Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ РК

КАФЕДРА РК-9

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовому проекту на тему:

Разработка программного обеспечения лабораторной работы по курсу Моделирование технологических и производственных процессов

Студент группы РК9-102 _____ А.О. Чернов
(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____ А.В. Урусов
(Подпись, дата) (И.О.Фамилия)

Москва, 2014

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине Проектирование КИП

Разработка программного обеспечения лабораторной работы по курсу
(Тема курсового проекта)
Моделирование технологических и производственных процессов

Студент Чернов А.О., РК9-102
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

1. Техническое задание

Разработать загружаемое расширение «Пятнашки» для системы дискретного имитационного моделирования RAO-studio и добиться его интеграции в рабочую версию.

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на 48 листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) 1 лист А1 – Постановка 2 лист А3 – Диаграмма компонентов подсистемы формирования исполняемого файла; 3 лист А3 – Диаграмма состояния процедуры слияния информации о плагинах; 4 лист А2 – Диаграмма последовательностей работы плагина «Пятнашки»; 5 лист А1 – Диаграмма классов плагина пятнашек; 6 лист А2 – Диаграмма активностей тестирования плагина; 7 лист А2 – Блок-схема алгоритма теста; 8 лист А1 – Результаты

Дата выдачи задания « ____ » _____ 20__ г.

Руководитель курсового проекта _____ А.В. Урусов
(Подпись, дата) (И.О.Фамилия)

Студент _____ А.О. Чернов
(Подпись, дата) (И.О.Фамилия)

1. Введение

Имитационное моделирование (ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, транспортные системы (морские порты, аэропорты) и комплексы перекачки нефти и газа, программное обеспечение сложных систем управления и вычислительные сети, а также многие другие.

Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

1. Сделать выводы о поведении СДС и ее особенностях:
   - без ее построения, если это проектируемая система
   - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно
   - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему
2. Синтезировать и исследовать стратегии управления
3. Прогнозировать и планировать функционирование системы в будущем
4. Обучать и тренировать управленческий персонал и т.д.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском Государственном Техническом Университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ,

связанных с системным анализом и организационным управлением сложными системами различной природы.

В языке имитационного моделирования РДО помимо возможности его использования для описания законов управления формализмов продукционных правил введены так называемые точки принятия решения, позволяющие осуществлять оптимальное управление[1].

Необходимость принятия оптимальных решений существует в различных областях деятельности человека. В тоже время сложные реальные системы и процессы не описываются математически, и поэтому при управлении и принятии решений требуется использовать имитационное моделирование. Становится необходимым решать задачи оптимизации, применяя имитационные модели в совокупности с алгоритмами оптимизации. Имитационная модель при таком подходе служит для оценки качества того или иного решения, а оптимизационный алгоритм осуществляет поиск наилучшего (или максимально приближенного к нему) решения. Механизм точек принятия решения в языке имитационного моделирования РДО позволяет гибко сочетать имитацию с оптимизацией. Для этого используется поиск на графе состояний.

Задачи, решаемые с помощью точек принятия решения:

- Транспортные задачи
- Задачи укладки грузов
- Задачи решения логических и других типов задач
- Задачи теории расписаний

2. Предпроектное исследование

Как уже было сказано ранее, РДО имеет механизм точек принятия решения и поиск по графу состояний, с помощью которого можно решать различные задачи, в том числе логические игры, такие как кубик Рубика, пятнашки и другие.

Поиск осуществляется по алгоритму A* – поиск по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины к другой.

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Этот алгоритм, однако, раскрывает не все вершины, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость».

Этому посвящена лабораторная работа по курсу Моделирование технологических и производственных процессов, в которой студенту предлагается найти оптимальное решение игры пятнашки, используя возможности РДО.

Основная цель лабораторной работы – освоить механизм точек принятия решения и алгоритм поиска A*, поняв предложенную модель, а также разобраться с эвристиками и предложить свою. Чтобы приступить к работе, необходима подготовка по двум темам:

1. Язык РДО. А именно синтаксис точек принятия решения и функций
2. Поиск на графе пространства состояний

В ходе лабораторной работы необходимо создать игровую ситуацию. Затем провести анализ трех уже предложенных эвристик. Для этого надо для одной начальной игровой ситуации провести три разных эксперимента, меняя предложенные эвристики. Результаты заносятся в таблицу. Нужно повторить эксперимент несколько раз с новыми начальными ситуациями, чтобы оценить эффективность каждой эвристики. После сбора статистики предложенных эвристик требуется разработать свою собственную. Для этого нужно поменять содержимое файла функций (закладка FUN), добавив туда необходимые функции. Имя новой эвристики вбивается в соответствующее поле настроек в интерфейсе. После получения статистики по эвристике и внесения результатов в таблицу, сделать вывод о её допустимости.

Игровая ситуация создается с помощью отдельного приложения. Это программное обеспечение представляет собой инструментарий для исследования работы механизма точек принятия решений, реализованных в

языке интеллектуального моделирования РДО. С помощью интерфейса программного обеспечения, пользователь может создать начальную ситуацию для “игры 5” (сокращенные “пятнашки”) и поменять законы алгоритма поиска решения. При этом модель создается автоматически. Цель и правила этой игры такие же, как и в “пятнашках”, т.е. расставить фишки в определенной последовательности, используя при перемещении только одну свободную ячейку. После моделирования имеется возможность визуально оценить полученный граф пространства состояний задачи, решение, если оно было найдено, и сделать выводы об эффективности заданных законов[RAO-game5, Руководство пользователя].

Вид главного окна приведен на Рисунок 1.

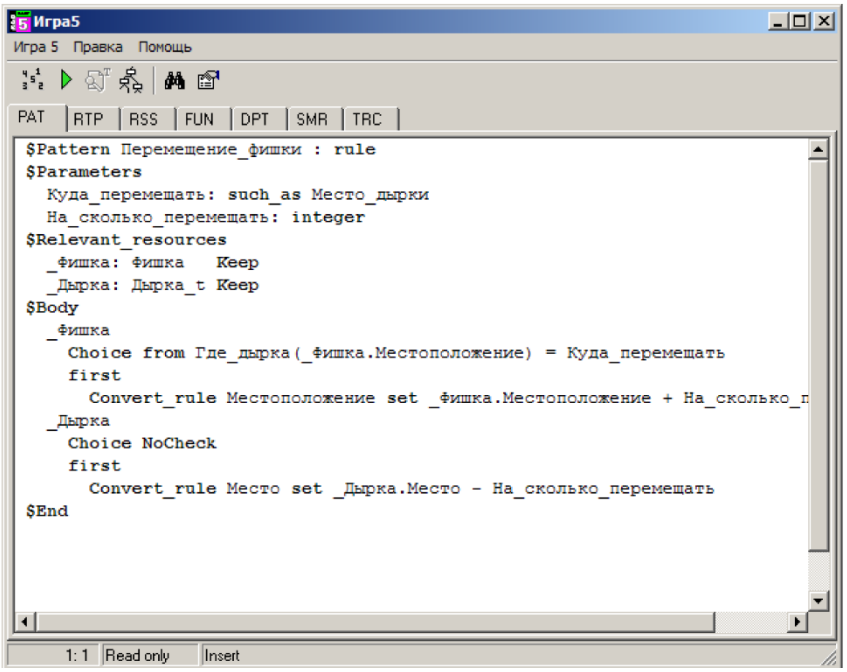


Рисунок 1

Для создания новой игровой ситуации можно воспользоваться пунктом меню **Игра 5/Разложить фишки**, после чего появится окно настроек, в котором можно задать расположение фишек вручную или получить случайную и правильную расстановки, используя соответствующие кнопки.

После вызова дополнительного диалогового окна у пользователя появляется возможность изменить содержимое файла точек принятия решений, то есть изменить законы алгоритма поиска решения на графе пространства состояний.

На Рисунок 2 представлено развернутое окно настроек.

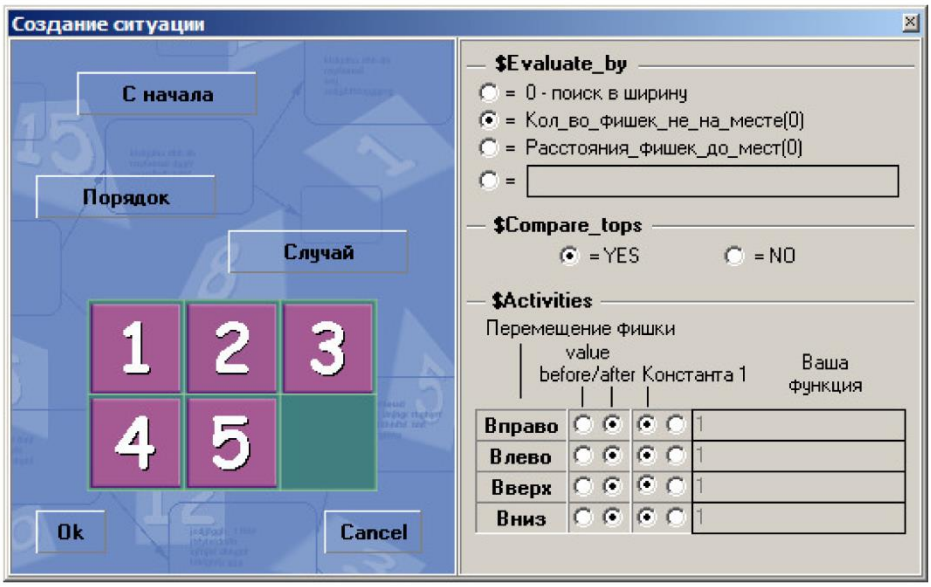


Рисунок 2

После формирования начальной ситуации необходимо нажать кнопку **Ок**, модель автоматически сгенерируется и запустится в системе моделирования RAO-studio.exe (РДО-имитатор). Сам имитатор должен располагаться в одной директории с RAO-game5.exe. Если это не так, то перед запуском модели необходимо указать место его расположения. В процессе моделирования, РДО-имитатор подготавливает файл трассировки, в который записывается ход расчета[RAO-game5, Руководство пользователя]

После удачной обработки трассировки можно построить и визуально оценить граф рассчитанного пространства состояний задачи и путь решения, если таковой был найден. Чтобы вывести граф на экран необходимо воспользоваться пунктом меню **Игра 5/Показать граф**, после чего откроется окно (Рисунок 3) с самим графом и результатами моделирования, необходимыми для отчета.

Приложение **Игра5** написано в среде разработки Delphi2 в 2007 году. С тех пор сама среда обновилась 15 раз[1]. Кроме того на сегодняшний день доля Delphi/Object Pascal среди разработчиков всего 0.691%[2]. В том время как сама система РДО активно развивается – с момента написания **Игры5** было совершено более 11000 изменений[2], само приложение не изменялось, соответственно, не поддерживалась работа с вновь выходящими версиями, так что теперь его сопровождение невозможно и нецелесообразно.

[1] Delphi 3, Inprise Delphi 4, Delphi 5, Delphi 6, Delphi 7, Delphi 8, Delphi 2005, Delphi 2006, Delphi 2009, Delphi 2010, Delphi XE, Delphi XE2, Delphi XE3, Delphi XE4, Delphi XE5

[2] Исходя из количества изменений в системе контроля версий[3]

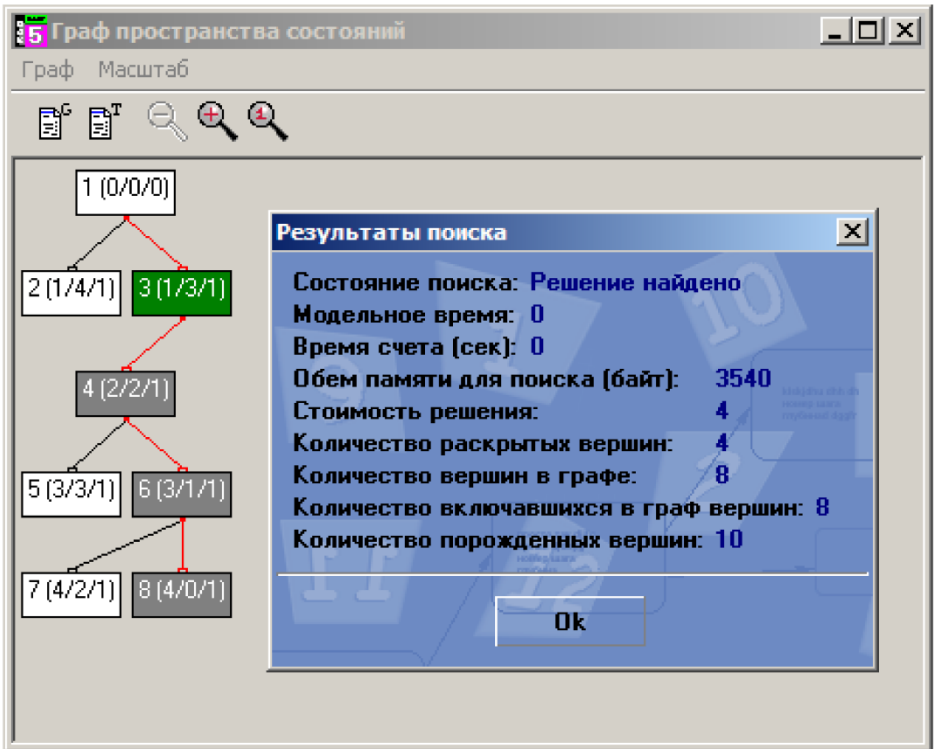


Рисунок 3

Поэтому было принято решение разработать с нуля подобного приложения, но с применением новых и поддерживаемых технологий. Поддержание актуальности становится доступным лишь при тесной интеграции «пятнашек» и РДО. Это является не только логичным требованием, но и большим плюсом, ведь лабораторная работа использует РДО и изменение в сторону того, что она целиком будет выполняться в этой системе, является положительным. Однако нагружать новым функционалом систему РДО на постоянной основе – не рационально, поэтому воспользуемся технологией **Плагинов**.

Плагин (англ. plugin, от plug in «подключать»), — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей.

В прошлом семестре в рамках курсового проекта мною уже была разработана подсистему загрузки плагинов в среду разработки RAO-studio. В рамках этого курсового проекта поставлена задача разработки и внедрения в эксплуатацию подключаемого расширения «Пятнашки».

Автоматическая генерация текста модели и тесная интеграция приводит к тому, что любое изменение в грамматике языка РДО ведет к созданию модели, которую невозможно запустить, по причине ошибочного, с точки зрения нового синтаксиса, текста модели. Поэтому отдельное внимание стоит уделить автоматическому тестированию.

2.1. Постановка задачи

Проектирование любой системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой.

В результате предпроектного исследования была выявлена необходимость в разработке динамически подключаемого к rdo-studio модуля, предназначенного для расширения возможностей основной программы, необходимых для выполнения лабораторной работы, а именно:

- Задание положение игровых фишек
- Задать настройки стоимости применения правил
- Автоматическое заполнение вкладок модели и её запуск
- Автоматический анализ трассировки прогона
- Вывод информации, построение графа

Автоматическое заполнение вкладок модели в условиях тесной интеграции с системой и командной разработки влечет за собой требование автоматического тестирования работы плагина.

Результаты разработки планируется использовать в качестве программного обеспечения лабораторной работки, что влечет за собой требование обновления документации:

- Постановка задачи на лабораторную работу
- Руководство пользователя для программного обеспечения

3. Концептуальный этап проектирования

3.1. Выбор общесистемной методологии проектирования

Задача, поставленная на этапе предпроектного обеспечения, может быть решена на основе следующих концепций:

- Модульность
- Объектная ориентированность

Модульность — это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне связанных между собой модулей. Применительно к конструированию технических систем модульность — принцип, согласно которому функционально связанные части группируются в законченные узлы — модули. В свою очередь модульность в программировании — принцип, согласно которому программное средство (ПС) разделяется на отдельные именованные сущности, называемые модулями. Модульность часто является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Выбранная модель проектирования, позволила производить разработку поэтапно и разделить разработку плагина и подсистемы, необходимой для его (и других плагинов) загрузки.

Как итог, необходимо разработать модуль реализующий интерфейс, необходимый для загрузки и который является связующим звеном

загружаемого расширения и системы в целом. Интерфейс был разработан вместе с подсистемой загрузки плагинов.

Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона. В отличие от концепции интерфейсов во многих других областях, интерфейс в ООП является строго формализованным элементом объектно-ориентированного языка и в качестве семантической конструкции широко используется кодом программы.

3.2. Диаграмма компонентов

На Рисунок 4 представлена «as is» диаграмма компонентов подсистемы, отвечающей за формирование исполняемого файла RAO-studio.

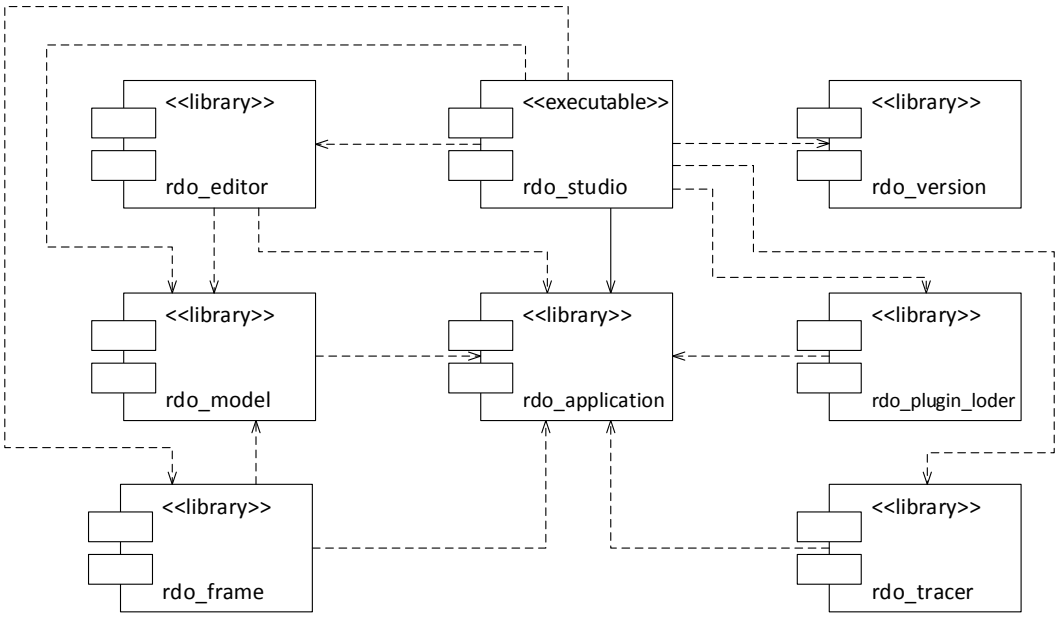


Рисунок 4

Базовый функционал этих пакетов:

rdo_studio – подсистема верхнего уровня

rdo_application – подсистема, реализующая графическую часть RAO-studio

rdo_version – подсистема, реализующая автоматический подсчет текущей версии из системы контроля версий.

rdo_editor – подсистема, реализующая функционал редактора.

rdo_frame – подсистема, реализующая функционал настройки отображения анимации.

rdo_tracer – подсистема, реализующая функционал сбора и отображения статистики (трассировка).

rdo_model – подсистема, реализующая функционал модели, в том числе её текстового редактора.

rdo_plugin_loader – подсистема, реализующая функционал загрузки плагинов.

3.3. Выделение системы из среды

Рассмотрим подробнее подсистему **rdo_plugin_loader**. Она обладает следующим функционалом:

- обнаружение, загрузка, сбор информации, "выгрузка" плагинов при запуске РДО
- отображение информации о плагине
- возможность запустить/остановить плагины
- возможность назначать автозапуск плагинов при загрузке РДО
- хранение информации о параметрах автозапуска для плагинов
- возможность удаления информации о плагине
- возможность удаление плагина с диска

Система имитационного моделирования РДО, для которой разрабатывается подсистема, написана с использованием кроссплатформенной библиотеки Qt, в частности эта библиотека предоставляет инструменты для работы с плагинами и хранения системной информации.

Для разрешения всех требований к системе, эта подсистема содержит класс, содержащий всю информацию о плагине и она должна хранить список, состоящий из этих классов.

Этот список необходим для реализации принципа, по которому плагины загружаются в систему. Подсистема рекурсивно пытается загрузить все плагины из всех подпапок папки **Plugins**. Это удобно, ведь добавляется возможность сортировки плагинов по папкам при большом количестве расширений. Следовательно, разрабатываемый плагин следует расположить в этой папке (или какой-нибудь подпапке этой директории)

Исходя из задачи возможности автозапуска плагина, возникает необходимость идентификация, причем, возникает необходимость уникальности, чтобы не допустить автозагрузки неизвестного плагина. Поэтому плагину нужна функция, сообщающая свой GUID[3] подсистеме, ведь по нему и будет идентифицироваться загружаемая динамическая

[3] GUID (Globally Unique Identifier) — статистически уникальный 128-битный идентификатор.

библиотека. Хотя GUID и обладает высокой уникальностью, это не защищает от его механического повторения, поэтому остальные обязательные поля класса PluginInfo являются вспомогательным ключом при идентификации плагина.

Каждый раз при загрузке студии РДО подсистема выясняет, для каких плагинов выполнить автозагрузку, для этого выполняется слияние двух списков: с информацией о вновь загруженных плагинах и хранимой информацией об автозагрузке плагинов.

Расширение может находиться в четырех состояниях:

- Новый плагин (**Unique**)
- Успешно повторно загруженный (**ExactMatched**)
- Загруженный плагин, информация о котором не совпадает с сохраненной (**IdOnlyMatched**)
- Плагин, о котором сохранена информация, но он не найден на диске (**Deleted**).

Это необходимо для того, чтобы сохранять информацию о параметрах автозапуска только для плагинов в состоянии ExactMatched или Unique.

По мере загрузки новых плагинов состояние остальных может меняться. На Рисунок 5 изображена диаграмма состояний изменения информации о плагине.

Все загружаемые расширения должны реализовывать один и тот же интерфейс, который используется при загрузке плагина, поэтому на этапе концептуального проектирования к плагину выдвигаются следующие требования:

- содержание информации о названии плагина, версии и авторе
- наличие методов, вызываемых при запуске/остановке
- идентификация с помощью GUID

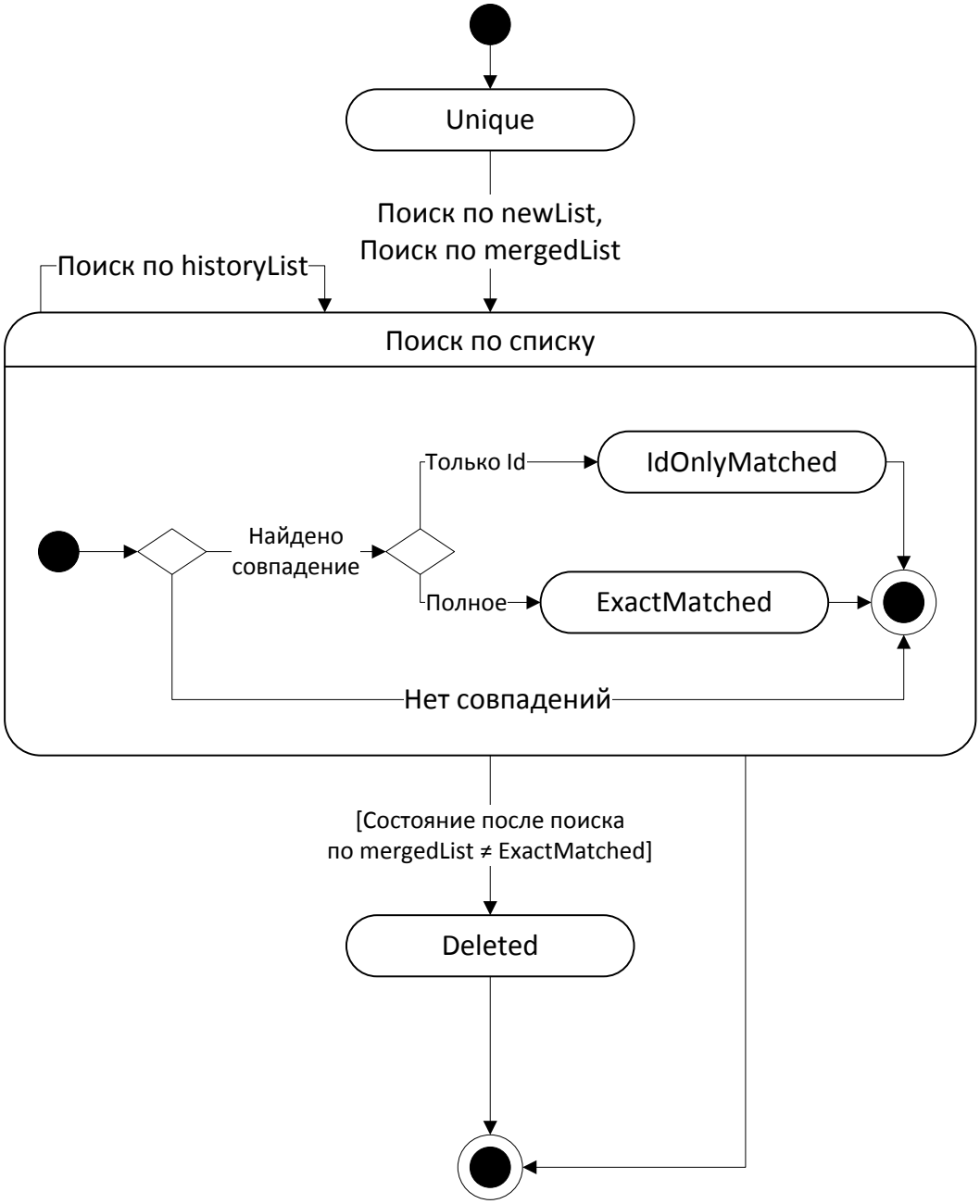


Рисунок 5

Блок-схема реализованного алгоритма слияния изображена в Приложение 1. Блок-схема алгоритма слияния списков информации о плагинах.

Блок-схема алгоритма функции **matchPluginInfo**(const PluginInfoList& list, const LPPluginInfo& plgnInfo), которая ищет соответствия plgnInfo по всему списку list и в зависимости от результата меняет его состояние в системе, представлена в приложении 2.

Программный код, реализующий приведенные алгоритмы, приведен в приложении 3.

4. Формирование технического задания

4.1. Введение

Программный комплекс RAO-studio предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса – обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов.

В языке имитационного моделирования РДО помимо возможности его использования для описания законов управления формализмов продукционных правил введены так называемые точки принятия решения и механизм поиска по графу состояний, позволяющие в сумме осуществлять оптимальное управление.

Задачи, решаемые с помощью точек принятия решения:

- Транспортные задачи
- Задачи укладки грузов
- Задачи решения логических и других типов задач
- Задачи теории расписаний

4.2. Общие сведения

- Полное наименование темы разработки: «Загружаемое расширение «Пятнашки» для системы дискретного имитационного моделирования РДО»
- Заказчик: Кафедра "Компьютерные системы автоматизации производства " МГТУ им. Н.Э.Баумана"
- Разработчик: студент кафедры "Компьютерные системы автоматизации производства" Чернов А.О.
- Основание для разработки: Задание на курсовой проект
- Плановые сроки начала работы: 31 августа 2013г.
- Плановые сроки окончания работы по созданию системы: 28 мая 2014г.

4.3. Назначение разработки

Разработать плагин «Пятнашки» и добиться его интеграции в рабочую версию RAO-studio.

Функциональным назначением объекта разработки является предоставление пользователю расширенных возможностей системы РДО.

Разработка должна эксплуатироваться студентами кафедры «Компьютерные системы автоматизации производства» в качестве программного обеспечения для выполнения лабораторной работы, цель которой изучить механизм точек принятия решения и алгоритм поиска A*, а также разобраться с моделью и эвристиками, предложить свою. А также другие пользователи в ознакомительных и других целях.

4.4. Требования к программе или программному изделию

Требования к функциональным характеристикам:

- Плагин должен реализовывать интерфейс подсистемы загрузки плагинов, то есть:
    1. содержание информации о названии плагина, версии и авторе
    2. наличие методов, вызываемых при запуске/остановке
    3. идентификация с помощью GUID
- Иметь возможность создания меню и панель инструментов в RAO-studio
- Реализовать окно создания начальной ситуации со следующими возможностями:
    1. Задать положение игровых фишек: вручную, т.е. передвигая фишки с помощью мыши, а так же полуавтоматически в случайном, заданном с клавиатуры или правильном порядках
    2. Установить флажок, при котором случайная расстановка фишек не генерирует ситуаций без решения
    3. Задать настройки стоимости применения правил
    4. Выбрать эвристику: одну из трех предложенных или свою, описанную во вкладке FUN, поле ввода эвристики должно обладать автозаполнением описанных функций для исключения опечатки
    5. Автоматическое заполнение вкладок модели и её запуск при нажатии на кнопку **Ок**
- Реализовать окно вывода графа со следующими возможностями
    1. Автоматический анализ трассировки прогона, обновление и отображения графа

2. Возможность масштабировать и панорамировать граф
3. Вывод информации о графе на экран
4. Выделение вершин графа, относящихся к решению
5. Вызов подробной информации о каждой вершине по клику

Требования к надежности:

Основное требование к надежности направлено на поддержание в исправном и работоспособном ЭВМ, на которой происходит использование программного комплекса RAO-Studio.

Проектные решения должны обеспечивать:

- Сохранение работоспособности системы при отказе по любым причинам подсистемы или её части
- Количество отказов из-за невыявленных ошибок не более 1 на 1000 сеансов работы с программой

Должны иметь защиту от некорректных действий пользователей и ошибочных исходных данных.

Условия эксплуатации:

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости
- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением

Требования к составу и параметрам технических средств:

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 256 Мб
- микропроцессор с тактовой частотой не менее 400 МГц
- требуемое свободное место на жестком диске – 40 Мб

Требования к информационной и программной совместимости:

- операционная система Windows XP и старше или Ubuntu 12.10 и старше[4]

[4] Microsoft, Windows являются зарегистрированными торговыми марками или торговыми марками Microsoft Corporation (в США и/или других странах).

- наличие в операционной системе средства просмотра справки в формате Qt - Qt Assistant

Требования к маркировке и упаковке:

Не предъявляются.

Требования к транспортированию и хранению:

Не предъявляются.

4.5. Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

- техническое задание
- технический и рабочий проекты
- внедрение

На стадии «Техническое задание» должен быть выполнен этап разработки и согласования настоящего технического задания.

На стадии «Технический и рабочий проект» должны быть выполнены перечисленные ниже этапы работ:

- разработка программы
- разработка методики тестирования
- разработка программной документации
- испытания программы

На стадии «Внедрение» должен быть выполнен этап разработки «Подготовка и передача программы».

4.6. Порядок контроля и приемки

Контроль и приемка работоспособности плагина «Пятнашки» должны осуществляться в процессе проверки функциональности (апробирования) системы имитационного моделирования путем многократных тестов в соответствии с требованиями к функциональным характеристикам системы.

Учитывая специфику плагина, а именно автоматическая генерация текста модели и тесная интеграция, в условиях командной разработки необходимо разработать автоматическое тестирование работы плагина.

---

5. Технический этап проектирования

Как уже было сказано ранее, плагин для системы РДО должен реализовывать интерфейс PluginInterface. Кроме того плагин должен реализовывать функции, ради которых он создавался, поэтому необходимо создать два класса:

- Окно создания начальной ситуации
- Окно вывода

На Рисунок 66 изображена упрощенная диаграмма классов, показывающая структуру плагина, разработанную на этапе технического проектирования.

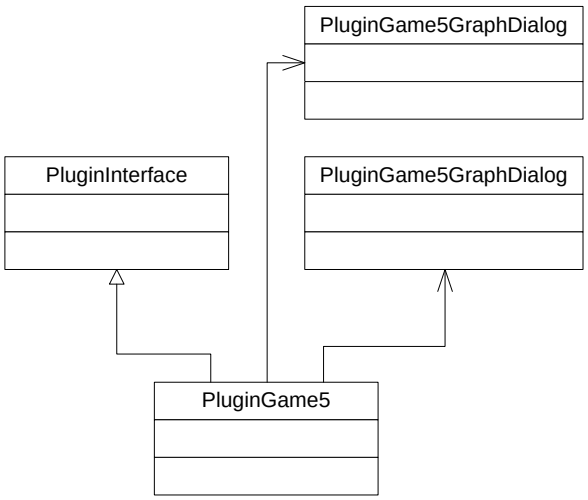


Рисунок 6

5.1. Разработка архитектуры класса диалогового окна для задания начальной ситуации

Для реализации требуемого функционала, а именно:

- Возможность задать положение игровых фишек: вручную, т.е. передвигая фишки с помощью мыши, а так же полуавтоматически в случайном, заданном с клавиатуры или правильном порядках
- Возможность задать настройки стоимости применения правил
- Автозаполнение описанных функций в поле выбора эвристики

необходимо вверить во владение диалогу задания начальной ситуации классы согласно структуре указанной на диаграмме на Рисунок 7.

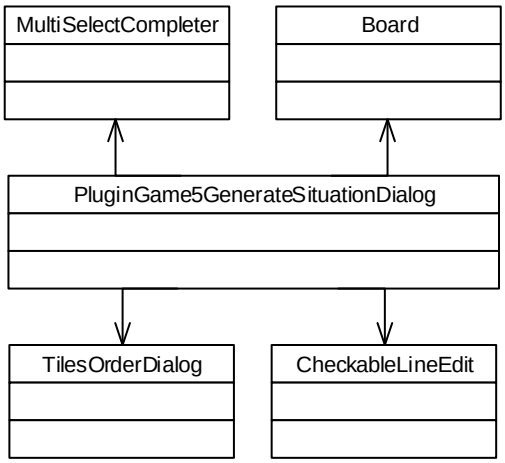


Рисунок 7

MultiSelectCompleter – класс, добавляющий в плагин автодополнение.

Board – класс, необходимый для отображения расстановки фишек и её ручной настройки.

TilesOrderDialog – класс, реализующий диалоговое окно расстановки фишек по местам, заданным с клавиатуры.

Для построения пользовательский интерфейса диалогового окна воспользуемся библиотекой Qt, однако, по задумке для настройки стоимости применения правила необходим элемент с флажком и текстовым полем, доступным для редактирования только при активном флажке.

CheckableLineEdit – класс –управляющий элемент, являющийся «контейнером» для поля ввода и флажка, объединяющим в себе их свойства и добавляющим необходимую особенность – текстовое поле, становится доступным для редактирования только при активном флажке.

Подробнее эта структура будет рассмотрена на этапе рабочего проектирования.

5.2. Разработка архитектуры класса диалогового окна для построения графа

Для реализации требуемого функционала, а именно:

- Отображения графа
- Вызов подробной информации о каждой вершине по клику

необходимо реализовать структуру, указанную на диаграмме классов на Рисунок 8.

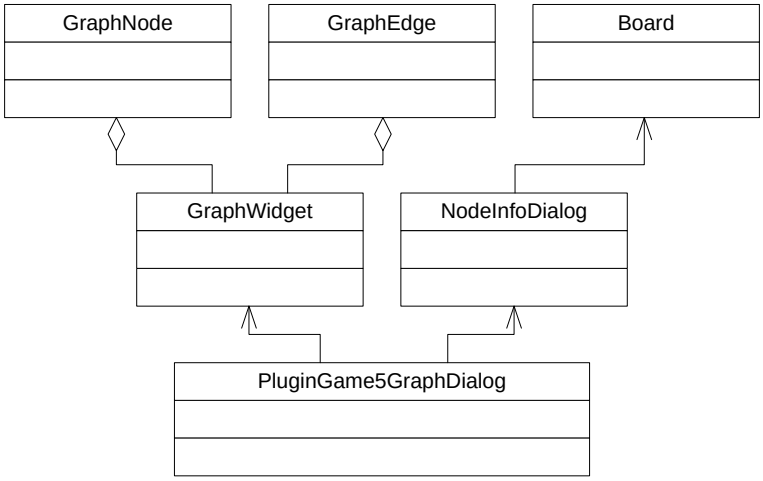


Рисунок 8

GraphWidget – класс, реализующий отрисовку на экране.

GraphNode – класс вершины графа.

GraphEdge – класс грани графи.

NodeInfoDialog – класс, реализующий диалоговое окно с подробной информацией он вершине.

Board – класс, необходимый для отображения расстановки фишек, описанный выше. В данном случае, фишки должны оставаться неподвижными. Было принято решение, сделать Board универсальным, добавив атрибут активности фишек, а не создавать два почти одинаковых класса, ведь дублирование – главный враг хорошо спроектированной системы[5].

Подробнее эта структура будет рассмотрена на этапе рабочего проектирования.

6. Рабочий этап проектирование

6.1. Структура плагина «Пятнашки»

Диаграмма классов загружаемого расширения приведена на Рисунок 9.

На ней видно, что основной класс плагина «Пятнашки» наследуется от **PluginInterface**, и реализует методы этого интерфейсного класса, эта необходимость описана на техническом этапе проектирования. Кроме того **PluginGame5** реализует методы класс **QObject**. Это необходимо, чтобы воспользоваться мощным функционалом парадигмы Сигнал/Слот, предоставляемой библиотекой Qt.

Сигналы и слоты - это фундаментальный механизм Qt, позволяющий связывать объекты друг с другом. Связанным объектам нет необходимости что-либо "знать" друг о друге. Сигналы и слоты гораздо удобнее механизма функций обратного вызова (callbacks) и четко вписываются в концепцию ООП[6].

Для использования этого механизма объявление класса должно содержать специальный макрос **Q_OBJECT** на следующей строке после ключевого слова **class**.

Класс **QObject** используется по той же причине и на остальных диаграммах класса.

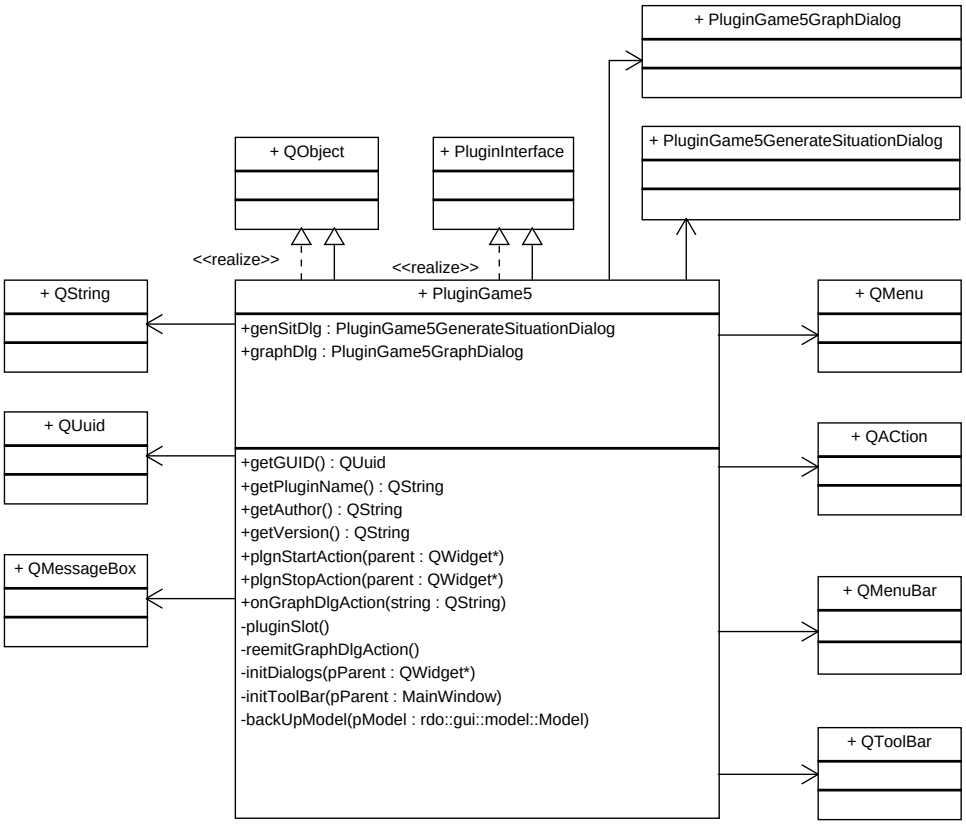


Рисунок 9

Что бы реализовать необходимый функционал плагина, а именно:

- Иметь возможность создания меню и панель инструментов в RAO-studio
- Реализовать окно создания начальной ситуации и окно построения графа

плагину необходимы экземпляры классов **QMenu, QAction, QMenuBar, QToolBar и PluginGame5GenerateSituationDialog, PluginGame5GraphDialog** соответственно.

Все кроме последних двух, которые разрабатываются в рамках работ по созданию плагина и будут рассмотрены подробнее ниже, предоставляются библиотекой Qt.

6.2. Структура класса диалогового окна создания начальной ситуации

Диаграмма классов представлена в приложении 4, она получена уточнением и подробным описанием структуры полученной на этапе технического проектирования. Как было сказано на этом этапе, классы **Board, TilesOrderDialog, CheckableLineEdit, MultiSlectCompleter** используются для реализации требуемого функционала, а именно:

- Возможность задать положение игровых фишек: вручную, т.е. передвигая фишки с помощью мыши, а так же полуавтоматически в случайном, заданном с клавиатуры или правильном порядках
- Возможность задать настройки стоимости применения правил
- Автозаполнение описанных функций в поле выбора эвристики

MultiSlectCompleter наследуется от класса **QCompleter,** реализует его виртуальные функции и расширяет его возможности, а именно автодополнение из списка доступных слов, в том числе нескольких в одном поле ввода и в середине строки.

TilesOrderDialog наследуется от класса **QDialog** и класса пользовательского интерфейса **Ui_TilesOrderDialog,** который имеет простую структуру: две кнопки **QPushButton** и однострочное текстовое поле **QLineEdit.** Класс, реализующий диалоговое окно расстановки фишек по местам, так же использует экземпляр класса **QRegExpValidator,** который не позволяет указать не правильные номера фишек в строке, задающей состояние на игровом поле, что должно уберечь от ошибок, вызванных опечатками.

Экземпляры классов **MultiSlectCompleter** и **TilesOrderDialog** принадлежат классу **PluginGame5GenerateSituationDialog,** который наследуется от класса **QDialog** и класса пользовательского интерфейса

Ui_PluginGame5GenerateSituationDialog, которому для выполнения функций всего диалогового окна в целом, а именно:

- Возможность задать положение игровых фишек вручную
- Возможность задать настройки стоимости применения правил

Необходимы экземпляры классов **Board** и **CheckableLineEdit.**

CheckableLineEdit – класс, наследуемый от **QWidget**, и являющийся «контейнером» для **QLineEdit** (поле ввода) и **QCheckBox** (флажок), объединяющий в себе их свойства и добавляющий необходимую особенность: текстовое поле, доступным для редактирования только при активном флажке.

Board – класс, необходимый для отображения расстановки фишек и её ручной настройки. Его структура будет рассмотрена ниже.

Свою главную функцию: Автоматическое заполнение вкладок модели и её запуск при нажатии на кнопку Ок, класс может выполнить благодаря указателю на объект класса **rdo::gui::model::Model**, который владеет текстовым редактором и имеет сигнал: управления стартом прогона.

6.3. Структура игрового поля

На Рисунок 10 – диаграмма классов, изображающая структуру игрового поля.

Класс **Board** наследуется от класса **QFrame**, что есть, по сути, **QWidget** с рамкой. Класс параметризованный, что вводит для разработчика удобный способ изменять игровое поле и масштабировать задачу.

Board владеет экземпляром шаблонного класса **std::vector<PlacedTile*>,** который является динамическим массивом произвольного доступа с автоматическим изменением размера при добавлении/удалении элемента, в данном случае указателей на экземпляр класса **PlacedTile.**

PlacedTile – класс фишки игрового поля, он наследуется от класса **Tile**, расширяя его функционал наличием позиции этой фишки. Ведь принципиально фишка не должна знать, где она находится, этим управляет и следит игровое поле.

Класс **Tile** является реализацией игровой фишки. Он наследуемого от кнопки **QPushButton**, расширяя её функционал наличием атрибута – номер фишки и соответствующих методов, возвращающих значение атрибута при нажатии на фишку.

Настройка внешнего вида **Board** и **Tile** происходит с помощью класса **QStyleSheet.**

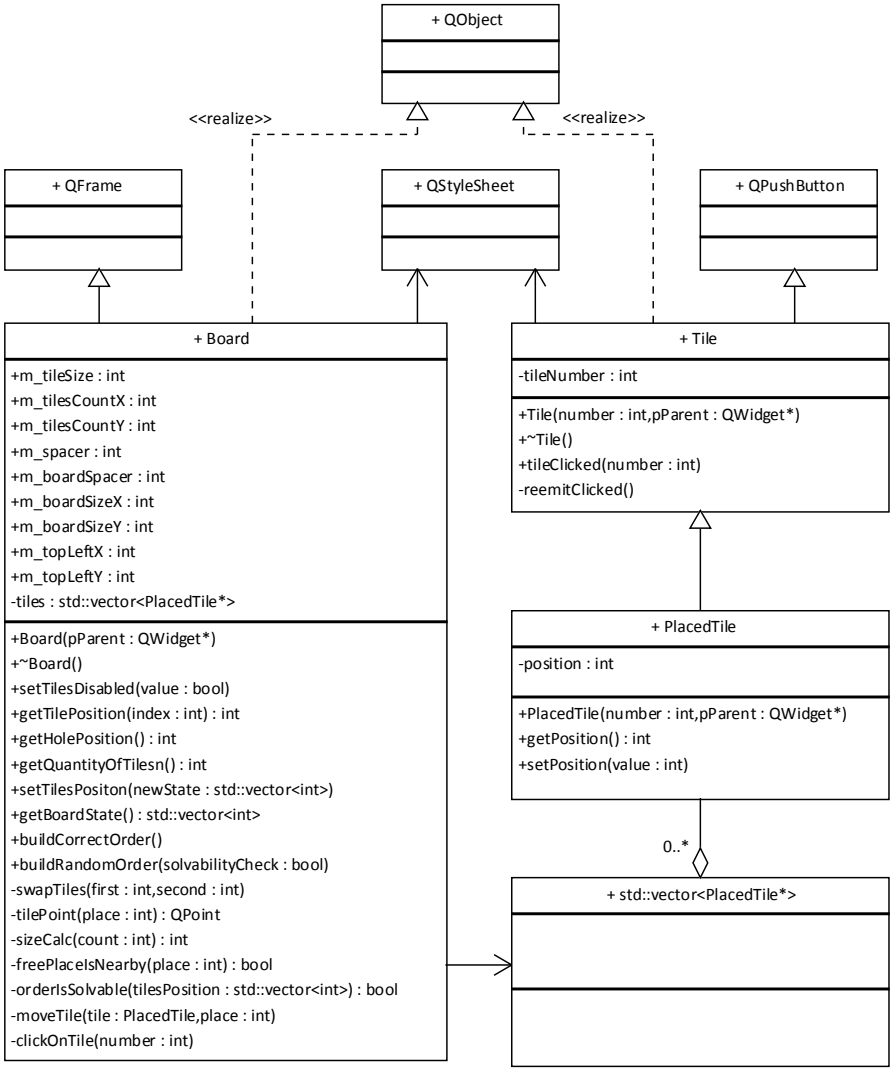


Рисунок 10

6.4. Структура класса диалогового окна построения графа

В приложении 5 представлена диаграмма классов, показывающая структуру окна построения графа.

Основной на этой диаграмме это собственно класс окна построения графа – **PluginGame5GraphDialog.**

Он наследуется от **QDialog** и класса, реализующего пользовательский интерфейс **Ui_PluginGame5GraphDialog** и владеющий одним лишь экземпляром класса **GraphWidget.**

Информация, необходимая для построения графа получается непосредственно из файла трассировки РДО-студии.

Для работы с файлом используется экземпляр класса **QFile,** необходимое расположение файла на диске получают из класса **rdo::gui::model::Model,** который упоминался ранее.

Нужная информация получается путем разбора всего текста трассировки с помощью класса **QRegExp,** Qt реализация регулярных выражений, формального язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Это эффективный способ лексического разбора текста.

GraphInfo – внутренний класс, созданный для удобства разбора информации о графе и дальнейшего обновления отображения этих данных.

На основе данных из файла трассировки создаются экземпляры класса **GraphNodeInfo.** Это класс, хранящий в себе разобранную информацию о вершине. Результат разбора хранится в виде экземпляра шаблонного класса **std::vector<GraphNodeInfo>.** От **GraphNodeInfo** наследуется класс **GraphNode,** расширяя функционал до требуемого. Эти два класса будут рассмотрены подробнее ниже. Строимый граф хранится в классе диалогового окна в виде экземпляра шаблонного класса **std::vector<std::vector<GraphNode>>,** что является двумерным динамическим массивом с произвольным доступом. Представление в таком виде, как двумерный массив, где ряд соответствует одному уровню вложенностей, позволяет производить быструю сортировку.

Для этих нужд создана внутренняя структура **UnbuildRange,** которая нужна для удобного хранения в системе информации о листовых вершинах графа, положение которых не может быть определено, через вершины-потомки.

PluginGame5GraphNodeInfoDialog – класс диалогового окна с подробной информацией о вершине, наследуется от **QDialog** и **Ui_PluginGame5GraphNodeInfoDialog.**

Как видно по диаграмме классов графический интерфейс **Ui_PluginGame5GraphNodeInfoDialog** состоит из нескольких текстовых выводов **Qlabel,** двух кнопок **QPushButton**, необходимых для навигации по графу из диалога и класса игрового поля **Board**, структура которого уже была рассмотрена.

6.5. Структура класса GraphWidget

В приложении 6 представлена структура класса **GraphWidget,** который наследуется от **QGraphicsView.** Для реализации вывода на экран графа ему необходимо владение экземпляром класса **QGraphicsScene**, который в свою очередь связан отношением композиция с классами **GraphNode** и **GraphEdge,** что означает жёсткую зависимость времени существования экземпляра «контейнера» и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Сам класс **GraphNode** хранит список вершин-потомков и вершину-предка, то есть экземпляры своего класса, это необходимо для построения

графа. Для хранения списка используется экземпляр шаблонного класса + **std::list<GraphNode>.** Класс вершины графа закрыто наследуется от **GraphNodeInfo,** класса хранящего информацию, полученную из разбора файла трассировки.

GraphEdge наследуется от класса **QGraphicsItem,** как и класс **QGraphicsObject,** который обладает к тому же функциями **QObject,** т.е. умеет обрабатывать сигналы(такие как клик), а так как вершине графа для отображения подробной необходимо обрабатывать клики, то класс **GraphNode** наследуется именно от **QGraphicsObject.**

За вывод подробной информации о графе отвечает класс **PluginGame5GraphNodeInfoDialog,** который был описан выше.

6.6. Алгоритм отрисовки графа

Алгоритм отрисовки графа представлен в приложении 7.

Программный код, реализующий этот алгоритм, представлен в приложении 8.

Для работы алгоритма требуется предварительная сортировка всего массива вершин графа, а именно необходимо создать шаблонный класс **std::vector<std::vector<int>>**, то есть двумерный упорядоченный массив с произвольным доступом по индексу элемента. Такая структура данных удобна для хранения произвольного количества вершин на каждом «уровне» графа. Вершины записываются в этот массив в порядке появления на графе. Сортировка проводится методом **std::sort()**, это функция из стандартной библиотеки c++, хороша она тем, что сложность её алгоритма $O(n \bullet \ln_2 n)$. В качестве функции сравнения используется положение вершины-родителя на графе. Доступ к этому значению получается через указатель на вершину родителя, который нужен еще и для навигации по графу.

Согласно алгоритму каждая «строка» графа два раза.

На первом проходе обрисовываются вершины, у которых есть потомки, так как их горизонтальная позиция может быть высчитана как средняя величина из позиций вершин-потомков. В этом же проходе формируются массив из структур **UnbuildRange,** который хранит информацию обо всех последовательностях непостроенных вершин.

На втором проходе достраиваются оставшиеся вершины. Возможны три варианта расположения последовательности листовых вершин: слева, в середине и справа. Нижний ряд вершин никогда не имеет вершин-потомком, поэтому он строится отдельно: за один раз все вершины с одинаковым шагом. Расположение последовательности справа или слева не представляет трудности и по сути не отличается от отрисовки первого ряда, нужно

поместить вершины на граф с фиксированным шагом с соответствующей стороны от крайней вершины.

Ряд листовых вершин в середине графе требует особой обработки, ведь при маленьком расстоянии между будущими соседями этих вершин возможно наложение. Чтобы этого избежать при отрисовке этого случая проверяется расстояние, в которое надо вписать последовательность и в случае нехватки место под неё, в прямом смысле, раздвигается. Для этого у всех соседей, например, с правой стороны вызывается метод-член класса **GraphNode – forceShift(int)**, в качестве параметра передается недостаток расстояния. Чтобы не потерять стройный порядок, достигнутый при построении на предыдущих шагах, этот метод в своем теле рекурсивно вызывается для всех потомков экземпляра класса, если такие есть.

Максимальная вложенность циклов этого алгоритма тройная, она обуславливается использованием двумерного массива, окончательно сложность алгоритма $O(n \bullet \ln_2 n) + O(n^2) = O(n^2)$, что неплохо для построения упорядоченного дерева.

На Рисунок 11 представлен результат работы алгоритма построения графа.

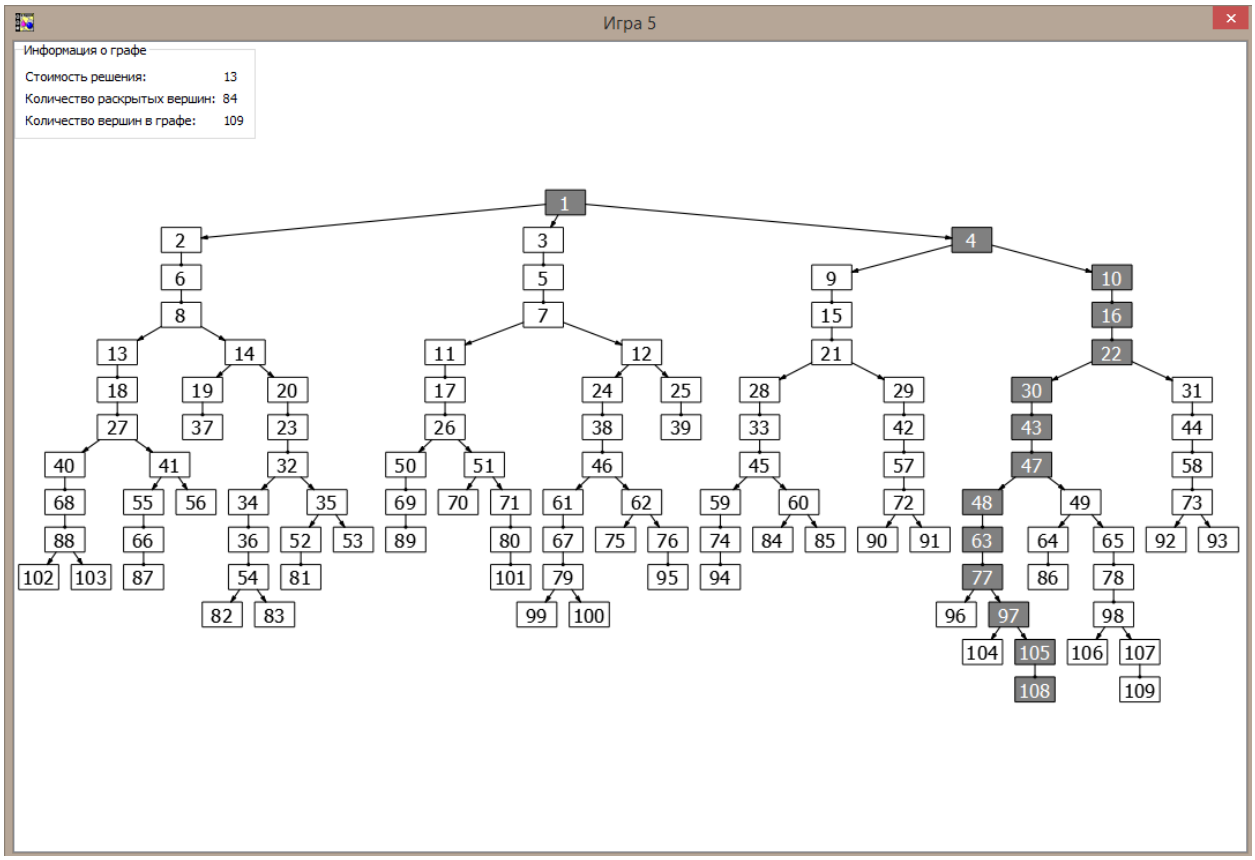


Рисунок 11

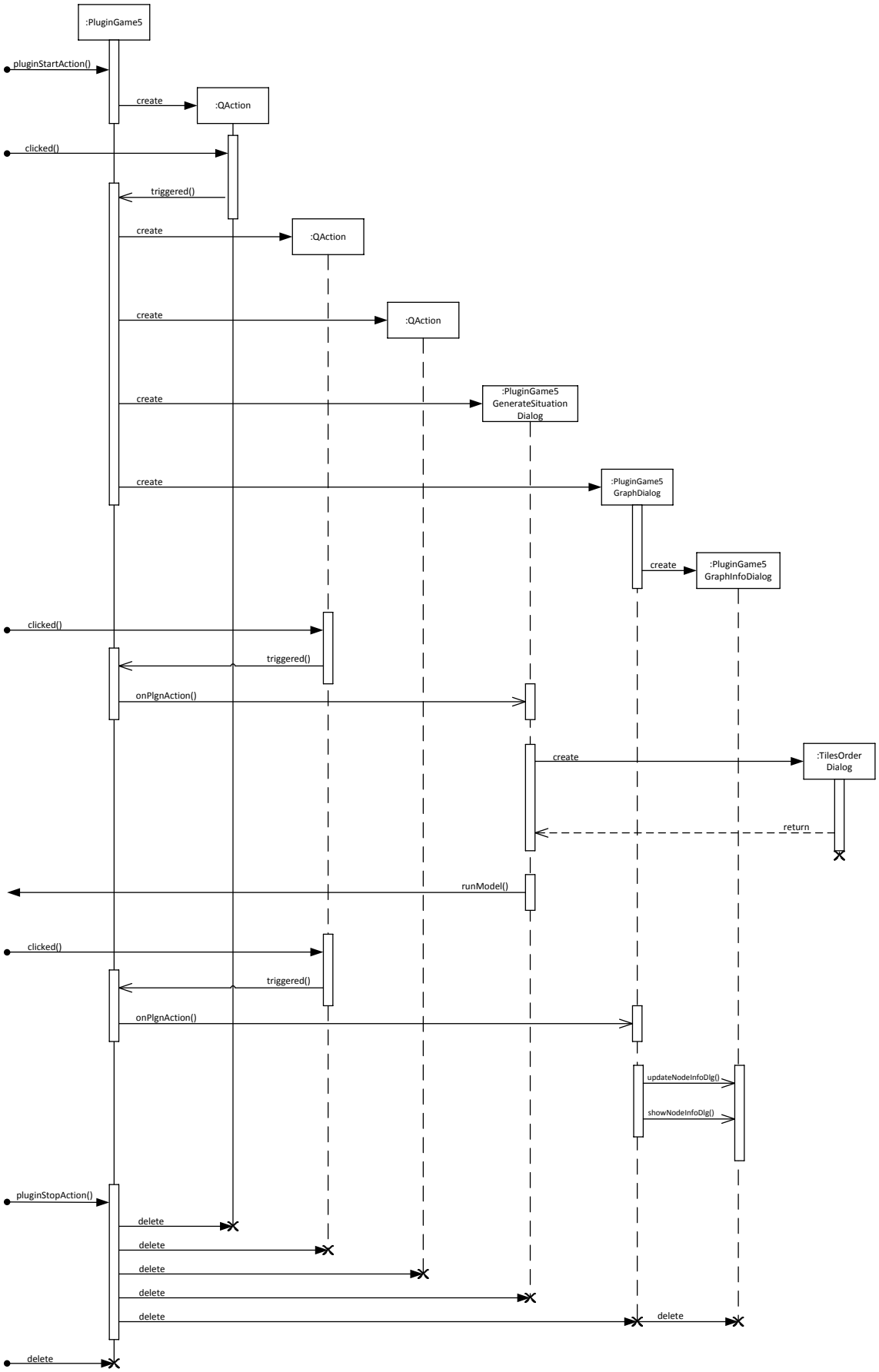
6.7. Диаграмма последовательностей вызовов диалоговых окон плагина «Пятнашки»

Для того чтобы разобраться в работе такого большого количества различных диалоговых окон необходимо рассмотреть их взаимодействии, для этого была разработана диаграмма последовательностей вызовов. На ней отображается течение времени при деятельности объекта и стрелки, показывающие выполнение действий объектами.

На **Ошибка! Источник ссылки не найден.** изображена диаграмма последовательностей, описывающая взаимодействие классов в плагине «Пятнашки» и распределение во времени: обработки действий по старту и остановке плагина, а также клипов по созданным кнопкам на панели инструментов.

Рассмотрен следующий сценарий:

- Старт плагина
- Клик по созданному меню **Плагины**
- Клик по созданной на панели инструментов кнопке **Расставить фишки**
- Задание порядка с клавиатуры при помощи специального диалогового окна. *Необязательно*
- Клип по кнопке **Ок**, окно графа откроется автоматически.
- Клик по созданной на панели инструментов кнопке **Построить граф.** *Необязательно*
- Двойной клик по вершине графа, для открытия информации о вершине графа. *Необязательно*
- Остановка плагина


:PluginGame5
pluginStartAction()
create
:QAction
clicked()
triggered()
create
:QAction
create
:QAction
create
:PluginGame5 GenerateSituation Dialog
create
:PluginGame5 GraphDialog
create
:PluginGame5 GraphInfoDialog
clicked()
triggered()
onPlgnAction()
create
:TilesOrder Dialog
return
runModel()
clicked()
triggered()
onPlgnAction()
updateNodeInfoDlg()
showNodeInfoDlg()
pluginStopAction()
delete
delete
delete
delete
delete
delete
delete

6.8. Разработка методики автоматического теста плагина «Пятнашки»

Автоматическая генерация текста модели и тесная интеграция приводит к тому, что любое изменение в грамматике языка РДО (а это в условиях командной разработки и бурного развития системы происходит часто) ведет к созданию модели, которую невозможно запустить, по причине ошибочного, с точки зрения нового синтаксиса, теста модели. Поэтому в рамках данного курсового проекта была разработано методика автотеста, диаграмма активностей которой приведена на листе 6, формат А2.

Суть непосредственно тестирования плагина в следующем:

- Преобразование относительного пути тестовой модели в абсолютный с помощью команды pwd
  `GAME5_TEST_MODEL=`pwd`/../models/test/plugin/game5_test/game5_test.rdox`
  Текст этой тестовой модель пустой при загрузке из удаленного репозитория, а эталонная трассировка соответствует модели игры «пятнашки»
- Запуск исполняемого файла rdo-studio с параметрами
  `./rdo_studio --game5_testcase="1 0 2 4 5 3" -i $GAME5_TEST_MODEL`
  Параметр -i с атрибутом $GAME5_TEST_MODEL откроет модель, абсолютный путь которой мы получили на предыдущем этапе.
  Параметр --game5_testcase с атрибутом "1 0 2 4 5 3" передаст сигнал плагину сгенерировать модель с начальной ситуацией как на Рисунок 12

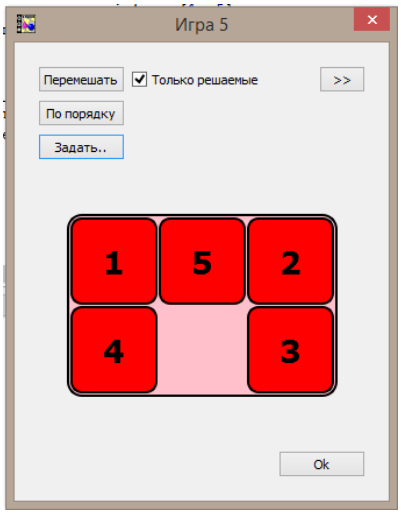


Рисунок 12

- Результатом такого запуска является открытие модели, редактирование модели, исходя из начальной ситуации, заданной в аргументе, сохранение модели и выход из исполняемого файла
- Отредактированная модель запускается в консольной версии
- Результаты прогона сравниваются с эталоном

Таким образом, в рамках теста проверяется работа плагина, а так же соответствие измененному синтаксису.

6.9. Разработка документации

Как уже было сказано, результаты разработки должны эксплуатироваться студентами кафедры «Компьютерные системы автоматизации производства» в качестве программного обеспечения для выполнения лабораторной работы, цель которой изучить механизм точек принятия решения и алгоритм поиска A*, а также разобраться с моделью и эвристиками, предложить свою. А также другие пользователи в ознакомительных и других целях.

По этой причине было разработано руководство пользователя плагином «Пятнашки», а так же методическое указание по выполнению лабораторной работы. За основу были взяты аналогичные материалы уже существующей лабораторной работы.

Руководство пользователя содержит следующие разделы:

- Общие сведения
- Загрузка плагина
- Создание новой игровой ситуации
- Запуск модели и просмотр результатов

Методическое указание содержит в себе постановку задачи и пример отчета, а также ссылается на учебные материалы, рекомендованные для подготовки к лабораторной работе.

При разработке документации был использован формат **Markdown** — облегчённый язык разметки.

Преимущества использования Markdown:

- Основан исключительно на текстовом вводе. Не нужно использовать сторонние редакторы и инструменты — только обычное текстовое поле. Таким образом, вы будете уверены, что ничего не поломается
- Минимальное количество кода. Разметка осуществляется при помощи простых тегов в тесте
- Исходный код максимально читабелен и наглядно
- Простая конвертация в популярные форматы: HTML, PDF и другие, а это значит нет необходимости поддерживать несколько версий электронной документации, достаточно выполнять конвертацию последней версии текста

- GitHub — самый крупный[5] веб-сервис для хостинга IT-проектов и их совместной разработки, на котором располагается исходный код rdo-studio, предоставляет удобный сервис хранения файлов в формате Markdown

[5] https://github.com/blog/865-github-dominates-the-forges

7. Апробирование разработанной системы для модельных условий

Апробирование разработанной системы осуществлялось при помощи многократного тестирования функционала. Выявленные в процессе тестирования ошибки и недочеты были исправлены на этапе рабочего проектирования. На этом же этапе для поддержания плагина «Пятнашки» в рабочем состоянии разработана методика автотеста, разработаны и опробованы необходимые для этого вспомогательные модули.

На листе результатов представлены снимки работающей системы с загруженным в неё плагином, а так же снимки результатов автоматического тестирования.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное исследование системы имитационного моделирования РДО и программного обеспечения для лабораторной работы в виде приложения **Игра5**
2. На этапе концептуального проектирования системы был сделан выбор общесистемной методологии проектирования, с помощью диаграммы компонентов нотации UML определена та часть системы в которой придется делать при разработке плагина. Сформулировано техническое задание
3. На этапе технического проектирования намечены необходимые для разработки классы, разработана структура плагина «Пятнашки», а также структура взаимодействия классов внутри этих элементов и их взаимодействие с системой. Предварительно разработано поведение для последующей реализации алгоритмов
4. На этапе рабочего проектирования написан программный код для реализации спроектированных ранее алгоритмов работы и архитектуры. Вновь разработанные классы показаны с помощью подробных диаграмм классов. Проведены отладка и тестирование нового функционала системы, в ходе которых исправлялись найденные ошибки, оптимизировалась работа разрабатываемой подсистемы. Разработан относительно быстрый алгоритм отрисовки графа. Реализован алгоритм тестирования разработанного плагина и показан на диаграмме активностей, а также разработан тестовая модель, для которой утверждены эталоны решения
5. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе новой функции системы
6. Была разработана документация по новому функционалу системы, а именно руководство пользователя и методические указания для выполнения лабораторной работы

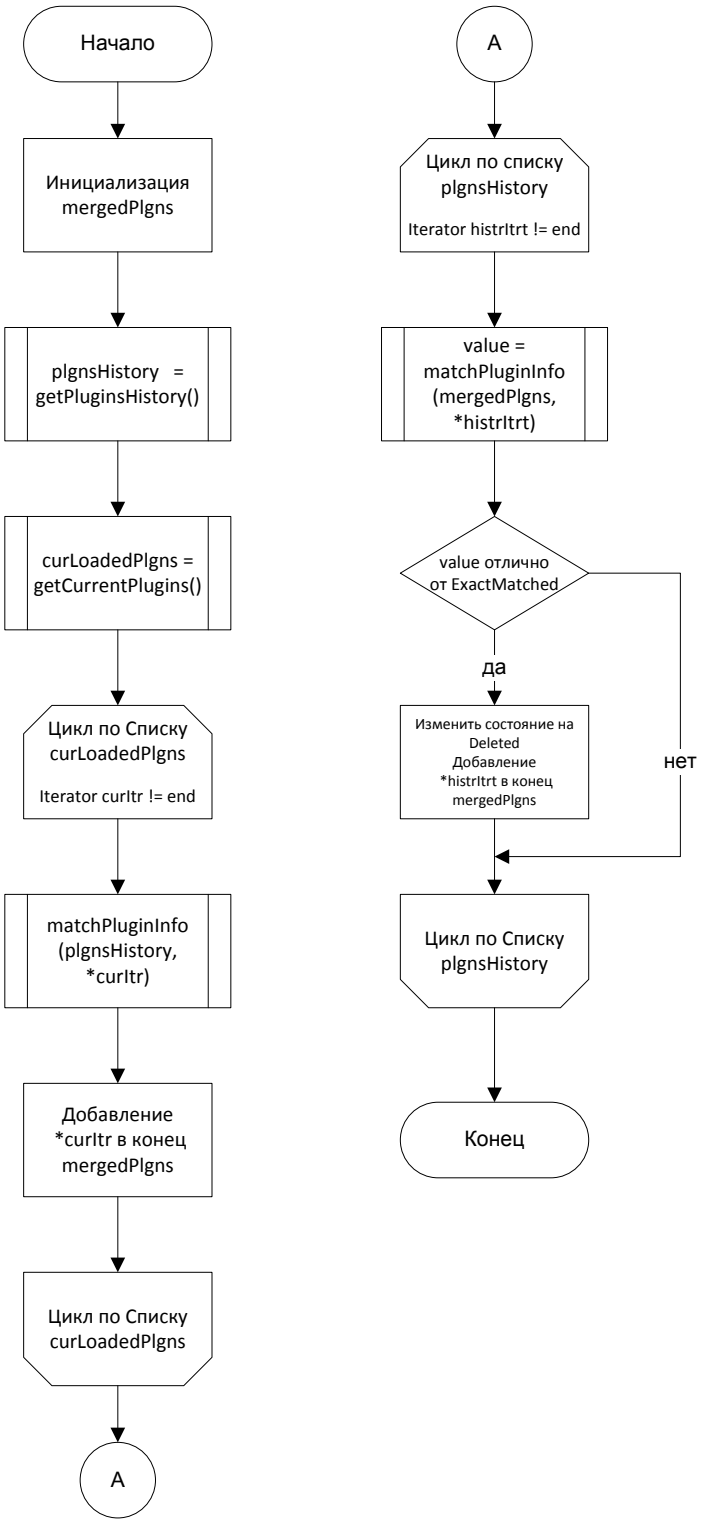
Список литературы

1. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем, язык и среда РДО. М.: МГТУ им. Н. Э. Баумана, 2009. -583с.
2. Рейтинг языков программирования по версии TIOBE [http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html/]
3. Справка по RAO-studio [http://rdo.rk9.bmstu.ru/help/]
4. Б. Страуструп. Язык программирования C++. Специальное издание / пер. с англ. – М.: ООО «Бином-Пресс», 2006. – 1104 с.: ил.
5. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с анг. Е. Матвеев – СПб.: Питер, 2010. – 464 стр.
6. Шлее М. Qt 4.8. Профессиональное программирование на C++. — СПб.: БХВ-Петербург, 2012. — 912 с.: ил.

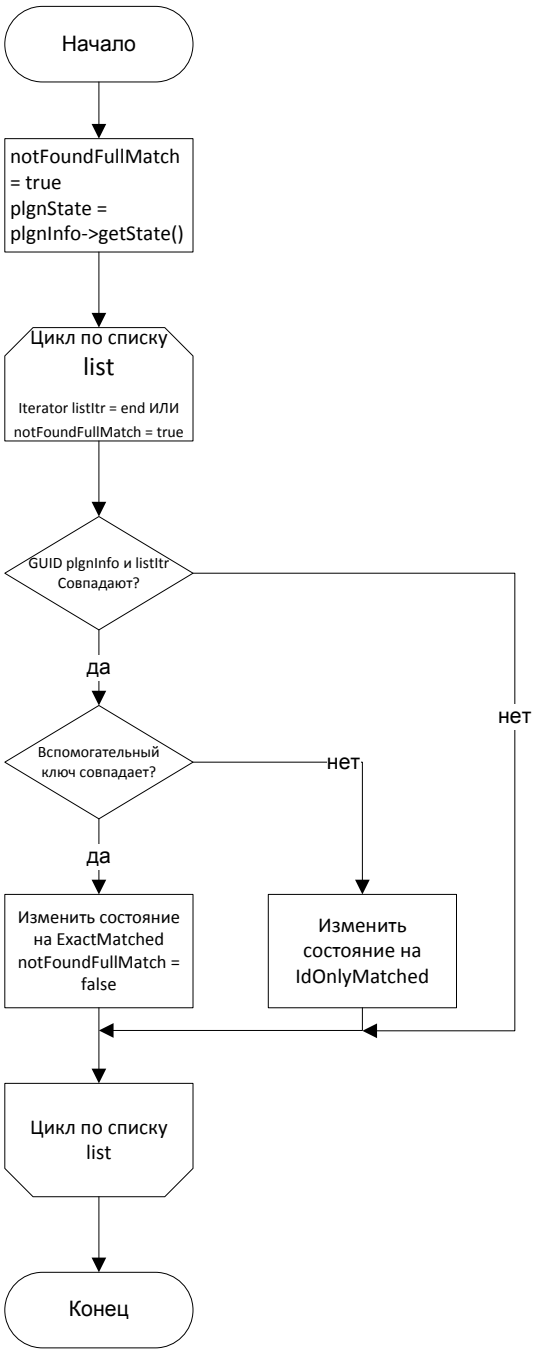
Список использованного программного обеспечения

1. RAO-Studio
2. ArgoUML
3. Microsoft®, Office Word 2003
4. Microsoft®, Office Visio 2007
5. Microsoft®, Visual Studio 2008 SP1
6. Qt Digia ©, Qt Designer 5.1.1
7. Qt Digia ©, Qt Creator 3.1.1

Приложение 1. Блок-схема алгоритма слияния списков информации о плагинах



Приложение 2. Блок-схема алгоритма функции для поиска соответствия информации о плагине в списке

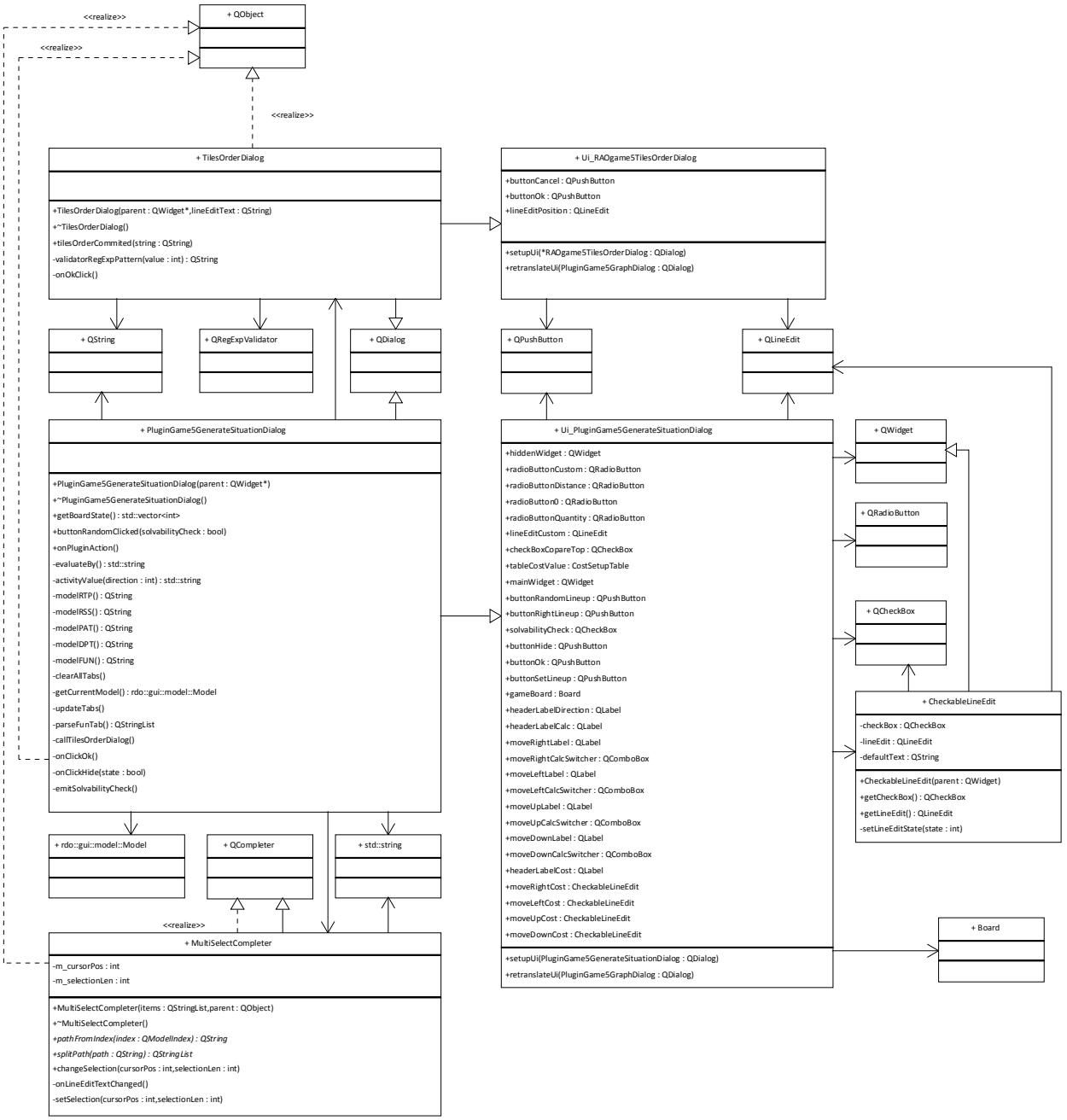


Приложение 3. Программный код, реализующий алгоритм слияния списков информации о плагинах

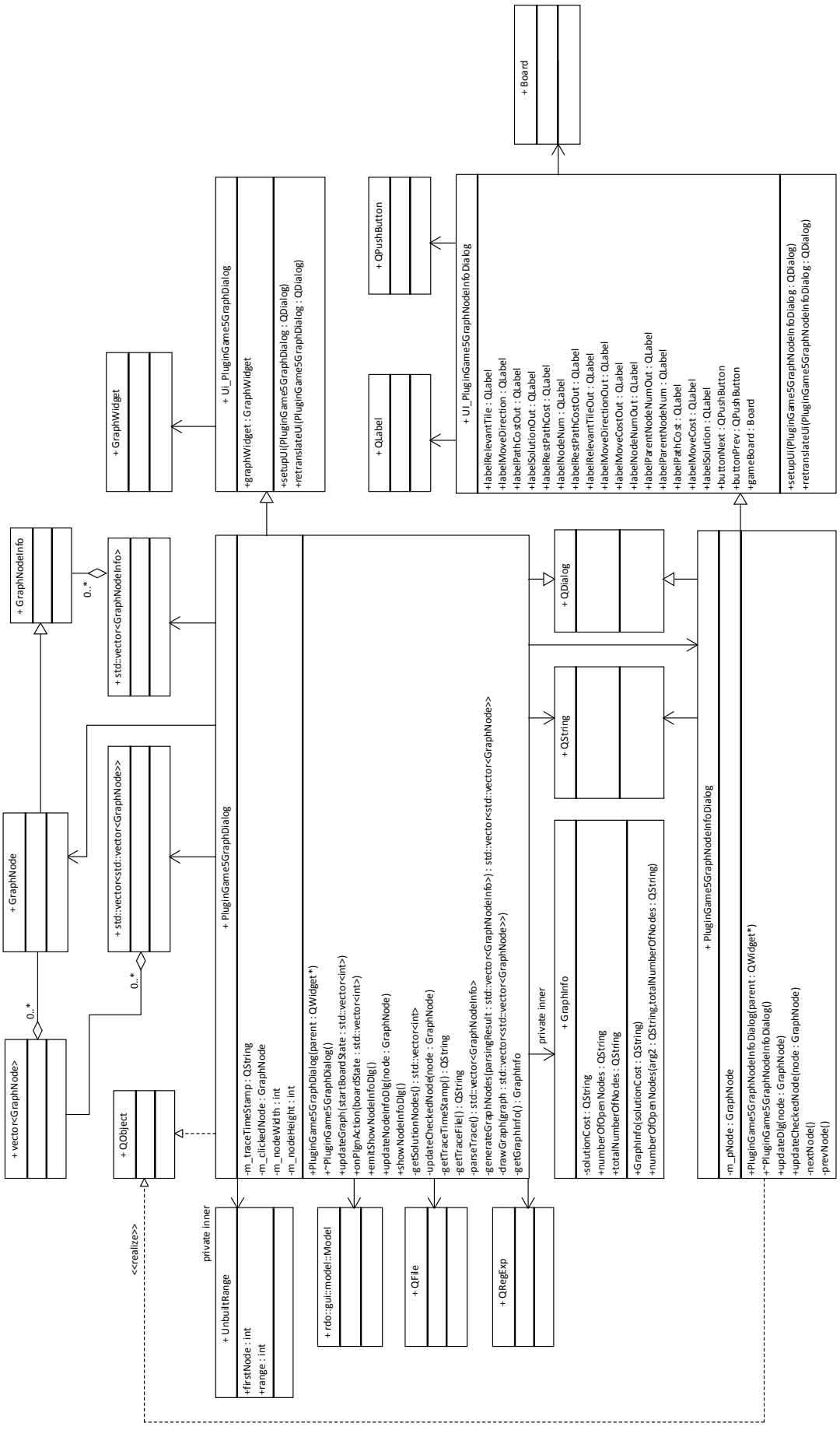
```
PluginInfoList Loader::getMergedPluginInfoList() const
{
   PluginInfoList plgnsHistory   = getPluginsHistory();
   PluginInfoList curLoadedPlgns = getCurrentPlugins();
   PluginInfoList mergedPlgns;
   for (PluginInfoList::const_iterator curItrt = curLoadedPlgns.begin();
        curItrt != curLoadedPlgns.end(); ++curItrt)
   {
      LPPluginInfo plgnInfo = *curItrt;
      matchPluginInfo(plgnsHistory, plgnInfo);
      mergedPlgns.push_back(plgnInfo);
   }
   for (PluginInfoList::const_iterator histrItrt = plgnsHistory.begin();
histrItrt != plgnsHistory.end(); ++histrItrt)
   {
      LPPluginInfo plgnInfo = *histrItrt;
      if (matchPluginInfo(mergedPlgns, plgnInfo) !=
rdo::Plugin::ExactMatched)
      {
         plgnInfo->setState(rdo::Plugin::Deleted);
         mergedPlgns.push_back(plgnInfo);
      }
   }
      return mergedPlgns;
}

int Loader::matchPluginInfo(const PluginInfoList& list, const LPPluginInfo&
                            plgnInfo) const
{
   bool notFoundFullMatch = true;
   int plgnState = plgnInfo->getState();
   for (PluginInfoList::const_iterator listItr = list.begin();
        listItr != list.end() && notFoundFullMatch; ++listItr)
   {
      if (plgnInfo->getGUID() == (*listItr)->getGUID())
      {
         if (plgnInfo->pluginSignInfoIsEqual(*(*listItr)))
         {
            notFoundFullMatch = false;
            plgnInfo->setAutoload((*listItr)->getAutoload());
            plgnState = rdo::Plugin::ExactMatched;
         }
         else
         {
            plgnState = rdo::Plugin::IdOnlyMatched;
         }
      }
   }
   plgnInfo->setState(plgnState);
   return plgnState;
}
```

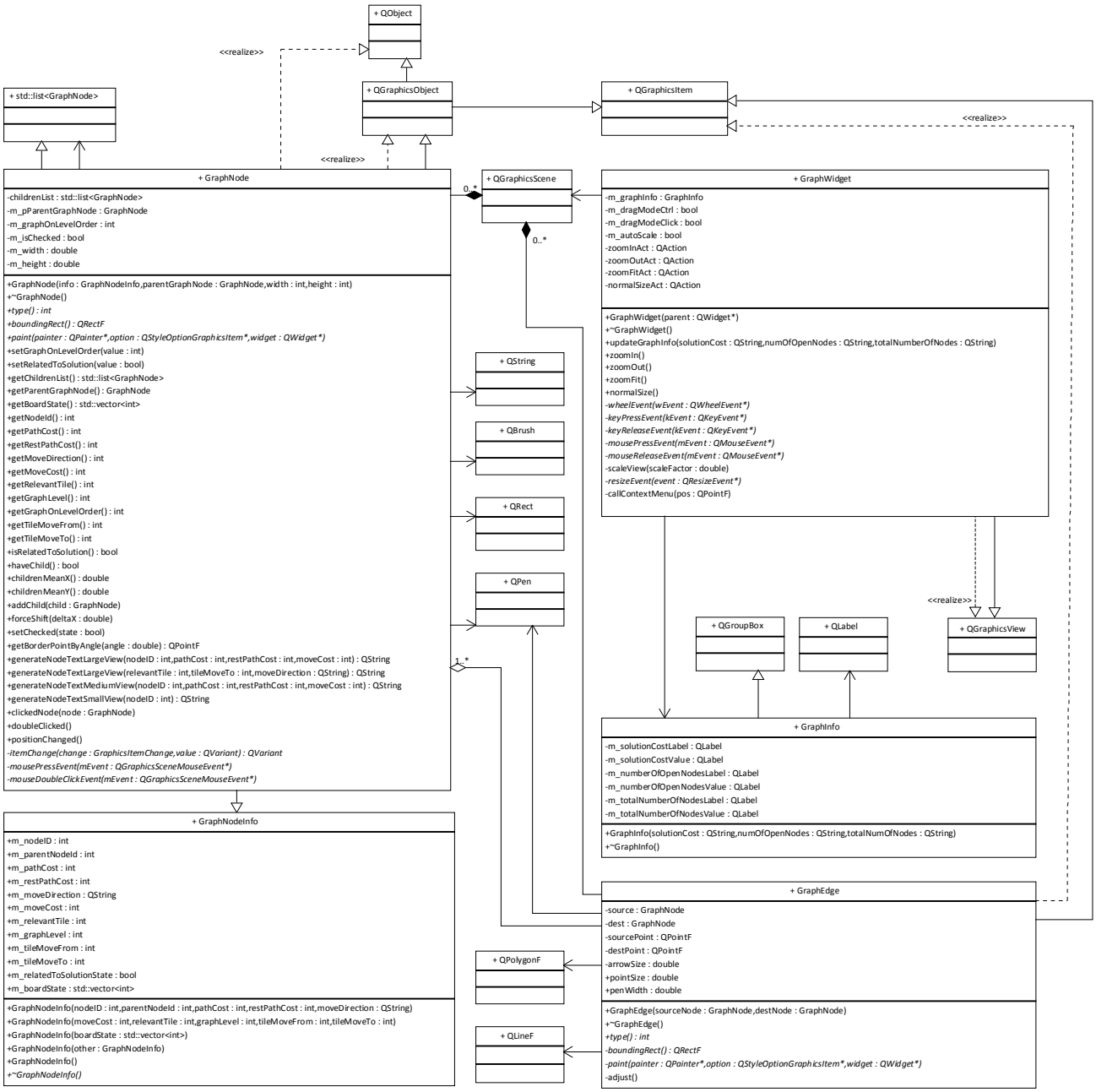
Приложение 4. Диаграмма классов PluginGame5GenerateSiationDialog



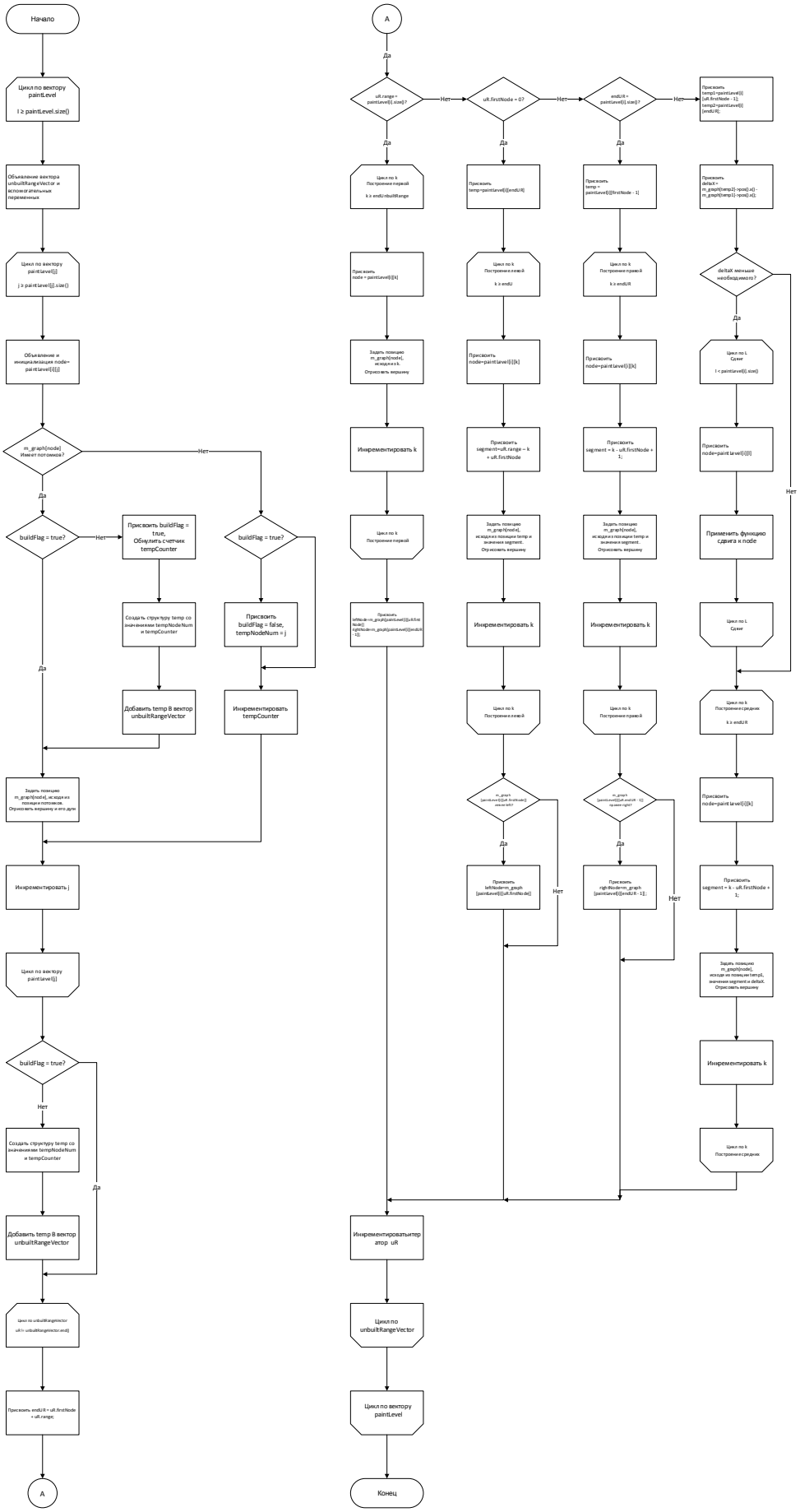
Приложение 5. Диаграмма классов PluginGame5GraphDialog



Приложение 6. Диаграмма классов GraphWidget



Приложение 7. Блок-схема алгоритма отрисовки



Приложение 8. Программный код, реализующий алгоритм отрисовки графа

```
for (int i = (int)paintLevel.size() - 1; i >= 0; i--)
{
   bool buildFlag   = true;
   int  tempNodeNum = 0;
   int  tempCounter = 0;
   std::vector<UnbuiltRange> unbuiltRangeVector;
   for (unsigned int j = 0; j < paintLevel[i].size(); j++)
   {
      int node = paintLevel[i][j];
      if (m_graph[node]->haveChild())
      {
         graphWidget->scene->addItem(m_graph[node]);
         m_graph[node]->setPos(m_graph[node]->childrenMeanX(),
                               m_graph[node]->childrenMeanY() - 40);
         BOOST_FOREACH(GraphNode* childNode,
                       m_graph[node]->getChildrenList())
         {
            graphWidget->scene->addItem(new GraphEdge(m_graph[node],
                                                      childNode));
         }
            if (!buildFlag)
         {
            buildFlag = true;
            UnbuiltRange temp = {tempNodeNum, tempCounter};
            unbuiltRangeVector.push_back(temp);
            tempCounter = 0;
         }
      }
      else
      {
         if (buildFlag)
         {
            buildFlag = false;
            tempNodeNum = j;
         }
         tempCounter++;
      }
   }
   if (!buildFlag)
   {
      buildFlag = true;
      UnbuiltRange temp = {tempNodeNum, tempCounter};
      unbuiltRangeVector.push_back(temp);
      tempCounter = 0;
   }
   BOOST_FOREACH(const UnbuiltRange& unbuiltRange, unbuiltRangeVector)
   {
      int endUnbuiltRange = unbuiltRange.firstNode + unbuiltRange.range;
      if (unbuiltRange.range == paintLevel[i].size())
      {
         for (int k = unbuiltRange.firstNode; k < endUnbuiltRange; k++)
         {
            int node = paintLevel[i][k];
               graphWidget->scene->addItem(m_graph[node]);
               m_graph[node]->setPos(40 * (k + 1), 20 +
                                     (paintLevel.size() - 1) * 40);
         }
         leftNode  = m_graph[paintLevel[i][unbuiltRange.firstNode]];
         rightNode = m_graph[paintLevel[i][endUnbuiltRange - 1]];
```

```
}
else
{
   if (unbuiltRange.firstNode == 0)
   {
      int temp  = paintLevel[i][endUnbuiltRange];
      for (int k = unbuiltRange.firstNode; k < endUnbuiltRange; k++)
      {
         int node  = paintLevel[i][k];
         int segment = unbuiltRange.range - k +
                       unbuiltRange.firstNode;

         graphWidget->scene->addItem(m_graph[node]);
         m_graph[node]->setPos(m_graph[temp]->pos().x()-40 * segment,
                               m_graph[temp]->pos().y());
      }
      if (leftNode->pos().x() >
          m_graph[paintLevel[i][unbuiltRange.firstNode]]->pos().x())
      {
         leftNode = m_graph[paintLevel[i][unbuiltRange.firstNode]];
      }
   }
   else if (endUnbuiltRange == paintLevel[i].size())
   {
      int temp = paintLevel[i][unbuiltRange.firstNode - 1];
      for (int k = unbuiltRange.firstNode; k < endUnbuiltRange; k++)
      {
         int node  = paintLevel[i][k];
         int segment = k - unbuiltRange.firstNode + 1;

         graphWidget->scene->addItem(m_graph[node]);
         m_graph[node]->setPos(m_graph[temp]->pos().x() +
                 40 * segment, m_graph[temp]->pos().y());
      }
      if (rightNode->pos().x() <
          m_graph[paintLevel[i][endUnbuiltRange - 1]]->pos().x())
      {
         rightNode = m_graph[paintLevel[i][endUnbuiltRange - 1]];
      }
   }
   else
   {
      int temp1  = paintLevel[i][unbuiltRange.firstNode - 1];
      int temp2  = paintLevel[i][endUnbuiltRange];
      double deltaX =  m_graph[temp2]->pos().x() -
                       m_graph[temp1]->pos().x();

      if (deltaX < (unbuiltRange.range + 1) * 40)
      {
         for (unsigned int l = endUnbuiltRange;
              l < paintLevel[i].size(); l++)
         {
            int node    = paintLevel[i][l];
            m_graph[node]->forceShift((unbuiltRange.range + 1) *
                                      40 - deltaX);
         }
         deltaX = m_graph[temp2]->pos().x() -
                  m_graph[temp1]->pos().x();
      }

      for (int k = unbuiltRange.firstNode; k < endUnbuiltRange; k++)
      {
         int node    = paintLevel[i][k];
         int segment = k - unbuiltRange.firstNode + 1;
```

```
                graphWidget->scene->addItem(m_graph[node]);
                m_graph[node]->setPos(m_graph[temp1]->pos().x() + segment *
                  deltaX/(unbuiltRange.range+1), m_graph[temp1]->pos().y());
            }
        }
    }
}
```