

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

КОМПОНЕНТ СТАТИЧЕСКОГО
АНАЛИЗАТОРА. ПОСТРОЕНИЕ СТЕКА
ВЫЗОВОВ ФУНКЦИЙ

Студент

А 25.05.2026 Амиров Имран Рафаилович
(подпись, дата) (ФИО полностью)

Руководитель

В.Ю. 25.05.2026 В. Ю. Егоров
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

А.С. 25.05.2026 А. С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от

15 июня 2026 № 31)

Секретарь ГЭК

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

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 07 » 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Амиров Имран Рафаилович гр. 22ВВП1 факультета вычислительной техники
2. Тема работы: «Компонент статического анализатора, построение стека вызовов функций»
Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026 г.
3. Руководитель бакалаврской работы к. т. н., доцент кафедры ВТ Егоров В. Ю

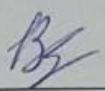
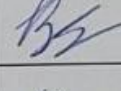
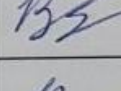
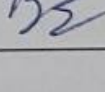
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение-компонент статического анализатора, выполняющую обработку исходного кода программ, с последующим построением стека вызовов функций. Функционал приложения включает в себя: выявление функций, классов, условных операторов в исходном коде программ. Приложение должно графический интерфейс, с визуализацией построенного графа вызовов и возможностями навигации по нему, по дереву функций, классов и ветвлений. Язык программирования Python 3.7, среда разработки Eclipse IDE, графическая библиотека PyQt5.

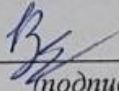
5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметных областей: изучить существующие статические анализаторы исходного кода, выполняющие построение стека вызовов с учетом ветвлений, выявить их достоинства и недостатки. Выбрать и изучить средства разработки, подходящие для выполнения работы.

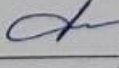
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Ознакомление с темой, разработка плана проведения работ	4	07.04.2026	
2	Анализ требований	16	07.04.2026 – 10.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	20	11.04.2026 – 14.04.2026	
4	Описание алгоритмов реализации	20	15.04.2026 – 19.04.2026	
5	Разработка компонента статического анализатора	60	20.04.2026 – 02.05.2026	
6	Разработка графической части приложения	60	03.05.2026 – 15.05.2026	
7	Оформление пояснительной записки	20	16.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  Егоров В. Ю.
(подпись, дата)

Задание к исполнению принял студент  Амиров И. Р.
(подпись, дата)

Работу к защите допустить
Декан факультета  09.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 96 листов, 30 рисунков, 1 таблицу, 10 источников и 3 приложения.

СТАТИЧЕСКИЙ АНАЛИЗ, СТЕК ВЫЗОВОВ, ПОСТРОЕНИЕ ГРАФА, ГРАФИЧЕСКОЕ ПРИЛОЖЕНИЕ, PYTHON, QT

Цель работы – разработка модуля статического анализатора исходного кода, выполняющего построение стека вызовов функций с учетом ветвлений для последующего анализа результатов.

Приложение должно обладать следующим функционалом:

– выявление пакетов, классов, функций, ветвлений в исходном коде Python программ;

- построение индекса функций, их вызовов и ветвлений;
- построение графа вызовов, в том числе в графическом виде;
- графический интерфейс для навигации по индексам и графам;
- сохранение и загрузка результатов.

В ходе выполнения работы разработано приложение, целиком соответствующее данным требованиям. Для разработки использовался язык программирования Python, библиотека графического интерфейса PyQt.

					ПГУ1.090301.01.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Компонент статического анализатора, построение стека вызовов функций Пояснительная записка	Лит.	Лист	Листов
Разраб.		Амиров И. Р.		25.05.25				
Провер.		Егоров В. Ю.		25.05.25			2	44
Н. Контр		Бычков А. С.		25.05.25		ФВТ, каф. ВТ гр. 22ВВП1		
Утверд.		Митрохин М. А.		25.05.25				

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Компонент статического анализатора, построение стека вызовов функций.....	7
1.1 Анализ предметной области	7
1.1.1 Анализ требований	7
1.1.2 Анализ существующих аналогов	8
1.1.3 Постановка задачи	11
1.2 Выбор средств и инструментов разработки	12
1.2.1 Выбор языка программирования.....	12
1.2.2 Выбор среды разработки.....	15
1.2.3 Выбор библиотеки графического интерфейса	15
1.3 Проектирование и разработка.....	18
1.3.1 Архитектура модуля анализа.....	18
1.3.2 Алгоритм работы модуля анализа.....	20
1.3.3 Архитектура графического интерфейса	21
1.3.4 Алгоритм работы графического интерфейса	25
1.4 Тестирование и отладка.....	26
1.4.1 Тестирование модуля анализа	27
1.4.3 Тестирование графического интерфейса.....	28
1.3 Анализ результатов.....	30
2 Охрана труда.....	39
2.1 Требования к рабочему месту.....	39
2.2 Требования к помещению	40
2.3 Правила размещения монитора	40
2.4 Режим труда и отдыха при использовании компьютера	41
Заключение	42
Список использованных источников	43
Приложение А – UML диаграммы	45

Приложение Б – Листинг программы	49
Приложение В – Презентация.....	87

					ПГУ1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Перечень сокращений и обозначений

В этой работе используются следующие сокращения и обозначения

ГОСТ – государственный стандарт

ООП - объектно-ориентированное программирование

ПК - персональный компьютер

ПО – программное обеспечение

ФСТЭК - федеральная служба по техническому и экспортному контролю

AST (Abstract Syntax Tree) – абстрактное синтаксическое дерево программы

CD (Continuous Deployment) – система автоматического развертывания измененного кода

CI (Continuous Integration) – система автоматической проверки интеграции изменений кода

DevSecOps (Development, Security, Operations) – методология непрерывной и безопасной разработки ПО

GIL (Global Interpreter Lock) – глобальная блокировка интерпретатора, способ синхронизации потоков в интерпретируемых языках.

GUI (Graphical User Interface) – графический интерфейс приложения

IDE (Integrated Development Environment) — интегрированная среда разработки

JIT (Just In Time) – точно в срок, технология динамической компиляции

JSON (JavaScript Object Notation) – текстовых формат обмена данных

JVM (Java Virtual Machine) – виртуальная машина Java

UI (User Interface) - пользовательский интерфейс

UML (Unified Modeling Language) - унифицированный язык моделирования.

					ПГУ1.090301.01.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Программное обеспечение становится всё более сложным: современные приложения состоят из десятков модулей, сотен функций и тысяч взаимосвязей между ними. В таких условиях обнаружение ошибок и уязвимостей традиционными методами, по типу ручного осмотра кода или запуска в режиме отладки, становится трудоёмким и ненадёжным. На помощь разработчикам приходит статический анализ: исследование исходного кода без его выполнения, позволяющее автоматически выявлять дефекты на ранних этапах разработки.

Актуальность темы обусловлена как практической потребностью в инструментах анализа кода на языке Python – одном из наиболее популярных языков в современной разработке, – так и нормативными требованиями: с апреля 2024 года в России действует ГОСТ Р 71207–2024, предписывающий применение межпроцедурного анализа и анализа потоков данных при разработке безопасного программного обеспечения. Оба подхода напрямую опираются на построение графа и стека вызовов [1].

На сегодняшний день существует очень мало решений этой задачи. Они реализуют построение графа вызовов как внутреннюю деталь реализации, недоступную для внешнего использования, или попросту ориентированы на другие языки программирования, или узкоспециализированные сценарии анализа.

Цель работы – исследовать предметную область, проанализировать существующие решения, разработать компонент статического анализатора, разработать графический интерфейс для навигации по построенному графу вызовов, протестировать разработанное ПО.

					ПГУ1.090301.01.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Компонент статического анализатора, построение стека вызовов функций

1.1 Анализ предметной области

1.1.1 Анализ требований

В соответствии с ГОСТ Р 71207–2024 «Защита информации. Разработка безопасного программного обеспечения. Статический анализ программного обеспечения. Общие требования», действующего с 1 апреля 2024 года, статический анализ подразумевает процесс исследования исходного кода, включая объектный и байт-код, без его непосредственного выполнения, целью которого является выявление уязвимостей, ошибок и нарушений стандартов безопасного программирования [1].

Актуальность развития и стандартизации средств статического анализа, в том числе компонентов для построения стека вызовов, в период 2022–2026 годов обусловлена следующими факторами:

– нормативное регулирование в области безопасной разработки (DevSecOps): Стандарт ГОСТ устанавливает единые требования к методам и инструментам статического анализа. Стандарт предписывает применение межпроцедурного контекстно-чувствительного анализа и анализа помеченных данных (taint analysis) для обнаружения сложных дефектов, что автоматически подразумевает необходимость в компоненте построения графа вызовов;

– необходимость верификации сложных межмодульных взаимодействий: В распределённых системах и микросервисных архитектурах критические ошибки, такие как уязвимости внедрения кода, потенциальные утечки конфиденциальной информации, часто возникают на стыках взаимодействия разных модулей и библиотек. Локализация таких ошибок требует построения полного дерева возможных путей выполнения программы, от точки входа до анализируемой лексемы;

					ПГУ1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

– импортозамещение средств анализа кода: утверждение стандарта на уровне ГОСТ стимулирует разработку и сертификацию отечественных анализаторов, удовлетворяющих требованиям ФСТЭК России в части полноты и точности выявления дефектов по классам критичности.

1.1.2 Анализ существующих аналогов

Стоит выполнить оценку и сравнение уже существующих анализаторов исходного кода, в частности их часть, отвечающую за построение стека вызовов.

Pyre.

Pyre представляет собой инструмент статического анализа исходного кода Python для проверки его на проблемы безопасности. Он включает в себя Pysa - анализатор для отслеживания потоков данных (data flow analysis), результат работы которого представлен на рисунке 1 [4].

```
(tutorial) x@ubuntu:~/Desktop/pyre-check/pysa_tutorial/exercisel$ pyre analyze
X `google.protobuf.message.Message.__init__` has 360 overrides, this might slow down the analysis c
onsiderably.
X `object.__init__` has 645 overrides, this might slow down the analysis considerably.
X `type.__call__` has 120 overrides, this might slow down the analysis considerably.
X `type.__init__` has 386 overrides, this might slow down the analysis considerably.
X `type.__new__` has 69 overrides, this might slow down the analysis considerably.
[
  {
    "line": 12,
    "column": 18,
    "stop_line": 12,
    "stop_column": 35,
    "path": "views.py",
    "code": 5001,
    "name": "Possible RCE:",
    "description":
      "Possible RCE: [5001]: User specified data may reach a code execution sink",
    "long_description":
      "Possible RCE: [5001]: User specified data may reach a code execution sink",
    "concise_description":
      "Possible RCE: [5001]: User specified data may reach a code execution sink",
    "inference": null,
    "define": "views.operate_on_twos"
  }
]
```

Рисунок 1 – Результат работы Pyre

Достоинства:

– максимальная поддержка Python: инструмент разработан специально для этого языка и учитывает его особенности (динамическая типизация, метаклассы, декораторы, генераторы);

– имеет алгоритм построения графа вызовов: в кодовой базе Pyre присутствуют модули DependencyGraph, ClassHierarchyGraph и CallGraph, реализующие построение графа и межпроцедурный анализ;

– экспорт результатов в удобных форматах: Pysa поддерживает вывод результатов в формате JSON, что упрощает интеграцию с внешними инструментами, к примеру, для визуализации графа.

Недостатки:

– специализация на безопасности: Pysa ориентирован на анализ помеченных данных (taint analysis), а не на построение стека вызовов для отдельного анализа. Многие элементы графа вызовов Pyre остаются внутренними и не предназначены для внешнего использования;

– высокая сложность: Pyre написан на языке OCaml, что затрудняет модификацию и расширение разработчиками, не знакомыми с данным языком программирования;

– компромисс точности и производительности: Разработчики Pysa сознательно выбрали стратегию эвристического анализа информации о типах, ценой снижения точности в пользу скорости анализа.

Infer.

Infer – инструмент для статического анализа, широко применяемый для поиска ошибок в крупных кодовых базах на языках Java, C/C++. В отличие от Pysa, Infer использует принципиально иной архитектурный подход к построению и использованию графа вызовов – композиционный анализ. На первом этапе Infer строит глобальный граф вызовов программы, определяя зависимости между функциями. Затем выполняется восходящий анализ: он начинается с "листовых" функций (не вызывающих других функций). Для каждой такой функции Infer вычисляет краткую сводку – компактное описание её поведения с точки зрения анализируемых свойств. Результат работы показан на рисунке 2 [3].

```

./Program.cs:53: error: Dotnet Resource Leak
Leaked [ 0 -> 1 ] resource(s) at type(s) System.IO.StreamWriter.
51.     /// Intraprocedural resource usage example, leaks expected.
52.     /// </summary>
53.     public void ResourceLeakIntraproceduralBad(){
54.         StreamWriter sw = new StreamWriter("everwhat.txt");
55.         sw.WriteLine("Guru99 - ASP.Net");

./Program.cs:92: error: Dotnet Resource Leak
Leaked [ n$0.Examples.Program.SWGlobal -> 1,
n$0.Examples.Program.SWGlobal -> 1 ] resource(s) at type(s) System.IO.StreamWriter, System.IO.StreamReader.
90.     public class MainClass {
91.         public static void Main(string[] args)
92.         {
93.             Program p = new Program("whatever.txt");
94.             // FIXME: should close the global streams by calling p.Cleanup()

./Program.cs:96: error: Null Dereference
указатель 'returned by NullDeReferenceBad()' может быть null и разменивается на стр. 96, кол. 13.
94.             // FIXME: should close the global streams by calling p.Cleanup()
95.             // Null dereference error report expected.
96.             p.NullDeReferenceBad().GetHashCode();
97.             // No null dereference error report expected.
98.             p.NullDeReferenceOK().GetHashCode();

Found 3 issues
Issue Type(ISSUED_TYPE_ID): #
Dotnet Resource Leak(DOTNET_RESOURCE_LEAK): 2
Null Dereference(NULL_DEREFERENCE): 1

```

Рисунок 2 – Результат работы infer

Достоинства:

- масштабируемый межпроцедурный анализ. Композиционный подход позволяет Infer выполнять глубокий межпроцедурный анализ без вычисления огромного количества возможных состояний программы;
- поддержка инкрементального анализа. Из-за изолированного анализа функций и кэширования их сводок, Infer может выполнять повторный анализ только изменённых функций и их зависимостей. Это делает инструмент особенно удобным для интеграции в CI/CD-пайплайны и системы код-ревью.

Недостатки:

- поддержка Python в состоянии ранней разработки;
- ограничения анализа псевдонимов. Точное определение модифицируемых параметров и отслеживание зависимостей через указатели требует качественного анализа псевдонимов, который трудно реализовать масштабируемо. Как отмечается в документации Infer, существующий механизм анализа псевдонимов "далёк от совершенства" и может приводить к ошибкам в определении чистоты функций;

					ПГУ1.090301.01.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

– сложность – кроме использования языка OCaml, внутренняя архитектура реализована довольно сложно, крайне вероятно появление проблем при расширении.

1.1.3 Постановка задачи

В рамках разработки модуля анализа необходимо, чтобы он обходил синтаксическое дерево исходного кода, идентифицируя точки входа, объявления функций и точки вызовов, включая вложенные функции, декораторы и импорты модулей. Должен быть реализован алгоритм определения имен для установления связей между местом вызова и определением функции. Должна быть обработка локальных и глобальных функций, методов классов с простейшей иерархией наследования. Разрабатываемый компонент должен находить все возможные пути выполнения от целевой функции до точек входа в программу, возвращая структурированные объекты в формате, пригодном для анализа.

Графический интерфейс должен быть понятным и удобным для работы как с небольшими, так и с объемными проектами. Должна быть возможность импорта директории с исходным кодом проекта или одного файла-источника, после чего должен возвращаться граф вызовов для указанных единиц кода. Также должна быть возможность экспорта результата.

Компонент должен быть протестирован на синтетических примерах и реальных открытых проектах для оценки точности разрешения вызовов и полноты построения стека. Код должен быть расширяемым, с учетом возможности его встраивания в инструменты статического анализа.

Технические требования: язык Python версии 3.7 и выше для упрощения поддержки и интеграции в инструменты анализа Python-кода. Отсутствие тяжелых внешних зависимостей, допускается использование упрощенных структур данных для хранения графов. Модульная архитектура, позволяющая заменять отдельные блоки, в частности модуль разрешения импортов, без переписывания всей логики компонента.

					ПГУ1.090301.01.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Выбор средств и инструментов разработки

1.2.1 Выбор языка программирования

Си – процедурный язык низкого уровня. Один из первых языков программирования, применяется для системного программирования, разработки операционных систем, драйверов, встроенных систем и высокопроизводительных вычислений.

Достоинства:

- высокая производительность – программы на С выполняются в 10–100 раз быстрее интерпретируемых языков при оптимизированной компиляции;
- низкое потребление памяти – отсутствие накладных расходов на виртуальную машину или интерпретатор;
- предсказуемое поведение – полный контроль над выделением и освобождением ресурсов;
- кроссплатформенность на низком уровне – компиляторы под любую архитектуру (x86, ARM, RISC-V);
- огромная база кода – ядро Linux, Windows, драйверы, базы данных (PostgreSQL, SQLite) написаны на Си.

Недостатки:

- ручное управление памятью – основной источник утечек памяти, висячих указателей и переполнения буфера (уязвимости типа CVE);
- низкая скорость разработки – даже простая задача (например, динамический массив строк) требует десятков строк кода;
- малое количество встроенных типов – нет списков, словарей, множеств, всё реализуется вручную или через сторонние библиотеки;
- сложность отладки – ошибки с памятью часто приводят к неопределенному поведению;
- минимальная абстракция – код сложно поддерживать и расширять при росте проекта.

Java – объектно-ориентированный язык со статической типизацией, компилируемый в байт-код для виртуальной машины JVM (Java Virtual Machine). Часто используется в корпоративной разработке, для серверных приложений и для разработки на Android.

Достоинства:

- автоматическое управление памятью – снижает количество ошибок по сравнению с Си;
- высокая производительность благодаря JIT-компиляции. По скорости уступает Си, но превосходит чистую интерпретацию;
- кроссплатформенность – один и тот же байт-код выполняется на любой ОС, в которой запускается JVM (Windows, Linux, macOS);
- развитые инструменты для многопоточности, множество встроенных типов и функции для написания параллельно исполняемого кода;
- строгая типизация – многие ошибки выявляются на этапе компиляции.

Недостатки:

- более низкая производительность, чем у Си – накладные расходы JVM (до 30–50% в некоторых задачах);
- высокое потребление оперативной памяти – JVM «забирает» от 256 МБ минимум, типичные серверные приложения – 1–4 ГБ;
- строгое ООП – даже простая программа требует написания классов;
- медленный старт – время загрузки JVM и первичной JIT-компиляции делает Java неподходящей для короткоживущих скриптов;
- сложность в изучении – необходимо знать ООП, уметь работать с исключениями, коллекциями, потоками ввода-вывода.

Python – интерпретируемый язык с динамической типизацией, поддерживающий несколько парадигм (процедурное и функциональное программирование, ООП). За последние 5–10 лет стал одним из самых популярных языков в мире, особенно в области машинного обучения, автоматизации, веб-разработки и образования [2].

Достоинства:

- высокая скорость разработки – объем кода, выполняющего типовые действия на Python в 2–5 раз короче, чем аналогичный на Java, и в 10–20 раз короче, чем на C.
- низкий порог входа – простой синтаксис позволяет сосредоточиться на алгоритмах, а не на управлении памятью или типах.
- динамическая типизация – гибкость при создании прототипов и исследовательском программировании.
- огромное количество готовых библиотек для решения любых задач
- кроссплатформенность – интерпретатор доступен для всех основных ОС.
- отличная интеграция с C/C++ – через ctypes, Cython, pybind можно ускорять критические к скорости выполнения участки кода.
- универсальность – от скриптов автоматизации до веб-серверов, от анализа данных до встраиваемых систем.

Недостатки:

- низкая производительность – в 10–100 раз медленнее C и в 2–5 раз медленнее Java для вычислительных задач.
- высокое потребление памяти – каждый объект имеет большой заголовок, к примеру, типичный int занимает 28 байт.
- GIL – ограничивает параллельные вычисления, так как в пределах одного процесса одновременно может выполняться только один поток Python-кода.

По итогам проведенного сравнения для разработки данного проекта был выбран язык Python. Выбор обусловлен в первую очередь темой проекта, так как количество инструментов статического анализа кода на этом языке крайне мало, несмотря на растущую популярность языка.

1.2.2 Выбор среды разработки

PyCharm от JetBrains – профессиональная среда разработки, созданная специально для Python. Выполняет глубокий анализ кода, имеет возможность умного автодополнения, встроенный терминал, мощный отладчик, поддержку юнит-тестирования, интеграцию с Git, а также инструменты для работы с базами данных и веб-фреймворками.

Visual Studio Code – легковесный, но расширяемый текстовый редактор от Microsoft. Благодаря плагину Python, редактор превращается в полноценную среду разработки с автодополнением на основе сервера языка, отладчиком, поддержкой Jupyter и встроенным терминалом.

Eclipse – свободная интегрированная среда разработки модульных кроссплатформенных приложений. Наиболее известные приложения на основе Eclipse – «Eclipse IDE» для разработки ПО на множестве языков программирования. Наиболее популярные: «Java IDE», «Eclipse CDT» для разработки приложений на C/C++, «PyDev» для разработки приложений на python [9].

В результате сравнений и выбора остановился на Eclipse IDE по причинам бесплатности, кроссплатформенности, расширяемости, поддержки многих языков программирования и длительного опыта работа с ней.

1.2.3 Выбор библиотеки графического интерфейса

При выборе графической библиотеки для реализации пользовательского интерфейса были рассмотрены три варианта: Tkinter, Matplotlib и PyQt.

Tkinter имеет весомое преимущество, она входит в стандартную библиотеку языка. При этом имеет множество недостатков: библиотека имеет устаревший внешний вид виджетов, скромный набор компонентов, сложность создания собственных элементов интерфейса и проблемы, связанные с отображением на разных платформах. На рисунке 3 изображен пример приложения, использующего эту библиотеку.

Address Entry Form

First Name:

Last Name:

Address Line 1:

Address Line 2:

City:

State/Province:

Postal Code:

Country:

Clear Submit

Рисунок 3 – Пример окна на Tkinter

Matplotlib в основном предназначена для визуализации данных и работы с NumPy, поэтому обладает слабыми интерактивными возможностями, и не годится для созданий полноценных и удобных GUI. На рисунке 4 показан пример использования библиотеки.

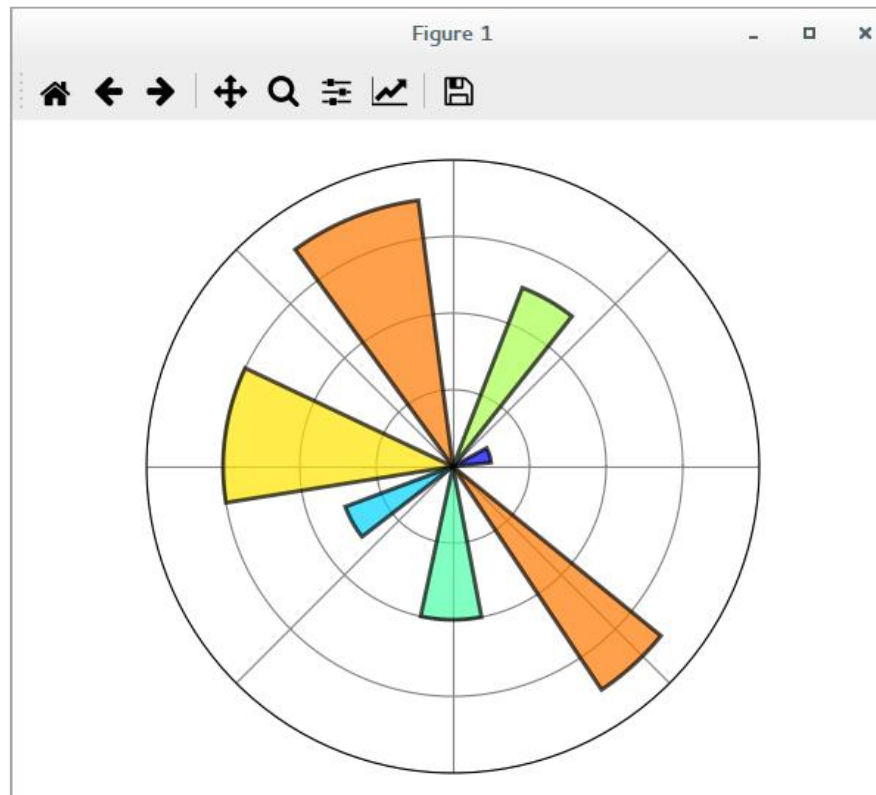


Рисунок 4 – Пример использования Matplotlib

Выбор был сделан в пользу PyQt. В отличие от предыдущих, PyQt имеет более широкие возможности: несколько сотен готовых классов и функций для работы с самыми разными элементами графического интерфейса, в том числе файлами, графикой. Отличается хорошей кроссплатформенностью, позволяет приложениям одинаково выглядеть как в Windows, так и в macOS и Linux.

Особое удобство в библиотеке предоставляет механизм сигналов и слотов, которые сильно упрощают взаимодействие между разными виджетами и элементами интерфейса, делая интерфейс максимально интерактивным с минимальной ручной синхронизацией. Пример использования библиотеки изображен на рисунке 5.

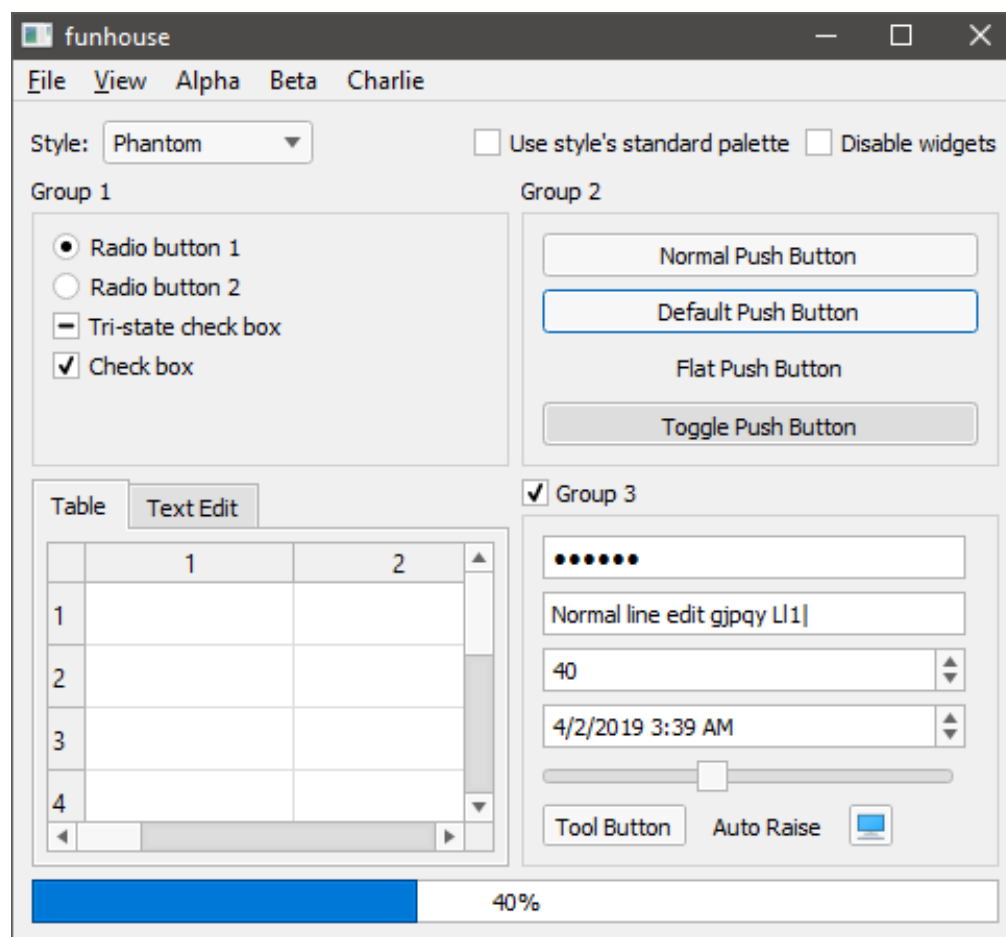


Рисунок 5 – Пример использования PyQt

1.3 Проектирование и разработка

1.3.1 Архитектура модуля анализа

Модуль анализа реализован в пакете graph и отвечает за анализ исходного кода. Там же выполняется построение графа вызовов функций. Модуль состоит из двух файлов: `resolvers.py` – вспомогательные функции разбора и разрешения имён, и `builder.py` – основного файла, содержащим логику обхода AST.

`resolvers.py` включает в себя следующие функции:

- `get_source_filepaths(source_dir)` – рекурсивный обход директории проекта для поиска файлов с исходным кодом программ;
- `parse_ast(filepath)` – чтение файла и построение абстрактного синтаксического дерева с помощью стандартной библиотеки `ast`;
- `resolve_attribute_chain(node)` – восстановление полной цепочки атрибутов вызова;
- `resolve_callee(raw, caller, defined)` – разрешение имени вызываемой функции, путем поиска среди известных определений с учётом `self`, `cls` и контекста модуля.

`builder.py` содержит три класса-обходчика AST, диаграмма которых изображена на рисунке 6:

- `DefinitionVisitor` – первый проход по синтаксическому дереву, собирает множество полностью квалифицированных имён всех определённых функций и методов в проекте, формирует словарь индекс `defined_functions`;
- `CallVisitor` – вспомогательный обходчик, выявляет вызовы функций внутри произвольного выражения или тела блока кода, используется как инструмент для `BranchVisitor`;
- `BranchVisitor` – основной обходчик, выполняет обход дерева, отслеживает стек имён (модуль – класс – функция – вложенная функция и т. п.), фиксирует безусловные вызовы функций и разбирает ключевые слова блоков кода (`if`, `elif`, `else`, `for`, `while`, `try`, `except`, `finally`, `with`) в структуры `BranchNode` и `BranchArm` [6].

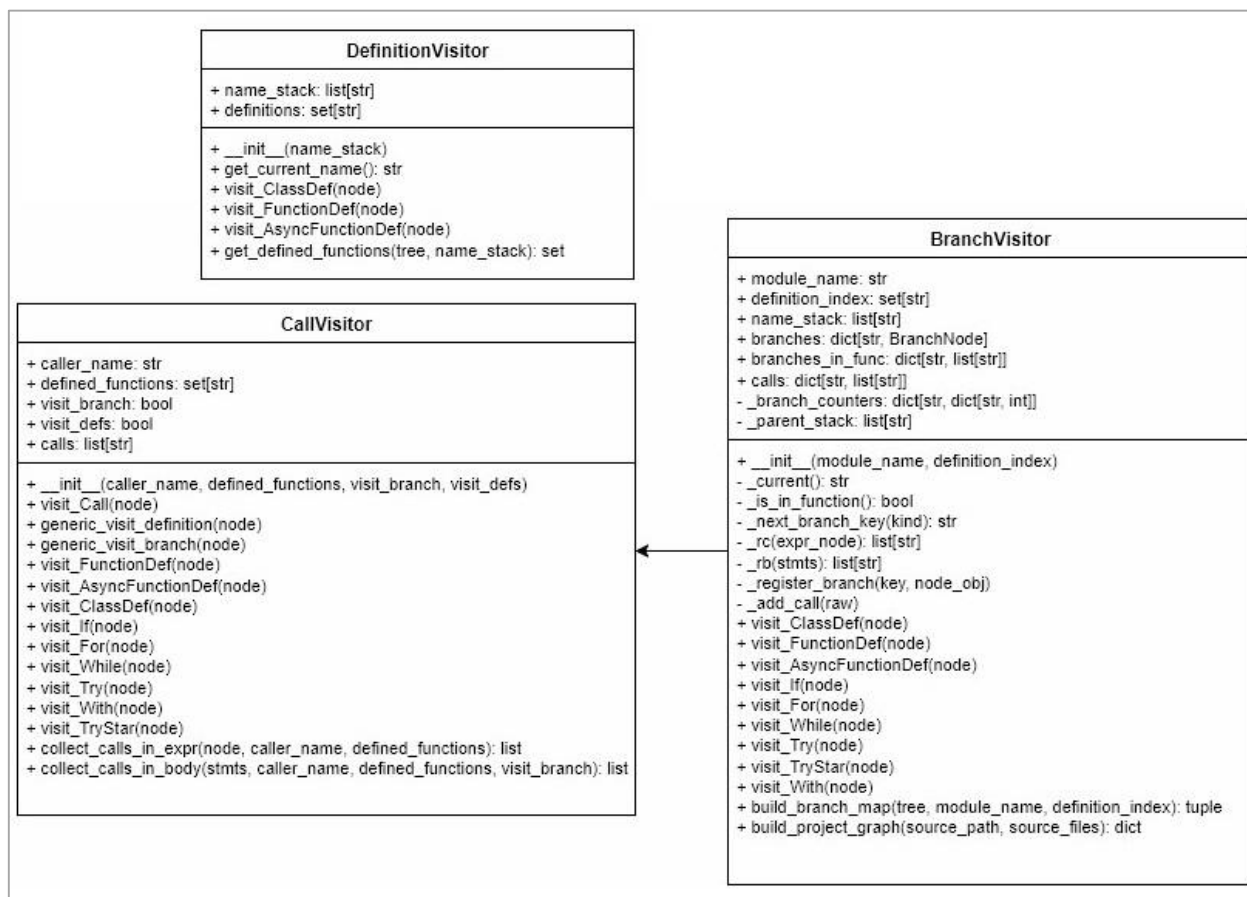


Рисунок 6 – UML-диаграмма классов Visitor

BranchNode и BranchArm – структуры данных, описывающие ветвление. BranchNode хранит список рёбер-ветвей (arms), где каждый BranchArm – это текст заголовка ветви, тип ветви (if, elif, else, for, while, try, except, finally, with), вызовы в условии (calls_in_cond) и вызовы в теле ветви (calls). Диаграмму этих классов можно увидеть на рисунке 7.

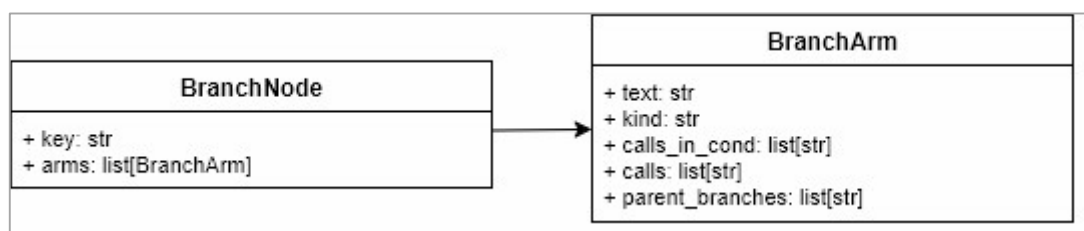


Рисунок 7 – UML-диаграмма классов BranchNode и BranchArm

`build_project_graph(source_path, source_files)` – основная функция. Принимает путь к проекту и список файлов, выполняет два прохода по всем AST: построение индекса определений, затем полный анализ. Возвращает словарь с множеством декларируемых функций, общей картой ветвлений, картой ветвлений в функциях, и картой вызовов. На рисунке 8 показаны данные, созданные в результате анализа.

▼ retval	dict: {'defined_functions': {'draw.QT5LabyrinthDraw.draw', 'win
> 'defined_functions' (1366957592128)	set: {'draw.QT5LabyrinthDraw.draw', 'window.MWindow.initMe
> 'branches' (1366957624304)	dict: {'draw.QT5LabyrinthDraw.draw_styled_3d_rectangle:if_1': l
> 'branches_in_func' (1366957593248)	dict: {'draw.QT5LabyrinthDraw.draw_styled_3d_rectangle': ['dra
▼ 'calls' (1366949204656)	dict: {'draw.QT5LabyrinthDraw.draw': ['draw.QT5LabyrinthDraw
> protected variables	
> 'draw.QT5LabyrinthDraw.draw' (1366984183520)	<class 'list'>: ['draw.QT5LabyrinthDraw.draw_styled_3d_rectang
> 'window.MWindow.newMap' (1366984190640)	<class 'list'>: ['window.MWindow.requestMapDimensions']
> 'window.MWindow.genMap' (1366984191200)	<class 'list'>: ['window.MWindow.requestMapDimensions']
> 'window.MWindow.solve' (1366984192160)	<class 'list'>: ['logic.Labyrinth.lee']
> 'window.MWindow.reload' (1366984192880)	<class 'list'>: ['window.MWindow.openMap']

Рисунок 8 – Результат анализа

1.3.2 Алгоритм работы модуля анализа

Работа модуля анализа делится на четыре этапа – подготовка, два логических прохода синтаксического дерева и сбор результатов в завершении.

Вызов `build_project_graph` инициирует вызов функции `get_source_filepaths` для определения списка файлов исходного кода в директории проекта. Затем для каждого найденного файла выполняется вызов `parse_ast` с целью построения синтаксического дерева. После этого выполняется непосредственно анализ.

На первом проходе для каждого исходного файла по его AST запускается `DefinitionVisitor`. Обходчик рекурсивно посещает узлы `ClassDef`, `FunctionDef`, `AsyncFunctionDef` сохраняя стек имён. При посещении функции, к стеку добавляется её локальное имя. Так формируется полное квалифицированное имя вида `module/subdir.ClassName.method_name`, которое добавляется в словарь `defined_functions`. После первого прохода по всем файлам проекта формируется единый индекс известных функций.

На втором проходе для каждого AST запускается `BranchVisitor`. Обходчик поддерживает:

– `name_stack` – текущий путь: модуль - класс - функция;

- `parent_stack` – стек активных ветвей, для вложенных условий;
- `branch_counters` – счётчики ветвей для генерации уникальных ключей.

При посещении узла `FunctionDef` регистрируются использования декораторов как безусловные вызовы функции. Затем выполняется вызов `CallVisitor.collect_calls_in_body` для сбора вызовов в теле функции, то есть без углубления в дочерние ветви. Результат записывается в `calls[current_func]`. После этого выполняется рекурсивный тела функции для поиска ветвлений.

При посещении ключевых слов, содержащих блоки кода (`If`, `For`, `While`, `Try`, `With`) генерируется уникальный ключ вида `module.Class.func::if_1`, создаётся объект `BranchNode`, и для каждой ветви создаётся `BranchArm` с текстовым заголовком, списком вызовов в условии и в теле ветви, после чего `BranchNode` регистрируется в `branches` и `branches_in_func`.

Функция `resolve_callee` реализует логику разрешения имён в виде трех проверок: если это точное совпадение с элементом `defined_functions`, выполняется возврат этого же имени. Если имя начинается с `self.` или `cls.`, то выполняется поиск метода в рамках текущего класса, затем расширенный поиск по суффиксу среди всех определений. Если точного совпадения нет – осуществляется поиск по суффиксу `.method_name`, с разрешением неоднозначностей приоритетом одноимённых функций из того же модуля.

Результаты обработки нескольких файлов объединяются в `build_project_graph`: словари `branches`, `branches_in_func` и `calls` накапливаются в общие структуры, а в списке вызовов убираются дубликаты. Итоговый словарь передаётся в модуль графического интерфейса, пример его значений приведен на рисунке 9.

```
defined_functions: set[str] = set()
all_branches: dict[str, BranchNode] = {}
all_branches_in_func: dict[str, list[str]] = {}
all_calls: dict[str, list[str]] = {}
```

Рисунок 9 – Переменные, заполняемые обходчиками

1.3.3 Архитектура графического интерфейса

Графический интерфейс реализован с использованием библиотеки PyQt5 и хранится в пакете gui. Архитектурно GUI следует шаблону «Представление – Данные – Контроллер», с разделением на компоненты отображения, бизнес-логики построения графа и логики управления состоянием [7].

Пакет состоит из нескольких модулей, описываемых ниже.

lib.py – разнообразные константы, перечисления и вспомогательные функции:

- Const – строковые константы цветов, размеры узлов и стрелок;
- QtColors / QtConst – объекты Qt (QColor, QFont), используемые для отрисовки интерфейса;
- EdgeType, IconType, SortBy – перечисления (enum) типов рёбер, иконок и режимов сортировки;
- вспомогательные функции create_icons, confirm_warning, _cond_color.

builders.py – алгоритмические функции построения и разметки графа:

- build_edges_from_data – преобразование словаря graph_data в список рёбер с учётом настроек построения графа: показ ветвлений, объединения равноценных ветвлений, показ только внешних функций;
- hierarchical_layout – алгоритм иерархического размещения узлов: определение уровней, различные сортировки для минимизации пересечений рёбер, итеративное сглаживание положений узлов.

main_window.py – главное окно приложения:

- MainWindow(QMainWindow) – создаёт компоновку: QSplitter с боковой панелью и холстом, меню, соединяет сигналы между компонентами;
- GraphSerializer – сериализация/десериализация графа в JSON-формат файла-проекта.

gui/widgets/ – пакет визуальных компонентов, включенных в диаграммы на рисунках 10 и 11:

- `NodeItem(QGraphicsItem)` – визуальный элемент узла, поддерживает перетаскивание, выделение, события наведения; вызывает сигналы `position_changed` и `clicked`;
- `EdgeItem(QGraphicsPathItem)` – визуальный элемент ребра графа (стрелки), поддерживает прямолинейный и криволинейный режимы отрисовки, подсветку при наведении;
- `CallGraphView(QGraphicsView)` – главный холст; управляет сценой (`QGraphicsScene`), координирует построение и перестройку графа, обрабатывает навигацию (масштаб колесом мыши, прокрутку, выделение узлов), реализует анимацию плавного центрирования и мигания узла;
- `BaseTree(QWidget)` – базовый класс боковых панелей с деревом (`QTreeWidget`), общие контекстные меню с обработчиками событий;
- `NavigatorTree(BaseTree)` – панель навигатора: иерархическое дерево всех функций проекта с фильтрацией, сортировкой и контекстным меню;
- `SelectionTree(BaseTree)` – панель выделения: отображает текущий набор выделенных узлов с группировкой по модулям/классам.

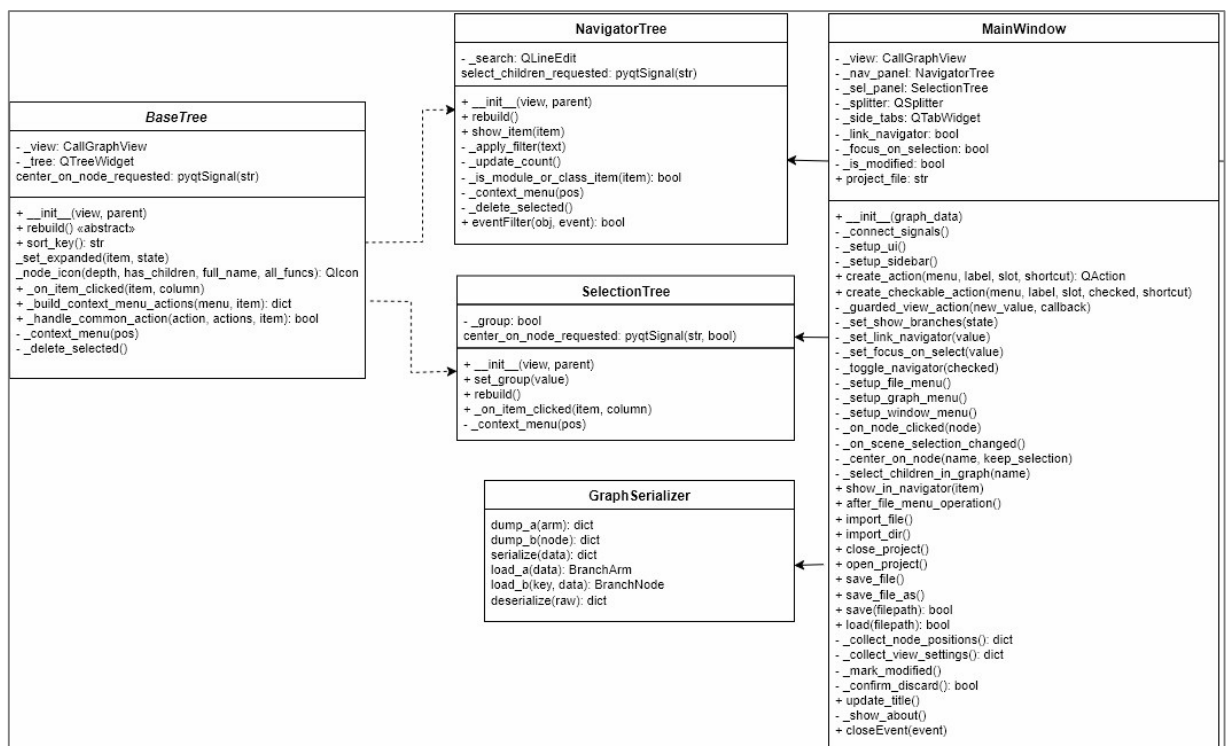


Рисунок 10 – UML-диаграмма классов, не связанных с отрисовкой графа

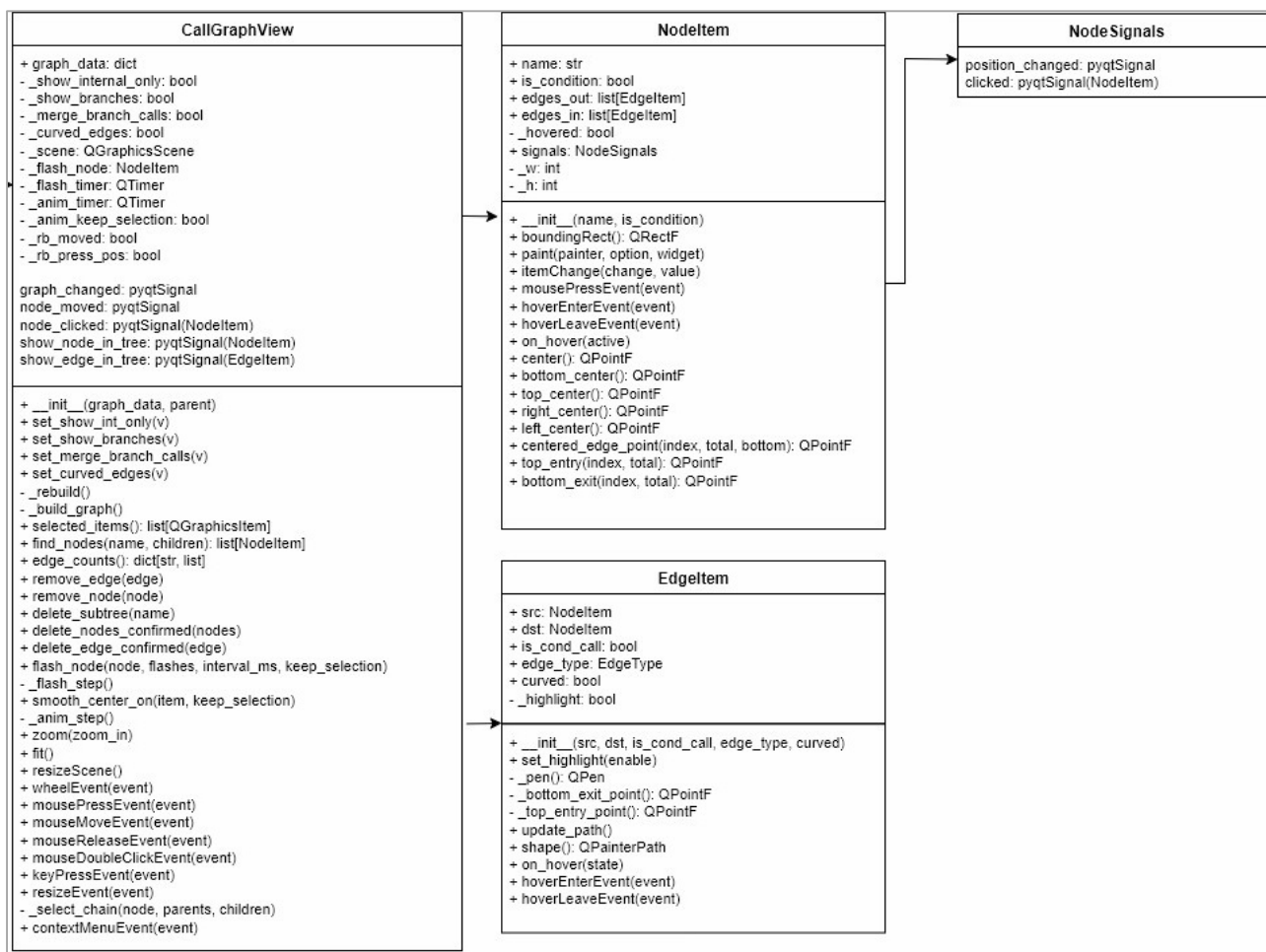


Рисунок 11 – UML-диаграмма классов, выполняющих отрисовку графа

Взаимодействие компонентов организовано через сигнально-слотовый механизм библиотеки Qt (таблица 1) [8].

Таблица 1 – Список сигналов:

Сигнал	Источник	Получатели
graph_changed	CallGraphView	NavigatorTree.rebuild, SelectionTree.rebuild, MainWindow._mark_modified, MainWindow.update_title
node_moved	CallGraphView	MainWindow._mark_modified
node_clicked	CallGraphView	MainWindow._on_node_clicked
show_node_in_tree	CallGraphView	MainWindow.show_in_navigator
show_edge_in_tree	CallGraphView	MainWindow.show_in_navigator
selectionChanged	QGraphicsScene	MainWindow._on_scene_selection_changed

Сигнал	Источник	Получатели
center_on_node_requested	NavigatorTree / SelectionTree	MainWindow._center_on_node
select_children_requested	NavigatorTree	MainWindow._select_children_in_graph

1.3.4 Алгоритм работы графического интерфейса

При запуске приложения выполняется следующий алгоритм:

- если путь к проекту передан аргументом командной строки, вызывается функция получения списка исходников и функция построения графа, результат сохраняется;

- создаётся главное окно, инициализируются константы, создаются CallGraphView, NavigatorTree, SelectionTree, настраивается компоновка и меню, соединяются сигналы, вызывается view.fit() и _nav_panel.rebuild();

- вызов build_edges_from_data – формируется набор имён узлов (all_node_names), список рёбер с метаданными (edges) и набор узлов-ветвей (condition_node_names);

- разбиение узлов на связанные (имеющие рёбра) и изолированные;

- вызов hierarchical_layout – вычисляются координаты узлов [5];

- создание NodeItem для каждого узла, размещение узла в сцене;

- создание EdgeItem для каждого ребра с привязкой к узлам-источнику и узлу-назначению, добавление в списки edges_out/edges_in соответствующих узлов;

- вызов fit() – масштабирование сцены по содержимому.

Алгоритм заполнения панелей навигации:

- построение словарного дерева имён name_tree из всех функций graph_data;

- рекурсивное заполнение QTreeWidget с назначением иконок (модуль, класс, функция) в зависимости от глубины и принадлежности к all_funcs;

- добавление дочерних элементов ветвлений (BranchNode, BranchArm), окрашенных по типу ветви;

- скрывание элементов, отсутствующих в текущей сцене;
- применение текстового фильтра.

Алгоритм обработки пользовательских действий:

- масштабирование: колесо мыши вызывает функцию `zoom`;
- панорамирование: нажатие левой кнопки мыши на пустое пространство активирует `ScrollHandDrag`;
- выделение рамкой: правая кнопка мыши активирует `RubberBandDrag`;
- навигация: клик по элементу дерева или узлу инициирует `smooth_center_on` – анимацию прокрутки сцены с последующим миганием узла;
- удаление: клавиша `Delete` или контекстное меню, показ диалога подтверждения удаление узла/ребра из сцены и из `graph_data` с подачей сигнала `graph_changed`;
- сохранение или загрузка: `GraphSerializer.serialize` или `.deserialize` преобразует `graph_data` в JSON-совместимый словарь, вместе с позициями узлов и настройками вида сохраняется в файл.

При изменении настроек «Показывать ветви», «Только внутренние вызовы» и др. `_rebuild()` очищает сцену и повторно запускает `_build_graph`. Перестройка панелей навигатора и выделения выполняется автоматически через сигнал `graph_changed`.

1.4 Тестирование и отладка

В процессе разработки выполнялась неоднократная проверка корректности работы приложения, функциональное тестирование модуля анализа на реальных проектах, интеграционное тестирование взаимодействия модуля анализа и графического интерфейса, а также ручное тестирование пользовательского интерфейса.

Для быстрой проверки логики анализа без запуска GUI использовался скрипт `main_cli_demo.py`, выводящий в консоль полный список определённых

функций, безусловных вызовов и всех ветвлений с их структурой, то есть всех данных, получаемых в результате анализа.

1.4.1 Тестирование модуля анализа

Проверялась правильность формирования `defined_functions` для различных структур кода:

- функции на уровне модуля;
- методы классов (включая `staticmethod` и `classmethod`);
- вложенные классы и функции;
- асинхронные функции (`async def`).

Тестовые сценарии включали проекты с несколькими файлами, чтобы убедиться в корректности формирования квалифицированных имён вида `subpackage/module.Class.method`.

Проверялась корректность разрешения вызовов функцией `resolve_callee` на следующих случаях:

- прямые вызовы локально определённых функций;
- вызовы метода через `self.method()` и `cls.method()`;
- вызовы методов через цепочки атрибутов (`os.path.join`, `self.attr.method()`);
- разрешение неоднозначности при совпадении суффиксов имён методов разных классов;
- корректное игнорирование вызовов стандартных библиотечных функций (не входящих в `defined_functions`).

Проверялась полнота извлечения ветвлений, обнаружение всех управляющих конструкций:

- `if/elif/else` с вложенными условиями;
- `for/else` и `while/else`;
- `try/except/else/finally` с несколькими обработчиками;
- `with` с множественными контекстными менеджерами;
- корректное отслеживание стека родительских ветвей (`parent_branches`)

при вложении.

					ПГУ1.090301.01.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

Одним из ключевых проверочных сценариев стал запуск анализатора на исходном коде самого приложения. Полученный граф вызовов сравнивался с ожидаемыми зависимостями между модулями, что позволило выявить и исправить ряд ошибок в логике разрешения цепочек атрибутов. Результат теста показан на рисунке 12.

```
<terminated> main_cli_demo.py [debug] [D:\Program Files\Python310\python.exe]
pydev debugger: starting (pid: 8284)
source search path: D:\Me\dev\univer\4 course\s8\dw\src
Located source files:
D:\Me\dev\univer\4 course\s8\dw\src\main_cli_demo.py
D:\Me\dev\univer\4 course\s8\dw\src\main_gui.py
D:\Me\dev\univer\4 course\s8\dw\src\graph\builder.py
D:\Me\dev\univer\4 course\s8\dw\src\graph\resolvers.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\builders.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\lib.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\main_window.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\edge.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\graph_view.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\node.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\tree_base.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\tree_navigator.py
D:\Me\dev\univer\4 course\s8\dw\src\gui\widgets\tree_selection.py

185 defined functions

unconditional calls:
graph\builder.BranchVisitor._add_call: ['graph\\builder.BranchVisitor._current', 'gra
graph\builder.BranchVisitor._next_branch_key: ['graph\\builder.BranchVisitor._current
graph\builder.BranchVisitor._rb: ['graph\\builder.CallVisitor.collect_calls_in_body',
graph\builder.BranchVisitor._rc: ['graph\\builder.CallVisitor.collect_calls_in_expr',
graph\builder.BranchVisitor._register_branch: ['graph\\builder.BranchVisitor._current
graph\builder.BranchVisitor.visit_for: ['graph\\builder.BranchVisitor._next_branch_ke
graph\builder.BranchVisitor.visit_if: ['graph\\builder.BranchVisitor._next_branch_key
graph\builder.BranchVisitor.visit_try: ['graph\\builder.BranchVisitor._next_branch_ke
graph\builder.BranchVisitor.visit_try_star: ['graph\\builder.BranchVisitor.visit_try']
graph\builder.BranchVisitor.visit_while: ['graph\\builder.BranchVisitor._next_branch_
graph\builder.BranchVisitor.visit_with: ['graph\\builder.BranchVisitor._next_branch_k
graph\builder.DefinitionVisitor.visit_FunctionDef: ['graph\\builder.DefinitionVisitor
gui\builders.hierarchical_layout: ['gui\\lib._center']
```

Рисунок 12 – Проверка модуля анализа на исходном коде этой работы

1.4.3 Тестирование графического интерфейса

Проверялась корректность построения и разметки графа:

- соответствие числа отображаемых узлов и рёбер данным из `graph_data` при различных комбинациях настроек (`show_branches`, `merge_branch_calls`, `internal_only`);
- корректная работа алгоритма `hierarchical_layout`: отсутствие перекрытий узлов одного уровня, правильное расположение изолированных узлов;
- применение типов рёбер (`TRUE`, `FALSE`, `REGULAR`, `CONDITION`) в соответствии со структурой ветвления, что подтверждается рисунком 13.

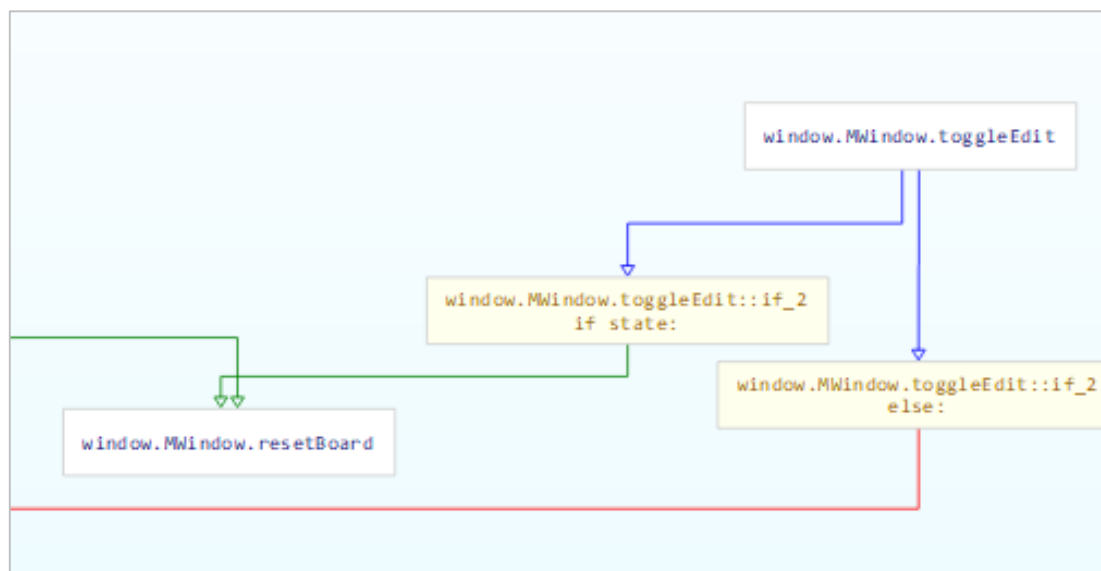


Рисунок 13 – Разные цвета в зависимости от типа рёбер

Проверялись все интерактивные сценарии:

- масштабирование колесом мыши и клавишами «плюс», «минус»;
- навигация по графу и выделение узлов рамкой с помощью мыши, как показано на рисунке 14;
- клик по узлу и синхронизация выбора с панелью навигатора;
- двойной клик по ребру с автоматическим центрированием на ближайшем узле;
- плавное центрирование (smooth_center_on) и анимация «мигания»;
- контекстное меню: выделение цепочки родителей/потомков, удаление с подтверждением.

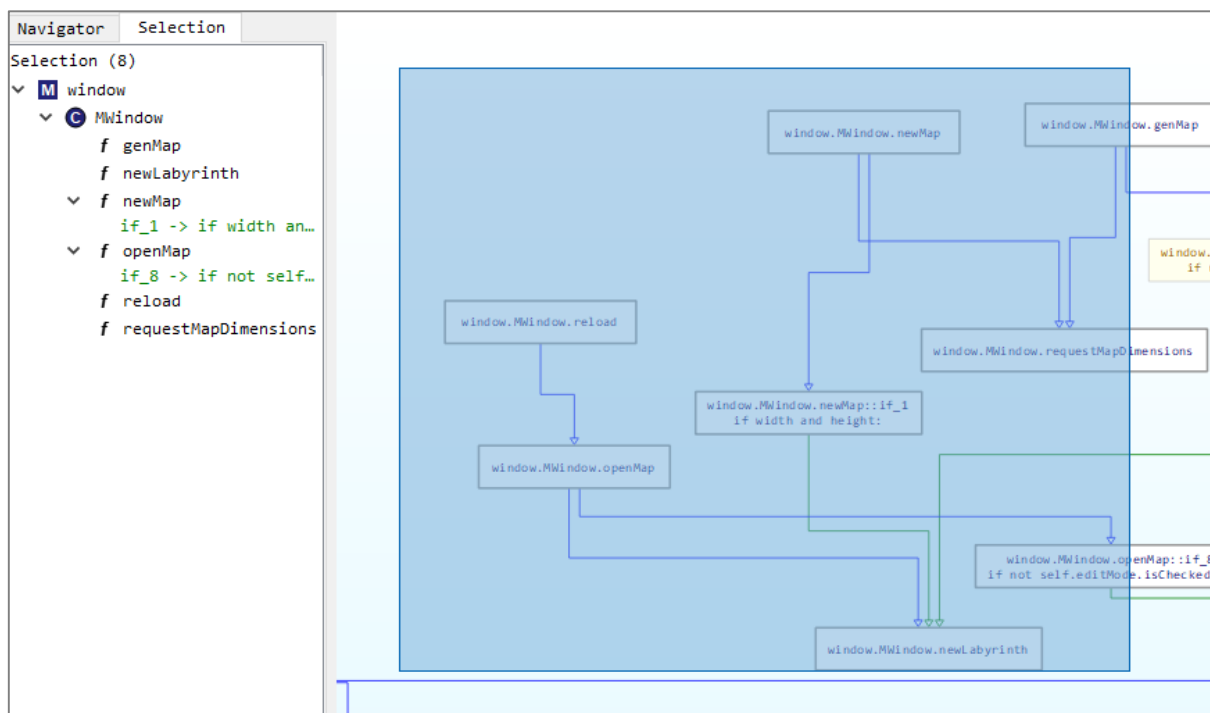


Рисунок 14 – Проверка соответствия выделенных элементов

Тестирование циклом «сохранение-загрузка» проверялся для следующих элементов состояния:

- graph_data (ветви и вызовы);
- позиции всех узлов;
- настройки вида (show_branches, merge_branch_calls, curved_edges, видимость боковой панели, размеры разделителя).

После загрузки файла проекта граф и позиции узлов восстанавливались идентично сохранённому состоянию.

1.3 Анализ результатов

Для начала работы необходимо создать, или открыть проект через меню «File», показанном на рисунке 15. Доступные варианты: создать проект из одного файла с исходным кодом, из директории с исходным кодом, или открыть уже существующий проект.

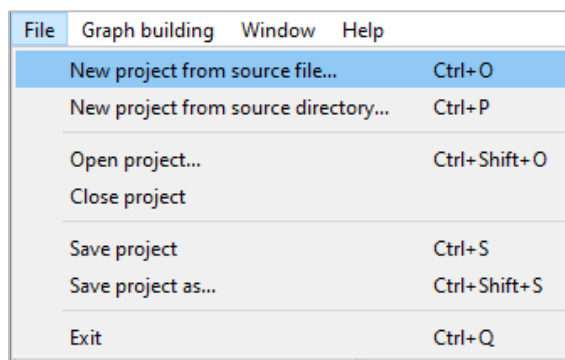


Рисунок 15 – Меню «File»

После этого выполняется построение графа, или его загрузка из сохранённых данных. На рисунке 16 показано основное окно приложения, на котором видно построенный граф вызовов, панель навигации по графу, поле ввода для фильтрации узлов в навигаторе, и панель меню.

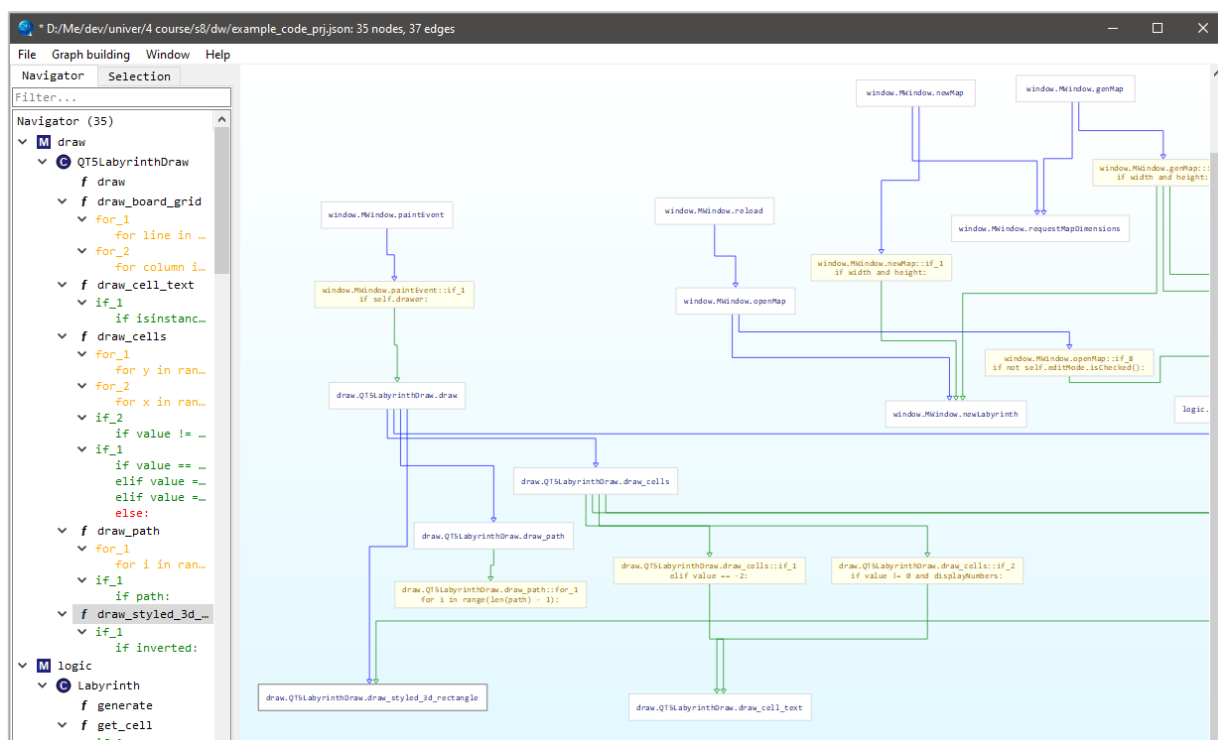


Рисунок 16 – Построенный граф вызовов с ветвлениями

Навигация по графу получилась довольно удобной. Перемещение можно осуществлять с помощью мыши, перемещая её, удерживая нажатой левую клавишу, и с помощью клавиш со стрелками на клавиатуре. Масштаб

отображаемого графа меняется с помощью колёсика мыши и клавиш «ПЛЮС» и «МИНУС» на клавиатуре.

В случае, если нет нужды работы с ветвлениями, можно отключить их отображение в меню «Graph building», показанном на рисунке 17. После этого будет показано предупреждение, и граф будет перестроен, что видно на рисунках 18-19. Аналогичное предупреждение показывается при изменении любой настройки в этом меню.

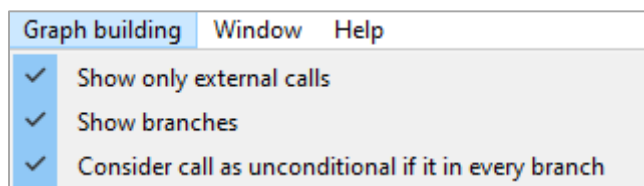


Рисунок 17 – Меню «Graph building»

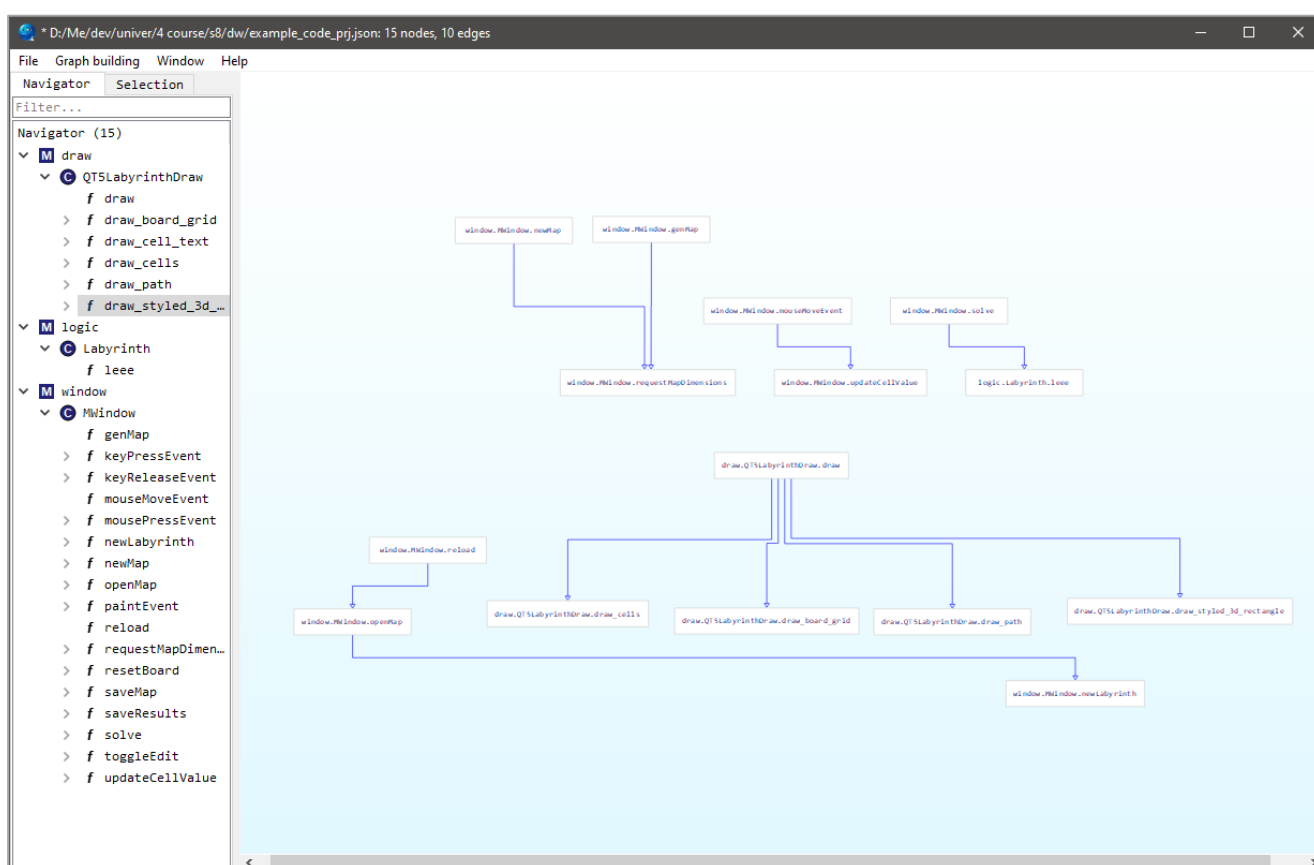


Рисунок 18 – Этот же граф, но без ветвлений

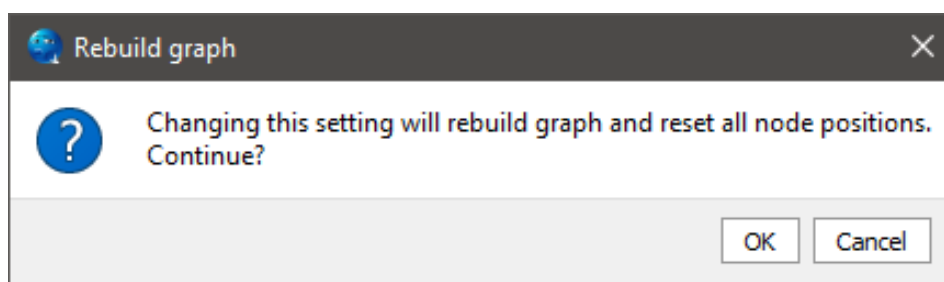


Рисунок 19 – Запрос подтверждения при изменении режима построения графа

Для работы с большой кодовой базой предусмотрен фильтр, который позволяет легко найти нужную функцию в панели навигации, через которую можно одним кликом перейти к соответствующему узлу в графе. Пример использования приведен на рисунке 20.

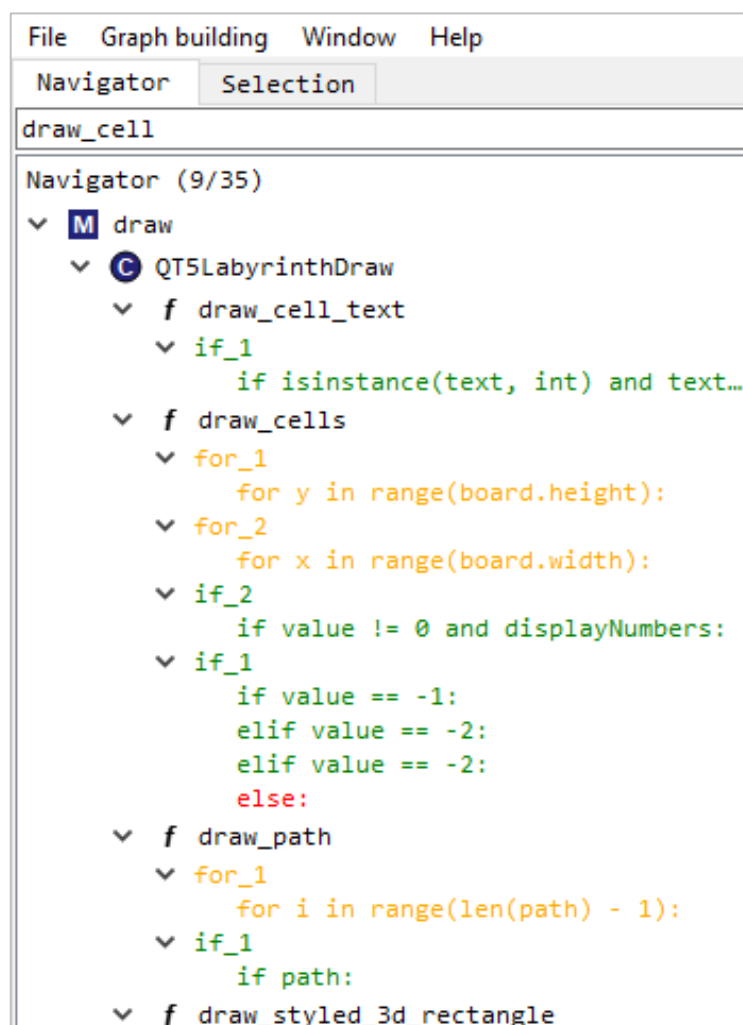


Рисунок 20 – Пример использования фильтра в навигаторе

При разработке приложения были предусмотрены функции для облегчения работы в крайне перегруженном графе:

- подсветка всех связанных ребер при наведении на узел, как на рисунке 21;
- подсветка ребра при наведении на него;
- возможность перехода к начальному и конечному узлу любого ребра через контекстное меню, показанном на рисунке 22, а также двойным кликом для перехода в противоположный конец ребра;
- возможность перехода к узлу в панели навигации, для более точного определения его принадлежности к модулю или классу через меню, показанном на рисунке 23.

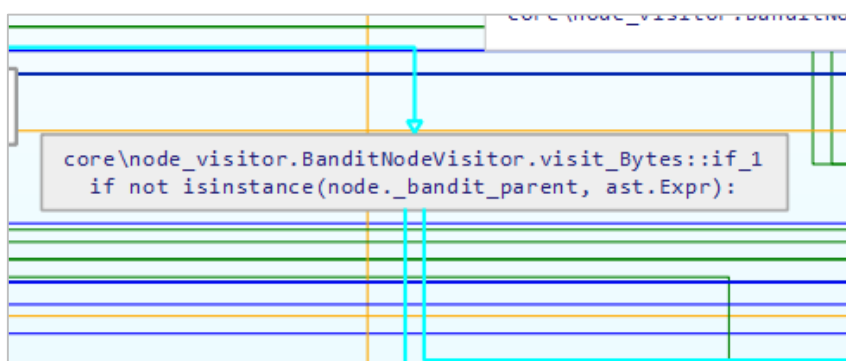


Рисунок 21 – Подсветка ребер при наведении на узел графа

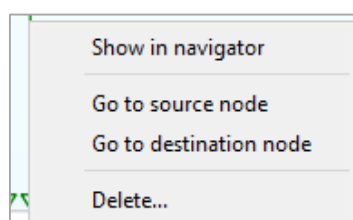


Рисунок 22 – Контекстное меню узлов графа

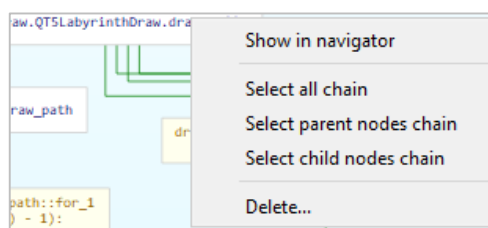


Рисунок 23 – Контекстное меню ребер графа

Построенный граф можно частично редактировать. Реализованный в приложении функционал по редактированию включает в себя:

- возможность свободного перемещения узлов по графу с помощью мыши, в том числе всех выделенных (положение ребер меняется автоматически);
- возможность удаления выделенных или конкретных узлов из графа (вместе со связанными ребрами) с помощью клавиши «Delete» и контекстного меню, показанном на рисунке 22;
- возможность удаления конкретных ребер с помощью контекстного меню, приведенном на рисунке 23;
- возможность удаления дочерних узлов выбранного модуля, класса, функции в панели навигации, в том числе нескольких с помощью клавиши «Ctrl» и контекстного меню, показанном на рисунке 24;
- возможность выделения цепочки всех родительских или дочерних узлов относительно выбранного узла через меню, показанном на рисунке 23;
- возможность выделения всей дочерних узлов относительно выбранного модуля, класса, или функции в панели навигации, что показано на рисунке 24;

При перемещении по графу, выделению узлов любым способом или переходу к нему, прошлое выделение сохраняется, что позволяет выполнять навигацию по графу и выбирать элементы без потери промежуточного результата.

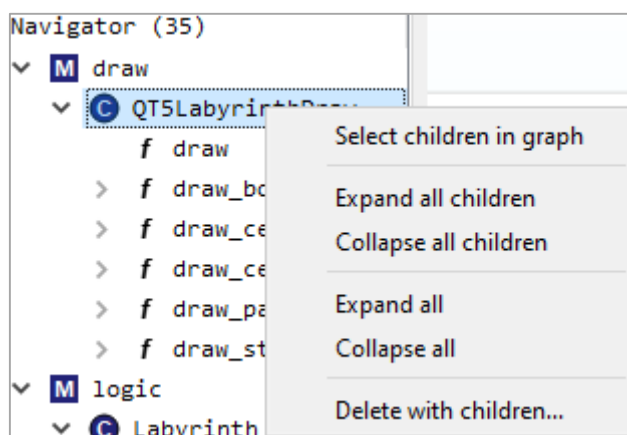


Рисунок 24 – Контекстное меню элементов в навигаторе

Для удобства, по умолчанию включена синхронизация панели навигации с графом. То есть, при клике на узел в графе, в панели навигации выбирается соответствующий узел, и наоборот. В случае выбора нескольких элементов, в интерфейсе разворачивается панель «Selection», отображающая выбранные узлы, что демонстрируется на рисунке 25.

При выборе множества узлов из разных модулей, классов, или функций, группировка по иерархии может мешать, в таком случае целесообразно её отключить. Должно получиться, как на рисунке 26. В панели меню «Window» доступен выбор типов сортировки для панелей «Navigator» и «Selection», там же доступны для отключения все вышеописанные функции, как показано на рисунке 27.

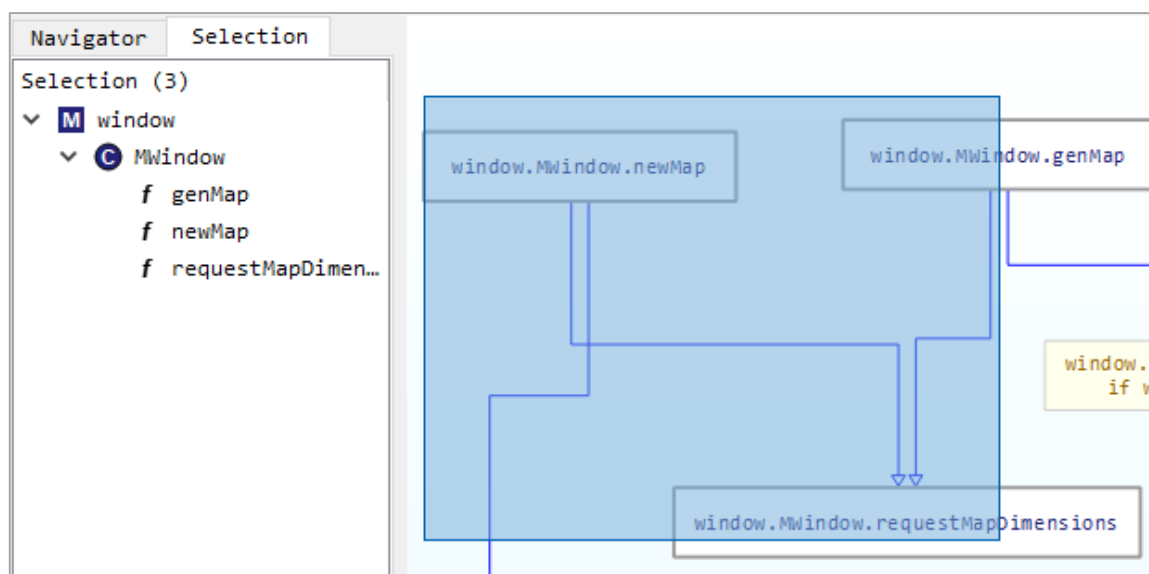


Рисунок 25 – Выделение узлов

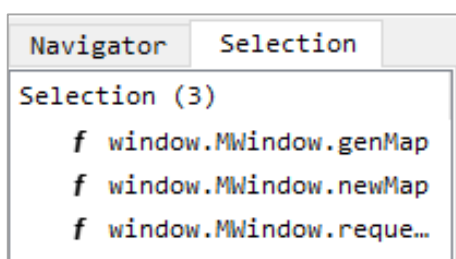


Рисунок 26 – Выбраны те же узлы, но группировка отключена

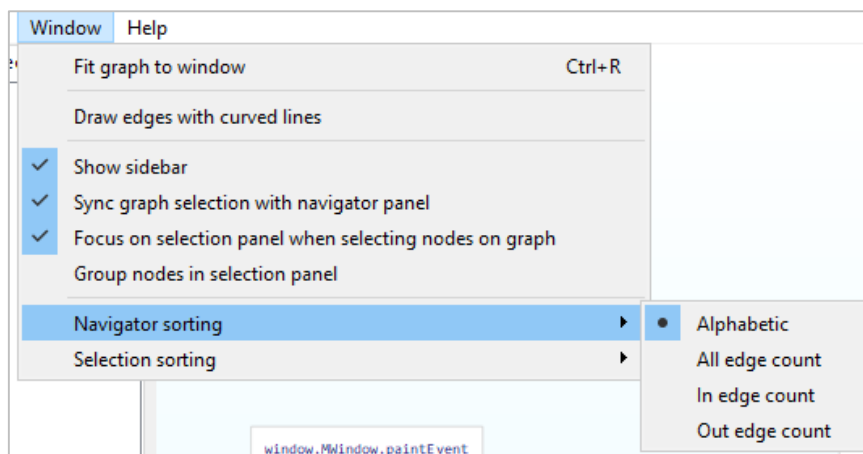


Рисунок 27 – Меню с визуальными настройками приложения

По умолчанию, рисование ребер графа выполняется с помощью прямых линий, так как при плотной компоновке узлов они воспринимаются проще. Возможно, в некоторых ситуациях, рисование с помощью кривых линий более предпочтительно. Этот режим можно включить в меню «Window», которое показано на рисунке 27. Пример результата показан на рисунке 28.

Отдельно стоит отметить, что перед любой операцией удаления элементов выполняется запрос подтверждения, как на рисунке 29. Аналогично при выходе из приложения или проекта без сохранения изменений, что видно на рисунке 30.

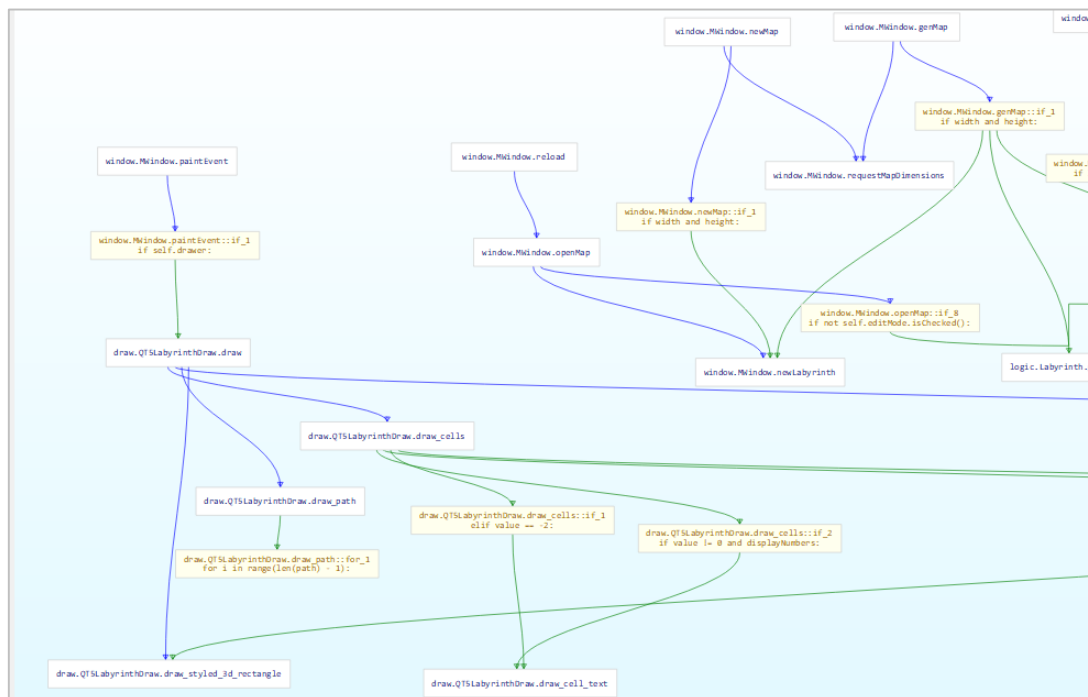


Рисунок 28 – Режим отрисовки ребер с помощью кривых

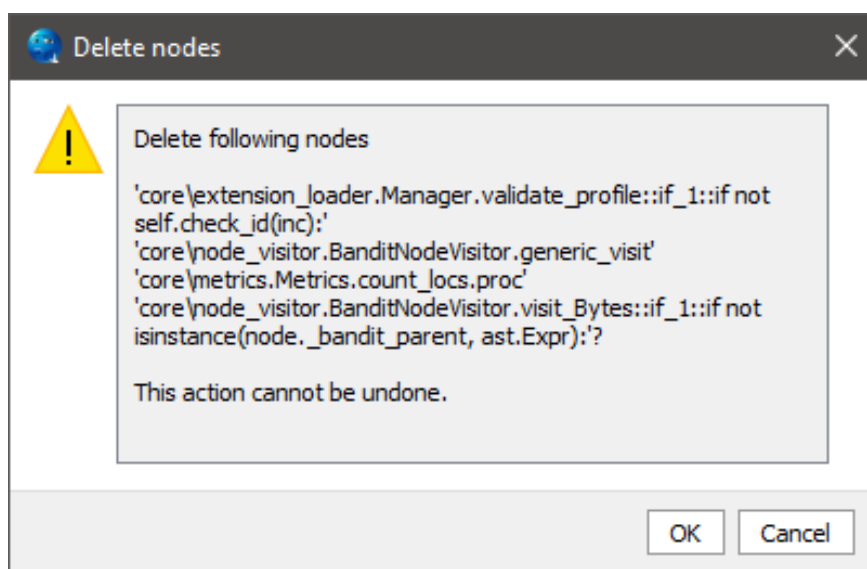


Рисунок 29 – Запрос подтверждения при удалении

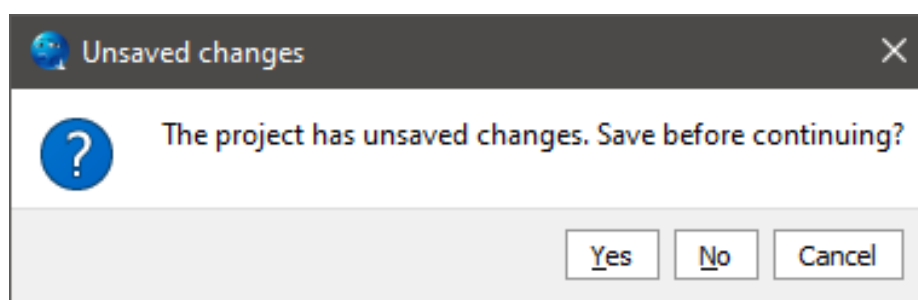


Рисунок 30 – Запрос подтверждении при несохранённых изменениях

2 Охрана труда

Работы за ПК требуют соблюдения различных мер безопасности для сохранения здоровья. Особенно это актуально для людей, специальность работы которых подразумевает длительную сидячую работу за компьютером. По этой причине современное законодательство в области охраны труда обязует выполнять несколько важных требований [10].

2.1 Требования к рабочему месту.

Основные нормы, которые позволяют минимизировать вред от работы за компьютером в долгосрочной перспективе:

- рабочие места должны быть изолированы друг от друга. Если сотрудники заняты творческой деятельностью, следует оборудовать их места перегородками высотой 1,5-2 метра;

- конструкция кресла должна регулироваться, чтобы соответствовать индивидуальным физическим параметрам сотрудников. Поверхность кресла должна быть нескользящей и должна обеспечивать возможность его легкой чистки;

- конструкция стола должна предусматривать просторное и удобное размещение используемого оборудования, при этом иметь коэффициент отражения поверхности, не превышающий 0,7;

- площадь одного рабочего места сотрудника должна составлять не менее 6 квадратных метров. Если в работе используются современные плоские компьютерные мониторы, то не менее 4,5 квадратных метров;

- внутренняя отделка помещений должна производиться с использованием материалов, имеющих соответствие необходимым нормам. Максимальный коэффициент отражения поверхности материала потолка составляет – 0,8, для стен – 0,6, для пола – 0,5.

2.2 Требования к помещению

Для обеспечения безопасности требуется соблюдение следующих требований к помещению, в котором размещены рабочие места:

- желательно, чтобы в помещении были окна, через которые может проходить естественный свет. При этом они должны быть ориентированы на север или северо-запад, и оборудованы устройствами для регулирования яркости естественного освещения, например, шторами, жалюзи и т. п.;

- в случае, если работа происходит в помещениях без естественного освещения, то необходимо обеспечить организованное искусственное освещение, соответствующее всем нормам;

- рабочие места должны находиться вдали от силовых кабелей, трансформаторов и другого оборудования, способного создавать помехи для используемой техники. В помещении должно быть организовано защитное заземление, обеспечивающее безопасную работу с компьютером и другим электрическим оборудованием;

- наличие приточно–вытяжной системы, скорость движения воздуха $\leq 0,1$ м/с. Концентрации различных летучих опасных соединений не должны превышать установленные нормы;

- температура воздуха должна быть от 20 до 24 °С, в идеале 22 °С, относительная влажность 40–60 %;

- уровень шума должен составлять не более 50 дБ, это позволит сотрудникам минимально отвлекаться и сосредоточено выполнять работу.

2.3 Правила размещения монитора

При размещении монитора для обеспечения безопасности здоровья необходимо соблюдать следующие требования:

- расстояние между столами не менее 2 метров, а между боковыми поверхностями мониторов – не менее 1,2 метра;

					ПГУ1.090301.01.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

– экран монитора находится на расстоянии не менее 0,6-0,7 метра от глаз, но и не более полутора метров.

– верхняя часть монитора должна быть чуть ниже уровня глаз, желательно наличие антибликового покрытия.

2.4 Режим труда и отдыха при использовании компьютера

В зависимости от категории выполняемой работы и уровня зрительной нагрузки, которая возникает у человека в течение смены, выделена определенная длительность перерывов, зависящая от продолжительности работы:

- при 8 - часовой смене – от 30 до 70 минут;
- при 12 - часовой смене – от 70 до 120 минут.

Для сохранения здоровья следует вставать с рабочего места хотя бы раз в час, всего на несколько минут. Кроме поддержания здоровья спины, это позволит немного отдохнуть глазам от экрана и снизить общее напряжение.

					ПГУ1.090301.01.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения данной работы был проведен анализ предметной области, рассмотрены существующие аналоги, их достоинства и недостатки. С учетом этого были выбраны средства и инструменты разработки, изучены алгоритмы построения графов, способы анализа исходного кода и выявления в нем различных конструкций.

Также были получены знания по работе с абстрактными синтаксическими деревьями, файлами исходного кода на языке Python. Получен опыт написания графического приложения с использованием библиотеки PyQt на языке программирования Python. Изучены и применены на практике методы работы с файлами, графами, и графическими примитивами.

В процессе разработки было создано приложение, выполняющее построение графа вызовов с учетом ветвлений, и позволяющее выполнять удобную навигацию по полученному графу с возможностью его редактирования и сохранения результата.

В дальнейшем возможно улучшение и расширение функционала программы, в том числе добавление поддержки других языков программирования и возможностей более глубокого анализа исходного кода.

					ПГУ1.090301.01.001 ПЗ	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. ГОСТ Р 71207— 2024. Защита информации. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. Статический анализ программного обеспечения. Общие требования. (утв. и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 18 января 2024 г. № 25-ст) (дата обращения 08.04.26)

2. Яворски Михал, Зиате Тарек – Python. Лучшие практики и инструменты. - 3-е издание. Питер, 2021. – 560 с. (дата обращения 13.04.26)

3. Документация статического анализатора Infer. [Электронный ресурс] / Режим доступа: <https://fbinfer.com/docs/about-Infer>, свободный (дата обращения 20.04.26)

4. Документация статического анализатора Pyre. [Электронный ресурс] / Режим доступа: <https://pyre-check.org/docs/pysa-explore/>, свободный (дата обращения 20.04.26)

5. Послойное рисование графа - Википедия. [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Послойное_рисование_графа, свободный (дата обращения 20.04.26)

6. Официальная документация по библиотеке ast языка Python 3. [Электронный ресурс] / Режим доступа: <https://docs.python.org/3/library/ast.html>, свободный (дата обращения 20.04.26)

7. Официальная документация библиотеки PyQt 5. [Электронный ресурс] / Режим доступа: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>, свободный (дата обращения 03.05.26)

8. Официальная документация библиотеки Qt 5.15. [Электронный ресурс] / Режим доступа: <https://doc.qt.io/archives/qt-5.15/>, свободный (дата обращения 12.05.26)

9. Документация плагина Eclipse PyDev [Электронный ресурс] / Режим доступа: https://www.pydev.org/manual_adv_features.html, свободный (дата обращения 12.05.26)

					ПГУ1.090301.01.001 ПЗ	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

10. Инструкция по охране труда при работе с ЭВМ. [Электронный ресурс] / Режим доступа: https://kpfu.ru/portal/docs/F_1086113514/1815.pdf/, свободный (дата обращения 15.05.26).

					ПГУ1.090301.01.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

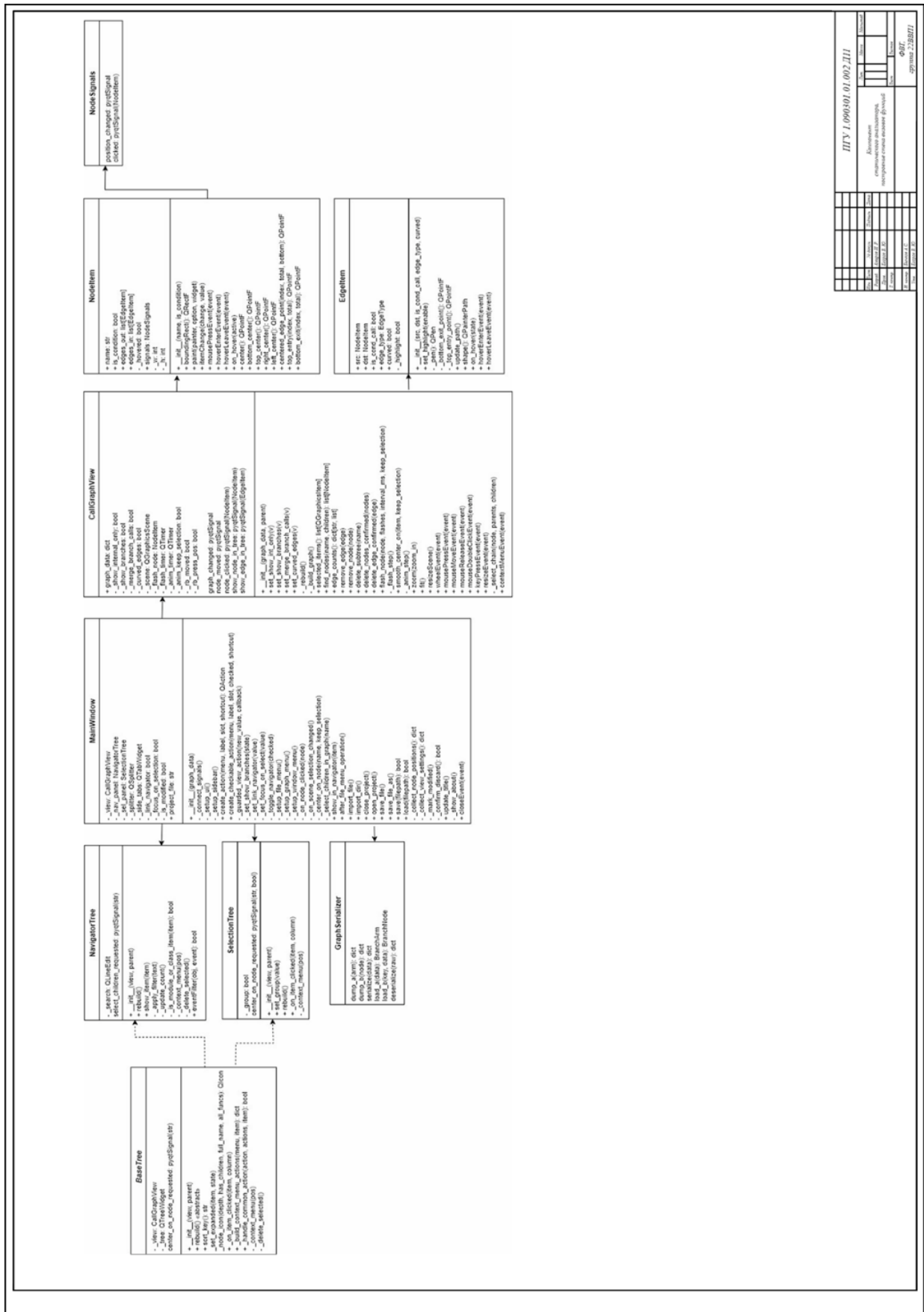
Приложение А

(Обязательное)

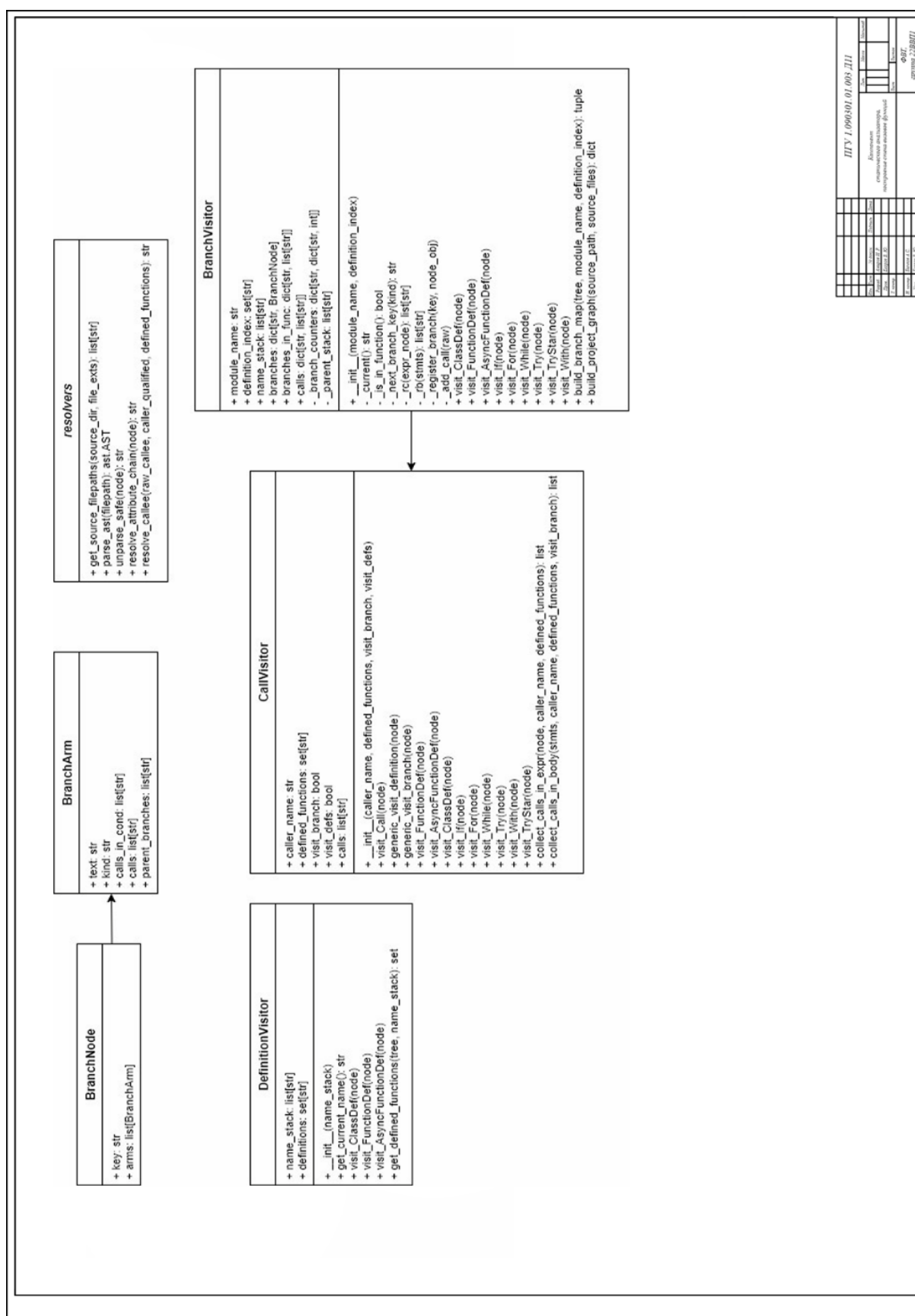
Компонент статического анализатора, построение стека вызовов функций
UML-диаграммы

					ПГУ1.090301.01.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

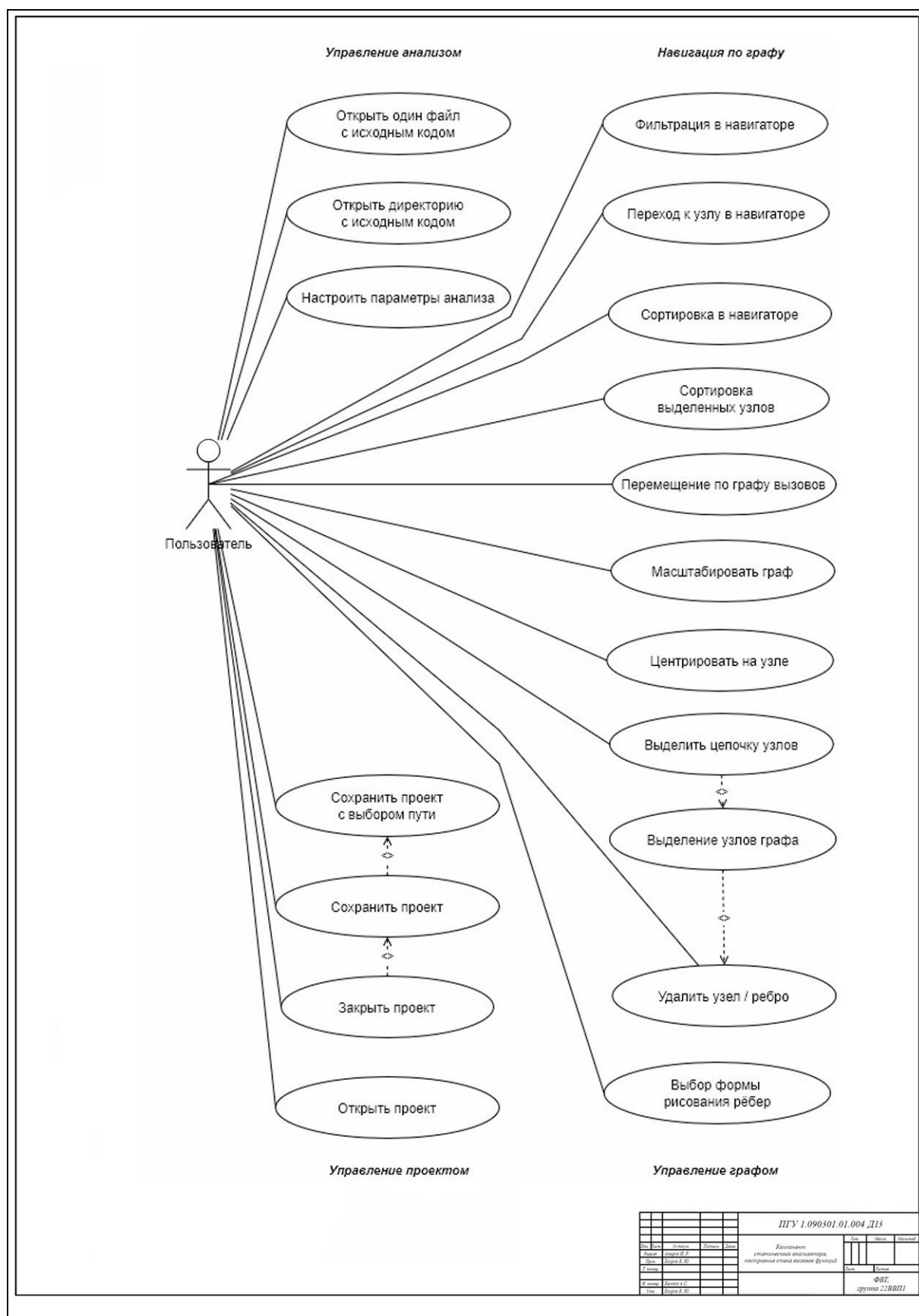
Приложение А.1 – Диаграмма классов графического интерфейса



Приложение А.2 – Диаграмма классов модуля анализа



Приложение А.3 – Диаграмма прецедентов



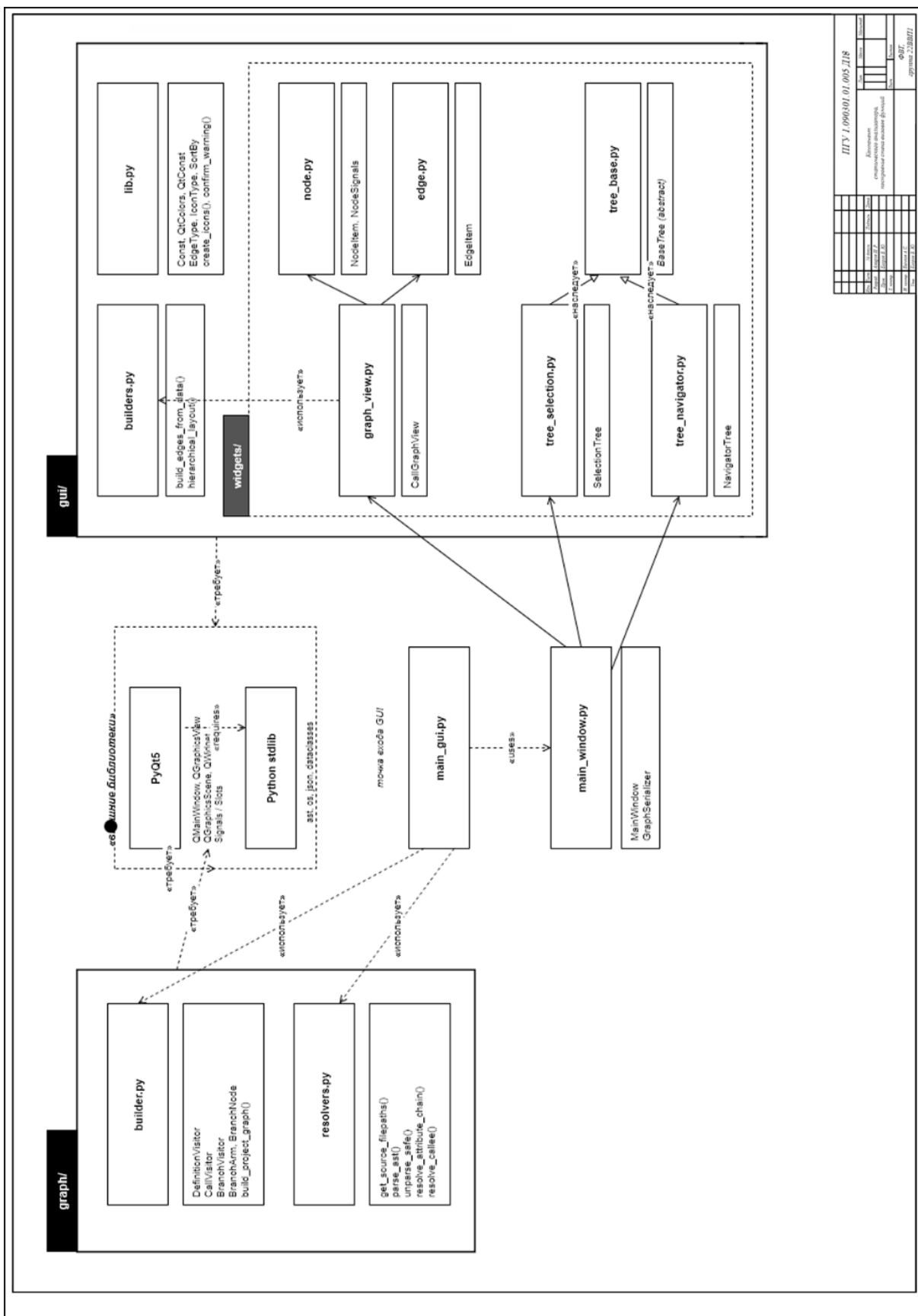
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.01.001 ПЗ

Лист

48

Приложение А.4 – Диаграмма компонентов



Приложение Б

(Обязательное)

Компонент статического анализатора, построение стека вызовов функций

Листинг программы

					ПГУ1.090301.01.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Файл graph/builder.py

```
import ast
from dataclasses import dataclass, field
import os

from graph.resolvers import parse_ast,
resolve_attribute_chain, resolve_callee, unparse_safe

@dataclass
class BranchArm:
    text: str
    kind: str # if elif else for while try except finally
    with
    calls_in_cond: list[str] = field(default_factory=list)
    calls: list[str] = field(default_factory=list)
    parent_branches: list[str] = field(default_factory=list)

@dataclass
class BranchNode:
    key: str # module.Class.func::if_1
    arms: list[BranchArm] = field(default_factory=list)

class DefinitionVisitor(ast.NodeVisitor):

    def __init__(self, name_stack: list[str]):
        self.name_stack: list[str] = name_stack
        self.definitions: tuple[str] = set()

    def get_current_name(self) -> str:
        return ".".join(self.name_stack)

    def visit_ClassDef(self, node: ast.ClassDef):
        self.name_stack.append(node.name)
        self.generic_visit(node)
        self.name_stack.pop()

    def visit_FunctionDef(self, node: ast.FunctionDef):
        self.name_stack.append(node.name)
        self.definitions.add(self.get_current_name())
        self.generic_visit(node)
        self.name_stack.pop()

    visit_AsyncFunctionDef = visit_FunctionDef

    @staticmethod
    def get_defined_functions(tree: ast.AST, name_stack:
[str]) -> (str):
        v = DefinitionVisitor(name_stack.copy())
```

```

        v.visit(tree)
        return v.definitions

class CallVisitor(ast.NodeVisitor):

    def __init__(self, caller_name: str, defined_functions:
set[str],
                    visit_branch: bool, visit_defs: bool):
        self.caller_name = caller_name
        self.defined_functions = defined_functions
        self.visit_branch = visit_branch
        self.visit_defs = visit_defs

        self.calls: list[str] = []

    def visit_Call(self, node: ast.Call):
        if isinstance(node.func, ast.Name):
            raw = node.func.id
        elif isinstance(node.func, ast.Attribute):
            raw = resolve_attribute_chain(node.func) or
f"<expr>.{node.func.attr}"
        else:
            raw = None

        if raw:
            resolved = resolve_callee(raw, self.caller_name,
self.defined_functions)
            if resolved and resolved != self.caller_name and
resolved not in self.calls:
                self.calls.append(resolved)
            self.generic_visit(node)

    def generic_visit_definition(self, node):
        if self.visit_defs:
            self.generic_visit(node)

    def generic_visit_branch(self, node):
        if self.visit_branch:
            self.generic_visit(node)

    visit_FunctionDef = generic_visit_definition
    visit_AsyncFunctionDef = generic_visit_definition
    visit_ClassDef = generic_visit_definition

    visit_If = generic_visit_branch
    visit_For = generic_visit_branch
    visit_While = generic_visit_branch
    visit_Try = generic_visit_branch
    visit_With = generic_visit_branch
    visit_TryStar = generic_visit_branch

```

```

        @staticmethod
        def collect_calls_in_expr(node: ast.expr, caller_name:
str,
                                defined_functions: set[str]) ->
[str]:
    v = CallVisitor(caller_name, defined_functions, True,
True)
    v.visit(node)
    return v.calls

        @staticmethod
        def collect_calls_in_body(stmts: list[ast.stmt],
caller_name: str,
                                defined_functions: set[str],
visit_branch: bool=True) -> [str]:
    v = CallVisitor(caller_name, defined_functions,
visit_branch, False)
    for stmt in stmts:
        v.visit(stmt)
    return v.calls

class BranchVisitor(ast.NodeVisitor):

    def __init__(self, module_name: str, definition_index:
set[str]):
        self.module_name = module_name
        self.definition_index = definition_index

        self.name_stack: list[str] = [module_name]

        # результаты
        self.branches: dict[str, BranchNode] = {}
        self.branches_in_func: dict[str, list[str]] = {}
        self.calls: dict[str, list[str]] = {} # безусловные

        self._branch_counters: dict[str, dict[str, int]] = {}
        self._parent_stack: list[str] = []

    def _current(self) -> str:
        return ".".join(self.name_stack)

    def _is_in_function(self) -> bool:
        return len(self.name_stack) > 1

    def _next_branch_key(self, kind: str) -> str:
        func = self._current()
        if func not in self._branch_counters:
            self._branch_counters[func] = {}
        cnt = self._branch_counters[func]
        cnt[kind] = cnt.get(kind, 0) + 1
        return f"{func}::{kind}_{cnt[kind]}"

```

```

def _rc(self, expr_node: ast.expr) -> list[str]:
    return CallVisitor.collect_calls_in_expr(expr_node,
self._current(), self.definition_index)

def _rb(self, stmts: list[ast.stmt]) -> list[str]:
    return CallVisitor.collect_calls_in_body(stmts,
self._current(), self.definition_index, False)

def _register_branch(self, key: str, node_obj:
BranchNode):
    func = self._current()
    self.branches[key] = node_obj
    self.branches_in_func.setdefault(func,
[]).append(key)

def _add_call(self, raw: str):
    func = self._current()
    resolved = resolve_callee(raw, func,
self.definition_index)
    if resolved is None or resolved == func:
        return
    self.calls.setdefault(func, [])
    if resolved not in self.calls[func]:
        self.calls[func].append(resolved)

def visit_ClassDef(self, node: ast.ClassDef):
    self.name_stack.append(node.name)
    self.generic_visit(node)
    self.name_stack.pop()

def visit_FunctionDef(self, node: ast.FunctionDef):
    for decorator in node.decorator_list: # декораторы
это безусловные вызовы
        if isinstance(decorator, ast.Name):
            self._add_call(decorator.id)
        elif isinstance(decorator, ast.Call):
            if isinstance(decorator.func, ast.Name):
                self._add_call(decorator.func.id)
            elif isinstance(decorator.func,
ast.Attribute):
                chain =
resolve_attribute_chain(decorator.func)
                if chain:
                    self._add_call(chain)
            elif isinstance(decorator, ast.Attribute):
                chain = resolve_attribute_chain(decorator)
                if chain:
                    self._add_call(chain)

    self.name_stack.append(node.name)

```

```

        for c in self._rb(node.body):
            self.calls.setdefault(self._current(), [])
            if c not in self.calls[self._current()]:
                self.calls[self._current()].append(c)

        for stmt in node.body:
            self.visit(stmt)

        self.name_stack.pop()

visit_AsyncFunctionDef = visit_FunctionDef

def visit_If(self, node: ast.If):
    if not self._is_in_function():
        self.generic_visit(node)
        return

    key = self._next_branch_key("if")
    branch = BranchNode(key=key)

    current_if = node
    is_first = True
    while True:
        if is_first:
            arm_kind = "if"
            is_first = False
        else:
            arm_kind = "elif"

        cond_text = unparse_safe(current_if.test)
        arm = BranchArm(
            text=f"{arm_kind} {cond_text}:",
            kind=arm_kind,
            calls_in_cond=self._rc(current_if.test),
            calls=self._rb(current_if.body),
            parent_branches=self._parent_stack.copy(),
        )
        branch.arms.append(arm)

        self._parent_stack.append(arm.text)
        for stmt in current_if.body:
            self.visit(stmt)
        self._parent_stack.pop()

        orelse = current_if.orelse
        if not orelse:
            break
        if len(orelse) == 1 and isinstance(orelse[0],
ast.If):
            current_if = orelse[0]
        else:

```

```

        else_arm = BranchArm(text="else:",
kind="else",
        calls_in_cond=[], calls=self._rb(orelse),

parent_branches=self._parent_stack.copy())
        branch.arms.append(else_arm)

        self._parent_stack.append(else_arm.text)
        for stmt in orelse:
            self.visit(stmt)
        self._parent_stack.pop()
        break

    self._register_branch(key, branch)

def visit_For(self, node: ast.For):
    if not self._is_in_function():
        self.generic_visit(node)
        return

    key = self._next_branch_key("for")
    target_text = unparse_safe(node.target)
    iter_text = unparse_safe(node.iter)

    branch = BranchNode(key=key)
    for_arm = BranchArm(
        text=f"for {target_text} in {iter_text}",
kind="for",
        calls_in_cond=self._rc(node.iter),
calls=self._rb(node.body),
        parent_branches=self._parent_stack.copy())
    branch.arms.append(for_arm)

    if node.orelse:
        else_arm = BranchArm(text="else:", kind="else",
calls_in_cond=[],
calls=self._rb(node.orelse),
        parent_branches=self._parent_stack.copy())
        branch.arms.append(else_arm)

        self._parent_stack.append(else_arm.text)
        for stmt in node.orelse:
            self.visit(stmt)
        self._parent_stack.pop()

    self._register_branch(key, branch)

    self._parent_stack.append(for_arm.text)
    for stmt in node.body:
        self.visit(stmt)
    self._parent_stack.pop()

```

```

def visit_While(self, node: ast.While):
    if not self._is_in_function():
        self.generic_visit(node)
        return

    key = self._next_branch_key("while")
    cond_text = unparse_safe(node.test)

    branch = BranchNode(key=key)
    while_arm = BranchArm(text=f"while {cond_text}:",
kind="while",
        calls_in_cond=self._rc(node.test),
calls=self._rb(node.body),
        parent_branches=self._parent_stack.copy())
    branch.arms.append(while_arm)

    if node.orelse:
        else_arm = BranchArm(text="else:", kind="else",
calls_in_cond=[],
        calls=self._rb(node.orelse),
parent_branches=self._parent_stack.copy())
        branch.arms.append(else_arm)

        self._parent_stack.append(else_arm.text)
        for stmt in node.orelse:
            self.visit(stmt)
        self._parent_stack.pop()

    self._register_branch(key, branch)

    self._parent_stack.append(while_arm.text)
    for stmt in node.body:
        self.visit(stmt)
    self._parent_stack.pop()

def visit_Try(self, node: ast.Try):
    if not self._is_in_function():
        self.generic_visit(node)
        return

    key = self._next_branch_key("try")
    branch = BranchNode(key=key)

    try_arm = BranchArm(text="try:", kind="try",
        calls_in_cond=[], calls=self._rb(node.body),
        parent_branches=self._parent_stack.copy())
    branch.arms.append(try_arm)

    self._parent_stack.append(try_arm.text)
    for stmt in node.body:
        self.visit(stmt)
    self._parent_stack.pop()

```

```

for handler in node.handlers:
    if handler.type is not None:
        exc_text = unparse_safe(handler.type)
        if handler.name:
            exc_text += f" as {handler.name}"
        header = f"except {exc_text}:"
    else:
        header = "except:"
    exc_arm = BranchArm(text=header, kind="except",
        calls_in_cond=[],
calls=self._rb(handler.body),
        parent_branches=self._parent_stack.copy())
    branch.arms.append(exc_arm)

    self._parent_stack.append(exc_arm.text)
    for stmt in handler.body:
        self.visit(stmt)
    self._parent_stack.pop()

    if node.orelse:
        else_arm = BranchArm(text="else:", kind="else",
            calls_in_cond=[],
calls=self._rb(node.orelse),
            parent_branches=self._parent_stack.copy())
        branch.arms.append(else_arm)

        self._parent_stack.append(else_arm.text)
        for stmt in node.orelse:
            self.visit(stmt)
        self._parent_stack.pop()

    if node.finalbody:
        fin_arm = BranchArm(text="finally:",
kind="finally",
            calls_in_cond=[],
calls=self._rb(node.finalbody),
            parent_branches=self._parent_stack.copy())
        branch.arms.append(fin_arm)

        self._parent_stack.append(fin_arm.text)
        for stmt in node.finalbody:
            self.visit(stmt)
        self._parent_stack.pop()

    self._register_branch(key, branch)

def visit_TryStar(self, node):
    if not self._is_in_function():
        self.generic_visit(node)
    return

```

```

        self.visit_Try(node)

def visit_With(self, node: ast.With):
    if not self._is_in_function():
        self.generic_visit(node)
        return

    key = self._next_branch_key("with")

    lst = []
    for item in node.items:
        text = unparse_safe(item.context_expr)
        if item.optional_vars:
            text += f" as
{unparse_safe(item.optional_vars)}"
        lst.append(text)
    items_text = ", ".join(lst)

    branch = BranchNode(key=key)
    calls_in_cond = [c for item in node.items for c in
self._rc(item.context_expr)]
    with_arm = BranchArm(text=f"with {items_text}:",
kind="with",
        calls_in_cond=calls_in_cond,
calls=self._rb(node.body),
        parent_branches=self._parent_stack.copy())
    branch.arms.append(with_arm)

    self._register_branch(key, branch)

    self._parent_stack.append(with_arm.text)
    for stmt in node.body:
        self.visit(stmt)
    self._parent_stack.pop()

    @staticmethod
    def build_branch_map(tree: ast.AST, module_name: str,
definition_index: set[str]
        ) -> tuple[dict[str, BranchNode], dict[str,
list[str]], dict[str, list[str]]]:
        visitor = BranchVisitor(module_name,
definition_index)
        visitor.visit(tree)
        return visitor.branches, visitor.branches_in_func,
visitor.calls

def build_project_graph(source_path: str, source_files: str)
-> dict:
    ast_trees: [(ast.AST, str)] = []
    for path in source_files:
        ast_tree = parse_ast(path)

```

```

        module_name = os.path.relpath(path, source_path)[:3]
        ast_trees.append((ast_tree, module_name))

    defined_functions = set()
    for ast_tree, module_name in ast_trees:
        definitions =
DefinitionVisitor.get_defined_functions(ast_tree,
[module_name])
        defined_functions |= definitions

    all_branches: {str: BranchNode} = {}
    all_branches_in_func: {str: [str]} = {}
    all_calls: {str: list[str]} = {}

    for ast_tree, module_name in ast_trees:
        res = BranchVisitor.build_branch_map(ast_tree,
module_name, defined_functions)
        branches, branch_in_func, calls = res

        all_branches.update(branches)
        for f, keys in branch_in_func.items():
            all_branches_in_func.setdefault(f,
[])].extend(keys)
            for f, cs in calls.items():
                lst = all_calls.setdefault(f, [])
                for c in cs:
                    if c not in lst:
                        lst.append(c)

    retval = {
        "defined_functions": defined_functions,
        "branches": all_branches,
        "branches_in_func": all_branches_in_func,
        "calls": all_calls,
    }
    return retval

```

Файл graph/resolvers.py

```

import ast
import os

def get_source_filepaths(source_dir: str, file_exts:
tuple=None) -> list[str]:
    if file_exts is None:
        file_exts = (".py",)

    source_files = []
    for root, _dirs, files in os.walk(source_dir):
        for file in files:
            if file.endswith(file_exts):
                source_files.append(os.path.join(root, file))

```

					ПГУ1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

```

return source_files

def parse_ast(filepath: str) -> ast.AST:
    with open(filepath, mode="r", encoding="utf-8") as file:
        return ast.parse(file.read(), filename=filepath)

def unparse_safe(node: ast.expr) -> str:
    try:
        return ast.unparse(node)
    except Exception:
        return "<expr>"

def resolve_attribute_chain(node: ast.expr) -> str:
    # obj.method() -> 'obj.method', os.path.join() ->
    'os.path.join'
    parts = []

    current = node
    while isinstance(current, ast.Attribute):
        parts.append(current.attr)
        current = current.value
    if isinstance(current, ast.Name):
        parts.append(current.id)
        return ".".join(reversed(parts))
    return None

def resolve_callee(raw_callee: str, caller_qualified: str,
defined_functions: set[str]) -> str:
    if raw_callee in defined_functions:
        return raw_callee

    parts = raw_callee.split(".") # self.foo; cls.foo;
    self.attr.method()
    if parts[0] in ("self", "cls") and len(parts) >= 2:
        for end in range(1, len(parts)):
            method_name = ".".join(parts[end:])
            caller_parts = caller_qualified.split(".")
            for i in range(len(caller_parts), 0, -1):
                candidate = ".".join(caller_parts[:i]) + "."
+ method_name
                if candidate in defined_functions:
                    return candidate
            suffix = "." + method_name
            matches = [q for q in defined_functions if
q.endswith(suffix)]
            if len(matches) == 1:
                return matches[0]
            if len(matches) > 1:

```

```

        caller_module =
caller_qualified.split(".")[0]
        same_module = [m for m in matches if
m.startswith(caller_module + ".")]
        if same_module:
            return same_module[0]
        return matches[0]
    raw_callee = parts[-1]

    suffix = "." + raw_callee
    matches = [q for q in defined_functions if
q.endswith(suffix)]
    if len(matches) == 1:
        return matches[0]
    if len(matches) > 1:
        caller_module = caller_qualified.split(".")[0]
        same_module = [m for m in matches if
m.startswith(caller_module + ".")]
        if len(same_module) == 1:
            return same_module[0]
        return matches[0]

    return None

```

Файл gui/widgets/edge.py

```

import math
from typing import TYPE_CHECKING

from PyQt5.QtCore import QPointF
from PyQt5.QtGui import QPainterPath, QPainterPathStroker,
QPen
from PyQt5.QtWidgets import QGraphicsItem, QGraphicsPathItem

from gui.lib import Const, EdgeType, QtColors, QtConst

if TYPE_CHECKING:
    from .node import NodeItem

class EdgeItem(QGraphicsPathItem):

    def __init__(self, src: "NodeItem", dst: "NodeItem",
                 is_cond_call: bool=False,
                 edge_type: EdgeType=None,
                 curved: bool=False):
        super().__init__()

        self.src: "NodeItem" = src
        self.dst: "NodeItem" = dst
        self.is_cond_call: bool = is_cond_call

```

```

    if is_cond_call:
        edge_type = EdgeType.CONDITON
    if not edge_type:
        edge_type = EdgeType.REGULAR
    self.edge_type: EdgeType = edge_type

    self.curved: bool = curved
    self._highlight: bool = False

    self.setZValue(QtConst.EDGE_Z)

    self.setAcceptHoverEvents(True)
    self.setFlag(QGraphicsItem.ItemIsSelectable, False)
    self.update_path()

    def set_highlight(self, enable: bool):
        self._highlight = enable
        self.setZValue(QtConst.EDGE_Z_HOVER if enable else
QtConst.EDGE_Z)
        self.update_path()

    def _pen(self) -> QPen:
        if self._highlight:
            return QPen(QtColors.EDGE_HOVER, 2.0)
        return QPen(self.edge_type.value, 1.0)

    def _bottom_exit_point(self) -> QPointF:
        bottom_edges = [e for e in self.src.edges_out if not
e.is_cond_call]
        index = bottom_edges.index(self) if self in
bottom_edges else 0
        return self.src.bottom_exit(index, len(bottom_edges))

    def _top_entry_point(self) -> QPointF:
        top_edges = [e for e in self.dst.edges_in]
        index = top_edges.index(self) if self in top_edges
else 0
        return self.dst.top_entry(index, len(top_edges))

    def update_path(self):
        if self.is_cond_call:
            p1 = (self.src.left_center()
                if self.src.center().x() -
self.dst.center().x() > 0
                else self.src.right_center())
        else:
            p1 = self._bottom_exit_point()
            p2 = self._top_entry_point()

        path = QPainterPath()
        path.moveTo(p1)

```

```

        if self.curved:
            if self.is_cond_call:
                ctrl1 = QPointF(p2.x(), p1.y())
                ctrl2 = QPointF(p2.x(), p2.y() -
Const.SPACE_SIZE * 2)
                path.cubicTo(ctrl1, ctrl2, p2)
            elif p2.y() < p1.y():
                src_w = self.src._w
                go_right = p2.x() >= p1.x()
                side_x = (self.src.right_center().x() + src_w
* 0.5
                                if go_right
                                else self.src.left_center().x() -
src_w * 0.5)
                below_src = QPointF(side_x, p1.y() +
Const.SPACE_SIZE * 2)
                above_dst = QPointF(side_x, p2.y() -
Const.SPACE_SIZE * 2)
                path.cubicTo(QPointF(p1.x(), p1.y() +
Const.SPACE_SIZE * 2),
                                QPointF(side_x, p1.y() +
Const.SPACE_SIZE * 2), below_src)
                path.cubicTo(QPointF(side_x, (below_src.y() +
above_dst.y()) / 2),
                                QPointF(side_x, (below_src.y() +
above_dst.y()) / 2), above_dst)
                path.cubicTo(QPointF(side_x, p2.y() -
Const.SPACE_SIZE * 2),
                                QPointF(p2.x(), p2.y() -
Const.SPACE_SIZE * 2), p2)
            else:
                dy = p2.y() - p1.y()
                ctrl_offset = max(dy * 0.4, 20.0)
                path.cubicTo(QPointF(p1.x(), p1.y() +
ctrl_offset),
                                QPointF(p2.x(), p2.y() -
ctrl_offset), p2)
            else:
                mid_x = (p1.x() + p2.x()) / 2
                mid_y = (p1.y() + p2.y()) / 2
                if p1.y() + Const.SPACE_SIZE >= p2.y():
                    near_x = (self.src.right_center().x() +
Const.SPACE_SIZE
                                if p1.x() < p2.x()
                                else self.src.left_center().x() -
Const.SPACE_SIZE)
                if self.is_cond_call:
                    path.lineTo(near_x, p1.y())
                    path.lineTo(near_x, p2.y() -
Const.SPACE_SIZE)
                    path.lineTo(p2.x(), p2.y() -
Const.SPACE_SIZE)

```

```

        else:
            path.lineTo(p1.x(), p1.y() +
Const.SPACE_SIZE)
            path.lineTo(near_x, p1.y() +
Const.SPACE_SIZE)
            path.lineTo(near_x, p2.y() -
Const.SPACE_SIZE)
            path.lineTo(p2.x(), p2.y() -
Const.SPACE_SIZE)
        else:
            if self.is_cond_call:
                if self.src.left_center().x() -
Const.SPACE_SIZE <= p2.x() <= self.src.right_center().x() +
Const.SPACE_SIZE:
                    if p1.x() - Const.SPACE_SIZE >
p2.x():
                        path.lineTo(p1.x() +
Const.SPACE_SIZE, p1.y())
                        path.lineTo(p1.x() +
Const.SPACE_SIZE, self.src.bottom_center().y() +
Const.SPACE_SIZE)
                    else:
                        path.lineTo(p1.x() -
Const.SPACE_SIZE, p1.y())
                        path.lineTo(p1.x() -
Const.SPACE_SIZE, self.src.bottom_center().y() +
Const.SPACE_SIZE)
                        path.lineTo(p2.x(),
self.src.bottom_center().y() + Const.SPACE_SIZE)
                    else:
                        path.lineTo(mid_x, p1.y())
                        path.lineTo(p2.x(), p1.y())
                        path.lineTo(p2.x(), mid_y)
                else:
                    path.lineTo(p1.x(), mid_y)
                    path.lineTo(p2.x(), mid_y)

            angle = math.radians(90)
            center = QPointF(p2.x(), p2.y() - Const.ARROW_SIZE *
math.sin(angle - math.pi / 4))
            left = QPointF(p2.x() - Const.ARROW_SIZE *
math.cos(angle - math.pi / 8),
p2.y() - Const.ARROW_SIZE *
math.sin(angle - math.pi / 4))
            right = QPointF(p2.x() - Const.ARROW_SIZE *
math.cos(angle + math.pi / 8),
p2.y() - Const.ARROW_SIZE *
math.sin(angle + math.pi / 4))
            path.lineTo(center)
            path.lineTo(left)
            path.lineTo(p2.x(), p2.y() - 1)
            path.lineTo(right)

```

```

        path.lineTo(center)

        self.setPen(self._pen())
        self.setPath(path)

    def shape(self) -> QPainterPath:
        stroker = QPainterPathStroker()
        stroker.setWidth(5)
        return stroker.createStroke(self.path())

    def on_hover(self, state: bool):
        self.set_highlight(state)

        self.src._hovered = state
        self.dst._hovered = state

        self.src.update()
        self.dst.update()
        self.update()

    def hoverEnterEvent(self, event):
        super().hoverEnterEvent(event)
        self.on_hover(True)

    def hoverLeaveEvent(self, event):
        super().hoverLeaveEvent(event)
        self.on_hover(False)

```

Файл gui/widgets/node.py

```

from typing import TYPE_CHECKING

from PyQt5.QtCore import QObject, QPointF, QRectF, Qt,
pyqtSignal
from PyQt5.QtGui import QBrush, QPainter, QPen
from PyQt5.QtWidgets import QGraphicsItem

from gui.lib import Const, QtColors, QtConst

if TYPE_CHECKING:
    from .edge import EdgeItem

class NodeItem(QGraphicsItem):

    def __init__(self, name: str, is_condition: bool=False):
        super().__init__()

        self.name: str = name
        self.is_condition: bool = is_condition
        self.edges_out: list["EdgeItem"] = []
        self.edges_in: list["EdgeItem"] = []

```

```

        self._hovered: bool = False
        self.signals: NodeSignals = NodeSignals()

        self.setFlags(QGraphicsItem.ItemIsMovable |
QGraphicsItem.ItemIsSelectable |
                        QGraphicsItem.ItemSendsGeometryChanges)
        self.setAcceptHoverEvents(True)
        self.setZValue(QtConst.NODE_Z)

        text_w = QtConst.NODE_FONT_M.size(0,
self.name).width() + 24
        self._w: int = max(Const.NODE_W, text_w)
        self._h: int = Const.NODE_H

        def boundingRect(self) -> QRectF:
            return QRectF(-self._w / 2 - 2, -self._h / 2 - 2,
self._w + 4, self._h + 4)

        def paint(self, painter: QPainter, option, widget=None):
            rect = QRectF(-self._w / 2, -self._h / 2, self._w,
self._h)
            bg_color = QtColors.BG_HOVER if self._hovered else
QtColors.BG
            border_color = QtColors.BORDER
            text_color = QtColors.TEXT
            pen_width = 1.0

            if self.isSelected():
                border_color, pen_width = QtColors.BORDER_SEL,
2.0
            elif self._hovered:
                border_color, pen_width = QtColors.BORDER_HOVER,
1.5
            elif self.is_condition:
                bg_color = QtColors.CONDITION_BG
                text_color = QtColors.CONDITION_TEXT

            painter.setBrush(QBrush(bg_color))
            painter.setPen(QPen(border_color, pen_width))
            painter.drawRect(rect)

            painter.setPen(QPen(text_color))
            painter.setFont(QtConst.NODE_FONT)
            painter.drawText(rect, Qt.AlignCenter, self.name)

        def itemChange(self, change, value):
            if change == QGraphicsItem.ItemPositionHasChanged:
                for e in self.edges_out + self.edges_in:
                    e.update_path()
                self.signals.position_changed.emit()
            return super().itemChange(change, value)

```

```

def mousePressEvent(self, event):
    super().mousePressEvent(event)
    if event.button() == Qt.LeftButton:
        self.signals.clicked.emit(self)

def hoverEnterEvent(self, event):
    self.on_hover(True)

def hoverLeaveEvent(self, event):
    self.on_hover(False)

def on_hover(self, active: bool):
    self._hovered = active
    for e in self.edges_out + self.edges_in:
        e.set_highlight(active)
    self.update()

def center(self) -> QPointF:
    return self.mapToScene(QPointF(0, 0))

def bottom_center(self) -> QPointF:
    return self.mapToScene(QPointF(0, self._h / 2))

def top_center(self) -> QPointF:
    return self.mapToScene(QPointF(0, -self._h / 2))

def right_center(self) -> QPointF:
    return self.mapToScene(QPointF(self._w / 2, 0))

def left_center(self) -> QPointF:
    return self.mapToScene(QPointF(-self._w / 2, 0))

def centered_edge_point(self, index: int, total: int,
bottom: bool=False) -> QPointF:
    if total <= 1:
        return self.bottom_center() if bottom else
self.top_center()

    total_width = (total - 1) * Const.ARROW_SPACING
    if total_width > self._w:
        self._w = total_width + 6

    offset = -total_width / 2 + index *
Const.ARROW_SPACING
    height = self._h if bottom else -self._h
    return self.mapToScene(QPointF(offset, height / 2))

def top_entry(self, index: int, total: int) -> QPointF:
    return self.centered_edge_point(index, total, False)

def bottom_exit(self, index: int, total: int) -> QPointF:

```

```
return self.centered_edge_point(index, total, True)
```

```
class NodeSignals(QObject):
    position_changed: pyqtSignal = pyqtSignal()
    clicked: pyqtSignal = pyqtSignal(NodeItem)
```

Файл gui/widgets/graph_view.py

```
from PyQt5.QtCore import QPointF, Qt, QTimer, pyqtSignal
from PyQt5.QtGui import QContextMenuEvent, QKeyEvent,
QMouseEvent, QPainter, QWheelEvent
from PyQt5.QtWidgets import QFrame, QGraphicsItem,
QGraphicsScene, QGraphicsView, QMenu, QMessageBox
```

```
from gui.builders import build_edges_from_data,
hierarchical_layout
from gui.lib import EdgeType, confirm_warning
from gui.widgets.edge import EdgeItem
from gui.widgets.node import NodeItem
```

```
class CallGraphView(QGraphicsView):

    graph_changed: pyqtSignal = pyqtSignal()
    node_moved: pyqtSignal = pyqtSignal()
    node_clicked: pyqtSignal = pyqtSignal(NodeItem)

    show_node_in_tree: pyqtSignal = pyqtSignal(NodeItem)
    show_edge_in_tree: pyqtSignal = pyqtSignal(EdgeItem)

    def __init__(self, graph_data: dict, parent=None):
        super().__init__(parent)
        self.graph_data: dict = graph_data

        self._show_internal_only: bool = True
        self._show_branches: bool = True
        self._merge_branch_calls: bool = True
        self._curved_edges: bool = False

        self._scene: QGraphicsScene = QGraphicsScene(self)
        self.setScene(self._scene)

        self._flash_node: NodeItem = None
        self._flash_timer: QTimer = None

        self._anim_timer: QTimer = None
        self._anim_keep_selection: bool = False

        self._rb_moved: bool = False
        self._rb_press_pos: bool = None
```

```

        self.setRenderHint(QPainter.Antialiasing)

self.setTransformationAnchor(QGraphicsView.AnchorUnderMouse)
        self.setResizeAnchor(QGraphicsView.AnchorUnderMouse)
        self.setFrameShape(QFrame.NoFrame)

        self._build_graph()

def set_show_int_only(self, v: bool):
    self._show_internal_only = v
    self._rebuild()

def set_show_branches(self, v: bool):
    self._show_branches = v
    self._rebuild()

def set_merge_branch_calls(self, v: bool):
    self._merge_branch_calls = v
    self._rebuild()

def set_curved_edges(self, v: bool):
    self._curved_edges = v
    for item in self._scene.items():
        if isinstance(item, EdgeItem):
            item.curved = v
            item.update_path()

def _rebuild(self):
    self._scene.clear()
    self._build_graph()
    self.graph_changed.emit()

def _build_graph(self):
    if not self.graph_data:
        return

    all_node_names, edges, condition_node_names =
build_edges_from_data(
        self.graph_data, self._show_branches,
        self._merge_branch_calls,
self._show_internal_only)

    connected = {n for s, d, _, _ in edges for n in (s,
d)}

    isolated = sorted(all_node_names - connected)
    node_list = sorted(connected | set(isolated))
    layout_edges = [(s, d) for s, d, _, _ in edges]
    positions = hierarchical_layout(node_list,
layout_edges, x_gap=60, y_gap=110)

    node_items: dict[str, NodeItem] = {}
    for name in node_list:

```

```

        item = NodeItem(name, is_condition=(name in
condition_node_names))
        item.setPos(*positions.get(name, (0, 0)))

item.signals.position_changed.connect(self.node_moved)
item.signals.clicked.connect(self.node_clicked)

self._scene.addItem(item)
node_items[name] = item

for src_name, dst_name, is_cond_call, edge_type in
edges:
    if src_name == dst_name:
        continue

    src_item = node_items[src_name]
    dst_item = node_items[dst_name]
    if edge_type is True:
        node_type = EdgeType.TRUE
    elif edge_type is False:
        node_type = EdgeType.FALSE
    elif edge_type is None:
        node_type = EdgeType.REGULAR
    else:
        node_type = None
    edge = EdgeItem(src_item, dst_item,
edge_type=node_type,
                    is_cond_call=is_cond_call,
curved=self._curved_edges)
    self._scene.addItem(edge)
    src_item.edges_out.append(edge)
    dst_item.edges_in.append(edge)

self.fit()

def selected_items(self) -> [QGraphicsItem]:
    return self._scene.selectedItems()

def find_nodes(self, name: str, children: bool=False) ->
[NodeItem]:
    nodes = []
    for item in self._scene.items():
        if not isinstance(item, NodeItem):
            continue
        if item.name == name or (children and
item.name.startswith(name + ".")):
            nodes.append(item)
    return nodes

def edge_counts(self) -> dict[str, list]:
    counts: dict[str, list] = {}

```

```

        for si in self._scene.items():
            if isinstance(si, NodeItem):
                counts[si.name] = [len(si.edges_in),
len(si.edges_out)]
        return counts

    def remove_edge(self, edge: EdgeItem):
        if self.graph_data:
            for key, callees in
list(self.graph_data["calls"].items()):
                if key == edge.src.name:
                    callees = [c for c in callees if c !=
edge.dst.name]
                    self.graph_data["calls"][key] = callees

        edge.src.edges_out.remove(edge)
        edge.dst.edges_in.remove(edge)
        self._scene.removeItem(edge)

        self.graph_changed.emit()

    def remove_node(self, node: NodeItem):
        name = node.name

        if self.graph_data:
            gd = self.graph_data

            gd["calls"].pop(name, None)
            for key in list(gd["calls"].keys()):
                callees = [c for c in gd["calls"][key] if c !=
name]
                gd["calls"][key] = callees

            removed_branch_keys: set = set()
            for key in list(gd["branches_in_func"].keys()):
                if key == name:
                    for bk in gd["branches_in_func"][key]:
                        removed_branch_keys.add(bk)
                    del gd["branches_in_func"][key]
            for bk in removed_branch_keys:
                gd["branches"].pop(bk, None)

            for edge in list(node.edges_out):
                self.remove_edge(edge)
            for edge in list(node.edges_in):
                self.remove_edge(edge)
            self._scene.removeItem(node)

            self.graph_changed.emit()

    def delete_subtree(self, name: str):
        if not self.graph_data:

```

```

        return

    if "\n" in name:
        parent_name, arm_name = name.split("\n")
    else:
        parent_name, arm_name = name, None

    is_match = lambda i: (i == parent_name or
i.startswith(parent_name + "."))

    gd = self.graph_data
    for key in list(gd["calls"].keys()):
        if is_match(key):
            del gd["calls"][key]
        else:
            gd["calls"][key] = [c for c in
gd["calls"][key] if not is_match(c)]

    for key in list(gd["branches"].keys()):
        if is_match(key):
            arms = gd["branches"][key].arms
            for arm in arms.copy():
                if arm.text == arm_name:
                    arms.remove(arm)
            if not arms:
                func_name = key.split("::")[0]
                if parent_name in
gd["branches_in_func"][func_name]:
gd["branches_in_func"][func_name].remove(parent_name)

    for node in self.find_nodes(name, True):
        self.remove_node(node)

    def delete_nodes_confirmed(self, nodes: list[NodeItem]):
        names = "\n".join(["'" + node.name.replace("\n",
"::") + "'" for node in nodes])
        reply = confirm_warning(self, "Delete nodes",
f"Delete following nodes\n\n{names}? \n\nThis
action cannot be undone.")
        if reply == QMessageBox.Ok:
            for node in nodes:
                self.remove_node(node)

    def delete_edge_confirmed(self, edge: EdgeItem):
        src, dst = edge.src.name.replace("\n", "::"),
edge.dst.name.replace("\n", "::")
        reply = confirm_warning(self, "Delete edge",
f"Delete edge
from:\n'{src}'\nto:\n'{dst}'? \n\nThis action cannot be
undone.")
        if reply == QMessageBox.Ok:

```

```

        self.remove_edge(edge)

    def flash_node(self, node: NodeItem, flashes: int=4,
                    interval_ms: int=150, keep_selection:
bool=False):
        if self._flash_timer and
self._flash_timer.isActive():
            self._flash_timer.stop()
            if self._flash_node:
                self._flash_node._hovered = False
                self._flash_node.update()
            if not keep_selection:
                self._scene.clearSelection()

        self._flash_node = node
        self._flash_count = flashes * 2
        self._flash_state = False
        self._flash_timer = QTimer(self)
        self._flash_timer.timeout.connect(self._flash_step)
        self._flash_timer.start(interval_ms)

    def _flash_step(self):
        if self._flash_count <= 0:
            self._flash_timer.stop()
            self._flash_node._hovered = False
            self._flash_node.update()
            return

        self._flash_state = not self._flash_state
        self._flash_node._hovered = self._flash_state
        self._flash_node.update()
        self._flash_count -= 1

    def smooth_center_on(self, item: NodeItem,
keep_selection: bool=True):
        if self._anim_timer and self._anim_timer.isActive():
            self._anim_timer.stop()

        target_vp = self.mapFromScene(item.center())
        vp_center = QPointF(self.viewport().width() / 2,
self.viewport().height() / 2)
        delta = target_vp - vp_center
        self._anim_steps = 25
        self._anim_delta = delta / self._anim_steps
        self._anim_item = item
        self._anim_keep_selection = keep_selection
        self._anim_timer = QTimer(self)
        self._anim_timer.timeout.connect(self._anim_step)
        self._anim_timer.start(16)

    def _anim_step(self):
        if self._anim_steps <= 0:

```

```

        self._anim_timer.stop()
        self.flash_node(self._anim_item,
keep_selection=self._anim_keep_selection)
        return

        hb, vb = self.horizontalScrollBar(),
self.verticalScrollBar()
        hb.setValue(int(hb.value() + self._anim_delta.x()))
        vb.setValue(int(vb.value() + self._anim_delta.y()))
        self._anim_steps -= 1

    def zoom(self, zoom_in: bool):
        self.scale(1.15 if zoom_in else 1 / 1.15, 1.15 if
zoom_in else 1 / 1.15)

    def fit(self):
        self.resizeScene()
        self.fitInView(self._scene.sceneRect(),
Qt.KeepAspectRatio)

    def resizeScene(self):
        e = 100

self._scene.setSceneRect(self._scene.itemsBoundingRect().adju
sted(-e, -e, e, e))

    def wheelEvent(self, event: QWheelEvent):
        self.zoom(event.angleDelta().y() > 0)

    def mousePressEvent(self, event: QMouseEvent):
        if event.button() == Qt.LeftButton:
            self.setDragMode(QGraphicsView.ScrollHandDrag
                if self.itemAt(event.pos()) is
None
                else QGraphicsView.NoDrag)
        elif event.button() == Qt.RightButton:
            self.setDragMode(QGraphicsView.RubberBandDrag)
            self._rb_press_pos = event.pos()
            self._rb_moved = False
            super().mousePressEvent(event)

    def mouseMoveEvent(self, event: QMouseEvent):
        if (event.buttons() & Qt.RightButton) and
self._rb_press_pos:
            if (event.pos() -
self._rb_press_pos).manhattanLength() > 4:
                self._rb_moved = True
            super().mouseMoveEvent(event)

    def mouseReleaseEvent(self, event: QMouseEvent):
        self.setDragMode(QGraphicsView.NoDrag)
        super().mouseReleaseEvent(event)

```

```

        self.resizeScene()

    def mouseDoubleClickEvent(self, event: QMouseEvent):
        if event.button() == Qt.LeftButton:
            scene_pos = self.mapToScene(event.pos())
            item = self._scene.itemAt(scene_pos,
self.transform())
            if isinstance(item, EdgeItem):
                vp_center = QPointF(self.viewport().width() /
2, self.viewport().height() / 2)

                def dist(p):
                    d = p - vp_center
                    return d.x() ** 2 + d.y() ** 2

                src_vp = self.mapFromScene(item.src.center())
                dst_vp = self.mapFromScene(item.dst.center())
                self.smooth_center_on(item.src if
dist(src_vp) >= dist(dst_vp) else item.dst)
                return
            super().mouseDoubleClickEvent(event)

    def keyPressEvent(self, event: QKeyEvent):
        key = event.key()
        if key in (Qt.Key_Equal, Qt.Key_Plus):
            self.zoom(True)
        elif key == Qt.Key_Minus:
            self.zoom(False)

        hb, vb = self.horizontalScrollBar(),
self.verticalScrollBar()
        if key == Qt.Key_Left:
            hb.setValue(hb.value() - hb.singleStep())
        elif key == Qt.Key_Right:
            hb.setValue(hb.value() + hb.singleStep())
        elif key == Qt.Key_Up:
            vb.setValue(vb.value() - vb.singleStep())
        elif key == Qt.Key_Down:
            vb.setValue(vb.value() + vb.singleStep())
        elif key == Qt.Key_Delete:
            selected = self.selected_items()
            if selected:
                self.delete_nodes_confirmed(selected)

    def resizeEvent(self, event):
        super().resizeEvent(event)
        self.resizeScene()

    def _select_chain(self, node: NodeItem, parents: bool,
children: bool):
        visited: set[NodeItem] = set()

```

```

def _walk_parents(n):
    for e in n.edges_in:
        if e.src not in visited:
            visited.add(e.src)
            _walk_parents(e.src)

def _walk_children(n):
    for e in n.edges_out:
        if e.dst not in visited:
            visited.add(e.dst)
            _walk_children(e.dst)

if parents:
    _walk_parents(node)
if children:
    _walk_children(node)
for item in visited:
    item.setSelected(True)

def contextMenuEvent(self, event: QContextMenuEvent):
    scene_pos = self.mapToScene(event.pos())
    item = self._scene.itemAt(scene_pos,
self.transform())

    if isinstance(item, NodeItem):
        menu = QMenu(self)
        show_nav = menu.addAction("Show in navigator")
        menu.addSeparator()
        sel_all = menu.addAction("Select all chain")
        sel_parents = menu.addAction("Select parent nodes
chain")
        sel_children = menu.addAction("Select child nodes
chain")
        menu.addSeparator()
        delete = menu.addAction("Delete...")

        action = menu.exec_(event.globalPos())
        if action == show_nav:
            self.show_node_in_tree.emit(item)
        elif action == sel_all:
            item.setSelected(True)
            self._select_chain(item, True, True)
        elif action == sel_parents:
            item.setSelected(True)
            self._select_chain(item, True, False)
        elif action == sel_children:
            item.setSelected(True)
            self._select_chain(item, False, True)
        elif action == delete:
            selected = self.selected_items()
            if not selected:
                selected = [item]

```

```

        self.delete_nodes_confirmed(selected)
    elif isinstance(item, EdgeItem):
        menu = QMenu(self)
        show_nav = menu.addAction("Show in navigator")
        menu.addSeparator()
        jump_src = menu.addAction("Go to source node")
        jump_dst = menu.addAction("Go to destination
node")

        menu.addSeparator()
        delete = menu.addAction("Delete...")
        action = menu.exec_(event.globalPos())

        if action == show_nav:
            self.show_edge_in_tree.emit(item)
        elif action == jump_src:
            self.smooth_center_on(item.src)
        elif action == jump_dst:
            self.smooth_center_on(item.dst)
        elif action == delete:
            self.delete_edge_confirmed(item)
    else:
        if self._rb_moved:
            return
        menu = QMenu(self)
        sel_all = menu.addAction("Select all nodes")
        if menu.exec_(event.globalPos()) == sel_all:
            for si in self._scene.items():
                if isinstance(si, NodeItem):
                    si.setSelected(True)

```

Файл gui/widgets/tree_base.py

```

from PyQt5.QtCore import QPoint, Qt, pyqtSignal
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QAction, QMenu, QTreeWidget,
QTreeWidgetItem, QWidget

from gui.lib import IconType, QtConst
from gui.widgets.graph_view import CallGraphView

class BaseTree(QWidget):

    center_on_node_requested = pyqtSignal(str)

    def __init__(self, view: CallGraphView, parent=None):
        super().__init__(parent)
        self._view = view

        self._tree = QTreeWidget()
        self._tree.setFont(QtConst.NODE_FONT)
        self._tree.setContextMenuPolicy(Qt.CustomContextMenu)

```

```

self._tree.customContextMenuRequested.connect(self._context_m
enu)

        self._tree.itemClicked.connect(self._on_item_clicked)

    def rebuild(self):
        raise NotImplementedError

    def sort_key(self) -> str:
        return self.sort_group.checkedAction().data()

    @staticmethod
    def _set_expanded(item: QTreeWidgetItem, state: bool):
        item.setExpanded(state)
        for i in range(item.childCount()):
            BaseTree._set_expanded(item.child(i), state)

    @staticmethod
    def _node_icon(depth: int, has_children: bool, full_name:
str, all_funcs) -> QIcon:
        if depth == 0:
            return QtConst.ICONS[IconType.MODULE]
        if has_children and full_name not in all_funcs:
            return QtConst.ICONS[IconType.CLASS]
        return QtConst.ICONS[IconType.FUNCTION]

    def _on_item_clicked(self, item: QTreeWidgetItem,
_column: int):
        name = item.data(0, Qt.UserRole)
        if name:
            self.center_on_node_requested.emit(name)

    def _build_context_menu_actions(self, menu: QMenu, item:
QTreeWidgetItem) -> dict:
        actions: dict[str, QAction | None] = {
            "show_in_graph": None,
            "select_children": None,
            "expand_children": None,
            "collapse_children": None,
            "expand_all": None,
            "collapse_all": None,
            "delete_with_children": None,
        }
        if item and item.childCount() > 0:
            actions["expand_children"] =
menu.addAction("Expand all children")
            actions["collapse_children"] =
menu.addAction("Collapse all children")
            menu.addSeparator()

            actions["expand_all"] = menu.addAction("Expand all")

```

```

        actions["collapse_all"] = menu.addAction("Collapse
all")

        if item:
            menu.addSeparator()
            actions["delete_with_children"] =
menu.addAction("Delete with children...")

        return actions

    def _handle_common_action(self, action: QAction, actions:
dict, item: QTreeWidgetItem) -> bool:
        if action is None:
            return False

        if action == actions.get("expand_children") and item:
            self._set_expanded(item, True)
        elif action == actions.get("collapse_children") and
item:
            self._set_expanded(item, False)
        elif action == actions.get("expand_all"):
            self._tree.expandAll()
        elif action == actions.get("collapse_all"):
            self._tree.collapseAll()
        elif action == actions.get("delete_with_children")
and item:
            self._delete_selected()
        else:
            return False
        return True

    def _context_menu(self, pos: QPoint):
        item = self._tree.itemAt(pos)
        menu = QMenu(self._tree)
        actions = self._build_context_menu_actions(menu,
item)
        action =
menu.exec_(self._tree.viewport().mapToGlobal(pos))
        self._handle_common_action(action, actions, item)

```

Файл gui/builders.py

```

from gui.lib import _node_pixel_width, _center

def build_edges_from_data(graph_data: dict,
                           show_branches: bool,
merge_branch_calls: bool, internal_only: bool
                           ) -> tuple[set[str],
list[tuple[str, str, bool]]]:
    all_branches = graph_data["branches"]
    all_branches_in_func = graph_data["branches_in_func"]

```

```

all_unconditional = graph_data["calls"]

internal_funcs: set[str] = set(all_unconditional.keys())
| set(all_branches_in_func.keys())

unconditional: dict[str, set[str]] = {f: set(cs) for f,
cs in all_unconditional.items()}

if show_branches and merge_branch_calls:
    for func, branch_keys in
all_branches_in_func.items():
        unc_set = unconditional.setdefault(func, set())
        for bk in branch_keys:
            bn = all_branches[bk]

            arms_with_calls = [arm for arm in bn.arms if
arm.calls]

            all_arms_have = len(arms_with_calls) ==
len(bn.arms) and len(bn.arms) > 1
            if all_arms_have:

                common = set(arms_with_calls[0].calls)
                for arm in arms_with_calls[1:]:
                    common &= set(arm.calls)
                unc_set |= common

cond_expr_edges: list[tuple[str, str]] = [] # ВЫЗОВЫ В
условиях
if show_branches:
    for func, branch_keys in
all_branches_in_func.items():
        for bk in branch_keys:
            bn = all_branches[bk]
            for arm in bn.arms:
                for callee in arm.calls_in_cond:
                    cond_expr_edges.append((func,
callee))

all_nodes: set[str] = set()
edges: list[tuple[str, str, bool]] = []
condition_node_names: set[str] = set()

for func, callees in unconditional.items(): # для
безусловных вызовов
    all_nodes.add(func)
    for callee in callees:
        all_nodes.add(callee)
        edges.append((func, callee, False, None))

if show_branches:
    for func, branch_keys in
all_branches_in_func.items():

```

```

unc_set = unconditional.get(func, set())
all_nodes.add(func)

for bk in branch_keys:
    bn = all_branches[bk]

    for arm in bn.arms:
        if merge_branch_calls:
            shown_unc = unc_set
        else:
            shown_unc = set()

        remaining = [c for c in arm.calls if c
not in shown_unc]
        has_cond_calls = bool(arm.calls_in_cond)

        if not remaining and not has_cond_calls:
            continue

        cond_label = f"{bk}\n{arm.text}"
        condition_node_names.add(cond_label)
        all_nodes.add(cond_label)

        edge_type = True if arm.parent_branches
else None
        edges.append((func, cond_label, False,
edge_type))

        if arm.kind in ("if", "elif", "except",
"while"):
            edge_type = True
            elif arm.kind in ("else"):
                edge_type = False
            else:
                edge_type = None

            for callee in remaining:
                all_nodes.add(callee)
                edges.append((cond_label, callee,
False, edge_type))

            for callee in arm.calls_in_cond:
                all_nodes.add(callee)
                edges.append((cond_label, callee,
True, True))

        if internal_only:
            keep = internal_funcs | condition_node_names
            edges = [(s, d, ic, t) for s, d, ic, t in edges
                    if s in keep and d in keep]
            all_nodes = {n for s, d, _, _ in edges for n in (s,
d)}

```

```

seen: set[tuple[str, str, bool]] = set()
unique: list[tuple[str, str, bool]] = []
for e in edges:
    if e not in seen:
        seen.add(e)
        unique.append(e)

return all_nodes, unique, condition_node_names

def hierarchical_layout(nodes: list[str], edges:
list[tuple[str, str]],
                        x_gap: int=200, y_gap: int=90) ->
dict[str, tuple[float, float]]:
    node_set = set(nodes)
    children: dict[str, list[str]] = {n: [] for n in
node_set}
    parents: dict[str, list[str]] = {n: [] for n in node_set}

    for src, dst in edges:
        if src in node_set and dst in node_set and src !=
dst:
            if dst not in children[src]:
                children[src].append(dst)
                parents[dst].append(src)

    connected_nodes = [n for n in nodes if children[n] or
parents[n]]
    isolated_nodes = [n for n in nodes if not children[n] and
not parents[n]]

    if not connected_nodes:
        positions: dict[str, tuple[float, float]] = {}
        x = 0.0
        for n in nodes:
            positions[n] = (x, 0.0)
            x += _node_pixel_width(n) + x_gap
        return _center(positions)

    levels: dict[str, int] = {}
    roots = [n for n in connected_nodes if not parents[n]]
    if not roots:
        roots = [connected_nodes[0]]

    for r in roots:
        levels[r] = 0
    queue = list(roots)
    visited: set[str] = set()
    while queue:
        cur = queue.pop(0)
        if cur in visited:

```

```

        continue
    visited.add(cur)
    for ch in children[cur]:
        lv = levels[cur] + 1
        if ch not in levels or levels[ch] < lv:
            levels[ch] = lv
        if ch not in visited:
            queue.append(ch)

    for n in connected_nodes:
        if n not in levels:
            levels[n] = 0

    max_level = max(levels.values()) if levels else 0
    level_nodes: dict[int, list[str]] = {}
    for n in connected_nodes:
        level_nodes.setdefault(levels[n], []).append(n)

    def _barycenter_sort(layer: list[str], reference_level:
int,
                        use_parents: bool) -> list[str]:
        ref_positions = {n: i for i, n in
enumerate(level_nodes.get(reference_level, []))}

        def score(n: str) -> float:
            neighbours = parents[n] if use_parents else
children[n]
            refs = [ref_positions[nb] for nb in neighbours if
nb in ref_positions]
            return sum(refs) / len(refs) if refs else
float("inf")

        return sorted(layer, key=score)

    for _ in range(3):
        for lv in range(1, max_level + 1):
            if lv in level_nodes:
                level_nodes[lv] =
_barycenter_sort(level_nodes[lv], lv - 1, use_parents=True)
            for lv in range(max_level - 1, -1, -1):
                if lv in level_nodes:
                    level_nodes[lv] =
_barycenter_sort(level_nodes[lv], lv + 1, use_parents=False)

    PADDING = x_gap
    positions: dict[str, tuple[float, float]] = {}

    for lv, layer in level_nodes.items():
        x = 0.0
        for n in layer:
            positions[n] = (x, lv * y_gap)
            x += _node_pixel_width(n) + PADDING

```

```

for lv, layer in level_nodes.items():
    xs = [positions[n][0] for n in layer]
    cx = (min(xs) + max(xs)) / 2
    for n in layer:
        x, y = positions[n]
        positions[n] = (x - cx, y)

for _ in range(2):
    for lv in range(max_level + 1):
        layer = level_nodes.get(lv, [])
        if not layer:
            continue
        desired: dict[str, float] = {}
        for n in layer:
            nb_xs = (
                [positions[nb][0] for nb in parents[n]
                if nb in positions] +
                [positions[nb][0] for nb in children[n]
                if nb in positions]
            )
            desired[n] = sum(nb_xs) / len(nb_xs) if nb_xs
        else positions[n][0]

        layer_sorted = sorted(layer, key=lambda n:
positions[n][0])
        for n in layer_sorted:
            old_x = positions[n][0]
            positions[n] = (old_x + 0.4 * (desired[n] -
old_x), positions[n][1])

        for i in range(1, len(layer_sorted)):
            prev = layer_sorted[i - 1]
            cur = layer_sorted[i]
            min_x = (positions[prev][0]
                    +_node_pixel_width(prev) / 2
                    +PADDING
                    +_node_pixel_width(cur) / 2)
            if positions[cur][0] < min_x:
                positions[cur] = (min_x,
positions[cur][1])

        xs = [positions[n][0] for n in layer_sorted]
        cx = (min(xs) + max(xs)) / 2
        for n in layer_sorted:
            x, y = positions[n]
            positions[n] = (x - cx, y)

if isolated_nodes:
    iso_y = (max_level + 1) * y_gap
    x = 0.0
    for n in isolated_nodes:

```

```

        positions[n] = (x, iso_y)
        x += _node_pixel_width(n) + PADDING
iso_xs = [positions[n][0] for n in isolated_nodes]
cx = (min(iso_xs) + max(iso_xs)) / 2
for n in isolated_nodes:
    ox, oy = positions[n]
    positions[n] = (ox - cx, oy)

return _center(positions)

```

					ПГУ1.090301.01.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		86

Приложение В

(Обязательное)

Компонент статического анализатора, построение стека вызовов функций

Презентация

					ПГУ1.090301.01.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

КОМПОНЕНТ СТАТИЧЕСКОГО АНАЛИЗАТОРА, ПОСТРОЕНИЕ СТЕКА ВЫЗОВОВ ФУНКЦИЙ

Автор работы:

студент гр. 22ВВП1 Амиров Имран Рафаилович

Руководитель:

к. т. н., доцент Егоров Валерий Юрьевич



Цель работы

- Разработать компонент статического анализатора, выполняющую обработку исходного кода Python-программ, с последующим построением стека вызовов функций.

Функционал приложения включает в себя:

- выявление пакетов, функций, классов, условных операторов в исходном коде программ;
- построение графа вызовов с учетом ветвлений из полученных данных;
- визуализация построенного графа, с возможностью навигации по нему;
- сохранение и загрузка результатов работы приложения.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.01.001 ПЗ

Лист

88

Актуальность

- Растущий спрос на инструменты для безопасной разработки ПО
- Небольшое количество аналогов
- Стандарт ГОСТ Р 71207–2024 по разработке и внедрению средств статического анализа кода

Косвенные аналоги

Pyre

Достоинства:

- Максимальная поддержка Python
- Экспорт результатов в удобных форматах

Недостатки:

- Высокая сложность, написано на OCaml
- Компромисс точности и производительности
- Получаемый граф вызовов не предназначен для использования во внешних инструментах

Infer

Достоинства:

- Масштабируемый анализ больших проектов
- Поддержка инкрементального анализа

Недостатки:

- Поддержка Python находится в разработке
- Ограничение анализа псевдонимов
- Высокая сложность, написано на OCaml

Косвенные аналоги

Pyre

```

1 google.protobuf.message.Message.__init__ has 360 overrides, this might slow down the analysis c
considerably.
2 object.__init__ has 645 overrides, this might slow down the analysis considerably.
3 type.__call__ has 120 overrides, this might slow down the analysis considerably.
4 type.__init__ has 386 overrides, this might slow down the analysis considerably.
5 type.__new__ has 69 overrides, this might slow down the analysis considerably.
6
7 {
8   "line": 12,
9   "column": 18,
10  "stop_line": 12,
11  "stop_column": 35,
12  "path": "views.py",
13  "code": "5001",
14  "name": "Possible RCE:",
15  "description":
16    "Possible RCE: [5001]: User specified data may reach a code execution sink",
17  "long description":
18    "Possible RCE: [5001]: User specified data may reach a code execution sink",
19  "concise description":
20    "Possible RCE: [5001]: User specified data may reach a code execution sink",
21  "inference": null,
22  "define": "views.operate_on_twos"
23 }

```

Infer

```

1 (Program.cs:11) error: DotNet Resource Leak
2
3 // Intraprocedural resource usage example, leaks expected.
4
5 // </summary>
6 public void ResourceLeakIntraproceduralBad() {
7     StreamWriter sw = new StreamWriter("everwhat.txt");
8     sw.WriteLine("Goruss - ASP.NET");
9 }
10
11 (Program.cs:11) error: DotNet Resource Leak
12 (Leaked) [obj, Instance Program, Dispose] = 1
13 (ResourceLeakIntraproceduralBad) [ResourceLeakIntraproceduralBad] = 1
14
15 public class MainClass {
16     public static void Main(string[] args)
17     {
18         Program p = new Program("whatever.txt");
19         // FIXME: should close the global streams by calling p.Cleanup()
20     }
21 }
22
23 (Program.cs:11) error: Null Dereference
24 (ResourceLeakIntraproceduralBad) [ResourceLeakIntraproceduralBad] = 1
25 // FIXME: should close the global streams by calling p.Cleanup()
26 // Null dereference error report expected.
27 p.NullDereferenceBad().GetHashCode();
28
29 // No null dereference error report expected.
30 p.NullDereferenceOK().GetHashCode();
31
32 Found 3 Issues
33 Issue Type(ISSUED_TYPE_ID): #
34 DotNet Resource Leak(DOTNET_RESOURCE_LEAK): 2
35 Null Dereference(NULL_DEREFERENCE): 1

```

Диаграмма вариантов использования



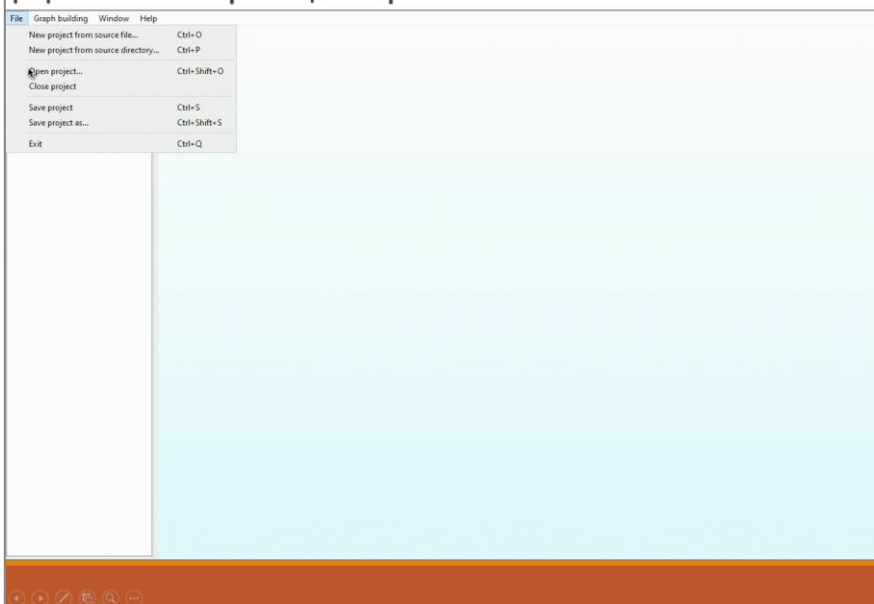
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.01.001 ПЗ

Лист

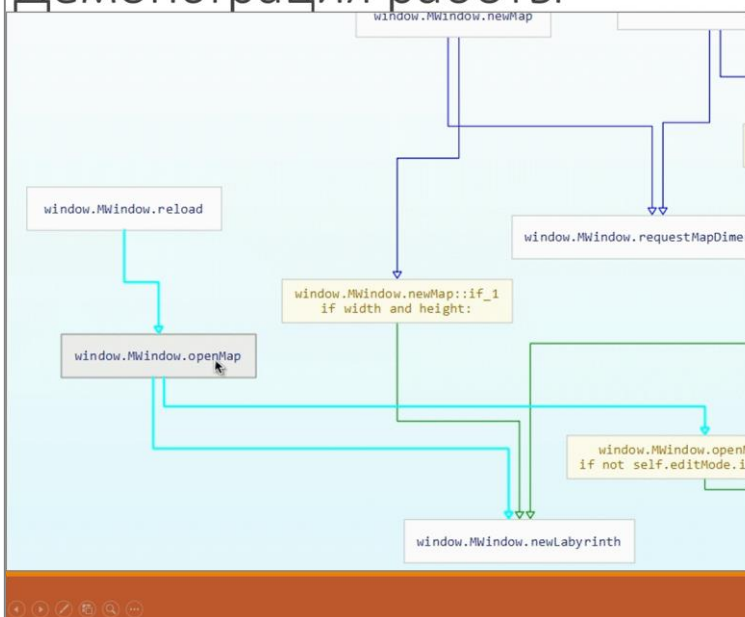
90

Демонстрация работы



Загрузка проекта и навигация по графу

Демонстрация работы



Подсветка ребер и узлов при наведении на них

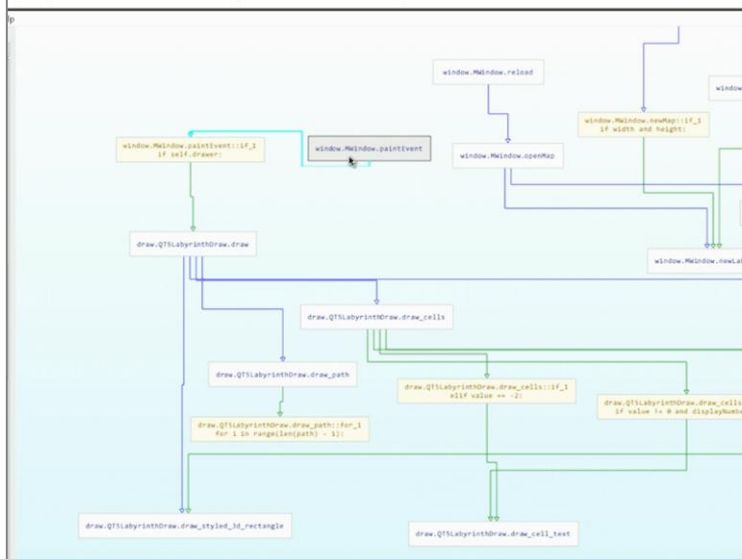
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.01.001 ПЗ

Лист

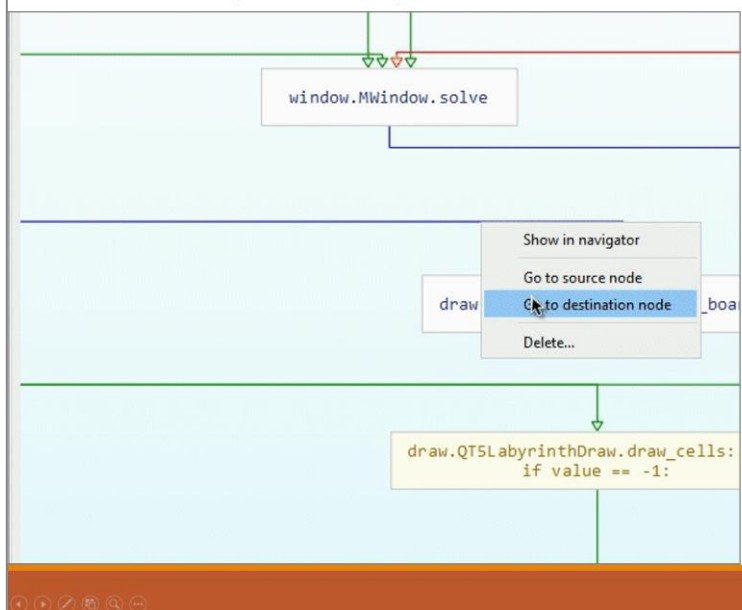
91

Демонстрация работы



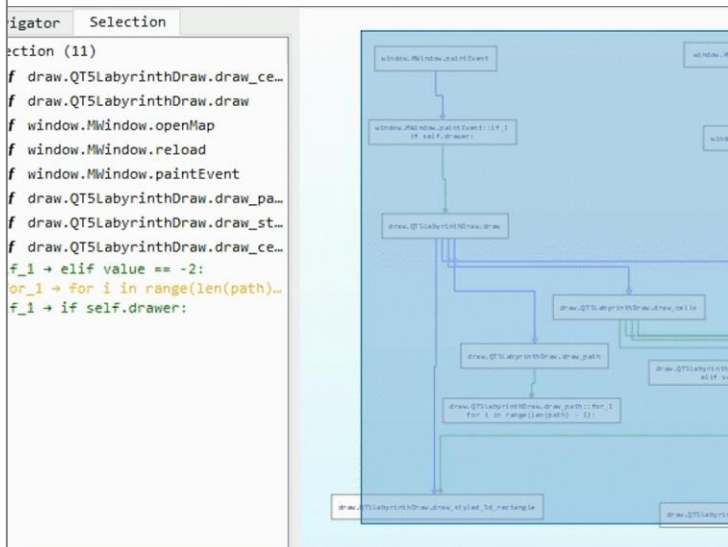
Перемещение узлов с автоматическим изменением пути ребер

Демонстрация работы



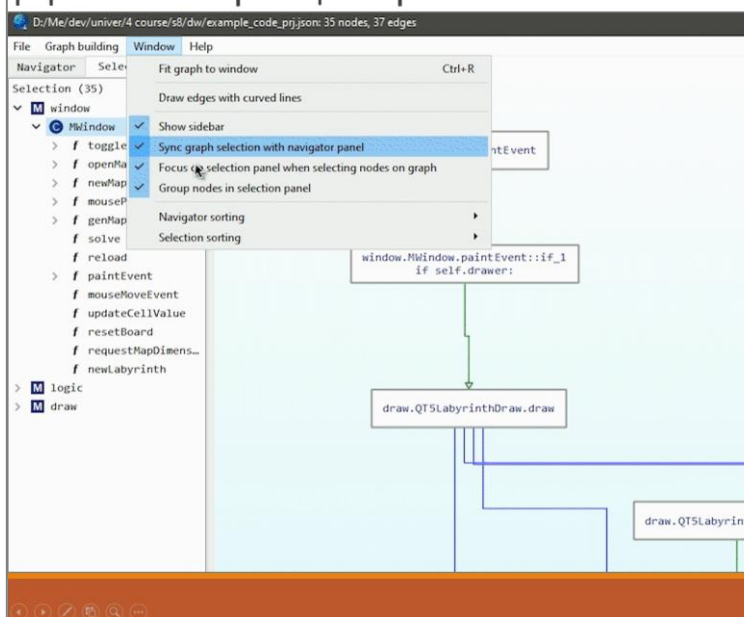
Переход к начальному и конечному узлу через контекстное меню и двойным кликом

Демонстрация работы



Показ выбранных элементов в виде списка с иерархией

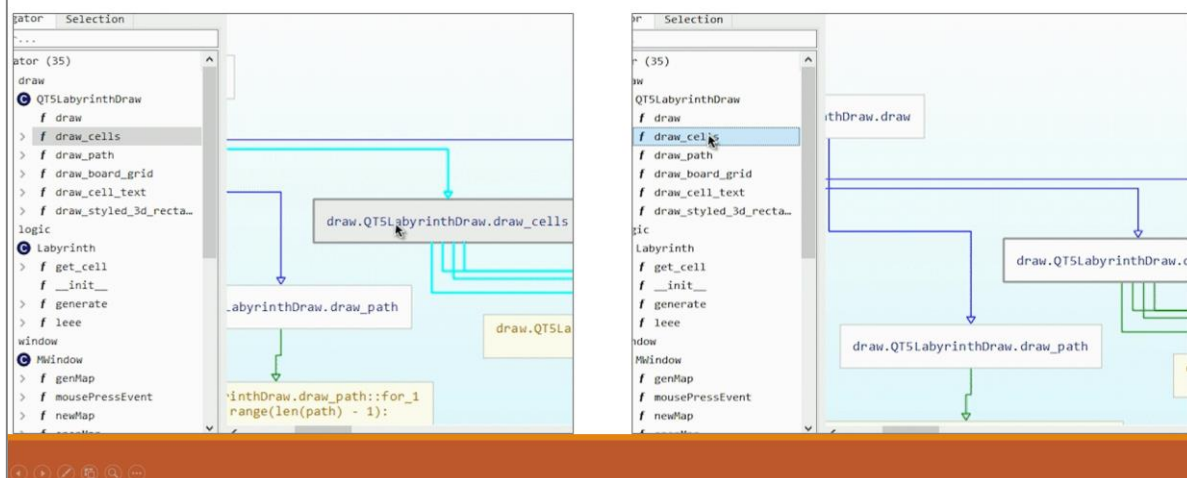
Демонстрация работы



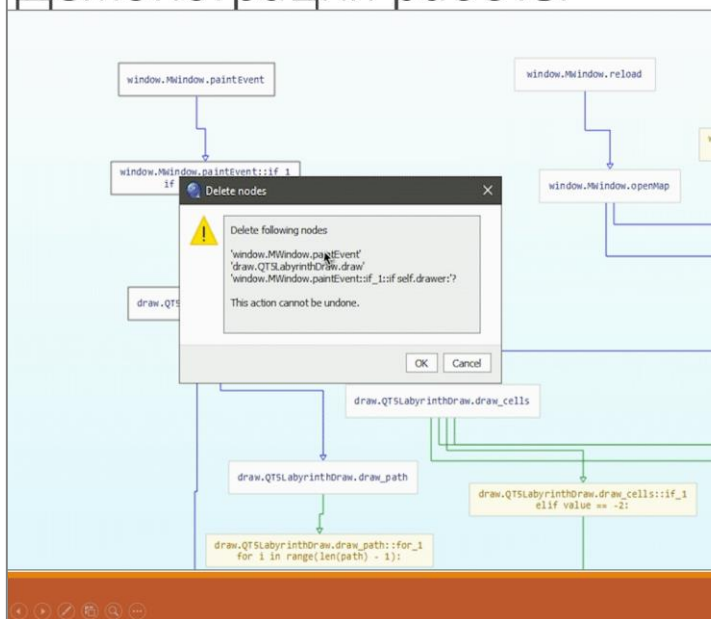
Возможность показывать выбранные элементы в виде плоского списка, без иерархии

Демонстрация работы

Синхронизация выбора между панелью навигации и графом
Обе функции возможно отключить

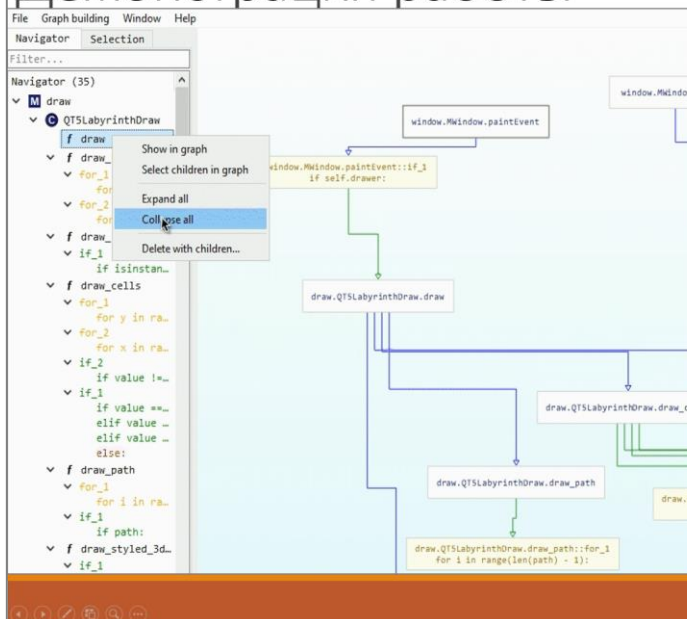


Демонстрация работы



Удаление узлов через
контекстное меню на графе
Также возможно через панель
навигации и клавишей «Delete»

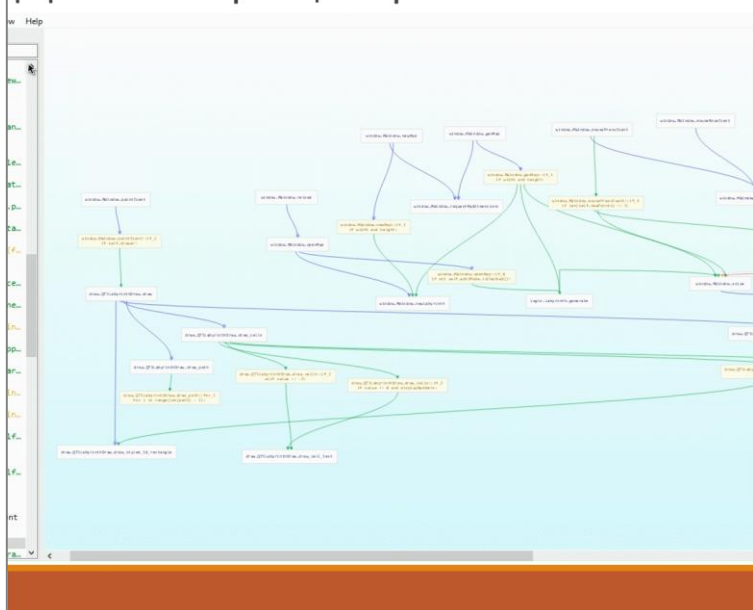
Демонстрация работы



Удобная работы с вложенными элементами в панели навигации:

- сворачивание и разворачивание
- тоже самое, но для всех элементов

Демонстрация работы



Режим рисования ребер с помощью кривых

Выводы

В ходе выполнения работы были выполнены все поставленные задачи:

- проанализирована предметная область, существующие аналоги;
- разработан компонент статического анализатора;
- разработан графический интерфейс пользователя.

Дальнейшее развитие:

- улучшение алгоритма сортировки узлов на графе;
- расширение списка поддерживаемых языков программирования;
- более глубокий анализ с вычислением условий.



СПАСИБО ЗА ВНИМАНИЕ!



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.01.001 ПЗ

Лист

96