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 лабораторной работы по курсу Моделирование технологических и производственных процессов

Студент группы РК9-92 _____ А. О. Чернов
(Подпись, дата) (И.О.Фамилия)

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

Москва, 2013

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 Технология КИП

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

Студент Чернов А. О., РК9-92
(Фамилия, инициалы, индекс группы)

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

1. Техническое задание

Разработать и внедрить в рабочую версию RAO-studio подсистему загрузки плагинов, Разработать плагин «Пятнашки» и добиться его интеграции в рабочую версию RAO-studio.

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на 47 листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.)

1 лист А1 – Постановка задачи; 2 лист А1 –Диаграмма классов подсистемы плагинов; 3 лист А3 – Диаграмма состояния процедуры слияния информации о плагинах; 4 лист А3 – Диаграмма классов плагина пятнашек; 5 лист А2 – Диаграмма последовательностей плагина пятнашек; 6 лист А1 –Блок-схема алгоритма отрисовки графа; 7 лист А1 – Результаты разработки.

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 3
2. Предпроектное исследование 5
3. Концептуальный этап проектирования 9
3.1. Выбор общесистемной методологии проектирования 9
3.2. Выделение системы из среды 10
3.3. Диаграмма компонентов 11
4. Формирование технического задания 13
4.1. Техническое задание на подсистему загрузки плагинов 13
4.1.1. Введение 13
4.1.2. Общие сведения 13
4.1.3. Назначение разработки 13
4.1.4. Требования к программе или программному изделию 14
4.1.5. Стадии и этапы разработки 15
4.1.6. Порядок контроля и приемки 16
4.2. Техническое задание на плагин «Пятнашки» 16
4.2.1. Введение 16
4.2.2. Общие сведения 16
4.2.3. Назначение разработки 17
4.2.4. Требования к программе или программному изделию 17
4.2.5. Стадии и этапы разработки 19
4.2.6. Порядок контроля и приемки 19
5. Технический этап проектирования 20
5.1. Технический этап проектирования подсистемы загрузки плагинов 20
5.1.1. Разработка архитектуры подсистемы загрузки плагинов 20
5.1.2. Разработка механизма загрузки плагинов 20
5.2. Технический этап проектирования плагина «Пятнашки» 22
5.2.1. Разработка архитектуры плагина «Пятнашки» 22
5.2.2. Разработка архитектуры класса диалога задания начальной ситуации 23
5.2.3. Разработка архитектуры класса диалога построения графа 24
6. Рабочий этап проектирование 26

6.1. Рабочий этап проектирования подсистемы загрузки плагинов 26
6.1.1. Структура подсистемы загрузки плагинов 26
6.1.2. Алгоритм слияния списков плагинов 27
6.2. Рабочий этап проектирования плагина «Пятнашки» 29
6.2.1. Структура плагина «Пятнашки» 29
6.2.2. Структура класса диалогового окна создания начальной ситуации 30
6.2.3. Структура игрового поля 31
6.2.4. Структура класса диалогового окна построения графа 32
6.2.5. Структура класса GraphWidget 33
6.2.6. Алгоритм отрисовки графа 34
6.2.7. Диаграмма последовательностей вызовов диалоговых окон плагина «Пятнашки» 35
7. Апробирование разработанной системы для модельных условий 39
8. Заключение 40
Список литературы 41
Список использованного программного обеспечения 41
Приложение 1. Диаграмма классов подсистемы плагинов 42
Приложение 2. Программный код, реализующий алгоритм слияния списков информации о плагинах. 43
Приложение 3. Диаграмма классов PluginGame5GenerateSiationDialog . 44
Приложение 4. Программный код, реализующий алгоритм отрисовки графа. 45

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 – изучить механизм точек принятия решения и алгоритм поиска A*, а также изучить модель, разобраться с эвристиками, 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, занести результаты в таблицу, сравнить их с предыдущими и 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 рисунке 1.

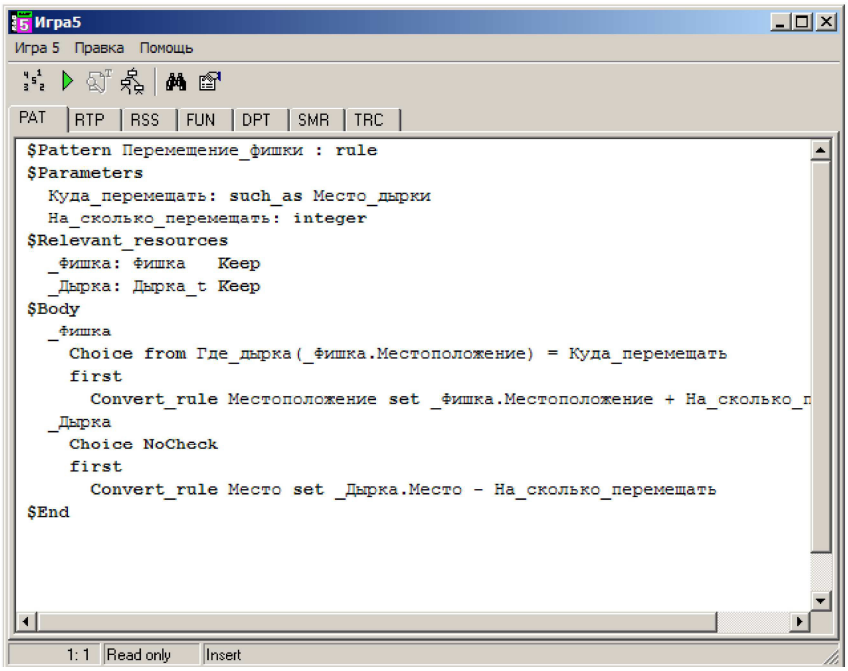


Рисунок 1

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 рисунке 2 представлено развернутое окно настроек.

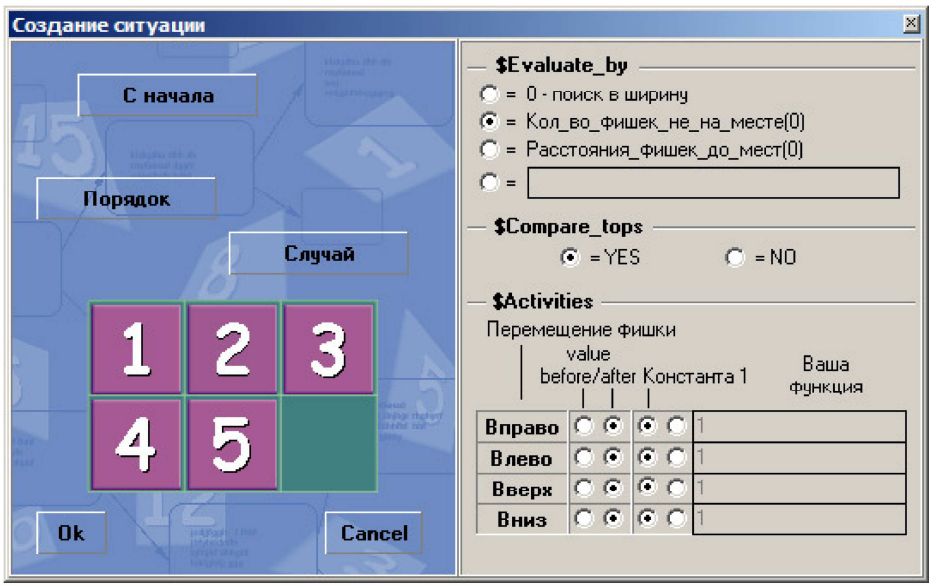


Рисунок 2

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 графоми 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]

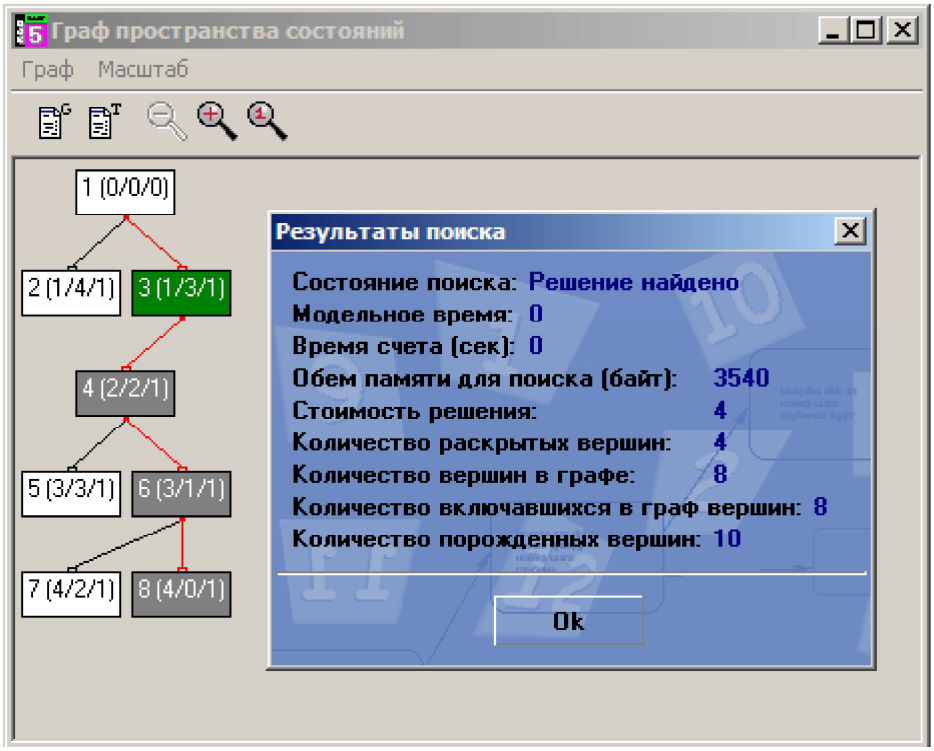


Рисунок 3

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. plug-in, от plug in «подключать»), — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей. Итак, задача разбивается на две: разработать подсистему плагинов и подключаемое расширение «Пятнашки».

3. Концептуальный этап проектирования

3.1. Выбор общесистемной методологии проектирования

Проектирование любой системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой. Одна из них была выявлена при предпроектном исследовании и заключается в отсутствии подсистемы плагинов. Проблема 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, позволяет производить разработку поэтапно и разделить разработку плагина и подсистемы, необходимой для его (и других плагинов) загрузки.

Как итог, необходимо разработать модуль загрузки плагинов и интерфейс, необходимый для загрузки и который будет связующим звеном загружаемого расширения и системы в целом.

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. Выделение системы из среды

Исходя их задачи выдвигаются требования к подсистеме:

- 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, также необходимо создать класс, содержащий всю информацию о плагине. Подсистема должна хранить список, состоящий из этих классов.

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

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

- идентификация с помощью GUID[3]

3.3. Диаграмма компонентов

На рисунке 4 представлена «to be» диаграмма компонентов подсистемы, отвечающей за формирование исполняемого файла RAO-studio.

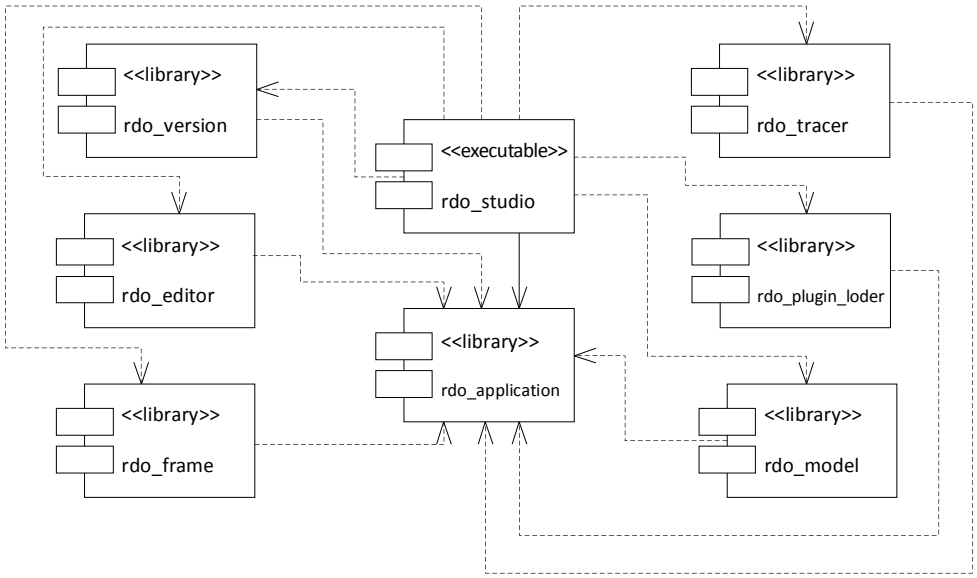


Рисунок 4

Базовый функционал этих пакетов:

rdo_studio – подсистема верхнего уровня

rdo_appliction – 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. **Разрабатываемая подсистема**

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

4. Формирование технического задания

4.1. Техническое задание на подсистему загрузки плагинов

4.1.1. 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.

Для эффективного использования широкого функционала RAO-studio ей необходима подсистема загрузки плагинов – загружаемых расширений.

4.1.2. Общие сведения

- Полное наименование темы разработки: «Добавление в систему дискретного имитационного моделирования РДО подсистемы загрузки плагинов»
- 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: 19 декабря 2012г.

4.1.3. Назначение разработки

Разработать и внедрить в рабочую версию RAO-studio подсистему загрузки плагинов.

Функциональным назначением объекта является предоставление разработчику инструмента для проектирования плагинов с возможностью последующей загрузки в систему РДО, а пользователю возможность

эффективно пользоваться возможностями RAO-studio при помощи загружаемых расширений.

Воспользоваться разработкой должны разработчики плагинов и их (плагинов) пользователи.

4.1.4. Требования к программе или программному изделию

Требования к функциональным характеристикам:

- 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

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 не выявленных ошибок не более 1 на 1000 сеансов работы с программой

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];
- 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

4.1.5. 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 программной документации;
- испытания программы;

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

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

Ubuntu является зарегистрированной торговой маркой Canonical Ltd.

Названия реальных компаний и продуктов, упомянутых в данной пояснительной записке, могут быть торговыми марками соответствующих владельцев.

4.1.6. Порядок контроля и приемки

Контроль и приемка работоспособности подсистемы загрузки плагинов должны осуществляться в процессе проверки функциональности (апробирования) системы имитационного моделирования на примере плагина «Пятнашки» в соответствии с требованиями к функциональным характеристикам системы.

4.2. Техническое задание на плагин «Пятнашки»

4.2.1. 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

4.2.2. Общие сведения

- Полное наименование темы разработки: «Загружаемое расширение для системы дискретного имитационного моделирования РДО «Пятнашки»»
- 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: 19 декабря 2013г.

4.2.3. 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.

Изменения должны эксплуатироваться студентами кафедры «Компьютерные системы автоматизации производства» как программным обеспечением 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

4.2.4. 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 или автоматически в случайном, заданном с клавиатуры и 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 не выявленных 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;
- 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

4.2.5. 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 программной документации;
- испытания программы;

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

4.2.6. 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.

5. Технический этап проектирования

5.1. Технический этап проектирования подсистемы загрузки плагинов

5.1.1. Разработка архитектуры подсистемы загрузки плагинов

На этапе концептуального проектирования было принято, что для разрешения всех требований к системе, необходимо создать класс, содержащий всю информацию о плагине. Класс информации назовем PluginInfo, подсистема должна хранить список, состоящий из этих классов.

Следовательно, нужно создать класс, состоящий из PluginInfo, это и будет базовый класс подсистемы загрузки плагинов, назовем его Plugin::Loader.

Кроме того на этапе концептуального проектирования было установлено, что все загружаемые расширения должны реализовывать один и тот же интерфейс, который используется при загрузке плагина, этот интерфейс используется в подсистеме загрузки, поэтому логично описать его в ней же. Назовем его PluginInterface. Интерфейсный класс мало что делает – если бы делал, то не был бы интерфейсным классом. Он просто приспосабливает внешнее представление некоторых услуг к местным требованиям[4]

На рисунке 5 представлена диаграмма классов, показывающая структуру подсистемы на техническом этапе проектирования.

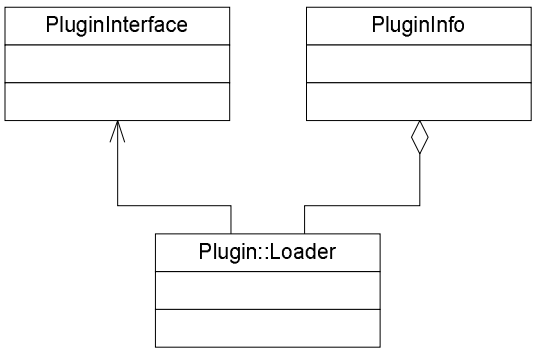


Рисунок 5

5.1.2. Разработка механизма загрузки плагинов

Определившись со структурой подсистемы, необходимо разработать принцип, по которому плагины будут загружаться в систему. Библиотека Qt, используемая в РДО, дает такой механизм, следовательно, наш класс Plugin::Loader должен использовать QPluginLoader, подробнее это будет рассмотрено при рабочем проектировании. На этапе технического

проектирования выберем, что QPluginLoader будет рекурсивно пытаться загрузить все плагины из всех подпапок папки **Plugins**. Это удобно, ведь добавляется возможность сортировки плагинов по папкам при большом количестве расширений. Исходя из задачи возможности автозапуска плагина, возникает необходимость идентификация, причем, возникает необходимость уникальности, чтобы не допустить автозагрузки неизвестного плагина. Учитывая то, что они должны иметь GUID, по нему и будем идентифицировать плагины. Хотя GUID и обладает высокой уникальностью, это не защищает от его механического повторения, поэтому остальные обязательные поля класса PluginInfo будут вспомогательным ключом при идентификации плагина.

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

Возникает необходимость в отслеживании состояния плагина в системе, то есть класс PluginInfo должен хранить такую информацию.

Исходя из совокупности задач:

- обнаружение, загрузка, сбор информации, "выгрузка" плагинов при запуске РДО
- хранение информации о параметрах автозапуска для плагинов
- возможность удаления информации о плагине
- возможность удаление плагина с диска

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. Необходимо разработать алгоритм слияния указанных выше списков. На этапе технического проектирования разработан принцип слияния, реализованный алгоритм будет рассмотрен на этапе рабочего проектирования. На рисунке 6 изображена диаграмма состояний изменения информации о плагине.

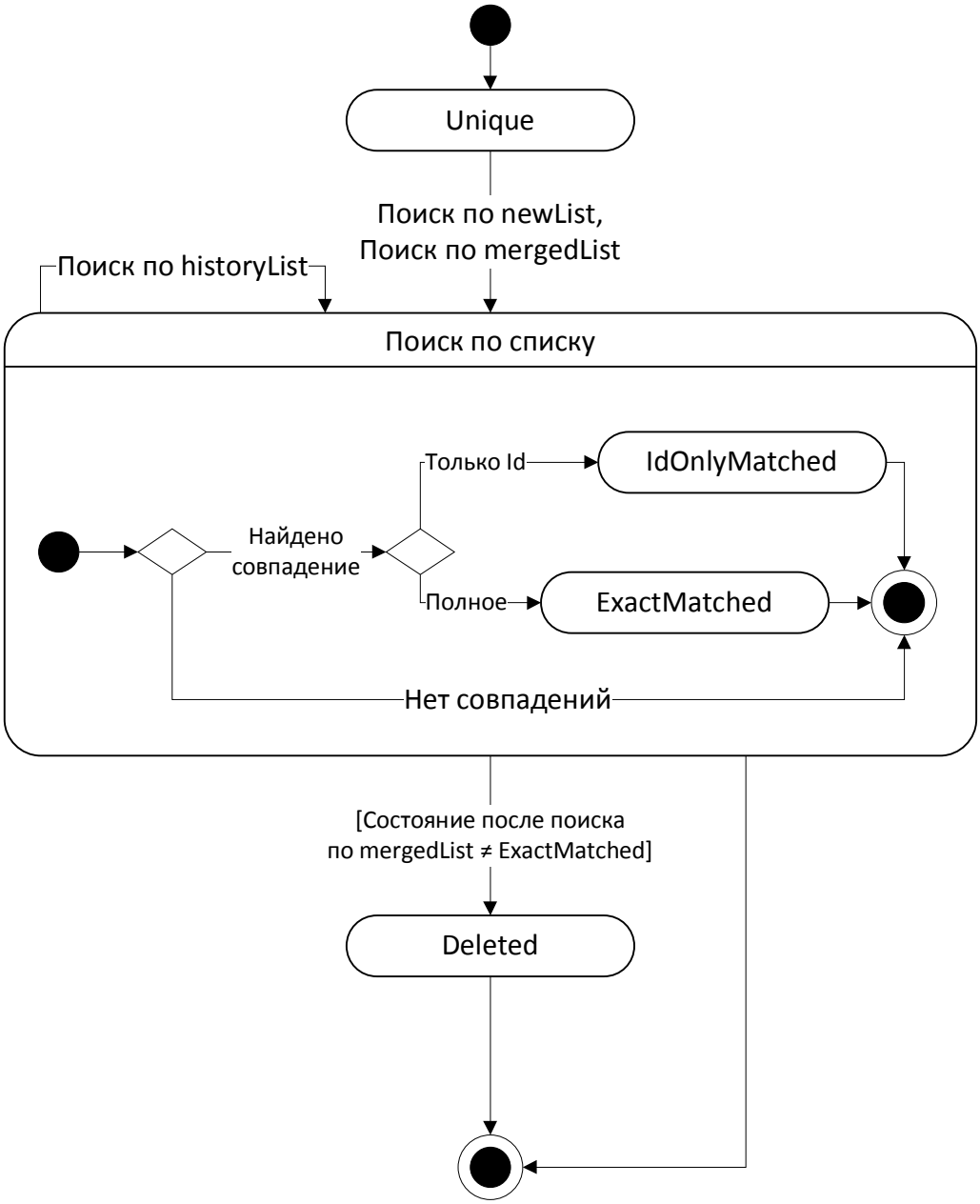


Рисунок 6

5.2. Технический этап проектирования плагина «Пятнашки»

5.2.1. Разработка архитектуры плагина «Пятнашки»

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

- Окно создания начальной ситуации
- Окно вывода

На рисунке 7 изображена упрощенная диаграмма классов, показывающая структуру плагина, разработанную на этапе технического проектирования.

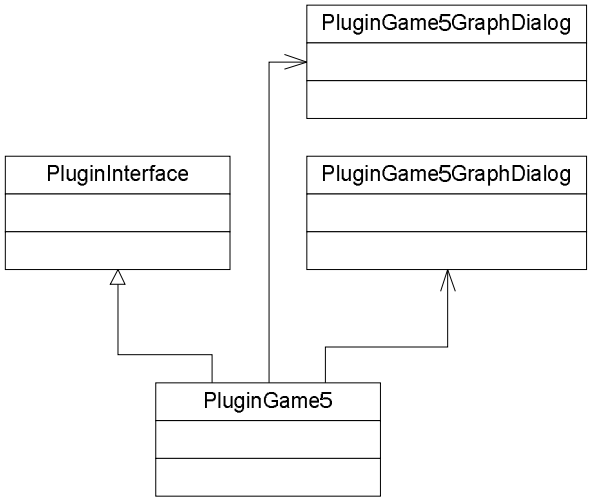


Рисунок 7

5.2.2. Разработка архитектуры класса диалога задания начальной ситуации

Для реализации требуемого функционала, а именно:

- Возможность задать положение игровых фишек: вручную, а так же автоматически в случайном, заданном с клавиатуры и 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 рисунке 8.

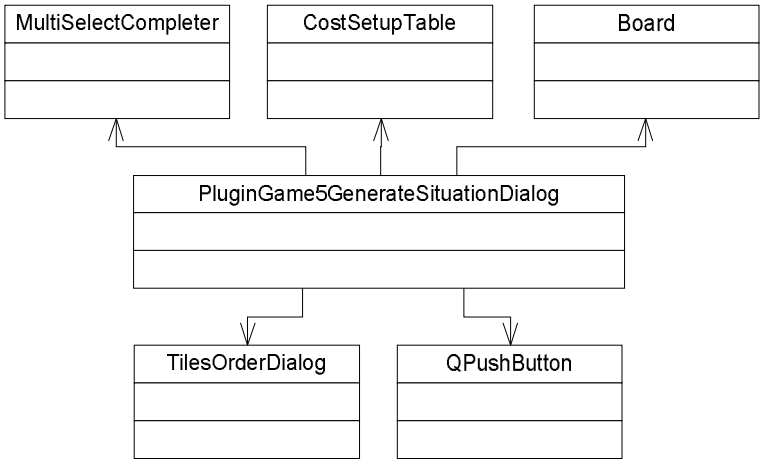


Рисунок 8

MultiSlectCompleter – класс, добавляющий в плагин автозаполнение

CostSetupTable – класс, наследуемый от таблицы, и расширяющий её возможности выпадающими списками и возможностью выставить в клетке таблицы флажок, необходимыми для настройки момента сложения стоимости применения правила и её величины.

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

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

Для расстановки в случайном и правильном порядке достаточно по кнопке, воспользуемся предоставляемыми библиотекой Qt классами **QPushButton**

Подробнее эта структура будет рассмотрена на этапе рабочего проектирования.

5.2.3. Разработка архитектуры класса диалога 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 рисунке 9.

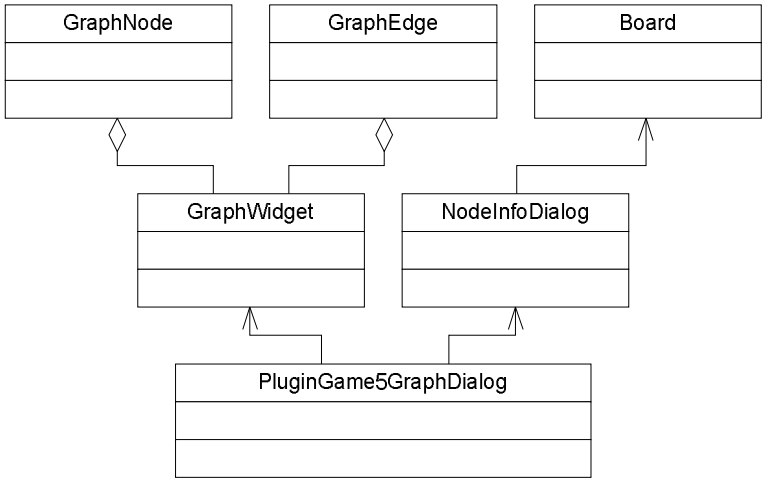


Рисунок 9

GraphWidget – класс, реализующий отрисовку на экране

GraphNode – класс вершины графа

GraphEdge – класс грани графи

NodeInfoDialog – класс, реализующий диалог 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 один атрибут, 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. Рабочий этап проектирования подсистемы загрузки плагинов

6.1.1. Структура подсистемы загрузки плагинов

Диаграмма классов подсистемы плагинов приведена в приложении 1

По этой диаграмме видно, что для класса информации о плагине **PluginInfo** заведена фабрика rdo::Factory, то есть мы можем проинициализировать умные указатели на объекты это класса, это сделано из-за необходимости передавать в качестве атрибутов функций указатели или как в нашем случае умные указатели, что безопаснее.

Так же фабрика была создана для класса **PluginInfoList**, который наследуется от класса std::list<LPPluginInfo>. Этот шаблонный класс, в свою очередь, просто список, из стандартной библиотеки, умных указателей. Выбор пал на этот тип контейнеров из-за быстрого добавления, удаления элементов списка, медленный у данного типа контейнеров поиск перебором не используется в используемом алгоритме работы, ведь в системе хранится умные указатели. Класс **PluginInfoList является** необходим для удобного хранения информации о плагинах.

Экземпляры классов **QSettings, QStringList, QString, QUuid** используются для загрузки сохраненный в системе списка-истории плагинов и формирование из него **PluginInfoList.**

Экземпляр класса **QDir** используется подсистемой плагинов во время рекурсивного поиска по подпапкам директории **plugins,** для поиска расположенных там загружаемых расширений с последующей загрузкой при помощи экземпляра класса **QPluginLoader.** Плагин удается загрузить, он только если он реализует интерфейсный класс PluginInterface.

На диаграмме классов также видно, что класс rdo::Plugin::Loader используется в качестве атрибута класса **Application**[5], что показывает взаимодействие разрабатываемой подсистемы плагинов и остальной системы.

Для отображения информации оплагинах, а также управлении ими в диалоговое окно настроек, реализуемое классом **ViewPreference** добавлена закладка плагины, через класс **Ui_ViewPreference**, реализующий пользовательский интерфейс этого диалога.

[5] На диаграмме у классов Application, ViewPreference и Ui_ViewPreference указаны только те методы иатрибуты, которые подверглись изменениям

6.1.2. Алгоритм слияния списков плагинов

На этапе технического проектирования был сделан вывод о необходимости алгоритма слияния списка информации о вновь загруженных с жесткого диска плагинах и восстановленный, сохраненный в историю, список информации, содержащий данные об автозагрузке. На этапе технического проектирования разработан принцип слияния, блок-схема реализованного алгоритма изображена на рисунке 10.

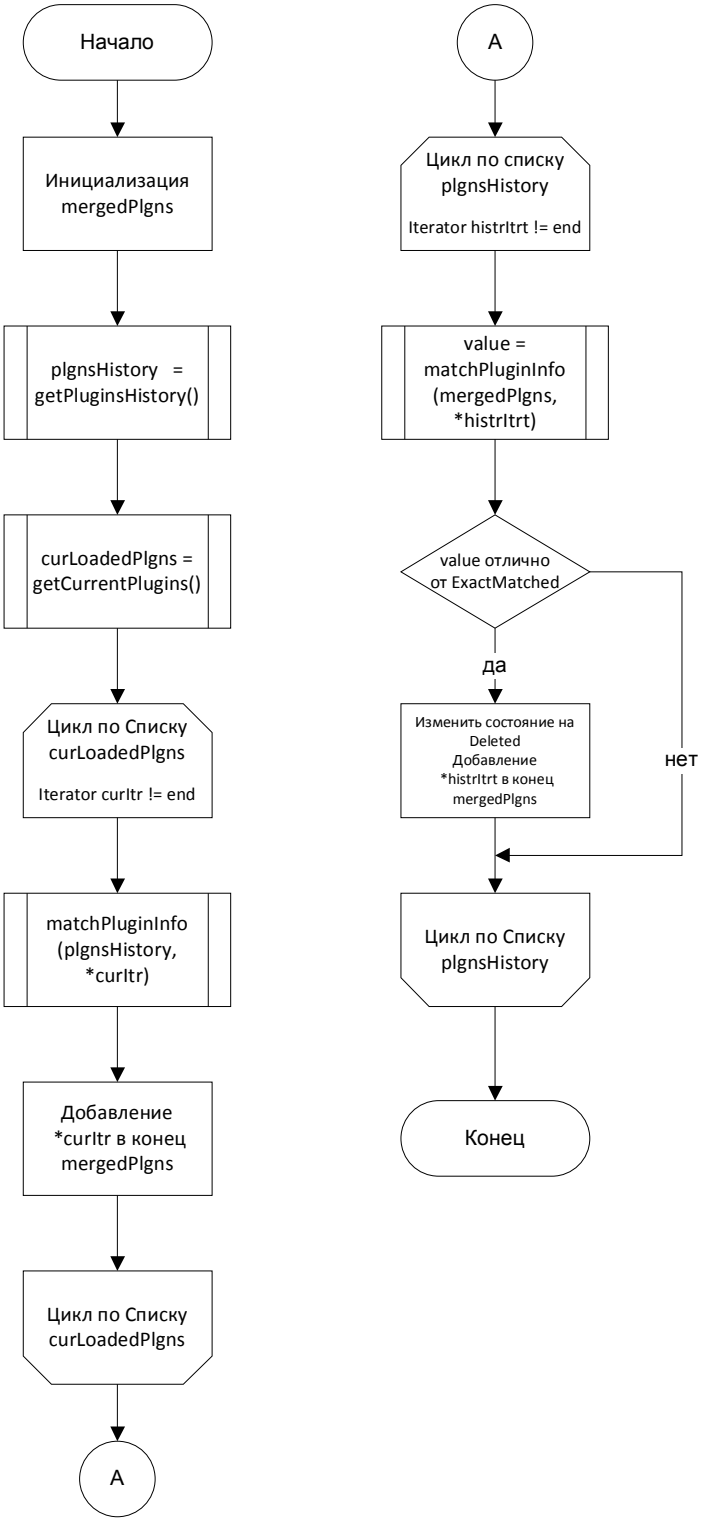


Рисунок 10

Внутренние методы **getCurrentPlugins()** и **getPluginsHistory()** возвращают вышеописанные списки.

Алгоритм 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, представлен на рисунке 11.

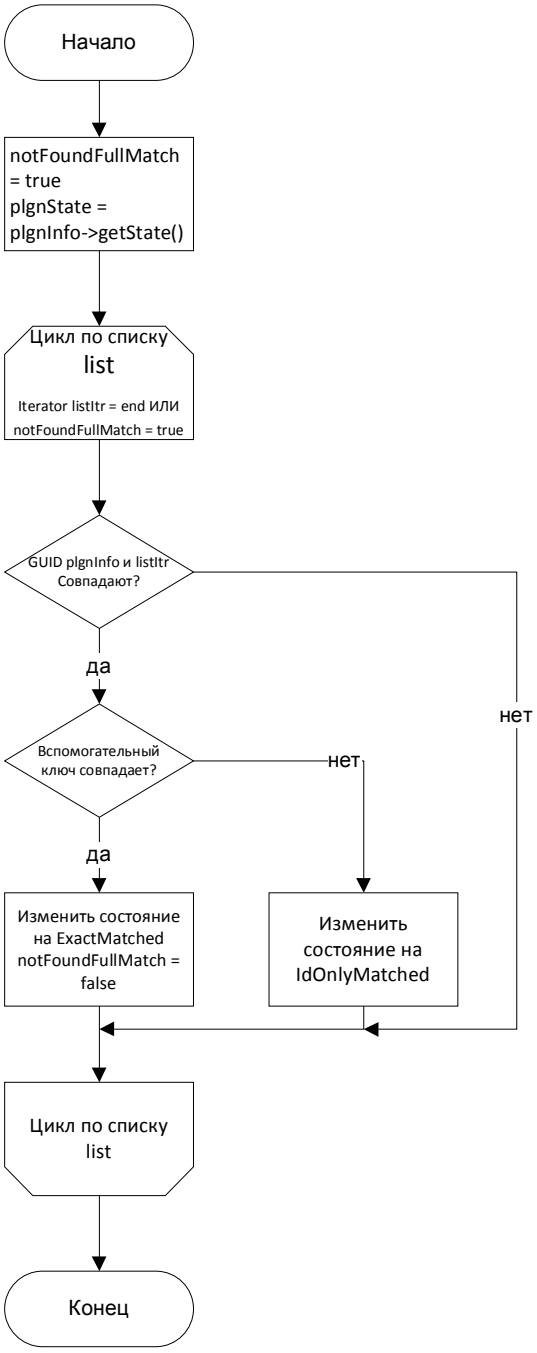


Рисунок 11

Программный код, реализующий приведенные алгоритмы, приведен в приложении 2.

6.2. Рабочий этап проектирования плагина «Пятнашки»

6.2.1. Структура плагина «Пятнашки»

Диаграмма классов загружаемого расширения приведена на рисунке 12.

На ней видно, что основной класс плагина «Пятнашки» наследуется от **PluginInterface**, и реализует методы этого интерфейсного класса, эта необходимасть 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 диграммах класса

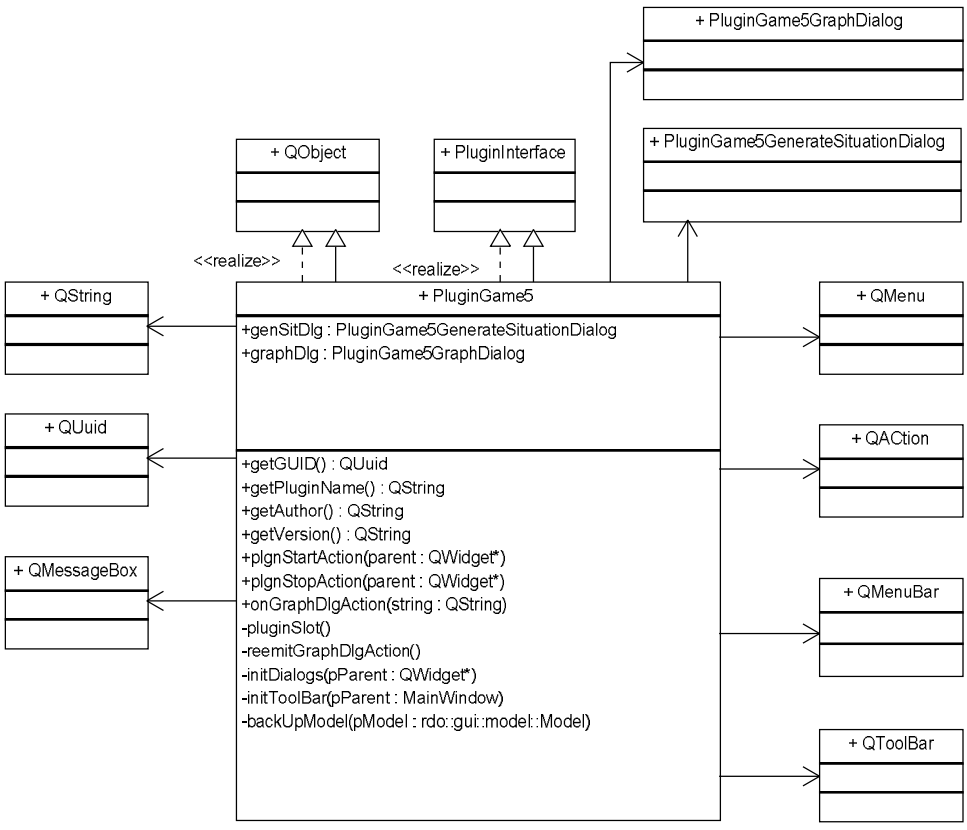


Рисунок 12

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** соответсвенно.

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

6.2.2. Структура класса диалогового окна создания начальной ситуации

Диаграмма классов представлена в приложении 3, она получена уточнением и подробным описанием структуры полученной на этапе технического проектирования. Как было сказано на этом этапе. Классы **Board, TileOrderDialog, CostSetupTable, MultiSlectCompleter** используются для реализации требуемого функционала, а именно:

- Возможность задать положение игровых фишек: вручную, а так же автоматически в случайном, заданном с клавиатуры и 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 до требуемых, а именно автозаполнение текста в тои числе при написании арифметических функций из слов, доступных для автозаполнения.

TileOrderDialog наследуется от класса **QDialog** и класса пользовательского интерфейса **Ui_ TileOrderDialog,** который имеет простую структуру: две кнопки **QPushButton** и однострочный редактор для ввода **QLineEdit**

Board и **TileOrderDialog** принадлежат классу **Ui_PluginGame5GenerateSituationDialog,** от которого наследуется класс **PluginGame5GenerateSituationDialog.** 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, которая владеет текстовым редактором и имеет сигнал управления стартом прогона.

6.2.3. Структура игрового поля

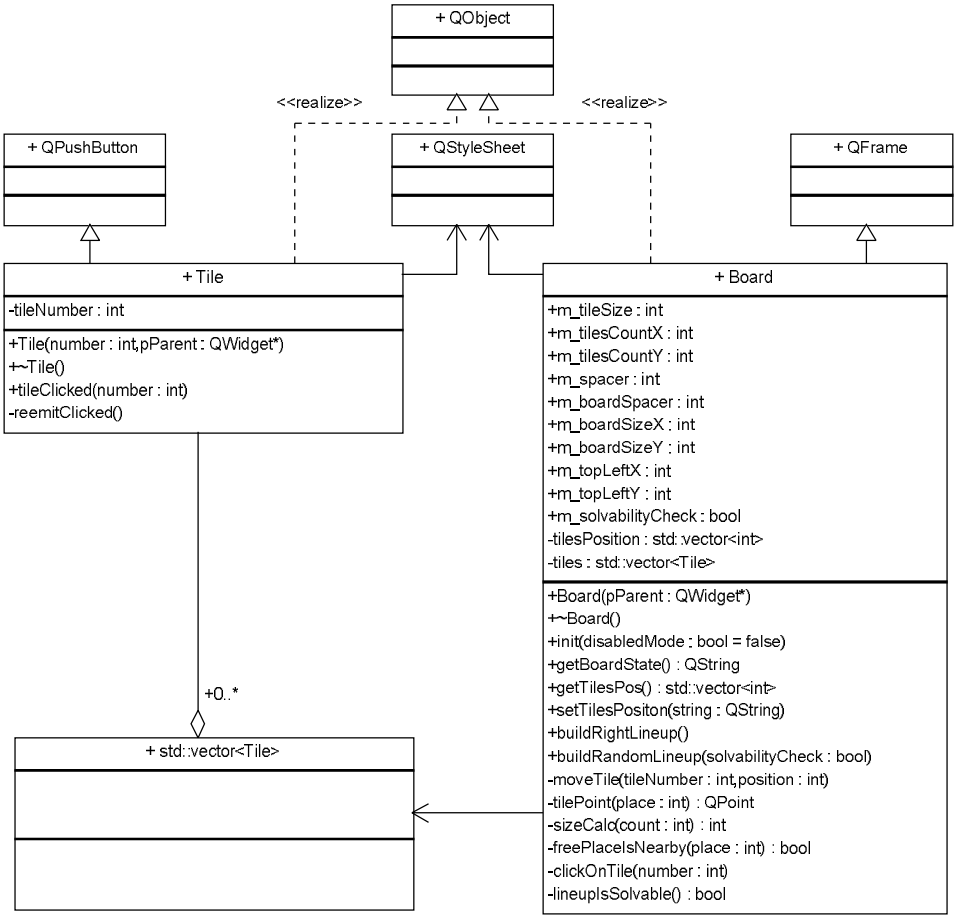


Рисунок 13

На рисунке 13 – диаграмма классов изображающая структуру игрового поля.

Класс **Board** наследуется от класса QFrame, что есть, по сути, QWidget с рамкой. Класс параметризован, вводя для разработчика удобный способ изменять игровое поле.

Board владеет экземпляром шаблонного класса **std::vector<Tile>,** который является динамический массив произвольного доступа с автоматическим изменением размера при добавлении/удалении элемента, в данном случае экземпляра класса **Tile,** фишки игрового поля, наследуемого от кнопки **QPushButton.**

Board и **Tile** реализуют свой вид с помощью класса **QStyleSheet**

6.2.4. Структура класса диалогового окна построения графа

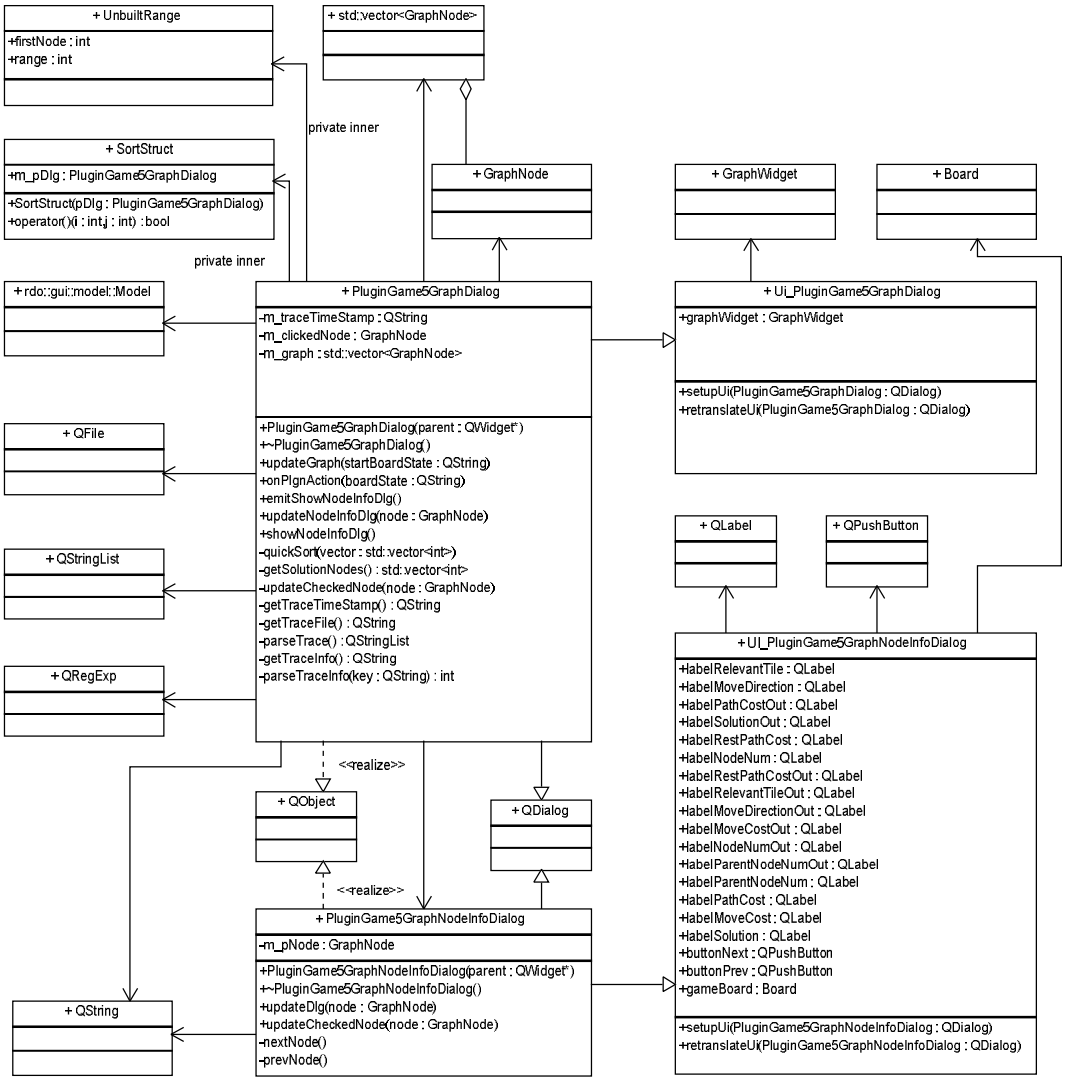


Рисунок 14

На рисунке 14 представлена диаграмма классов, показывающая структуру окна построения графа.

Для реализации основной задачи – отрисовки графа, не достаточно того, что родитель класса **PluginGame5GraphDialog Ui_PluginGame5GraphDialog,** который является реализацией графического интерфейса пользователя, владеет экземпляром класса **GraphWidget.** Необходимо получить информацию, которую в последствии можно будет вывести. Она получается непосредственно из файла трейсера РДО-студии, для этого в классе используетсяэкземпляр класса **QFile,** необходимое расположение файла на диске получают из класса **rdo::gui::model::Model,** который упоминался ранее.

Нужная информация “вырывается” из текста 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.

Помимо данных для отобрания на графе, необходима сортировка вершин для его корректного отображения. Для этого созданы внутренние структуры **UnbuildRange** и **SortStruct.** Первая нужна для удобного хранения в системе информации о листовых вершинах графа, положение которых не может быть определено, через вершины-потомки, вторая непосредственно участвует в сортировке, через неё в функцию быстрой сортировки в качестве оператора сравнении передается не статическая функция.

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.

6.2.5. Структура класса GraphWidget

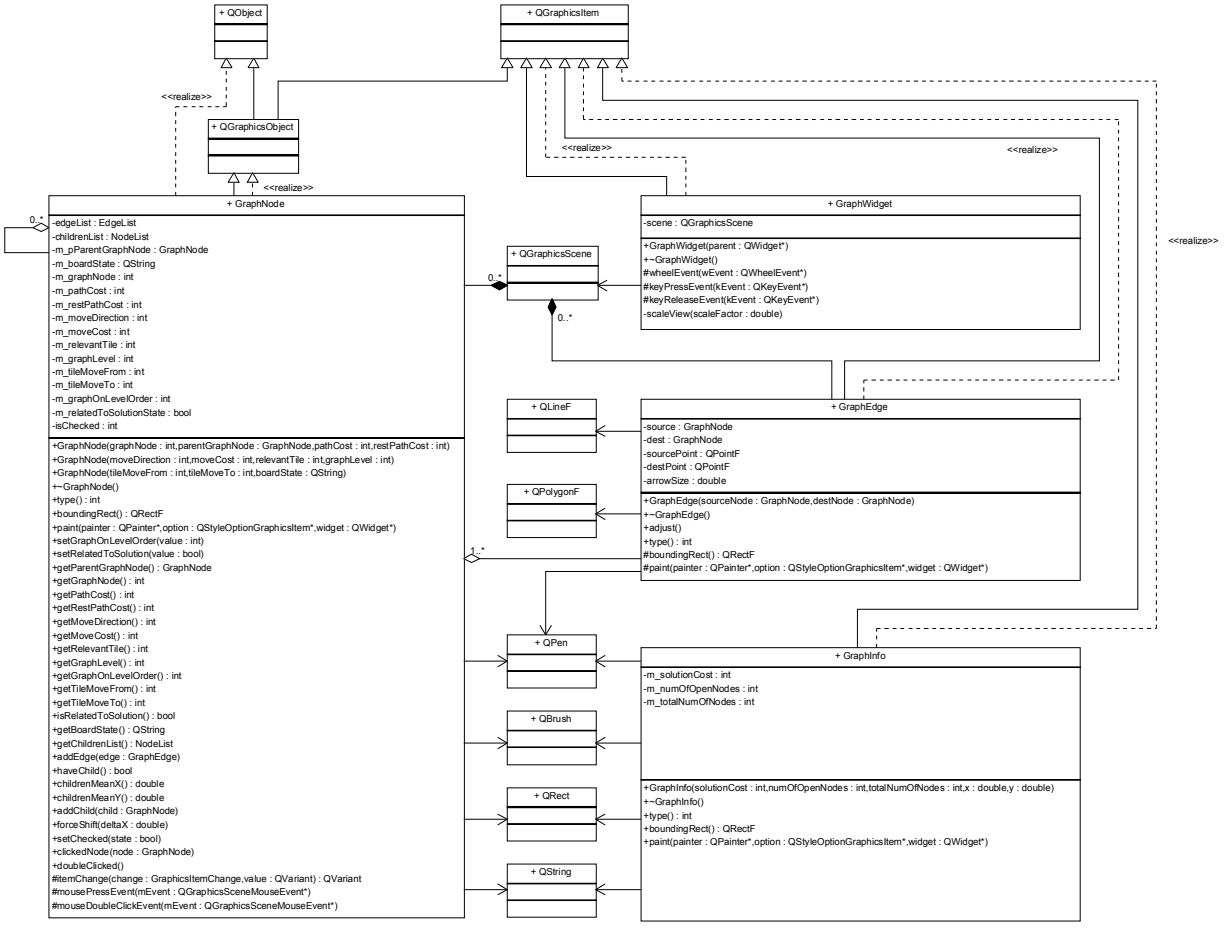


Рисунок 15

На рисунке 15 представлена структура класса **GraphWidget,** реализующий вывод на экран графа, для этого необходимо владение экземпляром класса **QGraphicsScene**, который в свою очередь связан

отношением композиция с классами **QGraphNode** и **QGraphEdge,** что означает жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Сам класс **QGraphNode** хранит список вершин-потомков, то есть экземпляры своего класса, а также список классов **QGraphEdge,** те грани графа которыми экземпляр соединен со своими вершинами-потомками и вершинами-родителями. Это необходимо для построения графа.

Все три графических элемента **QGraphNode, QGraphEdge** и **QGraphInfo,** так или иначе, реализуют функции класса QGraphicsItem и наследуются от него.

6.2.6. Алгоритм отрисовки графа.

Алгоритм отрисовки графа вынесен на лист 6, формат А1.

Программный код, реализующий этот алгоритм представлен в приложении 4.

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 индексы элемента. В его вершины в порядке появления на графе, сортировка проводится методом **quick_sort()**, в качестве функции сравнения используется положение вершины-родителя на графе, которое хранится в этом же массиве на строчку выше. Сложность алгоритма **quick_sort()** $O(n \bullet \ln_2 n)$, не зря он называется быстрым.

Алгоритм в цикле проходит по каждой «строке» графа, начиная с нижней, причем по каждой два раза.

На первом проходе отрисовываются 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 с фиксированным шагом отрисовать вершины последовательности с соответсвующей стороны от крайней вершины.

Последовательность 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, если такие есть, экземпляра класса.

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 рисунке 16 представлен результат работы алгоритма построения графа.

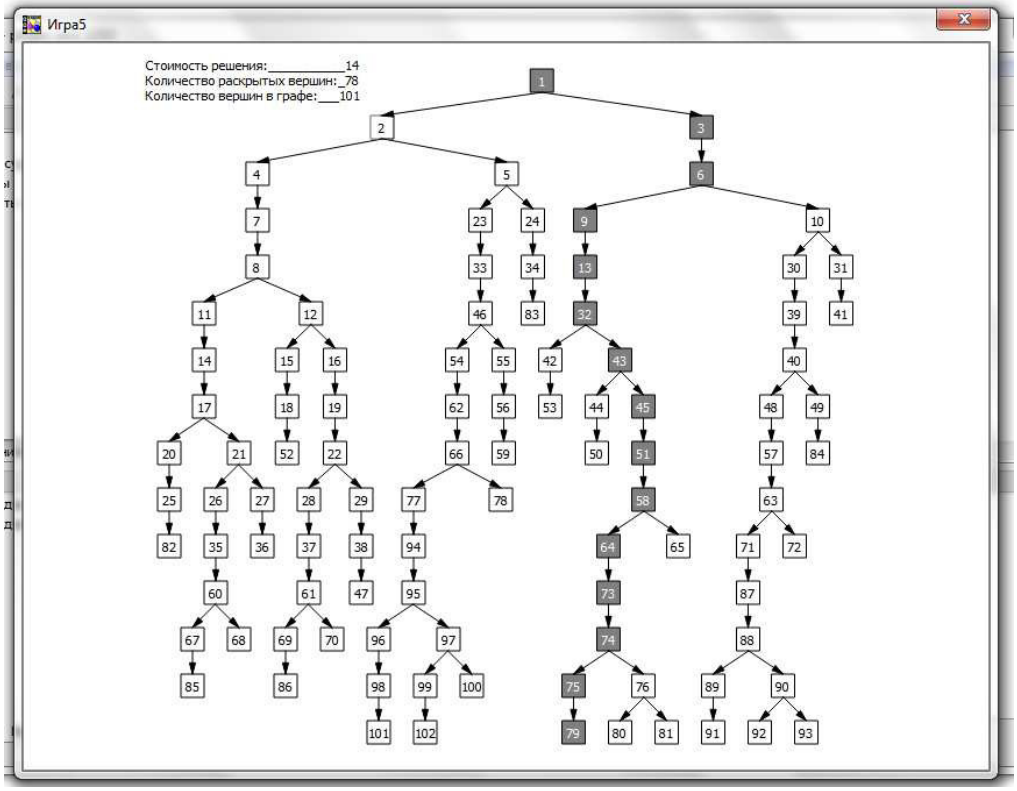


Рисунок 16

6.2.7. 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.

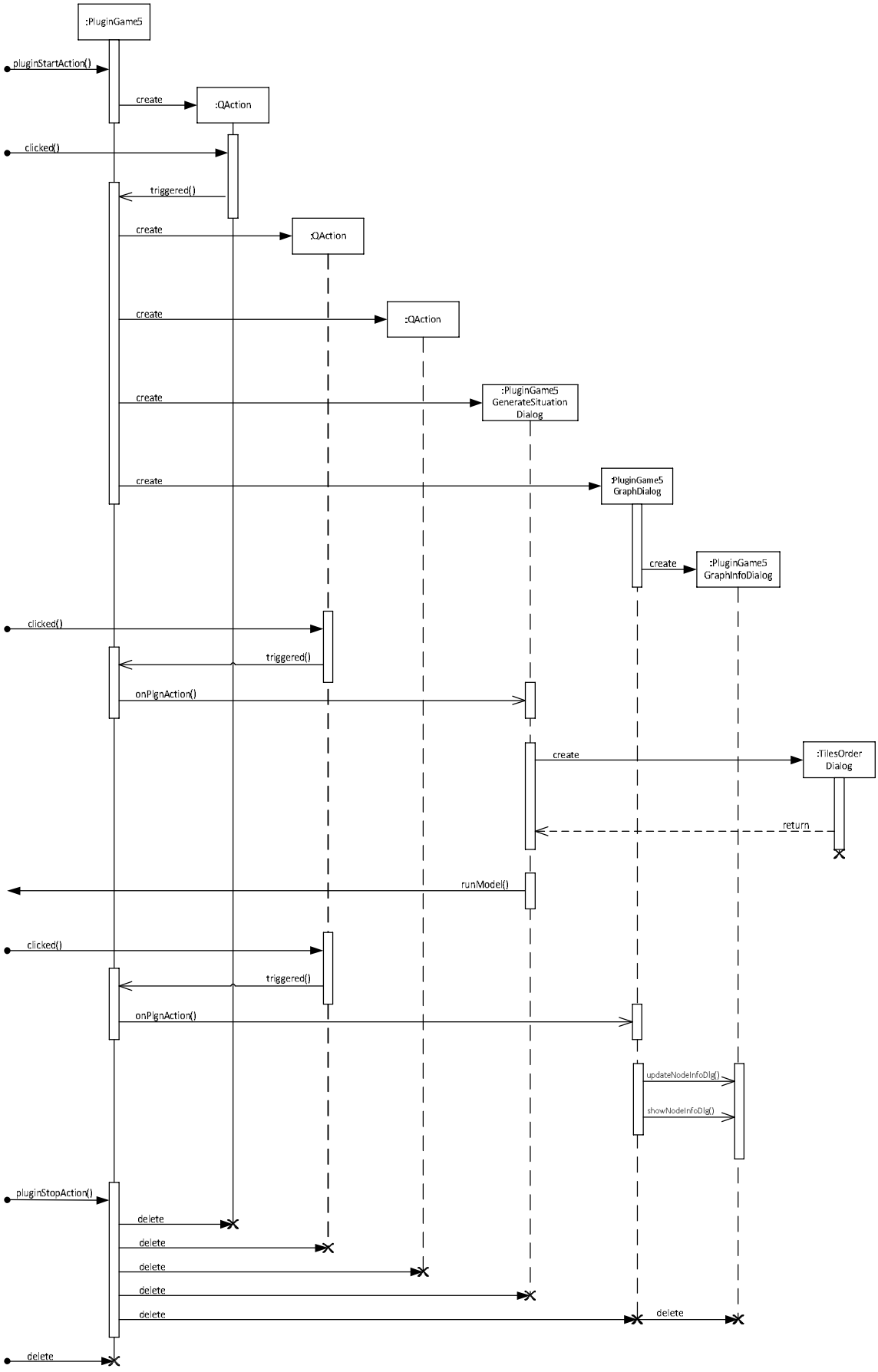

:PluginGame5
pluginStartAction()
create
:QAction
clicked()
triggered()
create
:QAction
create
:QAction
create
:PluginGame5
GenerateSituation
Dialog
create
:PluginGame5
GraphDialog
create
:PluginGame5
GraphInfoDialog
clicked()
triggered()
onPlgnAction()
create
:TilesOrder
Dialog
return
runModel()
clicked()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

Рисунок 17

На рисунке 17 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, клип по кнопке **Ок,** клик по созданной на панели инструментов кнопке **Построить граф**, двойной клик по вершине графа, остановка плагина.

7. Апробирование разработанной системы для модельных условий

Для контроля работоспособности подсистемы загрузки используется разработанный плагин «Пятнашки» в соответствии с требованиями к функциональным характеристикам системы.

Автоматический текст программного обеспечения с обширным графическим пользовательским интерфейсом – это отдельная сложная задача, поэтому в рамках данного курсового проекта, апробирование плагина «Пятнашки» – заключается в прогоне одинаковых моделей в плагине и в отдельном приложении **Игра5**, о котором было сказано на этапе предпроектного исследования, и сравнение результатов работы.

На листе результатов разработки представлены снимки работающей системы с загруженным в неё плагином.

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 укрупнено показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы. Сформулировано техническое задание.
3. На этапе технического проектирования намечены необходимые для разработки классы, разработаны структуры подсистемы загрузки плагинов и самого 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 раннее алгоритмов работы и архитектуры. Вновь разработанные классы показаны с помощью подробных диаграмм классов. Написан программный код, реализующий разработанную грамматику. На диаграмме активностей показан механизм создания объекта спрайта. Проведены отладка и тестирование нового функционала системы, в ходе которых исправлялись найденные ошибки, оптимизировалась работа разрабатываемой подсистемы. Разработан относительно быстрый алгоритм отрисовки графа.
5. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе новой функции системы.
6. В качестве справочной информации по внесенным в систему РДО изменениям предлагается использовать лист результатов данного курсового проекта.

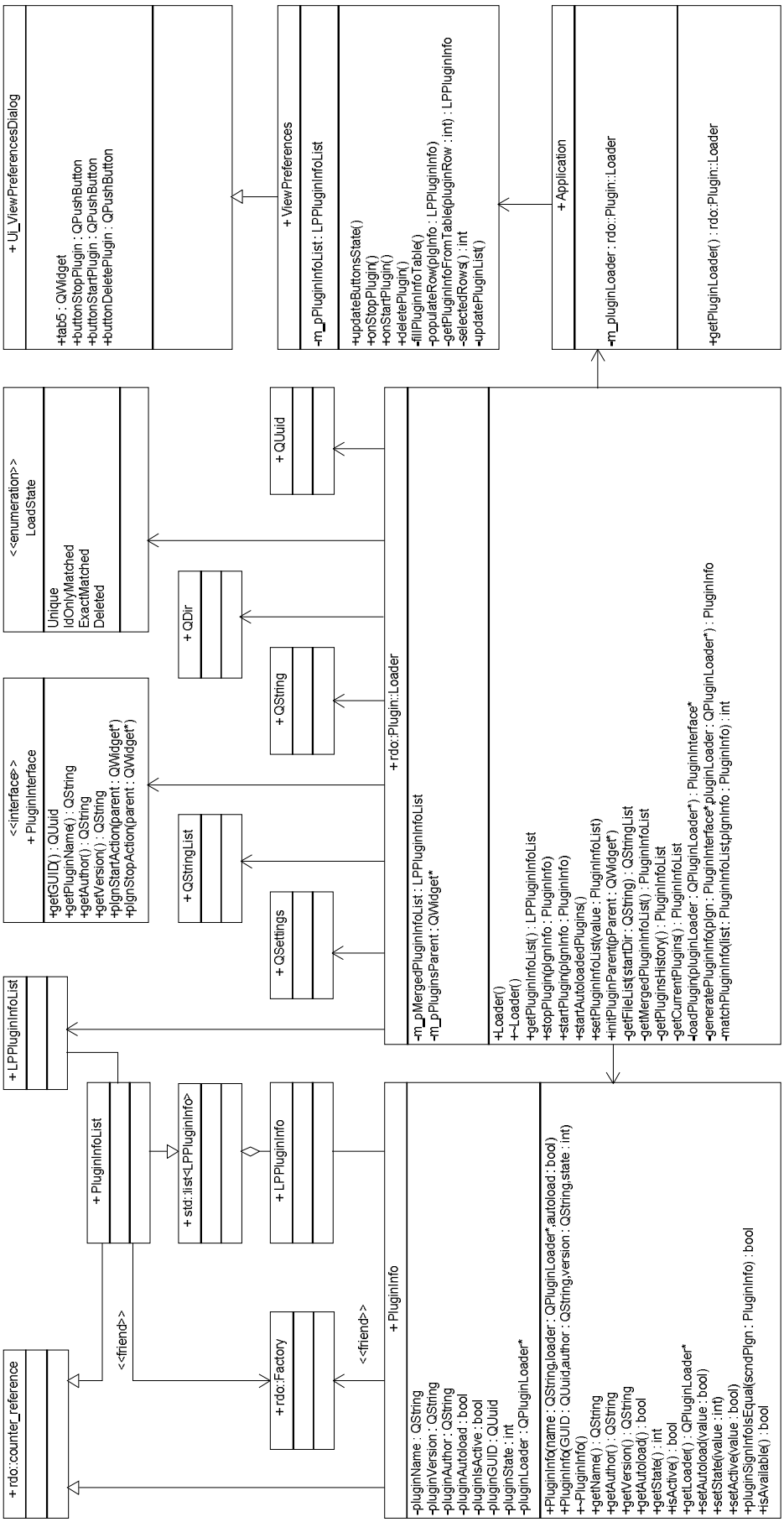
Список литературы

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. Autodesk®, Inc., AutoCAD® Mechanical 2009
4. Microsoft®, Office Word 2003
5. Microsoft®, Office Visio 2007
6. Microsoft®, Visual Studio 2008 SP1
7. Qt Digia, Qt Designer 5.1.1

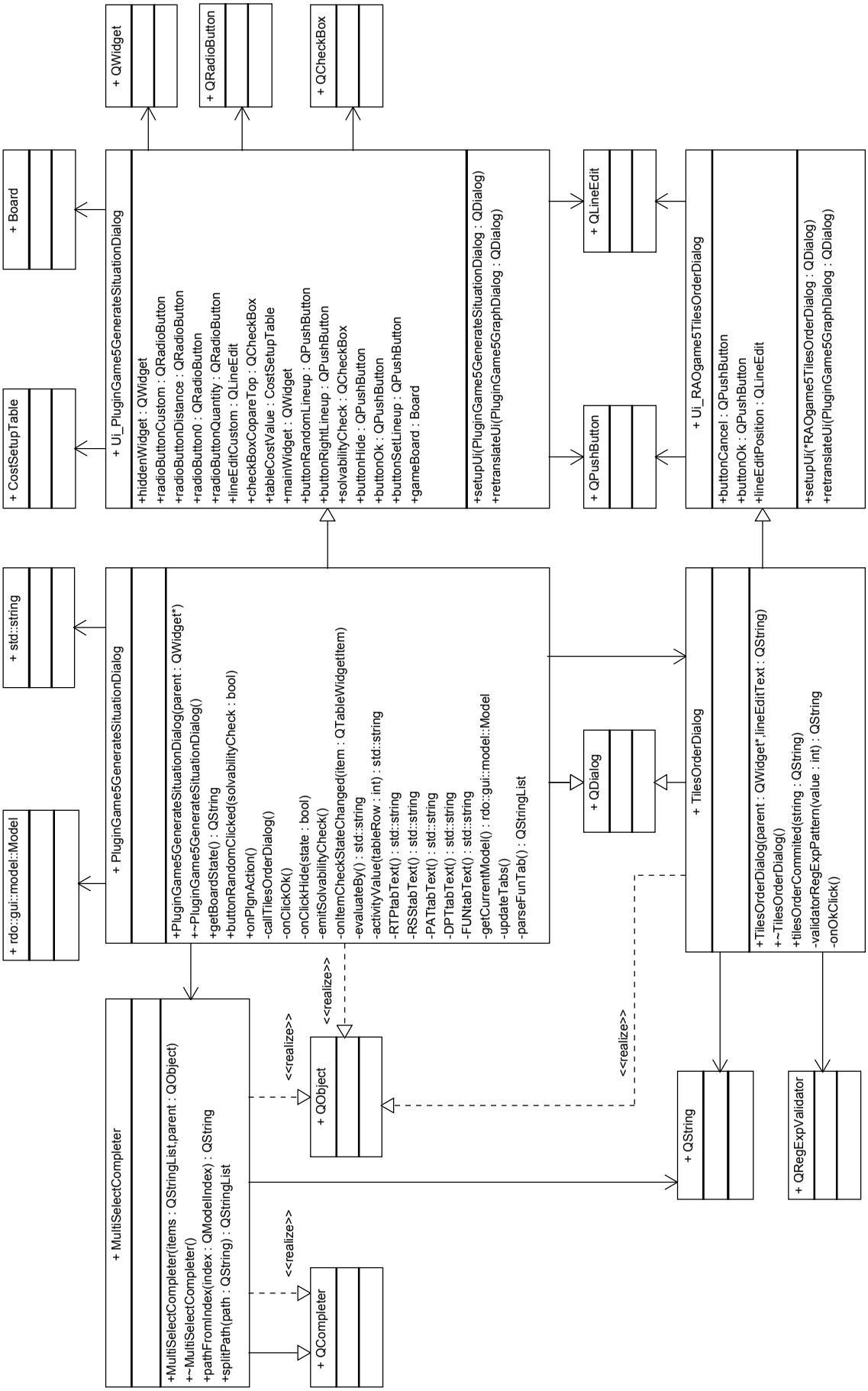
Приложение 1. Диаграмма классов подсистемы плагинов



Приложение 2. 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 3. Диаграмма классов PluginGame5GenerateSiationDialog



Приложение 4. 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());
            }
        }
    }
}
```