //Вычислим текущую высоту изображения в градусах широты
        double latHeight = calculateLatHeight(data.latitude);

        foreach (var obj in result)
        {
            MapObjectData1 objectData = new MapObjectData1();
            objectData.type = obj.Name.Name.ToString();
            objectData.X1 = data.longitude + longWidth18z *
((double)obj.Bounds.X / 450 - 1d / 2);
            objectData.Y1 = data.latitude + latHeight *
((double)(450 - obj.Bounds.Y) / 450 - 1d / 2);

```

```

        objectData.X2 = objectData.X1 +
((double)obj.Bounds.Width / 450) * longWidth18z;
        objectData.Y2 = objectData.Y1 -
((double)obj.Bounds.Height / 450) * latHeight;
        resultObjects.Add(objectData);
    }
    return new DetectionResult(resultObjects);
}
}

```

Листинг Б.4 – Код метода MapController.calculateLatHeight()

```

private double calculateLatHeight(double lat)
{
    double latHeight = longWidth18z / lengthLat * Math.Cos(lat *
Math.PI / 180) * Math.PI / 180 * 6378137 * 0.001;
    return latHeight;
}

```

ПРИЛОЖЕНИЕ В

(Обязательное)

Система поддержки принятия решений по развёртыванию мобильных пунктов
временного размещения при чрезвычайных ситуациях

Алгоритм поиска ближайшего населённого пункта и объектов
жизнеобеспечения

Листинг В.1 – Функция setRoutes() из представления «Map.cshtml»

```
function setRoutes(){
    //Рисуем маркер для точки, которую пользователь отметил на
    карте.
    const destinationPlacemark = new ymaps.Placemark(coords, {
        balloonContent: `Выбранная точка на карте`
    });
    myMap.geoObjects.add(destinationPlacemark);

    var ObjectsList = ['locality'];
    // Обратное геокодирование: найти ближайший населённый пункт
    ymaps.geocode(coords, {
        kind: 'locality', // ищем населённый пункт
        results: 1
    }).then(function (res) {
        const firstGeoObject = res.geoObjects.get(0);

        if(firstGeoObject){
            const nearestLocalityCoords =
firstGeoObject.geometry.getCoordinates();
            const localityName =
firstGeoObject.getLocalities().join(', ');

            //Marker: Это просто отметка ближайшего населённого
пункта. Её можно убрать.
            const destinationPlacemark = new
ymaps.Placemark(nearestLocalityCoords, {
                balloonContent: `Ближайший населённый пункт:
${localityName}`
            });

            myMap.geoObjects.add(destinationPlacemark);

            ymaps.route([
                coords,
```

```

        nearestLocalityCoords
    ]).then(function (route) {
        myMap.geoObjects.add(route);

        const waypoints = route.getWaypoints();
        waypoints.each(function (waypoint) {
            waypoint.options.set('visible', false);
        });

        // Получаем активный маршрут и вычисляем его
        // длину (первый по умолчанию)
        const activeRoute = route.getPaths().get(0);

        if (activeRoute) {
            //Получаем длину маршрута
            const distance = activeRoute.getLength();
            // в метрах

            console.log(`Расстояние: ${distance} м`);

            //Отображаем полученную длину на html-
            //странице

            document.getElementById("localityRouteLength").innerText =
            `Расстояние до ближайшего населённого пункта:
            ${distance.toFixed(2)}м`;
        }

        }, function(error) {
            console.error('Ошибка построения маршрута:',
            error);
        });
    } else {
        console.warn('Населённый пункт не найден.');
```

```

    });

    //Далее делаем запрос на сервер, чтобы найти ближайшие
    больницу и пожарную часть.
    fetch('/Map/SearchNearestObjects', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json' // ВАЖНО!
      },
      body: JSON.stringify({
        latitude: coords[0],
        longitude: coords[1],
        objects: ["поликлиника или больница", "пожарная часть или
пожарное отделение"]
      })
    })
    .then(response => response.json())
    .then(data => {
      //С сервера мы получили список ближайших искомым
муниципальных объектов и теперь их обрабатываем по очереди.
      //Сначала выводим суммарный JSON ответ от сервера в
консоль браузера.
      console.log('Ответ сервера:', data);
      //Теперь обрабатываем каждый элемент в JSON-ответе от
сервера отдельно.
      data.forEach(obj => {

        //Задаём координаты и название объекта для того,
чтобы потом на карте отобразить маркер и маршрут до объекта.
        const nearestLocalityCoords = [obj.latitude,
obj.longitude];
        const localityName = obj.name;

        //Рисуем маркер.

```

//Marker: Это просто отметка ближайшего населённого пункта. Её можно убрать.

```
const destinationPlacemark = new
ymaps.Placemark(nearestLocalityCoords, {
    balloonContent: `Название: ${localityName};
    Расстояние до объекта: ${obj.distance}`
});

myMap.geoObjects.add(destinationPlacemark);

//Строим маршрут.
ymaps.route([
    coords,
    nearestLocalityCoords
]).then(function (route) {
    myMap.geoObjects.add(route);

    const waypoints = route.getWayPoints();
    waypoints.each(function (wayPoint) {
        wayPoint.options.set('visible', false);
    });
}, function(error) {
    console.error('Ошибка построения маршрута:',
error);
});
});
});
.catch(error => console.error('Ошибка:', error));
}
```

Листинг В.2 – Код метода MapController.SearchNearestObjects()

```
public async Task<IActionResult> SearchNearestObjects([FromBody]
RequestModel data)
{
    //Сюда будем сохранять данные о каждом из заданных объектов
    var resultObjects = new List<MapObjectData2>();
```

```

foreach (string obj in data.objects)
{
    if (data == null)
    {
        return Json(new { error = "Входные данные не указаны"
});
    }
    //Делаем запрос в геосаггест и получаем адрес.
    using (var httpClient = new HttpClient())
    {
        // Формирование URL запроса
        string requestURL =
        $"{geosuggestApiUrl}apikey={apiKey3}&text={obj}&results=10&ll={data
a.longitude.ToString(CultureInfo.InvariantCulture)}, {data.latitude
.ToString(CultureInfo.InvariantCulture)}&spn=1,1&strict_bounds=0&p
rint_address=1";

        var response = await httpClient.GetAsync(requestURL);

        if (!response.IsSuccessStatusCode)
        {
            //TODO: Надо узнать корректно ли на английском моё
сообщение об ошибке.
            return BadRequest("Запрос к Geosuggest API был
неуспешен!");
        }

        string responseContent = await
response.Content.ReadAsStringAsync();

        //Объявим свойства, которые будем доставать из
Geosuggest API
        string? name;
        string? distance;
        string? country;

```

```

        string? formatted_address;

        using (JsonDocument doc =
JsonDocument.Parse(responseContent))
        {
            JsonElement root = doc.RootElement;
            JsonElement results = root.GetProperty("results");
            //Здесь мы будем искать ближайший объект.
            if (results.GetArrayLength() == 0)
            {
                return BadRequest("Geosuggest API не нашёл
объектов заданного типа.");
            }
            int minIndex = 0;
            double? minDist = results[0]
                .GetProperty("distance")
                .GetProperty("value")
                .GetDouble();
            for (int i = 1; i < results.GetArrayLength(); i++)
            {
                double dist = results[i]
                    .GetProperty("distance")
                    .GetProperty("value")
                    .GetDouble();
                if (dist < minDist)
                {
                    minDist = dist;
                    minIndex = i;
                }
            }
            //Итак, по идее мы нашли ближайший объект. Теперь
            надо для него достать все необходимые нам данные.
            JsonElement addressComponent = results[minIndex]
                .GetProperty("address")
                .GetProperty("component");

```

```

        name = results[minIndex]
            .GetProperty("title")
            .GetProperty("text")
            .GetString();
        distance = results[minIndex]
            .GetProperty("distance")
            .GetProperty("text")
            .GetString();
        country = addressComponent[0]
            .GetProperty("name")
            .GetString();
        formatted_address = results[minIndex]
            .GetProperty("address")
            .GetProperty("formatted_address")
            .GetString();
    }
    //Делаем запрос в геокодер и получаем название
организации и координаты.

    // Формирование URL запроса
    requestURL =
    $"{geocodeUrl}?apikey={apiKey1}&geocode={country}+{formatted_addre
ss}+{name}&format=json&results=1";

    var response2 = await httpClient.GetAsync(requestURL);

    if (!response2.IsSuccessStatusCode)
    {
        return BadRequest("Запрос к Geocoder API был
неуспешен!");
    }

    string response2Content = await
response2.Content.ReadAsStringAsync();

```

```

        string? longitude;
        string? latitude;

        using (JsonDocument doc =
JsonDocument.Parse(response2Content))
        {
            JsonElement root = doc.RootElement;
            JsonElement featureMember =
root.GetProperty("response")
                .GetProperty("GeoObjectCollection")
                .GetProperty("featureMember");
            string? pos = featureMember[0]
                .GetProperty("GeoObject")
                .GetProperty("Point")
                .GetProperty("pos")
                .GetString();

            string[] posParts = pos.Split(' ');

            longitude = posParts[0];
            latitude = posParts[1];
        }
        MapObjectData2 objData = new MapObjectData2();
        objData.name = name;
        objData.formattedAddress =
$"{country},{formatted_address},{name}";
        objData.distance = distance;
        objData.latitude = double.Parse(latitude,
CultureInfo.InvariantCulture);
        objData.longitude = double.Parse(longitude,
CultureInfo.InvariantCulture);
        resultObjects.Add(objData);
    }
}

return Json(resultObjects);

```

}