Многопользовательский графический редактор

Исследовательская работа
ученика 11“А” класса лицея №281
Еникеева Дмитрия.

Введение

Существуют много графических редакторов, но все они однотипные. Я, вдохновившись Google docs, в котором люди могут вместе работать над общими документами, поставил перед собой задачу реализации многопользовательского графического редактора. В отличие от обычного, многопользовательский редактор позволяет нескольким людям одновременно рисовать одну и ту же картину. Рисовать с кем-то одновременно — это не только весело, но и хороший способ обучаться рисованию. Ты можешь смотреть как другой художник рисует в реальном времени, а также этот же художник может тебя направлять, обучать, рассказывать полезные вещи и сразу же их демонстрировать.

Я разработал свое приложение, чтобы оно было удобным и понятным любому.

Векторное рисование

Существует два вида графических редакторов: растровые[1] и векторные[2]. Основное различие между ними состоит в том, что растровый позволяет редактировать изображение как набор точек, а векторный — как набор геометрических фигур.

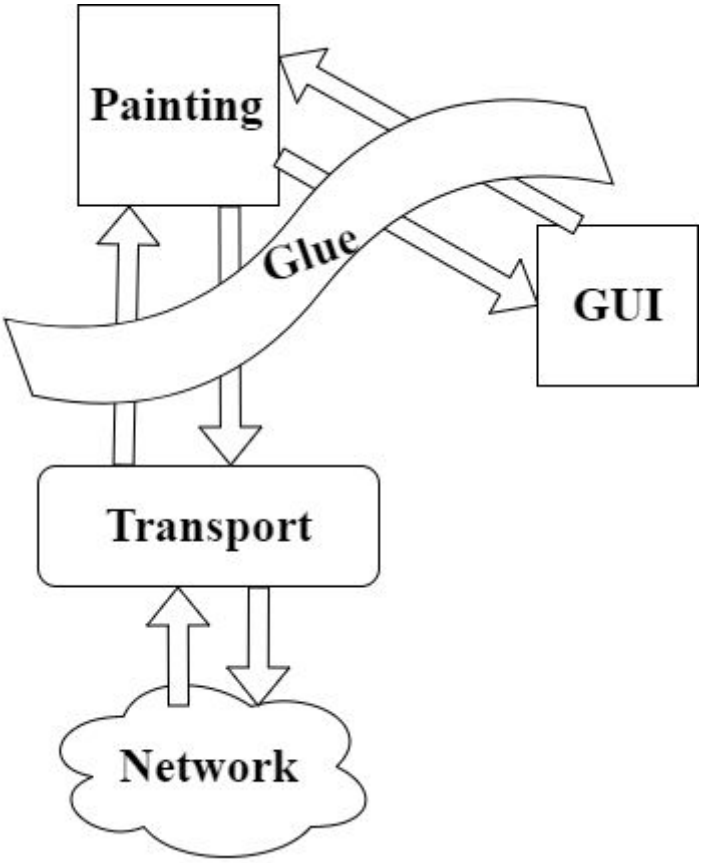
Векторные редакторы позволяют неограниченно масштабировать изображение, а нарисованные элементы не влияют друг на друга, в то время как растровые редакторы предоставляют больше разнообразных инструментов и лучше подходят для редактирования фотографий.

Я решил написать векторный редактор, так как для поставленных целей видеть процесс рисования каждого пользователя важнее, чем растеризованный результат.

Структура программы

Редактор написан на **C++11** с помощью фреймворка **Qt5**. Отличительными особенностями Qt являются сигналы и слоты[3] — возможность посылать события и подписываться на них. Это позволяет различным объектам в программе взаимодействовать не зная друг о друге.

Клиент состоит из четырех блоков: внутреннее представление изображения, пользовательский интерфейс, сетевой транспорт и связующий код.

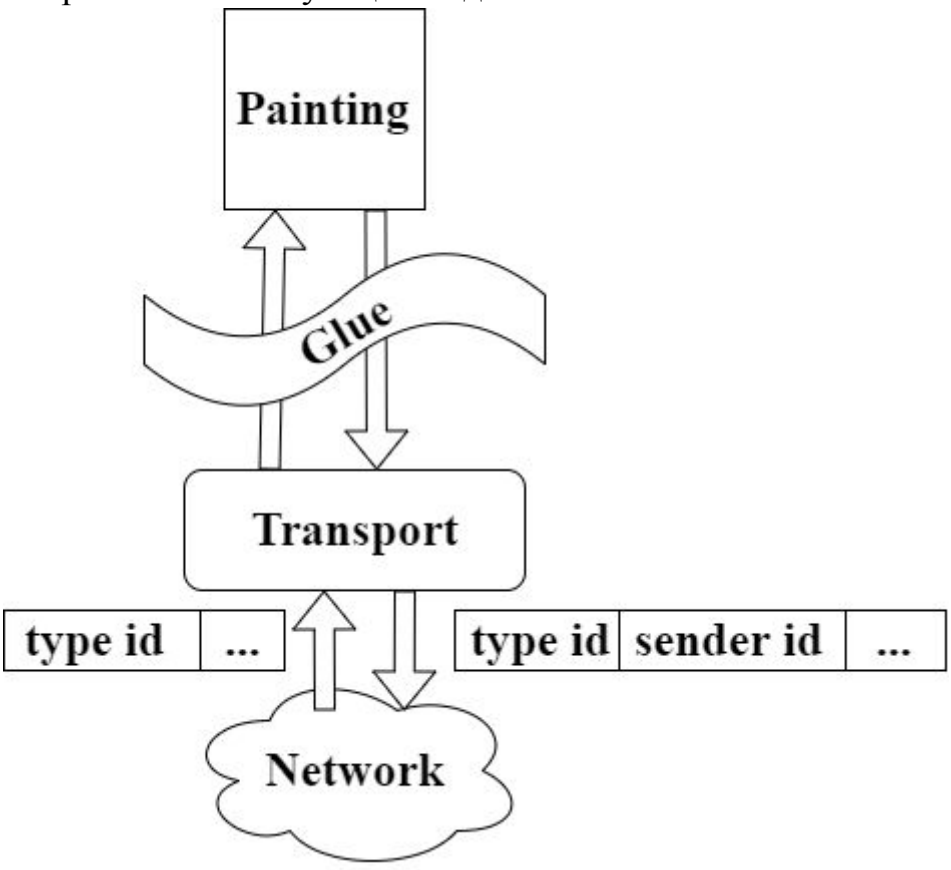


- Внутреннее представление изображения (**Painting**) содержит в себе все данные изображения и предоставляет методы и слоты для его чтения и модификации. При модификации изображения **Painting** отправляет сигнал, в котором содержится вся информация об изменении.
- Пользовательский интерфейс (**GUI**) — совокупность кнопок, меню и прочих элементов управления. При взаимодействии с пользователем элемент отправляет соответствующий сигнал, но не меняет свое состояние; вместо этого элемент содержит слоты для приема сигналов о смене состояния. Например, если попытаться поднять выше верхний слой, то сигнал отправится, но **Painting** определит,

что это действие невозможно и в свою очередь не отправит сигнал о смене состояния списку слоев.

- Сетевой транспорт (**Transport**) — это набор объектов для подключения к серверу и передачи информации между клиентами через свои сигналы и слоты. Когда в слот **Transport** одного клиента приходит сигнал, информация о нем пересылается через сервер другим клиентам и их **Transport** отправляет соответствующие сигналы.
- Связующий код (**Glue**) подключает сигналы об изменении от **GUI** и **Transport** к слотам **Painting**, а сигналы от **Painting** к слотам **GUI** и **Transport**. Таким образом информация о любом изменение изображения не только отображается в интерфейсе, но и автоматически отсылается другим клиентам.

Сервер состоит из трех блоков: сетевой транспорт, внутреннее представление изображения и связующий код.



- Сетевой транспорт (**Transport**) получает сообщения от клиентов и пересылает их остальным. Также на каждое сообщение **Transport** отправляет сигнал с его содержимым.
- Внутреннее представление изображения (**Painting**) в сервере используется для того, чтобы при отключении клиентов от сервера результаты работы не терялись.

- Связующий код (**Glue**) подключает сигналы от **Transport** к слотам **Painting** для того, чтобы дублировать действия всех клиентов в серверный **Painting**.

Базовые возможности графического редактора

От любого графического редактора пользователем ожидается, что он имеет некоторый минимум возможностей: перо, ластик, выбор цвета, масштабирование холста. Рассмотрим детали реализации.

Перо и ластик

Инструменты перо и ластик являются классами-потомками абстрактного инструмента. Абстрактный инструмент имеет методы для получение событий перетаскивания от мыши и возможность нарисовать себя, а также отправляет сигнал после завершения рисования. И перо и ластик реализованы как добавление ломаной линии из большого количества точек, но перо рисует заданным цветом, а ластик — прозрачным. В процессе рисования неоконченная линия рисуется самим инструментом, и только после завершения перетаскивания отправляется сигнал с готовой линией.

Выбор цвета

В **Qt5** есть **QColorDialog**, который позволяет выбирать цвет через всплывающее окно как это сделано в программе **MSPaint**. Но я решил, что всплывающие окна неудобны и не выглядят красиво, поэтому решил реализовать свой вариант, виджет **ColorChooserWidget** — чтобы была возможность выбирать цвет не отвлекаясь от рисования на всплывающие окна, как в **SAI**[4]. **ColorChooserWidget** устроен следующим образом: координаты мыши на палитре это **H** и **S** (тон и насыщенность), а бегунок справа — **L**(светлота). Также можно ввести вручную параметры в поля (**HSL** или **RGB**). После выбора цвета **ColorChooserWidget** отправляет сигнал о том, что цвет выбран.

Масштаб и сдвиг

Перетаскивание мышью холста на меняет координаты линий, а происходит через камеру. Камера содержит в себе сдвиг холста относительно экрана и масштаб. Чтобы масштабирование ощущалось одинаково вне зависимости от текущего масштаба, он меняется

мультипликативно, домножением или делением на $\sqrt[4]{2}$. Коэффициент выбран специально, чтобы 4 увеличения подряд давали двухкратный масштаб. Для удобства, при масштабировании одновременно происходит сдвиг так, чтобы точка холста под курсором мыши сохранила свое положение на экране.

Слои

Крайне удобной функциональностью графического редактора является поддержка слоев. Слой — изначально полностью прозрачное изображение, на котором можно рисовать; итоговая картина получается наложением всех слоев друг на друга в порядке их расположения. В моем редакторе поддерживается создание, копирование и удаление слоев, а также перетаскивание слоя для изменения его порядка.

Клиент-серверное взаимодействие

Подключение к серверу происходит по протоколу TCP. Общение между клиентами происходит только через сервер.

Структура сообщений

TCP — потоковый протокол, поэтому в транспорте реализована самостоятельная разбивка потока на сообщения. Перед каждым сообщением передается его длина. Само сообщение состоит из id типа сообщения, id отправителя и содержимого. В качестве содержимого могут передаваться числа, логические значения, строки, массивы и структуры. Логический тип передается одним байтом 0 или 1. У массивов передается сначала количество элементов, потом по порядку каждый элемент. Строки передаются как массив байт в кодировке **UTF8**. Структуры передаются как последовательность составляющих полей.

Кодирование чисел

В моем транспорте есть два способа передачи чисел: фиксированного и переменного размера. Например, цвет передается как компоненты RGB, каждая из которых находится в фиксированном диапазоне, поэтому для него используется фиксированный размер. А вот у id пользователей диапазон неизвестен. Для таких чисел используется семибитное кодирование[5]. Число представляется последовательностью байт, в каждом байте семь бит содержат биты числа в порядке от старшего к

младшему (big-endian), а старший бит отвечает за конец последовательности: если старший бит ноль, то это последний байт. Таким образом числа до 127 включительно кодируются одним байтом, до 16383 двумя и так далее.

Особенности надежного кодирования и декодирования

При передаче сообщения оно может прийти несколькими отдельными частями[6]. Возникает проблема, когда наивный алгоритм декодирования может обратиться в память за частью сообщения, которая еще не пришла. Эту проблему можно решить ужасно сложным и большим кодом декодирования, который не падает посредине сообщения, вне зависимости от места где сообщение оборвалось. Также можно декодировать только длину (все еще учитывая случай когда даже длина может прийти не полностью), после чего читать все в буфер и декодировать только когда придет все сообщение целиком. Я решил использовать второй подход, более затратный по памяти но более простой в написании и отладке. Для того чтобы исключить возможные ошибки в кодировании/декодировании сообщений, я написал программу на языке **Python**, которая по описанию содержимого сообщения генерирует код для кодирования и декодирования такого сообщения. Также для каждого сообщения вручную написаны тесты на кодирование/декодирование, чтобы возможные будущие ошибки в генераторе кода были сразу обнаружены.

Взаимодействие между пользователями

Моей программой можно пользоваться и без подключения к серверу, как обычным графическим редактором. Подключение к серверу позволяет нескольким пользователям совместно рисовать.

Подключение к серверу

Во время подключение клиента к серверу, сервер выдает клиенту его id и имя. Сервер все действия клиента подписывает его id. После этого происходит синхронизация **Painting** сервера с **Painting** клиента.

Слои пользователей

Каждый пользователь видит все слои других пользователей, но может рисовать только на своих. Таким образом пользователи могут рисовать что-то в одном и том же месте, но при этом не мешать друг другу. Для удобства есть возможность локально отключать видимость всех слоев любого из пользователей.

Отмена/повтор действия

Векторному редактору для отмены действия одного пользователя не требуется пересылать изображение в том виде, в котором оно было до этого действия. Когда пользователь просит отменить действие, пересылается сообщение с обратным действием. Например, если пользователь создал слой, а потом отменил действие, то отправляется сообщение об удалении этого слоя. Остальным клиентам достаточно применить это обратное действие к своим изображениям.

100% недоверие клиенту

Общим правилом в клиент-серверной архитектуре является максимальное недоверие данным, приходящим от клиента[7]. Даже специально модифицированный клиент не должен позволять пользователю выходить за рамки своих прав, например, действовать от лица другого пользователя. Для этого в транспорте предусмотрено поле id отправителя. Все сообщения, которые клиент отсылает серверу, уходят без этого поля. Сервер, который единственный точно знает какой клиент что отправил, частично декодирует сообщение и дописывает в его начало это поле, после чего рассылает модифицированное сообщение остальным клиентам. Таким образом клиент не может, например, указать, чей слой он редактирует — транспорт гарантирует, что клиент редактирует всегда только свой слой.

Заключение

Был разработан многопользовательский редактор. Написан расширяемый транспорт и кодогенератор для кодирования сообщений. Реализованы как базовые возможности редактора, так и отмена/повтор действий, работа со слоями. В то же время есть большой простор для расширения возможностей: больше инструментов рисования, поддержка пера, экспорт в растровые форматы и т.п.

Исходный код проекта на GitHub: http://github.com/kracav4ik/pencil_mob

Список литературы

1. https://ru.wikipedia.org/wiki/Растровый_графический_редактор
2. https://ru.wikipedia.org/wiki/Векторный_графический_редактор
3. Qt Documentation, Signals & Slots https://doc.qt.io/qt-5/signalsandslots.html
4. Paint Tool SAI https://www.systemax.jp/en/sai/
5. https://en.wikipedia.org/wiki/Variable-length_quantity
6. Congestion Control in IP/TCP Internetworks https://tools.ietf.org/html/rfc896
7. Matt Bishop, "Robust Programming"
   http://nob.cs.ucdavis.edu/bishop/secprog/robust.html