**ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ЯРОСЛАВСКОЙ ОБЛАСТИ**

государственное профессиональное образовательное учреждение   
Ярославской области

Рыбинский полиграфический колледж

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| Разработка класса визуального компонента «Редактор графов поиск кратчайшего пути» | |
|  | |
| по дисциплине | Объектно-ориентированное программирование |
|  | |

Пояснительная записка

|  |
| --- |
| КП.090207.11.000000.00 ПЗ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы | 3-ИС-3 |  |  |  | Малышкин И А |
|  | *(Код учебной группы)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |
| Руководитель | преподаватель |  |  |  | Смирнов В Б |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *( И.О.Фамилия)* |
| Нормоконтроль | преподаватель |  |  |  | Ермолычева С В |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |

г. Рыбинск

2023

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc115562458)

[1 Исследовательский раздел 3](#_Toc115562459)

[2 Конструкторский раздел 9](#_Toc115562460)

[2.1 Математическая модель 9](#_Toc115562461)

[2.2 Разработка структуры компонента 10](#_Toc115562462)

[2.3 Разработка графического интерфейса компонента 14](#_Toc115562463)

[2.4 Описание входных и выходных данных компонента 14](#_Toc115562464)

[2.5 Разработка алгоритмов компонента 18](#_Toc115562465)

[3 Технологический раздел 36](#_Toc115562466)

[3.1 Отладка компонента 36](#_Toc115562467)

[3.2 Инструкция по установке компонента 40](#_Toc115562468)

[3.3 Инструкция по эксплуатации компонента 44](#_Toc115562469)

[4 Раздел охраны труда 46](#_Toc115562470)

[Заключение 49](#_Toc115562471)

[Список используемых источников 50](#_Toc115562472)

[Приложение А 51](#_Toc115562473)

Введение

В наше время вычислительная техника является неотъемлемой частью нашей жизни и широко применяется в различных отраслях. Она играет особенно важную роль в научных исследованиях, где позволяет изучать и анализировать разнообразные аспекты окружающего мира, микроорганизмы, историю, космос и многое другое. Вычислительная техника также используется для обработки, хранения и анализа огромного объема данных, а также для создания цифровых моделей объектов реального мира.

Ключевым компонентом вычислительной техники является компьютер, для управления которым необходимо программное обеспечение. Языки программирования были созданы для упрощения процесса написания программного кода. С самых ранних низкоуровневых языков программирования, разработанных специально для конкретных компьютеров, до появления языков высокого уровня, которые стали более удобными и абстрактными, прошло много времени. Первым “настоящим” языком программирования стал Plankalkül [1] (Расчет плана) разработанный человеком по имени Конрад Цузе где-то между 1944 и 1945 годами.

Середина 1950-х годов стала переломным периодом в развитии языков программирования, когда появились первые языки высокого уровня, не привязанные к конкретным типам компьютеров. Первые алгоритмические языки эпохи ЭВМ Фортран и Ангол. Они были ориентированы в основном на системные расчеты. Причем Фортран был первым языком, для которого были созданы компиляторы для многих ЭВМ. Компилятор переводит программу с языка высокого уровня на язык ЭВМ. Алгол был разработан в 1958 году в международном комитете под руководством Питера Наура [6]. Алгоритмические языки предложили новый подход, позволяющий повторно использовать код и имеющий более "человеческий" синтаксис. Это существенно упростило процесс написания программ и сократило объем кода.

Однако с развитием программ стало очевидно, что структура программы имеет большое значение, особенно при работе с длинными и сложными программами. В результате появилась методология структурного программирования, которая ввела понятие процедур и функций, улучшая читаемость и организацию кода.

Однако для решения ещё более сложных задач требовался новый подход, который бы позволил более эффективно организовывать программы и разбивать их на составные части. Таким подходом стало объектно-ориентированное программирование (ООП), которое позволяет описывать программу в терминах объектов и классов, формируя иерархию наследования и взаимодействуя между собой с помощью сообщений. Первым языком программирования, в котором были предложены основные понятия, впоследствии сложившиеся в парадигму, была Simula-1, но термин «объектная ориентированность» не использовался в контексте использования этого языка. В момент его появления в 1967 году в нём были предложены революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Фактически, Симула была «Алголом с классами», упрощающим выражение в процедурном программировании многих сложных концепций [7].

Объектно-ориентированное программирование предлагает ряд принципов, таких как инкапсуляция, наследование и полиморфизм, которые способствуют более гибкой и модульной разработке программного кода. Этот подход не только повышает эффективность работы программистов, но и позволяет лучше отражать реальный мир в структуре программы.

В рамках данного курсового проекта будет разработан класс, представляющий собой визуальный компонент "Редактор графов поиск кратчайшего пути". Этот проект имеет актуальность, так как способствует развитию у людей любого возраста путём решения различных задач с поиском кратчайшего пути в графе. Он также полезен для начинающих программистов, позволяя им увидеть практическое применение изучаемых алгоритмов и использовать методы класса или сам класс при разработке собственных визуальных компонентов, но самая главная причина актуальности класса является то что до сих пор не было разработано ни одного визуального компонента "Редактор графов поиск кратчайшего пути" на windows forms app.

1 Исследовательский раздел

В связи с необходимостью визуального отображения в нашем проекте, мы ориентируемся на среду разработки, которая обладает визуальным редактором. После проведения аналитической работы были выявлены две сходные среды разработки с визуальным редактором: "Visual Studio" [8] и "NetBeans" [9].

В процессе анализа среды Visual Studio мы выявили ее преимущества и недостатки. Среди основных преимуществ данной среды следует отметить наличие бесплатной версии "Community Edition", быстрая и удобная установка на персональный компьютер, легкий для освоения интерфейс, возможность работы с визуальными компонентами, поддержку объектно-ориентированных языков, наличие полной документации и русскоязычной версии, быстрая интерпретация кода, интегрированная поддержка рефакторинга кода (апгрейд кодовой базы), создание приложений для любой платформы, широкую поддержку языков и наличие git manager. Однако следует учесть, что данная среда требовательна к ресурсам компьютера и то что редактор может не подойти для новичков.

Проанализируем теперь среду "NetBeans". Среди главных преимуществ этой среды разработки стоит отметить лёгкую установку, настоящий open-source, совместную разработку, широкую поддержку языков и бесконечное развитие среды которое вытекает из открытого кода софта. Общим недостатком для всех сред, которые запускаются через JDK/SDK, будет существенное потребление ресурсов устройства. Как Eclipse, так и NetBeans не застрахованы от этого. Но от обновления к обновлению все становится лучше. Да и Microsoft понемногу смягчает агрессивную политику по отношению к Java.

На основе выявленных достоинств и недостатков сред разработки мы приняли решение выбрать среду Visual Studio, поскольку она обладает более удобным интерфейсом по сравнению с NetBeans.

После выбора среды разработки необходимо определиться с языком программирования. Для этого также была проведена аналитическая работа, в результате которой были найдены два языка программирования: C# [10] и Java [11]. Решение о выборе конкретного языка будет зависеть от требований проекта, характеристик языка и наших собственных предпочтений.

C#, также известный как C-Sharp, представляет собой типобезопасный язык программирования общего назначения, который следует конструкциям C и C ++. Он следует нескольким парадигмам программирования, включая объектно-ориентированное, структурированное, императивное, управляемое задачами, функциональное, управляемое событиями, параллельное, рефлексивное и общее.

Преимущества заключаются в том что C# - простой, надежный и масштабируемый язык программирования, Динамически типизированный характер C# облегчает разработчикам поиск ошибок в коде. C# устраняет проблему утечки памяти, он имеет знакомый синтаксис, идентичный языкам C и C++.

Главным недостатком является то что у C# крутая кривая обучения, поэтому он не идеален для начинающих. Те, кто имеет базовые знания C, C ++ или Java, могут легко изучить C#.

Главные преимущества языка Java: кроссплатформенность. Написанный вами код будет работать на любой платформе с поддержкой Java; надёжность. Этим качеством Java обязан строгой статистической типизации. Любая переменная или выражение имеют определённый тип на момент компиляции, что сводит к минимуму вероятность ошибки.

Недостатки Java. Минусами Java принято считать: более низкую скорость, если сравнивать с C и C++. Как и у других языков высокого уровня, низкая производительность Java связана с компиляцией и абстракцией с помощью виртуальной машины.

По совокупности достоинств и недостатков был выбран язык С#, потому что данный язык полностью, без каких-либо манипуляций поддерживается средой разработки Visual Studio.

В интернете существует большое количество редакторов графов «Редактор графов поиск кратчайшего пути». Одним из таких примеров можно ознакомиться на сайте «github» [12]. Данная версия редактора – это пример классического редактора поиска кратчайшего пути через алгоритм Дейкстры. В этой версии можно рисовать вершины и грани, удалять их, менять вес грани и ставить вершину от которой нужно просчитать кратчайшие пути. Управление осуществляется с помощью манипулятора типа «мышь». Недостатками данной версии редактора является обязательное подключение к интернету и то, что она не является визуальным компонентом.

Цель данного курсового проекта – создание визуального редактора «Редактор графов поиск кратчайшего пути» лишённой таких недостатков.

Родительским классом был выбран класс визуальных компонентов «Control», потому что данный класс имеет метод «OnPaint», который позволяет обновлять содержимое компонента по средствам перерисовывания.

Основным процессом в проекте будет алгоритм, а именно, алгоритм поиска кратчайшего пути в графе через алгоритм Дейкстры.

2 Конструкторский раздел

2.1 Математическая модель

По правилам редактора на компоненте будет присутствовать поле прямоугольной формы. На поле можно будет добавлять вершины и грани с определённым весом между ними.

В данном проекте будет присутствовать один основной процесс. Для работы алгоритмов потребуются отдельные списки типа классов, первый класс будет иметь 2 свойства для хранения имени вершины и списка соединённых граней, второй класс будет иметь тоже 2 свойства первое для хранения веса ребра, а второе для присоединённой вершины. Третий класс имеет одно свойство для хранения вершин. Четвёртый класс имеет 4 свойства вершина, состояние, сумма весов рёбер и предыдущая вершина.

Главным алгоритмом будет алгоритм Дейкстры для поиска кратчайшего пути в графе.

Идея алгоритма Дейкстры заключается в постепенном поиске кратчайших путей от начальной вершины ко всем остальным вершинам графа. Алгоритм поддерживает набор вершин, для которых уже найдены кратчайшие пути, и постепенно расширяет этот набор, выбирая вершину с наименьшим известным кратчайшим путем и обновляя кратчайшие пути для соседних вершин. Во входных парамметрах алгоритма будут две вершины.

Алгоритм Дейкстры будет работать следующий образом. Для начала устанавливаем начальную вершину и присваиваем ей кратчайший путь равным 0, а все остальные вершины — бесконечности. Создаем также пустое множество для хранения посещенных вершин. Пока есть непосещенные вершины, выбираем вершину с наименьшим известным кратчайшим путем, помечаем выбранную вершину как посещенную. Для каждой соседней вершины, не посещенной ранее рассчитываем временную длину пути от начальной вершины до текущей соседней вершины через выбранную вершину. Если временная длина пути меньше текущей известной длины пути до текущей соседней вершины, обновляем кратчайший путь до нее. По завершении алгоритма получаем кратчайшие пути от начальной вершины до всех остальных вершин.

2.2 Разработка структуры компонента

Для описания метода проектирования компонента будет применяться последовательная декомпозиция, начиная с концепции "черной сферы". Черная сфера представляет собой систему, внутреннее устройство которой не имеет значения. В эту систему вводятся входные данные, а на выходе получаются выходные данные.Черная сфера представлена на рисунке 2.1.



Рисунок 2.1 – Черная сфера

Входными данными компонента будут, размеры, цвета, а также управляющее воздействие пользователя. Выходными данными станет кратчайший путь и обновлённое изображение содержимого компонента. Цвета будут использоваться для таких объектов, как поле редактора, вершины, грани, текст, кратчайший путь и рамки. Размеры объектов будут задаваться в свойствах компонента. Управляющее воздействие пользователя будет происходить с помощью манипуляторов типа «мышь» и «клавиатура». Размер вершин будет задаваться в свойствах компонента Изображение компонента будет выводиться на экран. Черная сфера с перечисленными входными и выходными параметрами представлена на рисунке 2.2.

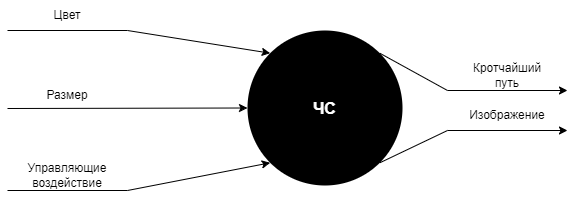


Рисунок. 2.2. - Черная сфера с параметрами

Проведем первый этап декомпозиции с выявлением основных внутренних структур данных, необходимых для хранения информации, и процессов компонента. Размеры компонента и размеры его элементов (игрового поля и шариков) будут зависеть от изменения размеров окна формы пользователем. Обрабатывать изменения будет метод, фиксирующий изменения размеров формы и корректирующий размеры элементов компонента таким образом, чтобы компонент соответствовал размерам формы. При изменении цвета элементов, компонент будет перерисовываться в соответствии с заданными цветами. Обработка нажатия клавиш пользователем будет осуществляться событием, регистрирующим нажатия и совершающим добавление, удаление вершины или ребра, в соответствии с нажатой клавишей. Поиск кратчайшего пути между вершинами будет осуществлён через публичный метод который будет возвращать его в виде строки и рисовать.

Первый этап декомпозиции представлен на рисунке 2.3.



Рисунок. 2.3 – Результат первого этапа декомпозиции

В первом этапе декомпозиции были получены основные структуры данных, необходимых для хранения информации, и процессов компонента, которые позволят построить базовую логику. Для получения более точных структур данных и процессов компонента проведем второй этап декомпозиции. Управляющее воздействие пользователя будет влиять на такие методы, как метод добавления и удаления вершин, метод добавления и удаления граней и метод поиска кратчайшего пути. Методы добавления, удаления вершин и рёбер смогут работать как с помощью слушателя нажатия кнопок мыши с включённым флагом нужного режима, так и с помощью передачи нужных имён и весов.

При изменении цвета элементов, компонент будет перерисовываться в соответствии с заданными цветами. Размеры компонента и размеры его элементов (вершины и т.д.) будут зависеть от изменения размеров окна формы пользователем. Обрабатывать изменения будет метод, фиксирующий изменения размеров формы и вызывающий метод, корректирующий размеры элементов компонента таким образом, чтобы компонент соответствовал размерам формы. Второй этап декомпозиции представлен на рисунке 2.4.

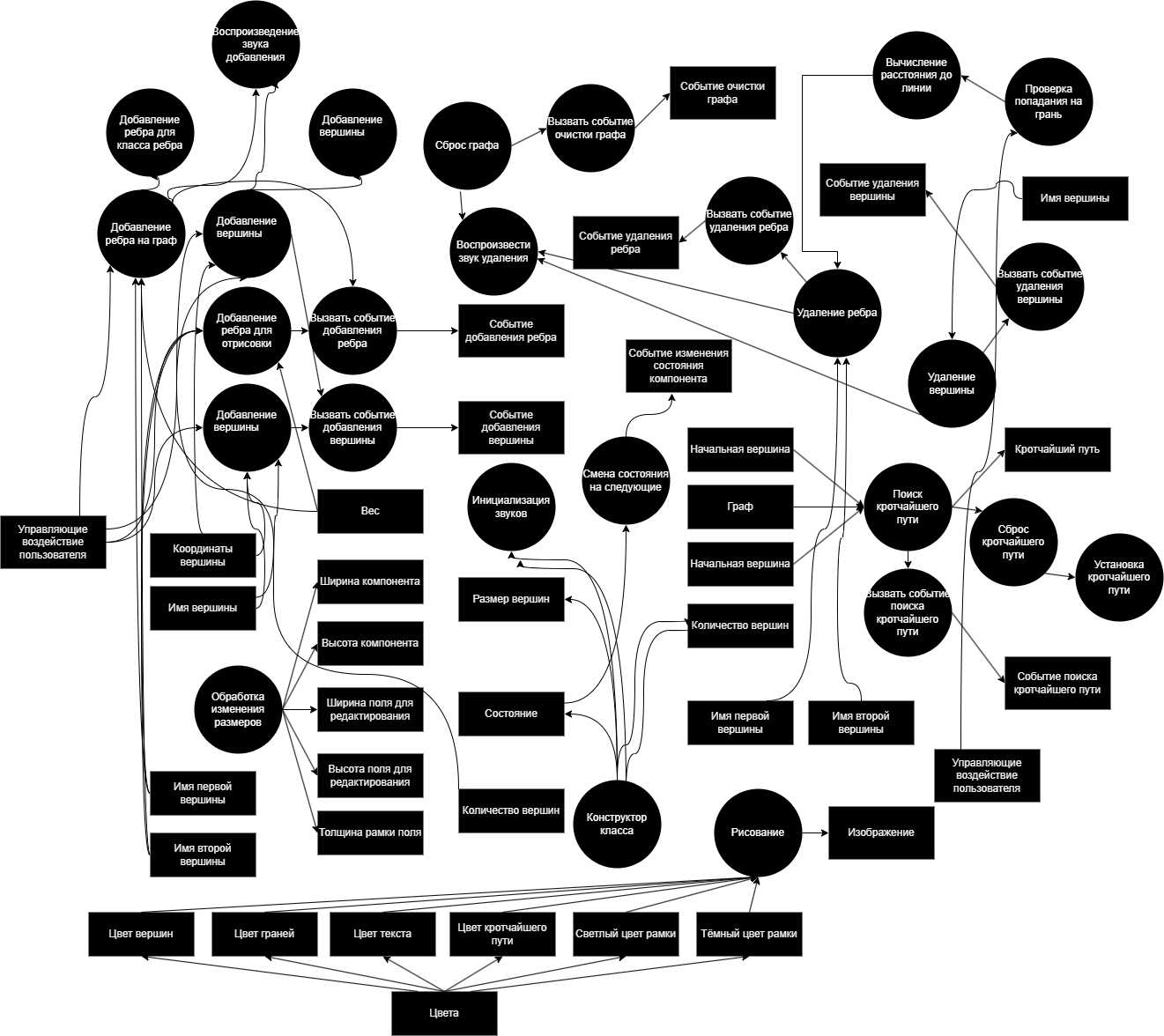


Рисунок .2.4 - Результат второго этапа декомпозиции

2.3 Разработка графического интерфейса компонента

Компонент будет представлять из себя фон настраиваемого цвета с рамкой по краям. Ширина поля будет определяться, как размер компонента с вычетом размеров рамки по краям. Изначально поле редактора будет пустым, но на него можно добавить вершины и рёбра путём включения соответствующих режимов и нажатия кнопки «ЛКМ» на манипуляторе мышь, ребро также можно добавить указав соответствующие вершины и вес. Вершины и рёбра можно будет удалять путём включения соответствующего режима и нажатия кнопки «ЛКМ» на манипуляторе мышь по удаляемому объекту. По этим вершинам и рёбрам можно будет найти кратчайший путь указав начальную вершину и вершину назначения. При нахождении кратчайшего пути если путь возможно построить, то он нарисуется указанным в параметрах компонента цветом и вернёт кратчайший путь в виде строки перечисления через запятую. Также в свойствах компонента можно будет изменить цвет рамки поля редактора, цвет фона поля редактора, цвет текста, кратчайшего пути, вершин и рёбер. Также в свойствах можно указать состояние рамки включить или отключить звуки и включить любой из предложенных режимов.

2.4 Описание входных и выходных данных компонента

В результатах раздела [2.2] получена структурная схема компонента, из которой следует необходимость присутствия тех или иных переменных и событий для полноценной работы компонента. Для удобства все свойства и события сведены в табличном виде.

Свойства класса представлены в таблице 2.1.

Таблица 2.1 – Свойства класса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Метод доступа на чтение | Метод доступа на запись |
| ObjState | ObjStates | Чтение/запись | Напрямую из внутреннего поля \_ObjState | Прямая запись внутри метода set |
| ShortestPathColor | Color | Чтение/запись | Напрямую из внутреннего поля \_ShortestPathColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| VertexColor | Color | Чтение/запись | Напрямую из внутреннего поля \_VertexColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| EdgeColor | Color | Чтение/запись | Напрямую из внутреннего поля \_EdgeColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| TextColor | Color | Чтение/запись | Напрямую из внутреннего поля \_TextColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| DarkColor | Color | Чтение/запись | Напрямую из внутреннего поля \_DarkColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |

Продолжение таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Метод доступа на чтение | Метод доступа на запись |
| ShortestPathColor | Color | Чтение/запись | Напрямую из внутреннего поля \_ShortestPathColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| IsVertexAddMode | bool | Чтение/запись | Напрямую из внутреннего поля  \_IsVertexAddMode | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| IsEdgeAddMode | bool | Чтение/запись | Напрямую из внутреннего поля \_IsEdgeAddMode | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| IsDeleteMode | bool | Чтение/запись | Напрямую из внутреннего поля \_IsDeleteMode | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |
| SoundsOn | bool | Чтение/запись | Напрямую из внутреннего поля \_SoundsOn | Используется механизм записи в методе set для обновления значения внутреннего поля и |

Окончание таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Метод доступа на чтение | Метод доступа на запись |
| LightColor | Color | Чтение/запись | Напрямую из внутреннего поля \_LightColor | Используется механизм записи в методе set для обновления значения внутреннего поля и картинки на экране |

События класса представлены в таблице 2.2.

Таблица 2.2- События класса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя события | Тип данных | Тип доступа | Метод доступа на добавление | Метод доступа на удаление |
| OnChangeState | EventHandler | Добавление/удаление | Напрямую из внутреннего поля State | Напрямую из внутреннего поля |
| OnVertexAdd | EventHandler | Добавление/удаление | Напрямую из внутреннего поля addVertex | Напрямую из внутреннего поля |
| OnVertexRemove | EventHandler | Добавление/удаление | Напрямую из внутреннего поля  removeVertex | Напрямую из внутреннего поля |
| OnEdgeAdd | EventHandler | Добавление/удаление | Напрямую из внутреннего поля addEdge | Напрямую из внутреннего поля |
| OnEdgeRemove | EventHandler | Добавление/удаление | Напрямую из внутреннего поля  removeEdge | Напрямую из внутреннего поля |
| OnFindShortestPath | EventHandler | Добавление/удаление | Напрямую из внутреннего поля FindShortestPath | Напрямую из внутреннего поля |

2.5 Разработка алгоритмов компонента

В результатах подраздела 2.2, получена структурная схема компонента, из которой следует необходимость присутствия тех или иных методов для полноценной работы компонента. В соответствие с этой структурной схемой в данном разделе будет рассмотрена разработка алгоритмов всех методов.

FindShortestPath - метод, который ищет кратчайший путь между двумя вершинами. Для этого он вызывает ряд методов, которые настраивают отдельные элементы игры, такие как вызов алгоритма Дейкстры, сброс старого кратчайшего пути и установка нового пути. addEdge - метод, который добавляет ребро на поле редактора. removeEdge - метод, который удаляет ребро с поля редактора. addVertex - метод, который добавляет вершину на поле редактора. removeVertex - метод, который удаляет вершину с поля редактора. rеmoveEdgesForVertex – метод который удаляет все рёбра для указанной вершины. State – метод который меняет состояние рамки. ResetShortestPath - метод который сбрасывает старый кратчайший путь. SetShortestPath - метод который устанавливает новый кратчайший путь. IsPointOnEdge - метод который проверяет попадает ли указанная точка на линию. DistancePointToLine - метод который вычисляет расстояние от точки до линии. InitSounds - метод который инициализирует звуки. Блок-схема алгоритма метода Reset приведена на рисунке 2.5.

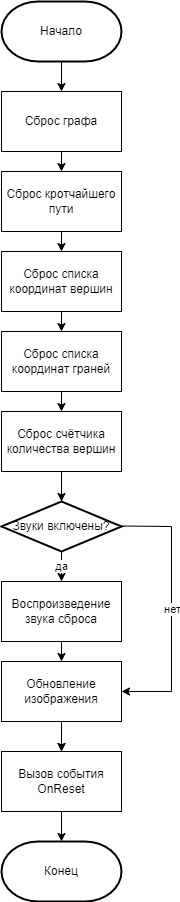


Рисунок. 2.5 - Блок-схема алгоритма метода Reset

Метод Reset вызывается при выполнении сброса графа и осуществляет сброс всех значений графа.

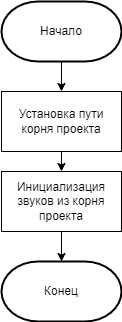


Рисунок. 2.6. – Блок-схема алгоритма метода InitSounds

Метод InitSounds нужен для того чтобы инициализировать звуки из корня проекта. Вызывается в конструкторе класса.

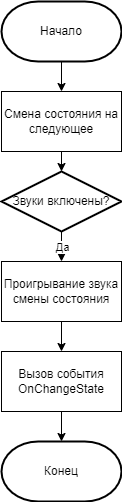
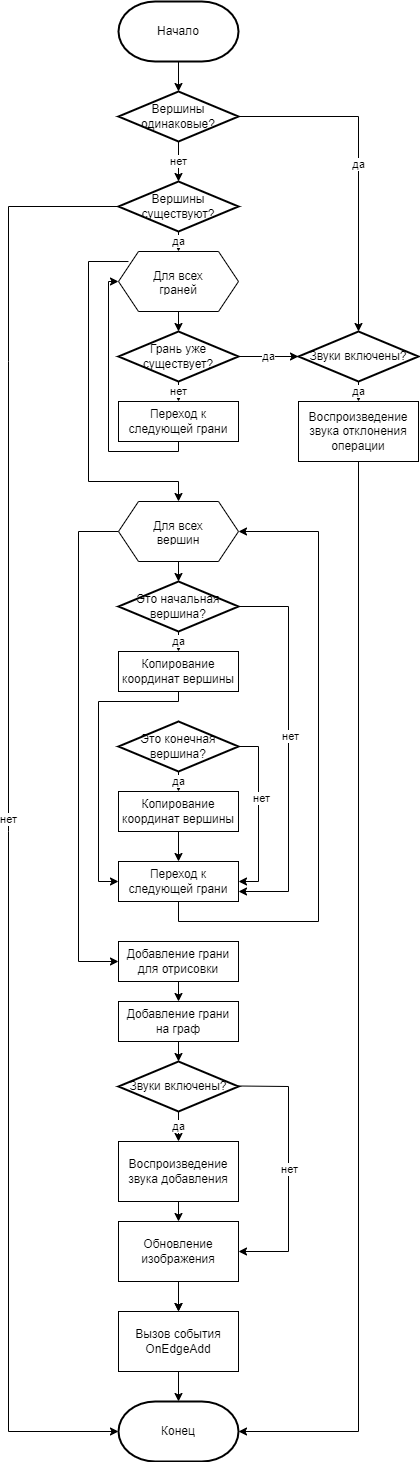


Рисунок. 2.7 – Блок-схема алгоритма метода State

State - этот метод сменяет состояние компонента на следующее.



Рисонок. 2.8 – Блок-схема алгоритма метода addEdge

Задача метода addEdge состоит в том, чтобы добавить новое ребро на граф если оно ещё не существует. Вершины и вес грани передаются во входных параметрах метода.

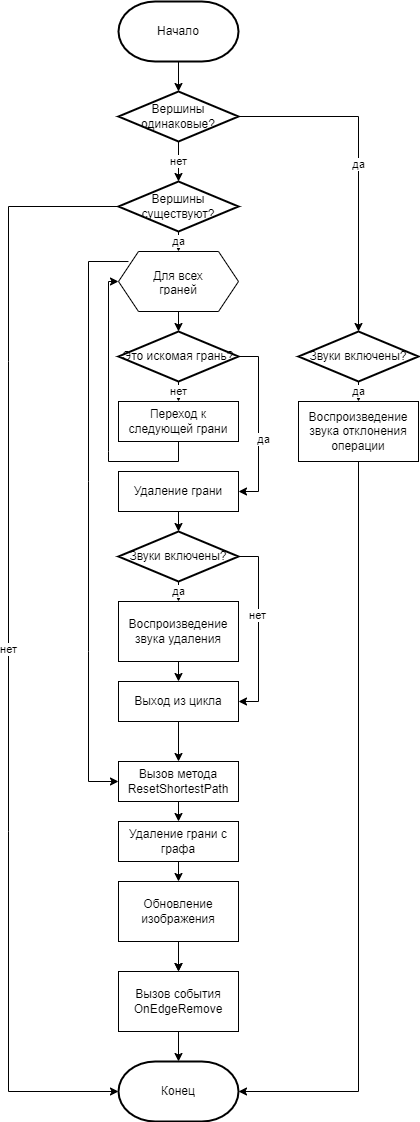


Рисунок 2.9 - Блок-схема алгоритма метода removeEdge

removeEdge - метод, который нужен для удаления грани. Вершины передаются во входные параметры метода. Блок-схема алгоритма метода ResetShortestPath приведена на рисунке 2.10.



Рисунок. 2.10 – Блок-схема алгоритма метода ResetShortestPath

ResetShortestPath – метод сбрасывает кратчайший путь для отрисовки.

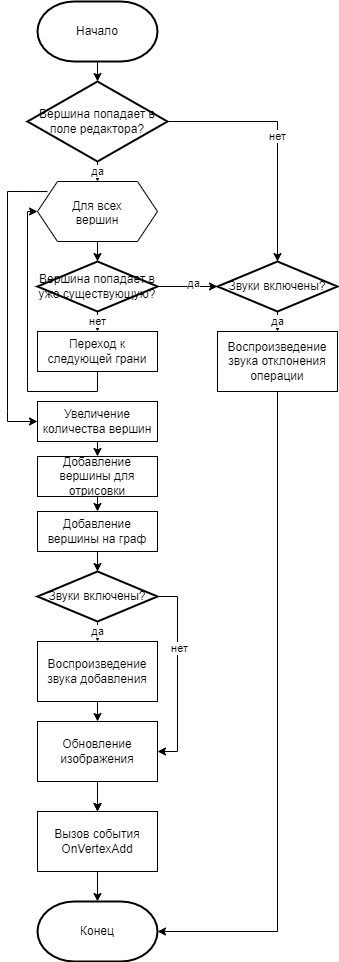


Рисунок. 2.11 – Блок-схема алгоритма метода addVertex

Метод addVertex добавляет новую вершину в месте нажатия. Координаты вершины передаются во входных параметрах метода

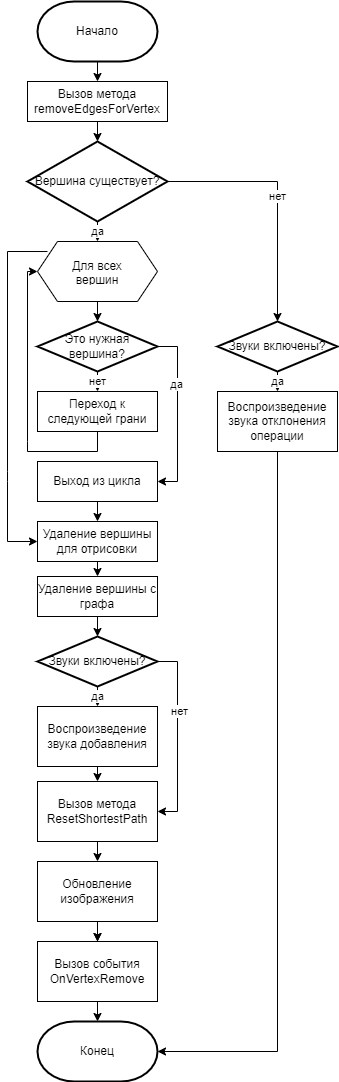


Рисунок. 2.12 – Блок-схема алгоритма метода removeVertex

Метод removeVertex удаляет указанную вершину с графа и из списка отрисовки. Вершина передаётся во входных параметрах. Блок-схема алгоритма метода ResetShortestPath приведена на рисунке 2.10. Блок-схема алгоритма метода removeEdgesForVertex приведена на рисунке 2.13.



Рисунок. 2.13 - Блок-схема алгоритма метода removeEdgesForVertex

removeEdgesForVertex – данный метод должен удалить все грани присоединённые к указанной вершине. Вершина передаётся во входных параметрах метода.

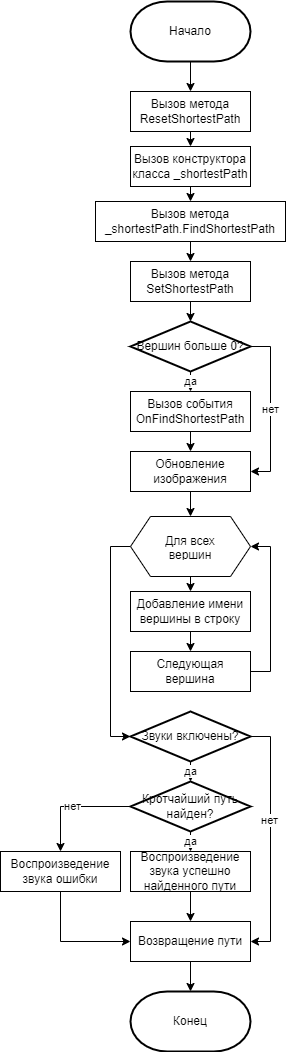


Рисунок. 2.14 – Блок-схема алгоритма метода FindShortestPath

FindShortestPath – метод, который отвечает за отображение кратчайшего пути между двумя вершинами. Метод вызывает метод FindShortestPath у класса с алгоритмом Дейкстры расставляет флаги для найденного кратчайшего пути и возвращает кратчайший путь между двумя вершинами в виде строки. Вершины передаются во входных параметрах. Блок-схема алгоритма метода SetShortestPath приведена на рисунке 2.15. Блок-схема алгоритма метода GraphShortestPath.FindShortestPath приведена на рисунке 2.16.

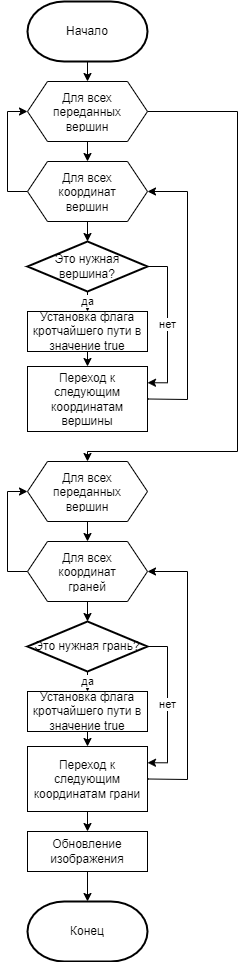


Рисунок. 2.15 – Блок-схема алгоритма метод SetShortestPath

Метод SetShortestPath ищет по переданным во входных параметрах вершинам кратчайшего пути подходящие вершины и грани, и ставит их флаги в состояние true.

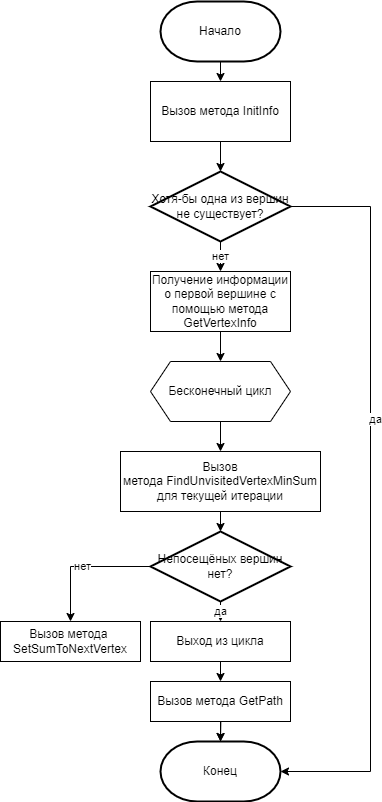


Рисунок. 2.16 – Блок-схема алгоритма метода GraphShortestPath.FindShortestPath

Метод GraphShortestPath.FindShortestPath ищет кратчайший путь между двумя вершинами через алгоритм Дейкстры. Вершины передаются через входные параметры. Блок-схема алгоритма метода InitInfo приведена на рисунке 2.17. Блок-схема алгоритма метода GetVertexInfo приведена на рисунке 2.18. Блок-схема алгоритма метода FindUnvisitedVertexMinSum приведена на рисунке 2.19. Блок-схема алгоритма метода SetSumToNextVertex приведена на рисунке 2.20. Блок-схема алгоритма метода GetPath приведена на рисунке 2.21.



Рисунок. 2.17 – Блок-схема алгоритма метода InitInfo

Метод InitInfo инициализирует внутреннюю переменную \_infos и добавляет в неё все вершины графа.

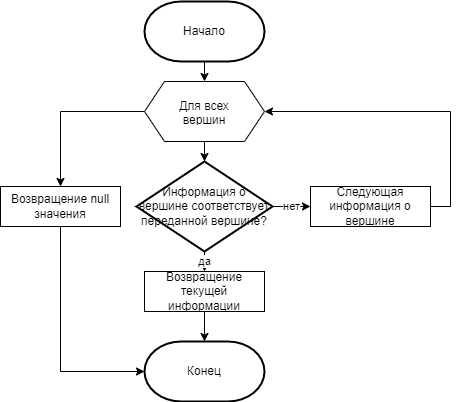


Рисунок. 2.18 – Блок-схема алгоритма метода GetVertexInfo

Метод GetVertexInfo ищет информацию о переданной во входных параметрах в списке информации и возвращает её. Если информация была не найдена будет возвращено null значение.

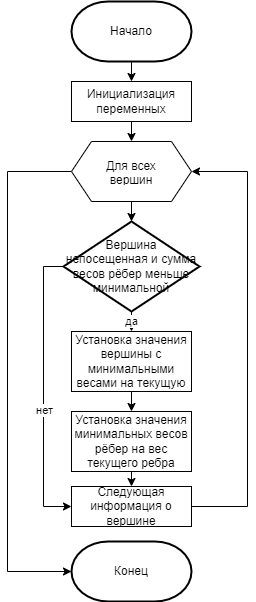


Рисунок. 2.19 – Блок-схема алгоритма метода FindUnvisitedVertexMinSum

Метод FindUnvisitedVertexMinSum ищет не посещённую вершину с минимальным весом граней.

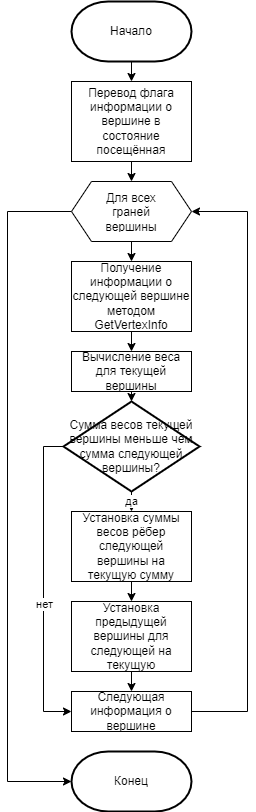


Рисунок 2.20 - Блок-схема алгоритма метода SetSumToNextVertex

Метод SetSumToNextVertex вычисляет суммы весов рёбер для следующей вершины. Информация о вершинах передаётся во входных параметрах. Блок-схема алгоритма метода GetVertexInfo приведена на рисунке 2.18.

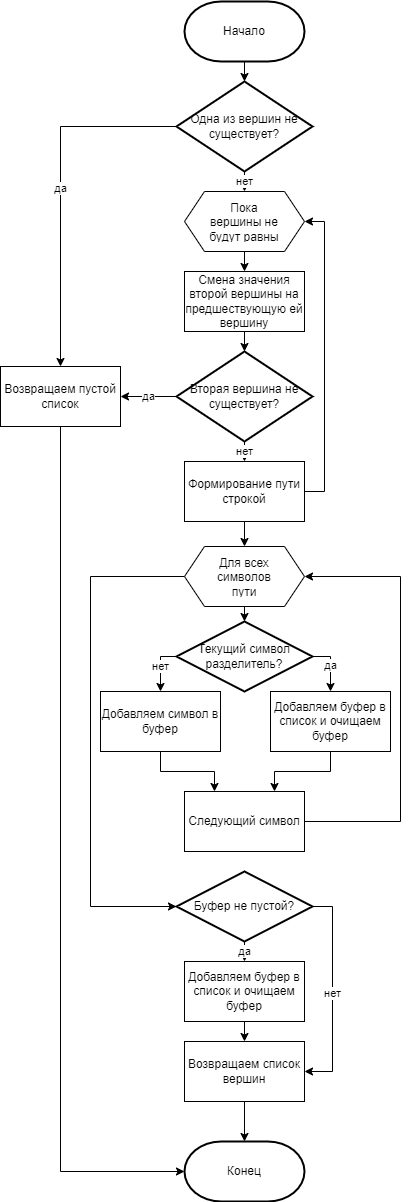


Рисунок. 2.21 – Блок-схема алгоритма метода GetPath

Метод GetPath формирует путь в формате листа из строк. Получает вершины в входных параметрах метода. Блок-схема алгоритма метода GetVertexInfo приведена на рисунке 2.18.

3 Технологический раздел

3.1 Отладка компонента

В процессе тестирования будет использоваться подход с использованием тест-кейсов, поскольку они позволяют проверить продукт без необходимости изучать всю документацию. Создание удобного и легко поддерживаемого тест-кейса, который будет написан один раз, сэкономит значительное количество времени и усилий у тестировщиков. Тест-кейс представляет собой документ, описывающий набор шагов, определенные условия и параметры, необходимые для проверки функциональности или ее части. Когда тест-кейсы передаются тестировщику, он должен последовательно пройти через каждый пункт и выполнить описанные действия, чтобы достичь определенных результатов. Тест-кейс для свойств представлен в таблице 3.1.

Таблица 3.1. Тест-кейс свойств класса

|  |  |  |
| --- | --- | --- |
| Название свойства | Управляющее воздействие | Результат воздействия |
| IsVertexAddMode | Изменить значение свойства, перевести свойства IsEdgeAddMode и IsDeleteMode в значение false | Изменение значения внутреннего поля \_IsVertexAddMode |
| IsEdgeAddMode | Изменить значение свойства, перевести свойства IsVertexAddMode и IsDeleteMode в значение false | Изменение значения внутреннего поля \_ IsEdgeAddMode |
| IsDeleteMode | Изменить значение свойства, перевести свойства IsEdgeAddMode и IsVertexAddMode в значение false | Изменение значения внутреннего поля \_IsDeleteMode |
| SoundsOn | Изменить значение свойства в свойствах компонента | Звуки будут включены либо выключены |
| ObjState | Изменить значение свойства в свойствах компонента | Состояние рамки будет изменено |
| VertexColor | Изменить значение свойства в свойствах компонента | Цвет вершин на поле будет изменено |

Окончание таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Название свойства | Управляющее воздействие | Результат воздействия |
| ShortestPathColor | Изменить значение свойства в свойствах компонента | Цвет кратчайшего пути будет изменён |
| EdgeColor | Изменить значение свойства в свойствах компонента | Цвет граней будет изменён |
| TextColor | Изменить значение свойства в свойствах компонента | Цвет текста будет изменён |
| DarkColor | Изменить значение свойства в свойствах компонента | Тёмный цвет рамки будет изменён |
| LightColor | Изменить значение свойства в свойствах компонента | Светлый цвет рамки будет изменён |
| \_VertexCount | Добавить вершину | Количество вершин увеличится |
| vertexCoordinates | Вызвать метод addVertex | Добавится новая вершина на граф для отрисовки |
| Вызвать метод  removeVertex | Удалится указанная вершина с графа для отрисовки |
| Вызвать метод removeEdge | Удалятся все рёбра привязанные к указанной вершине для отрисовки |
| edgeCoordinates | Вызвать метод addEdge | Добавится новое ребро на граф для отрисовки |
| Вызвать метод removeEdge | Удалится указанное ребро с графа для отрисовки |
| \_graph | Вызвать метод addVertex | Добавится новая вершина на граф |
| Вызвать метод  removeVertex | Удалится указанная вершина с графа |
| Вызвать метод removeEdge | Удалятся все рёбра привязанные к указанной вершине |
| Вызвать метод addEdge | Добавится новое ребро на граф |
| Вызвать метод removeEdge | Удалится указанное ребро с графа |
| Вызвать метод FindShortestPath | Найдёт кратчайший путь между 2-мя вершинами |
| \_shortestPath | Вызвать метод FindShortestPath | Найдёт кратчайший путь между 2-мя вершинами |

Тест кейс для методов представлен в таблице 3.2.

Таблица 3.2 – Тест-кейс для методов

|  |  |  |
| --- | --- | --- |
| Название метода | Управляющее воздействие | Результат воздействия |
| addVertex | Нажать «ЛКМ» на манипуляторе мышь в режиме добавления вершин | В месте нажатия на поле появится вершина |
| addEdge | Программно вызвать метод edgeAddByMouse | Добавится грань с указанным весом между выбранными вершинами |
| removeEdge | Программно вызвать метод deleteMode; Вызвать метод и передать в него имена вершин | Удалится выбранная грань |
| removeVertex | Программно вызвать метод deleteMode; Вызвать метод и передать в него имя вершины | Удалится выбранная вершина |
| removeEdgesForVertex | Программно вызвать метод removeVertex | Удалятся грани для выбранной вершины |
| State | Программно вызвать метод  State | Изменится состояние рамки поля |
| Reset | Программно вызвать метод  Reset | Удалятся все вершины и грани с поля редактора |
| FindShortestPath | Программно вызвать метод  FindShortestPath | Найдётся и отобразится кратчайший путь между указанными вершинами |
| ResetShortestPath | Программно вызвать метод FindShortestPath | Сбросится кратчайший путь |
| SetShortestPath | Программно вызвать метод  FindShortestPath | Задастся новый кратчайший путь |
| SetBoundsCore | Изменить размеры компонента | Изменение ширины и высоты компонента с сохранением всех пропорций всех элементов игрового поля |
| InitSounds | Вызвать конструктор компонента | Звуковые компоненты звука будут инициализированы |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| Название метода | Управляющее воздействие | Результат воздействия |
| IsPointOnEdge | Программно вызвать метод  deleteMode | Вернётся булево значение того, что попадает точка на грань или нет |
| DistancePointToLine | Программно вызвать метод  IsPointOnEdge | Вернётся число с плавающей точкой означающее расстояние от точки до линии |
| edgeAddByMouse | Нажать «ЛКМ» с взведённым флагом \_IsEdgeAddMode | Если есть попадание в вершину, то она запомнится и после следующего попадания в вершину добавится ребро с указанным весом |
| deleteMode | Нажать «ЛКМ» с взведённым флагом \_IsDeleteMode | Удалится выбранный элемент |
| OnMouseDown | Нажать кнопку на манипуляторе мышь | Выполнится проверка события нажатия |
| DrawVertices | Вызвать метод OnPaint | Нарисуются вершины графа |
| DrawEdges | Вызвать метод OnPaint | Нарисуются грани графа |
| DrawWindowState | Вызвать метод OnPaint | Нарисуются рамка поля компонента |
| OnPaint | Переопределить любое свойство компонента | Обновится визуальное содержание компонента |

3.2 Инструкция по установке компонента

Для добавления компонента в проект требуется выполнить определенные действия в обозревателе решений. Для этого нужно щелкнуть правой кнопкой мыши на ветке «Ссылки» в структуре текущего проекта, что вызовет контекстное меню. В этом меню необходимо выбрать опцию «Добавить ссылку», как показано на рисунке 3.1

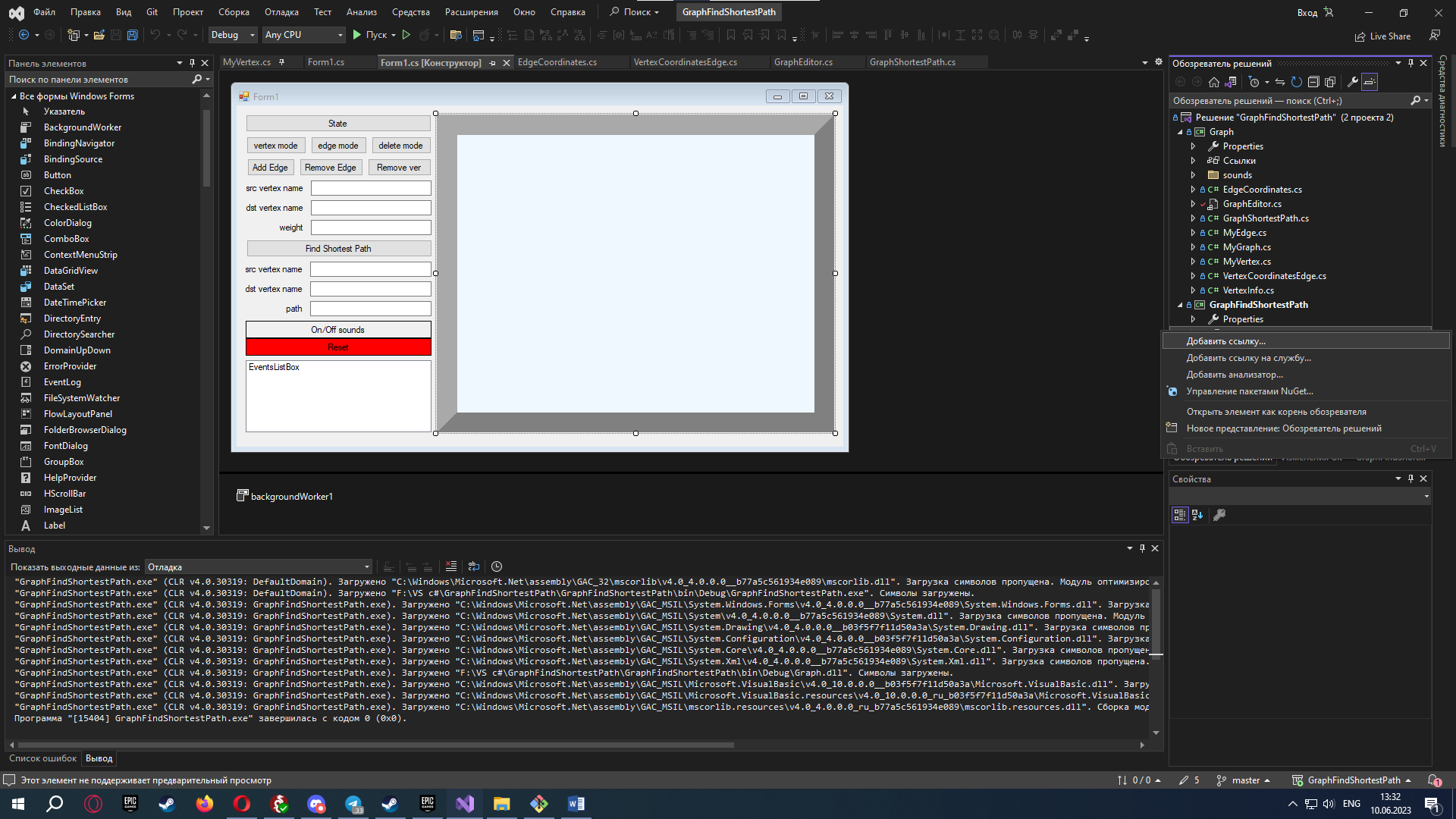


Рисунок. 3.1 - Добавление ссылки

После описанных действий на экране должно появиться окно Менеджера ссылок. Его внешний вид приведен на рисунке 3.2.

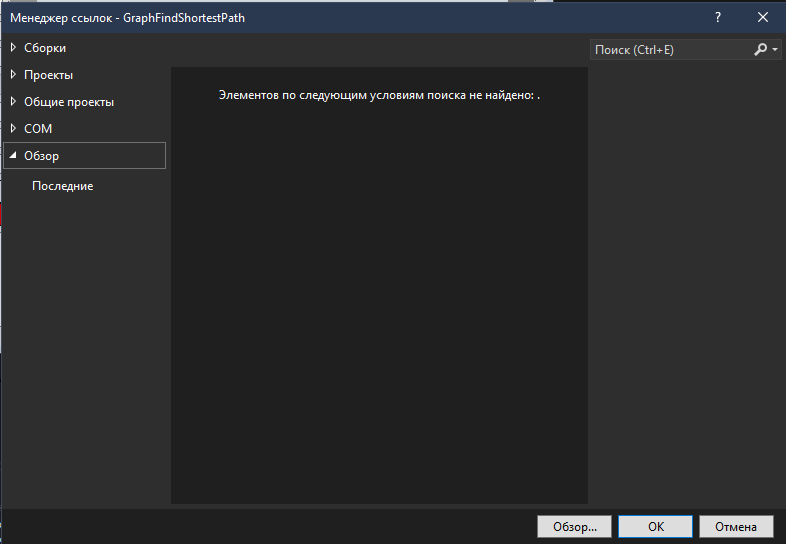


Рисунок 3.2 - Менеджер ссылок

Находясь в менеджере ссылок, нужно нажать на кнопку «Обзор». Сделать это можно из любого раздела. На рисунке 3.2 показан раздел «Обзор» в качестве примера. Находясь в окне выбора файлов необходимо найти файловый пакет компонента, выделить его нажатием левой кнопки мыши и нажать на кнопку «Добавить». Окно для выбора файлов показано на рисунке 3.3.

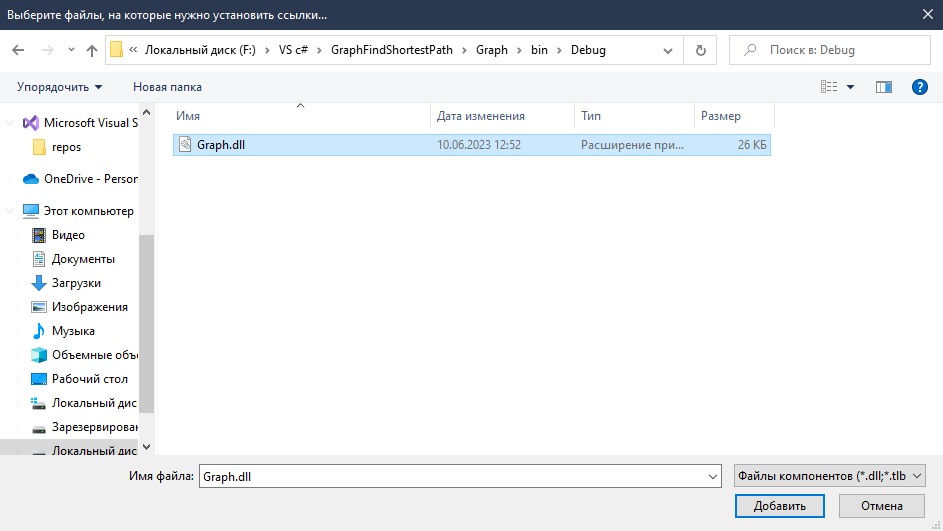


Рисунок 3.3 – Поиск файлового пакета в проводнике

После выбора файлового пакета в панели элементов, необходимо перейти на новую вкладку и вызвать контекстное меню, нажав правую кнопку мыши на этой вкладке. В появившемся меню нужно выбрать команду "Выбрать элементы", как показано на рисунке 3.4. Если группы "New" отсутствуют, то для этого следует щелкнуть правой кнопкой мыши в свободной области панели элементов. После этого алгоритм действий аналогичен описанному выше.

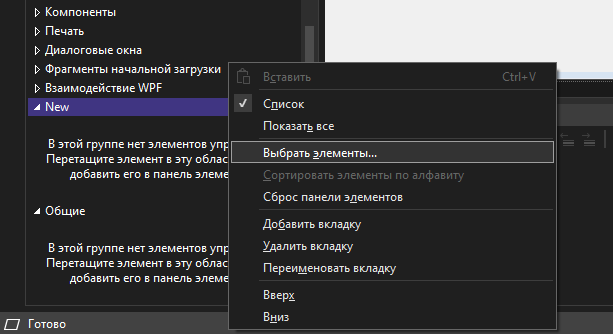


Рисунок 3.4 - Панель элементов

В появившемся окне «Выбор элементов панели элементов», в правом углу необходимо нажать на кнопку «Обзор» как показано на рисунке 3.5.

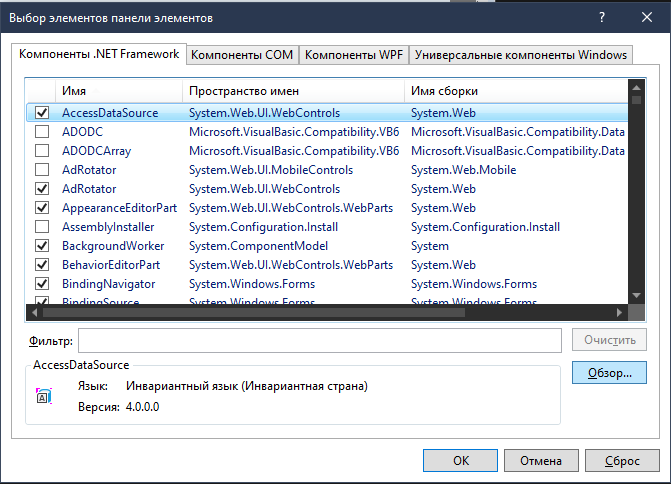


Рисунок 3.5 - Окно «Выбор элементов панели элементов»

Далее в появившемся окне проводника необходимо найти файловый пакет с компонентом и выбрать его, как показано на рисунке 3.6.

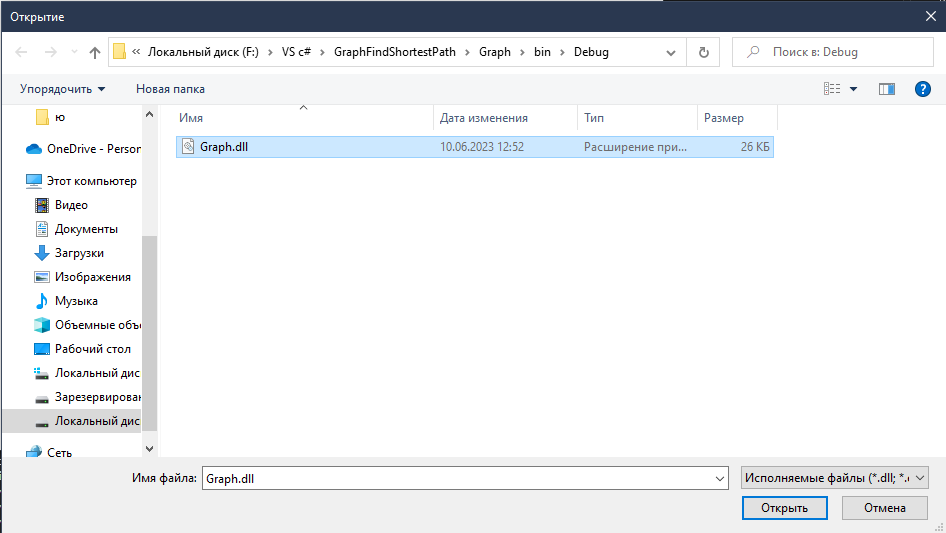


Рисунок 3.6 - Файловый пакет в проводнике

После завершения всех вышеописанных действий, компонент будет доступен для включения в прикладные программы через панель элементов.

Чтобы разместить компонент на форме приложения требуется в панели элементов выбрать вкладку «New». В этой вкладке необходимо выделить компонент и переместить его на форму. Результат представлен на рисунке 3.7. В случае , если не видно данной вкладки, то нужно найти компонент во вкладке «Все элементы».

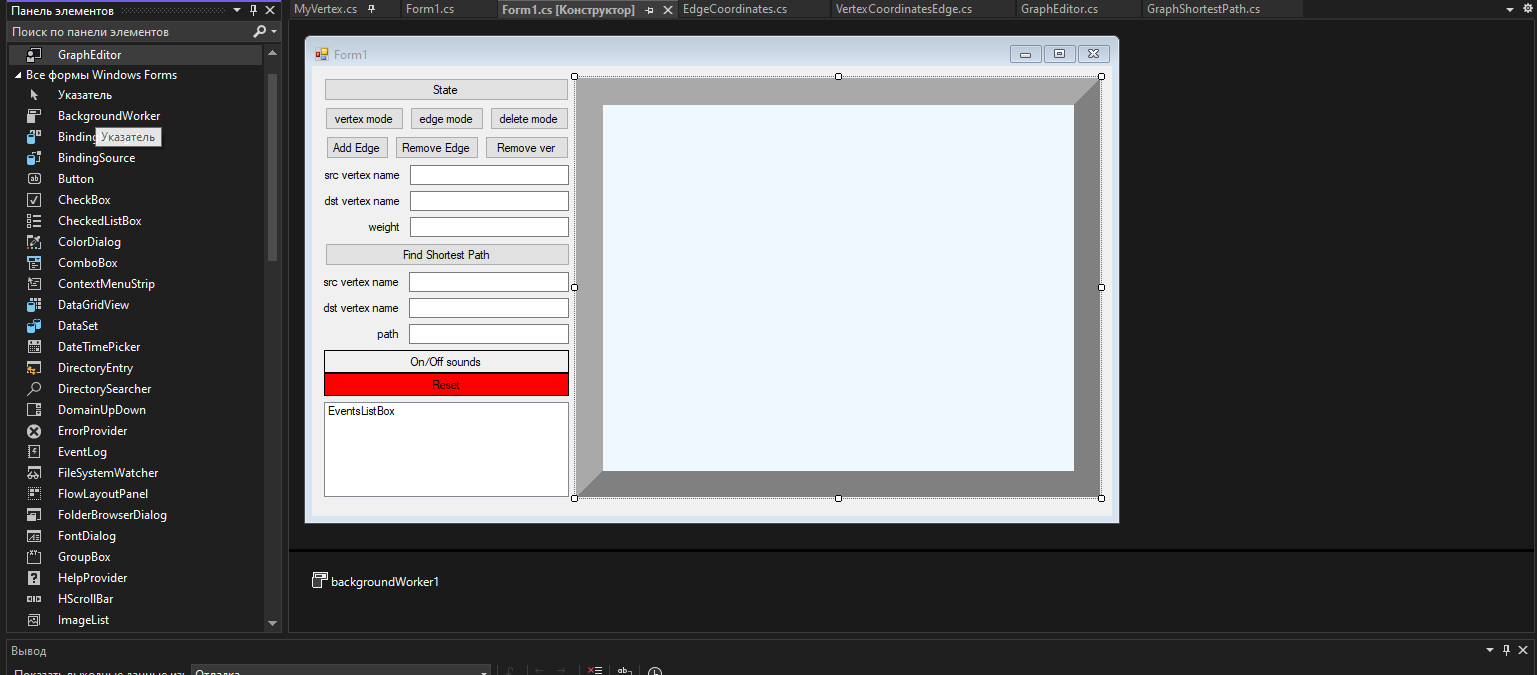


Рисунок 3.7 – Панель элементов

3.3 Инструкция по эксплуатации компонента

Для удобства вся инструкция по эксплуатации визуального компонента сведена в табличный вид. Все свойства компонента перечислены в таблице 3.1

Таблица 3.1 – Свойства компонента

|  |  |  |  |
| --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Назначение |
| ObjState | ObjStates | Чтение/запись | Задаёт состояние поля |
| IsVertexAddMode | bool | Чтение/запись | Включает/выключает режим добавления вершин |
| IsEdgeAddMode | bool | Чтение/запись | Включает/выключает режим добавления граней |
| IsDeleteMode | bool | Чтение/запись | Включает/выключает режим удаления вершин и граней |
| SoundsOn | bool | Чтение/запись | Включает/выключает звук компонента |
| VertexColor | Color | Чтение/запись | Задаёт цвет вершин |
| ShortestPathColor | Color | Чтение/запись | Задаёт цвет кратчайшего пути |
| EdgeColor | Color | Чтение/запись | Задаёт цвет граней |
| TextColor | Color | Чтение/запись | Задаёт цвет текста |
| DarkColor | Color | Чтение/запись | Задаёт тёмный цвет состояния |
| LightColor | Color | Чтение/запись | Задаёт светлый цвет состояния |
| VertexSize | int | Чтение/запись | Задаёт размер вершин |

События компонента представлены в таблице 3.2.

Таблица 3.2 – События компонента

|  |  |
| --- | --- |
| Название события | Назначение |
| OnChangeState | Уведомляет о смене состояния |
| OnVertexAdd | Уведомляет о добавлении вершины |
| OnVertexRemove | Уведомляет о удалении вершины |
| OnEdgeAdd | Уведомляет о добавлении грани |
| OnEdgeRemove | Уведомляет о удалении грани |
| OnFindShortestPath | Уведомляет о поиске кратчайшего пути |
| OnResetGraph | Уведомляет о очистке полотна |

Публичные методы компонента представлены в таблице 3.3.

Таблица 3.3 – Публичные методы компонента

|  |  |  |
| --- | --- | --- |
| Название метода | Входные/выходные параметры | Назначение |
| Reset | Нет входных параметров | Сброс графа |
| addEdge | Имя 1-ой вершины, имя 2-ой вершины, вес | Добавляет грань |
| removeEdge | Имя 1-ой вершины, имя 2-ой вершины | Удаляет грань |
| removeVertex | Имя вершины | Удаляет вершину |
| FindShortestPath | Имя 1-ой вершины, имя 2-ой вершины | Ищет кратчайший путь между вершинами |
| State | Нет входных параметров | Меняет состояние полотна |

4 Раздел охраны труда

Охрана труда - система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда. Научно-технический прогресс внес серьезные изменения в условия производственной деятельности работников умственного труда. Их труд стал более интенсивным, напряженным, требующим значительных затрат умственной, эмоциональной и физической энергии. Это потребовало комплексного решения проблем эргономики, гигиены и организации труда, регламентации режимов труда и отдыха.

Рабочее место - это часть пространства, в котором инженер осуществляет трудовую деятельность, и проводит большую часть рабочего времени. Рабочее место, хорошо приспособленное к трудовой деятельности инженера, правильно и целесообразно организованное, в отношении пространства, формы, размера обеспечивает ему удобное положение при работе и высокую производительность труда при наименьшем физическом и психическом напряжении.

При правильной организации рабочего места производительность труда инженера возрастает с 8 до 20 процентов.

Согласно ГОСТ 12.2.032-78 конструкция рабочего места и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы [13].

Для рабочего места с компьютером, оснащенным жидкокристаллическим или плазменным экраном, рекомендуется обеспечить площадь не менее 4,5 кв. м. Расстояние между столами с мониторами (от задней части одного монитора до экрана другого) должно быть не менее 2 метров. Монитор следует размещать на расстоянии от 50 до 70 см от глаз программиста. Характеристики рабочего стола для сотрудника включают возможность регулировки высоты рабочей поверхности или точно установленную высоту в 72,5 см, ширину стола варьирующуюся от 80, 100, 120 до 140 см, а также глубину рабочей поверхности в 80 или 100 см. Пространство под столешницей (для ног) должно иметь высоту и ширину не менее 50 см, глубина на уровне колен должна быть не менее 45 см, а на уровне вытянутых ног - не менее 65 см. Надлежащее освещение рабочего места является важным аспектом трудовой деятельности, влияющим на работоспособность. Отсутствие должного освещения может вызывать напряжение зрения, снижение внимания и преждевременную усталость. Слишком яркое освещение также нежелательно, поскольку может вызывать ослепление, раздражение и дискомфорт в глазах. При искусственном освещении обычно используются два типа ламп: накаливания и люминесцентные.

Известно, что шум ухудшает условия труда и оказывает вредное воздействие на организм человека. Согласно ГОСТ 12.1.003-88 «Шум для помещений расчетчиков и программистов, уровни шума не должны превышать соответственно: 71, 61, 54, 49, 45, 42, 40, 38 дБ», информация из работы [14].

В процессе эксплуатации компьютерной техники выделяется значительное количество тепла, что может представлять опасность возникновения пожара. Приборы, используемые для обслуживания, устройства электропитания и системы кондиционирования воздуха также могут стать источниками возгорания. Особенно опасны электроизоляционные материалы, применяемые для защиты от механических повреждений электронных компонентов. В связи с этим, зоны, где присутствует компьютерная техника, относятся к категории пожароопасных "В".

В случае пожара людям необходимо немедленно покинуть помещение. В помещениях с компьютерной техникой не рекомендуется использовать воду или пену, поскольку они могут нанести ущерб или полностью вывести из строя дорогостоящее электронное оборудование. Для тушения пожаров рекомендуется применять углекислотные и порошковые огнетушители, которые обладают высокой скоростью тушения, продолжительным действием, а также могут быть безопасно использованы для потушения пожаров в электроустановках. Эти огнетушители являются эффективным средством борьбы с огнем. Воду допускается использовать только во вспомогательных помещениях, где отсутствует электронное оборудование или информация из основной рабочей среды.

Заключение

По итогу работы был разработан класс-компонент для редактирования графов «Редактор графов поиск кратчайшего пути». Компонент является адаптивным и представляет из себя поле для редактирования графов с адаптивной размерностью, которая задаётся, размерами, указанными пользователем компонента. Поле компонента содержит контур с двумя состояниями (выпуклое и вогнутое), также на поле можно добавлять вершины и соединять их гранями. В компоненте присутствует система нахождения кратчайшего пути на графе через алгоритм Дейкстры. В зависимости от действий пользователя в компоненте будет появляться уведомление об одном из событий, предусмотренных в компоненте (Вершина добавлена, Вершина удалена, Грань добавлена, Грань удалена, Изменение состояния, Поиск кратчайшего пути, Граф очищен).

Главными достоинствами разработанного компонента можно считать уникальность компонента, систему добавления вершин и граней путём нажатия ЛКМ на манипуляторе мышь и звуковое сопровождение действий пользователя.

У компонента есть перспективы развития, а именно, внедрение анимации поиска кратчайшего пути, вершин и граней, а также потом можно будет добавить различные алгоритмы поиска кратчайшего пути.

Разработанный класс можно использовать для решения различных задач поиска кратчайшего пути в транспортных системах, сетях связи и телекоммуникациях. Так же компонент можно использовать в учебных целях. Компонент может проиллюстрировать различные методы рисования, объектно-ориентированный подход. В компоненте используются различного рода алгоритмы, на которые можно опираться при разработке других компонентов.

Список используемых источников

1. <https://itanddigital.ru/historycoding>
2. <https://programm.top/c-sharp/algorithm/search/dijkstra-algorithm/>
3. <https://programm.top/c-sharp/data-structures/graph/>
4. <https://sfxr.me>
5. <https://stackoverflow.com/questions/3502311/how-to-play-a-sound-in-c-net>
6. <https://studopedia.ru/19_420644_istoriya-programmirovaniya-i-algoritmicheskih-yazikov.html#:~:text=Первые%20алгоритмические%20языки%20эпохи%20ЭВМ,комитете%20под%20руководством%20Питера%20Наура>
7. <https://www.garshin.ru/it/computer-languages/object-pioners.html>
8. <https://hit-tool.com/code-editors/468-microsoft-visual-studio-community.html>
9. <https://webformyself.com/bitva-titanov-open-source-netbeans-ili-eclipse/>
10. <https://devdelphi.ru/blog/c-protiv-python-kakoi-variant-luchshe>
11. <https://imba.ru/blog-1/chto-takoe-java-obiasniaem-dlia-novichkov-63>
12. <https://mdahshan.github.io/dijkstra/>
13. <https://otherreferats.allbest.ru/life/00130223_0.html>
14. https://дцо.рф/opisanie-rabochego-mesta-programmista-na-predpriyatii/

Приложение А

Программный кодкласса Редактор графов

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.VisualStyles;

using Microsoft.VisualBasic;

namespace Graph

{

public class GraphEditor : Control

{

/// <summary>

/// Конструктор компонента

/// </summary>

public GraphEditor() : base()

{

BackColor = Color.AliceBlue;

\_VertexColor = Color.Black; EdgeColor = Color.Gray;

\_TextColor = Color.Red;

\_ShortestPathColor = Color.Green;

\_DarkColor = Color.DarkGray;

\_LightColor = Color.Gray;

\_graph = new MyGraph();

\_VertexCount = 0;

\_SoundsOn = true;

vertexCoordinates = new List<VertexCoordinatesEdge>();

edgeCoordinates = new List<EdgeCoordinates>();

InitSounds();

}

//внутренние переменные

private MyGraph \_graph;

private GraphShortestPath \_shortestPath;

private List<VertexCoordinatesEdge> vertexCoordinates;

private List<EdgeCoordinates> edgeCoordinates;

private System.Media.SoundPlayer addVertexOrEdge;

private System.Media.SoundPlayer changeState;

private System.Media.SoundPlayer findAction;

private System.Media.SoundPlayer prohibitionOfAction;

private System.Media.SoundPlayer resetAction;

private int \_selectedVertex = -1;

private int startNodeIndex = -1;

protected int \_VertexSize;

protected int \_VertexCount;

//свойства

protected Color \_VertexColor;

protected Color \_EdgeColor;

protected Color \_TextColor;

protected Color \_DarkColor;

protected Color \_LightColor;

protected Color \_ShortestPathColor;

private ObjStates \_ObjState;

protected bool \_IsVertexAddMode;

protected bool \_SoundsOn;

protected bool \_IsEdgeAddMode;

protected bool \_IsDeleteMode;

public enum ObjStates

{

osConvex,

osConcavity

}

public virtual bool IsVertexAddMode

{

get { return \_IsVertexAddMode; }

set

{

if (\_IsVertexAddMode != value)

{

\_IsVertexAddMode = value;

\_IsEdgeAddMode = false;

\_IsDeleteMode = false;

}

}

}

public virtual bool IsEdgeAddMode

{

get { return \_IsEdgeAddMode; }

set

{

if (\_IsEdgeAddMode != value)

{

\_IsEdgeAddMode = value;

\_IsVertexAddMode = false;

\_IsDeleteMode = false;

}

}

}

public virtual bool IsDeleteMode

{

get { return \_IsDeleteMode; }

set

{

if (\_IsDeleteMode != value)

{

\_IsDeleteMode = value;

\_IsVertexAddMode = false;

\_IsEdgeAddMode = false;

}

}

}

public virtual bool SoundsOn

{

get { return \_SoundsOn; }

set

{

if (\_SoundsOn != value)

{

\_SoundsOn = value;

}

}

}

public virtual ObjStates ObjState

{

get

{

return \_ObjState;

}

set

{

if (value != \_ObjState)

{

\_ObjState = value;

Invalidate();

}

}

}

public virtual Color VertexColor

{

get

{

return \_VertexColor;

}

set

{

if (value != \_VertexColor)

{

\_VertexColor = value;

Invalidate();

}

}

}

public virtual Color ShortestPathColor

{

get

{

return \_ShortestPathColor;

}

set

{

if (value != \_ShortestPathColor)

{

\_ShortestPathColor = value;

Invalidate();

}

}

}

public virtual Color EdgeColor

{

get

{

return \_EdgeColor;

}

set

{

if (value != \_EdgeColor)

{

\_EdgeColor = value;

Invalidate();

}

}

}

public virtual Color TextColor

{

get

{

return \_TextColor;

}

set

{

if (value != \_TextColor)

{

\_TextColor = value;

Invalidate();

}

}

}

public virtual Color DarkColor

{

get

{

return \_DarkColor;

}

set

{

if (value != \_DarkColor)

{

\_DarkColor = value;

Invalidate();

}

}

}

public virtual Color LightColor

{

get

{

return \_LightColor;

}

set

{

if (value != \_LightColor)

{

\_LightColor = value;

Invalidate();

}

}

}

/// <summary>

/// Метод сброса графа

/// </summary>

public virtual void Reset()

{

\_graph = new MyGraph();

\_shortestPath = new GraphShortestPath(\_graph);

vertexCoordinates = new List<VertexCoordinatesEdge>();

edgeCoordinates = new List<EdgeCoordinates>();

\_VertexCount = 0;

OnResetGraph();

if (SoundsOn)

resetAction.Play();

Invalidate();

}

/// <summary>

/// Метод добавления грани с графа

/// </summary>

public virtual void addEdge(string src, string dst, int weight)

{

if (src == dst)

{

if (SoundsOn) prohibitionOfAction.Play();

return;

}

if (\_graph.FindVertex(src) != null && \_graph.FindVertex(dst) != null)

{

lock (edgeCoordinates)

{

{

foreach (EdgeCoordinates edge in edgeCoordinates)

{

if ((edge.src == src && edge.dst == dst) || (edge.src == dst && edge.dst == src))

{

if (SoundsOn)

prohibitionOfAction.Play();

return;

}

}

}

int x1 = 0;

int x2 = 0;

int y1 = 0;

int y2 = 0;

foreach (VertexCoordinatesEdge vertex in vertexCoordinates)

{

if (vertex.name == src) { x1 = vertex.x; y1 = vertex.y; }

if (vertex.name == dst) { x2 = vertex.x; y2 = vertex.y; }

}

edgeCoordinates.Add(new EdgeCoordinates(x1, x2, y1, y2, src, dst, weight.ToString(), false));

\_graph.AddEdge(src, dst, weight);

if (SoundsOn)

addVertexOrEdge.Play();

OnEdgeAdd();

}

Invalidate();

}

}

/// <summary>

/// Метод удаления грани на граф

/// <param name="src">имя первой вершины</param>

/// <param name="dst">имя второй вершины</param>

/// </summary>

public virtual void removeEdge(string src, string dst)

{

if (src == dst)

{

if (SoundsOn) prohibitionOfAction.Play();

return;

}

if (src != null && dst != null)

{

EdgeCoordinates buf = null;

for (int i = 0; i <= edgeCoordinates.Count; i++)

{

EdgeCoordinates edge = edgeCoordinates[i];

if ((edge.src == src && edge.dst == dst) || (edge.src == dst && edge.dst == src))

{

edgeCoordinates.RemoveAt(i);

if (SoundsOn)

resetAction.Play();

break;

}

}

//edgeCoordinates.Remove(buf);

Invalidate();

\_graph.RemoveEdge(src, dst);

}

}

/// <summary>

/// Метод удаления вершины с графа

/// <param name="vertexName">имя вершины</param>

/// </summary>

public virtual void removeVertex(string vertexName)

{

removeEdgesForVertex(vertexName);

var vertex = \_graph.FindVertex(vertexName);

if (vertex == null)

{

if (SoundsOn)

prohibitionOfAction.Play();

return;

}

VertexCoordinatesEdge ver = null;

foreach (VertexCoordinatesEdge v in vertexCoordinates)

{

if (v.name == vertexName)

{

ver = v; break;

}

}

vertexCoordinates.Remove(ver);

\_graph.RemoveVertex(vertex);

if (SoundsOn)

resetAction.Play();

ResetShortestPath();

Invalidate();

}

/// <summary>

/// Метод поиска кратчайшего пути в графе

/// </summary>

public virtual string FindShortestPath(string src, string dst)

{

ResetShortestPath();

\_shortestPath = new GraphShortestPath(\_graph);

var vertices = \_shortestPath.FindShortestPath(src, dst);

SetShortestPath(vertices);

if (vertices.Count > 0) OnFindShortestPath();

Invalidate();

string buf = "";

foreach (string vertex in vertices)

{

if (buf != "") buf += ", ";

buf += vertex;

}

if (SoundsOn)

if (buf != "") findAction.Play(); else prohibitionOfAction.Play();

return buf;

}

/// <summary>

/// Метод смены состояний полотна

/// </summary>

public virtual void State()

{

int I = (int)\_ObjState + 1;

I = I > 1 ? 0 : I;

ObjState = (ObjStates)I;

if (SoundsOn)

changeState.Play();

OnChangeState();

}

/// <summary>

/// Метод инициализации звуков

/// </summary>

protected virtual void InitSounds()

{

var exePath = Environment.CurrentDirectory;

addVertexOrEdge = new System.Media.SoundPlayer(Path.Combine(exePath, @"sounds\", "addVertexOrEdge.wav"));

changeState = new System.Media.SoundPlayer(Path.Combine(exePath, @"sounds\", "changeState.wav"));

findAction = new System.Media.SoundPlayer(Path.Combine(exePath, @"sounds\", "findAction.wav"));

prohibitionOfAction = new System.Media.SoundPlayer(Path.Combine(exePath, @"sounds\", "prohibitionOfAction.wav"));

resetAction = new System.Media.SoundPlayer(Path.Combine(exePath, @"sounds\", "reset.wav"));

}

/// <summary>

/// Метод добавления вершины на граф

/// </summary>

/// <param name="x">x координата вершины</param>

/// <param name="y">y координата вершины</param>

protected virtual void addVertex(int x, int y)

{

if ((x <= Math.Min(Height, Width) / 15) || (x >= Width - Math.Min(Height, Width) / 15) || (y <= Math.Min(Height, Width) / 15) || (y >= Height - Math.Min(Height, Width) / 15))

{

if (SoundsOn)

prohibitionOfAction.Play();

return;

}

foreach (VertexCoordinatesEdge v in vertexCoordinates)

{

if (((x <= v.x + \_VertexSize && x >= v.x - \_VertexSize) && (y <= v.y + \_VertexSize && y >= v.y - \_VertexSize)))

{

if (SoundsOn)

prohibitionOfAction.Play();

return;

}

}

\_VertexCount++;

vertexCoordinates.Add(new VertexCoordinatesEdge(\_VertexCount.ToString(), x, y, false));

\_graph.AddVertex(\_VertexCount.ToString());

OnVertexAdd();

//\_IsVertexAddMode = false;

if (SoundsOn)

addVertexOrEdge.Play();

Invalidate();

}

/// <summary>

/// Метод удаления всех граней для вершины

/// <param name="src">имя вершины</param>

/// </summary>

protected virtual void removeEdgesForVertex(string src)

{

if (\_graph.FindVertex(src) != null)

{

for (int i = 0; i <= \_graph.FindVertex(src).edges.Count; i++)

{

try

{

var buf = edgeCoordinates;

foreach (EdgeCoordinates edge in edgeCoordinates)

{

if (edge.src == src || edge.dst == src)

{

buf.Remove(edge);

}

}

edgeCoordinates = buf;

Invalidate();

\_graph.RemoveEdge(src, "");

}

catch (Exception e) { }

}

}

}

/// <summary>

/// Метод сброса флагов кратчайшего пути для вершин и граней

/// </summary>

protected virtual void ResetShortestPath()

{

lock (vertexCoordinates)

{

foreach (var ver in vertexCoordinates)

{

ver.isShortestPath = false;

}

}

lock (edgeCoordinates)

{

foreach (var e in edgeCoordinates)

{

e.isShortestPath = false;

}

}

}

/// <summary>

/// Метод установки флагов кратчайшего пути для вершин и граней

/// </summary>

protected virtual void SetShortestPath(List<string> vertices)

{

foreach (var vertexName in vertices)

{

foreach (var ver in vertexCoordinates)

{

if (ver.name == vertexName)

{

ver.isShortestPath = true;

}

}

}

for (int i = 0; i < vertices.Count - 1; i++)

{

foreach (var e in edgeCoordinates)

{

if ((e.src == vertices[i] && e.dst == vertices[i + 1]) || (e.dst == vertices[i] && e.src == vertices[i + 1]))

{

e.isShortestPath = true;

}

}

}

Invalidate();

}

/// <summary>

/// Метод масштабирования полотна

/// </summary>

protected override void SetBoundsCore(int x, int y, int width, int height, BoundsSpecified specified)

{

if (width < 400)

{

width = 400;

}

if (height < 400)

{

height = 400;

}

\_VertexSize = 20;

base.SetBoundsCore(x, y, width, height, specified);

Invalidate();

}

/// <summary>

/// Метод отрисовки полотна

/// </summary>

protected override void OnPaint(PaintEventArgs e)

{

Brush VertexBrush = new SolidBrush(\_VertexColor);

Brush ShortestPathBrush = new SolidBrush(\_ShortestPathColor);

Pen ShortestPathPen = new Pen(\_ShortestPathColor, 5);

Pen EdgePen = new Pen(\_EdgeColor, 5);

Pen LightPen = new Pen(\_LightColor);

Pen DarkPen = new Pen(\_DarkColor);

int FontSize = (int)Math.Round((\_VertexSize - 16) \* 92 / e.Graphics.DpiY);

if (FontSize <= 12)

{

FontSize = 12;

}

Font font = new Font("Arial", FontSize);

SolidBrush TextBrush = new SolidBrush(\_TextColor);

StringFormat Fmt = new StringFormat();

Fmt.LineAlignment = StringAlignment.Center;

Fmt.Alignment = StringAlignment.Center;

DrawWindowState(e, DarkPen, LightPen);

DrawEdges(e, EdgePen, ShortestPathPen, font, TextBrush, Fmt);

DrawVertices(e, VertexBrush, ShortestPathBrush, font, TextBrush, Fmt);

base.OnPaint(e);

}

/// <summary>

/// Метод отрисовки выпуклостей/вогнутостей

/// </summary>

/// <param name="e">Графика</param>

/// <param name="DarkPen">Тёмный карандаш</param>

/// <param name="LightPen">Светлый карандаш</param>

protected virtual void DrawWindowState(PaintEventArgs e, Pen DarkPen, Pen LightPen)

{

for (int i = 0; i < Math.Min(Height, Width) / 15; i++)

{

Point[] D =

{

new Point(Width-i,i),

new Point(i,i),

new Point(i,Height-i)

};

Point[] L =

{

new Point(Width-i,i),

new Point(Width-i,Height-i),

new Point(i,Height-i)

};

if ((int)ObjState == 0)

{

e.Graphics.DrawLines(DarkPen, D);

e.Graphics.DrawLines(LightPen, L);

}

else

{

e.Graphics.DrawLines(DarkPen, L);

e.Graphics.DrawLines(LightPen, D);

}

}

}

/// <summary>

/// Метод отрисовки граней

/// </summary>

/// <param name="e">Графика</param>

/// <param name="EdgePen">Карандаш для отрисовки граней</param>

/// <param name="ShortestPathPen">Карандаш для отрисовки граней кратчайшего пути</param>

/// <param name="font">Шрифт</param>

/// <param name="TextBrush">Кисть для отрисовки текста</param>

/// <param name="Fmt">Формат строки</param>

protected virtual void DrawEdges(PaintEventArgs e, Pen EdgePen, Pen ShortestPathPen, Font font, SolidBrush TextBrush, StringFormat Fmt)

{

Fmt.LineAlignment = StringAlignment.Center;

foreach (var edge in edgeCoordinates)

{

if (edge != null)

{

if (edge.x1 - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

edge.x1 = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (edge.x1 + \_VertexSize / 2 > Width - Math.Min(Height, Width) / 15)

{

edge.x1 = Width - Math.Min(Height, Width) / 15 - \_VertexSize / 2;

}

}

if (edge.y1 - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

edge.y1 = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (edge.y1 + \_VertexSize / 2 > Height - Math.Min(Height, Width) / 15)

{

edge.y1 = Height - Math.Min(Height, Width) / 15 - \_VertexSize / 2;

}

}

if (edge.x2 - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

edge.x2 = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (edge.x2 + \_VertexSize / 2 > Width - Math.Min(Height, Width) / 15)

{

edge.x2 = Width - Math.Min(Height, Width) / 15 - \_VertexSize / 2;

}

}

if (edge.y2 - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

edge.y2 = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (edge.y2 + \_VertexSize / 2 > Height - Math.Min(Height, Width) / 15)

{

edge.y2 = Height - \_VertexSize / 2 - Math.Min(Height, Width) / 15;

}

}

if (!edge.isShortestPath)

{

e.Graphics.DrawLine(EdgePen, edge.x1, edge.y1, edge.x2, edge.y2);

}

else

{

e.Graphics.DrawLine(ShortestPathPen, edge.x1, edge.y1, edge.x2, edge.y2);

}

int x = (Math.Min(edge.x2, edge.x1) - \_VertexSize) + ((Math.Max(edge.x2, edge.x1) - \_VertexSize) - (Math.Min(edge.x1, edge.x2) - \_VertexSize)) / 2;

int y = (Math.Min(edge.y2, edge.y1) - \_VertexSize / 2) + ((Math.Max(edge.y2, edge.y1) - \_VertexSize / 2) - (Math.Min(edge.y1, edge.y2) - \_VertexSize / 2)) / 2;

RectangleF Rect = new RectangleF(x, y, \_VertexSize \* 2, \_VertexSize);

String S = edge.weight;

e.Graphics.DrawString(S, font, TextBrush, Rect, Fmt);

}

}

}

/// <summary>

/// Метод отрисовки граней

/// </summary>

/// <param name="e">Графика</param>

/// <param name="VertexBrush">Кисть для отрисовки вершин</param>

/// <param name="ShortestPathBrush">Кисть для отрисовки вершин кратчайшего пути</param>

/// <param name="font">Шрифт</param>

/// <param name="TextBrush">Кисть для отрисовки текста</param>

/// <param name="Fmt">Формат строки</param>

protected virtual void DrawVertices(PaintEventArgs e, Brush VertexBrush, Brush ShortestPathBrush, Font font, SolidBrush TextBrush, StringFormat Fmt)

{

Fmt.LineAlignment = StringAlignment.Center;

foreach (var vertex in vertexCoordinates)

{

if (vertex != null)

{

if (vertex.x - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

vertex.x = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (vertex.x + \_VertexSize / 2 > Width - Math.Min(Height, Width) / 15)

{

vertex.x = Width - Math.Min(Height, Width) / 15 - \_VertexSize / 2;

}

}

if (vertex.y - \_VertexSize / 2 < Math.Min(Height, Width) / 15)

{

vertex.y = Math.Min(Height, Width) / 15 + \_VertexSize / 2;

}

else

{

if (vertex.y + \_VertexSize / 2 > Height - Math.Min(Height, Width) / 15)

{

vertex.y = Height - Math.Min(Height, Width) / 15 - \_VertexSize / 2;

}

}

if (vertex.isShortestPath)

{

e.Graphics.FillEllipse(ShortestPathBrush, vertex.x - \_VertexSize / 2, vertex.y - \_VertexSize / 2, \_VertexSize, \_VertexSize);

}

else

{

e.Graphics.FillEllipse(VertexBrush, vertex.x - \_VertexSize / 2, vertex.y - \_VertexSize / 2, \_VertexSize, \_VertexSize);

}

RectangleF Rect = new RectangleF(vertex.x - \_VertexSize, vertex.y - \_VertexSize / 2, \_VertexSize \* 2, \_VertexSize);

String S = vertex.name;

e.Graphics.DrawString(S, font, TextBrush, Rect, Fmt);

}

}

}

/// <summary>

/// Слушатель нажатия кнопок мыши

/// </summary>

protected override void OnMouseDown(MouseEventArgs e)

{

int x = e.X;

int y = e.Y;

if (e.Button == MouseButtons.Left)

{

if (\_IsVertexAddMode)

{

addVertex(x, y);

}

else

{

if (\_IsEdgeAddMode)

{

edgeAddByMouse(e);

}

else

{

if (\_IsDeleteMode)

{

deleteMode(e);

}

}

}

base.OnMouseDown(e);

}

Invalidate();

}

//Метод удаления ребра или вершины мышкой

private void deleteMode(MouseEventArgs e)

{

// Проверяем, попали ли мы в какой-либо элемент (вершину или ребро)

for (int i = 0; i < vertexCoordinates.Count; i++)

{

Rectangle nodeBounds = new Rectangle(vertexCoordinates[i].x - 10, vertexCoordinates[i].y - 10, 20, 20);

if (nodeBounds.Contains(e.Location))

{

// Подсвечиваем выбранный элемент

\_selectedVertex = i;

Invalidate(); // Перерисовываем компонент

// Отображаем предупреждение об удалении

DialogResult result = MessageBox.Show("Вы уверены, что хотите удалить эту вершину и связанные с ней ребра?", "Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

// Удаляем вершину и связанные ребра

removeVertex(vertexCoordinates[i].name);

}

return;

}

}

// Проверяем, попали ли мы в какое-либо ребро

for (int i = 0; i < edgeCoordinates.Count; i++)

{

Tuple<Point, Point> edge;

if (IsPointOnEdge(e.Location, edgeCoordinates[i].x1, edgeCoordinates[i].y1, edgeCoordinates[i].x2, edgeCoordinates[i].y2))

{

// Отображаем предупреждение об удалении

DialogResult result = MessageBox.Show("Вы уверены, что хотите удалить это ребро?", "Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

// Удаляем ребро

removeEdge(edgeCoordinates[i].src, edgeCoordinates[i].dst);

}

return;

}

}

}

// Метод для проверки, попадает ли точка на ребро

private bool IsPointOnEdge(Point point, int x1, int y1, int x2, int y2)

{

const int tolerance = 3; // Допустимое отклонение от ребра

float distance = DistancePointToLine(point, x1, y1, x2, y2);

return distance <= tolerance;

}

// Метод для вычисления расстояния от точки до линии

private float DistancePointToLine(Point point, int x1, int y1, int x2, int y2)

{

float a = point.X - x1;

float b = point.Y - y1;

float c = x2 - x1;

float d = y2 - y1;

float dot = a \* c + b \* d;

float lenSq = c \* c + d \* d;

float param = dot / lenSq;

float xx, yy;

if (param < 0)

{

xx = x1;

yy = y1;

}

else if (param > 1)

{

xx = x2;

yy = y2;

}

else

{

xx = x1 + param \* c;

yy = y1 + param \* d;

}

float dx = point.X - xx;

float dy = point.Y - yy;

return (float)Math.Sqrt(dx \* dx + dy \* dy);

}

//метод добавления грани мышкой

private void edgeAddByMouse(MouseEventArgs e)

{

// Проверяем, попали ли мы в какую-либо вершину

for (int i = 0; i < vertexCoordinates.Count; i++)

{

Rectangle nodeBounds = new Rectangle(vertexCoordinates[i].x - 10, vertexCoordinates[i].y - 10, 20, 20);

if (nodeBounds.Contains(e.Location))

{

// Проверяем, является ли это начальной вершиной

if (\_selectedVertex == -1)

{

// Выбираем начальную вершину и меняем ее цвет

startNodeIndex = i;

\_selectedVertex = i;

Invalidate(); // Перерисовываем компонент

}

else

{

// Создаем ребро между начальной и конечной вершинами

int s = 0;

try

{

s = int.Parse(Interaction.InputBox("Введите вес: "));

}

catch (Exception ex)

{

}

addEdge(vertexCoordinates[startNodeIndex].name, vertexCoordinates[i].name, s);

// Сбрасываем начальную вершину

startNodeIndex = -1;

\_selectedVertex = -1;

Invalidate(); // Перерисовываем компонент

}

return;

}

}

}

protected override CreateParams CreateParams

{

get

{

CreateParams P = base.CreateParams;

P.ExStyle = P.ExStyle | 0x02000000;

return P;

}

}

//События

protected event EventHandler \_OnChangeState;

protected event EventHandler \_OnVertexAdd;

protected event EventHandler \_OnEdgeAdd;

protected event EventHandler \_OnFindShortestPath;

protected event EventHandler \_OnReset;

public event EventHandler ChangeState

{

add

{

\_OnChangeState += value;

}

remove

{

\_OnChangeState -= value;

}

}

public event EventHandler VertexAdd

{

add

{

\_OnVertexAdd += value;

}

remove

{

\_OnVertexAdd -= value;

}

}

public event EventHandler EdgeAdd

{

add

{

\_OnEdgeAdd += value;

}

remove

{

\_OnEdgeAdd -= value;

}

}

public event EventHandler FindedShortestPath

{

add

{

\_OnFindShortestPath += value;

}

remove

{

\_OnFindShortestPath -= value;

}

}

public event EventHandler ResetGraph

{

add

{

\_OnReset += value;

}

remove

{

\_OnReset -= value;

}

}

protected void OnChangeState()

{

if (\_OnChangeState != null)

{

\_OnChangeState.Invoke(this, new EventArgs());

}

}

protected void OnVertexAdd()

{

if (\_OnVertexAdd != null)

{

\_OnVertexAdd.Invoke(this, new EventArgs());

}

}

protected void OnEdgeAdd()

{

if (\_OnEdgeAdd != null)

{

\_OnEdgeAdd.Invoke(this, new EventArgs());

}

}

protected void OnFindShortestPath()

{

if (\_OnFindShortestPath != null)

{

\_OnFindShortestPath.Invoke(this, new EventArgs());

}

}

protected void OnResetGraph()

{

if (\_OnReset != null)

{

\_OnReset.Invoke(this, new EventArgs());

}

}

}

}

Программный код класса алгоритма Дектрейсы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Graph

{

public class GraphShortestPath

{

// Конструктор

public GraphShortestPath(MyGraph graph)

{

\_graph = graph;

}

private MyGraph \_graph;

private List<VertexInfo> \_infos;

// Инициализация информации

private void InitInfo()

{

\_infos = new List<VertexInfo>();

foreach (MyVertex vertex in \_graph.Vertices)

{

\_infos.Add(new VertexInfo(vertex));

}

}

// Получение информации о вершине

private VertexInfo GetVertexInfo(MyVertex v)

{

foreach(VertexInfo info in \_infos)

{

if (info.vertex.Equals(v))

{

return info;

}

}

return null;

}

// Поиск непосещённой вершины с наименьшим весом рёбер

public VertexInfo FindUnvisitedVertexMinSum()

{

int minValue = int.MaxValue;

VertexInfo minVertexInfo = null;

foreach (VertexInfo info in \_infos)

{

if(info.isUnvisited && info.edgesWeightSum < minValue)

{

minVertexInfo = info;

minValue = info.edgesWeightSum;

}

}

return minVertexInfo;

}

// Поиск кратчайшего пути

public List<string> FindShortestPath(MyVertex src, MyVertex dst)

{

InitInfo();

if (dst == null) return new List<string>();

if (src == null) return new List<string>();

var first = GetVertexInfo(src);

first.edgesWeightSum = 0;

while (true)

{

var current = FindUnvisitedVertexMinSum();

if (current == null)

{

break;

}

SetSumToNextVertex(current);

}

return GetPath(src, dst);

}

// Поиск кратчайшего пути

public List<string> FindShortestPath(string src, string dst)

{

if (dst == null) return new List<string>();

if (src == null) return new List<string>();

return FindShortestPath(\_graph.FindVertex(src), \_graph.FindVertex(dst));

}

// Вычисление суммы весов рёбер для следующей вершины

private void SetSumToNextVertex(VertexInfo info)

{

info.isUnvisited = false;

foreach(var edge in info.vertex.edges)

{

var nextInfo = GetVertexInfo(edge.connectedVertex);

var sum = info.edgesWeightSum + edge.weight;

if(sum < nextInfo.edgesWeightSum)

{

nextInfo.edgesWeightSum = sum;

nextInfo.previousVertex = info.vertex;

}

}

}

// Формирование пути в формате листа из строк

private List<string> GetPath(MyVertex src, MyVertex dst)

{

if (dst == null) return new List<string>();

if(src == null) return new List<string>();

var path = dst.ToString();

while (src != dst)

{

try

{

dst = GetVertexInfo(dst).previousVertex;

if (dst == null) return new List<string>();

path = dst.ToString() + "?" + path;

}

catch

{

return new List<string>();

}

}

string buf = "";

List<string> list = new List<string>();

for(int i=0; i < path.Length;i++)

{

if (path[i]!='?')

{

buf += path[i];

}

else

{

list.Add(buf);

buf = "";

}

}

if (buf != "")

{

list.Add(buf);

}

return list;

}

}

}

Программный код класса