

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc122596718)

[1.2 Анализ предметной области: система контроля версий 6](#_Toc122596719)

[1.3 Формулирование требований к разрабатываемому программному обеспечению 7](#_Toc122596720)

[2 РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА 9](#_Toc122596721)

[3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА 11](#_Toc122596722)

[3.1 Проектирование графического редактора 11](#_Toc122596723)

[3.2 Проектирование системы контроля версий 12](#_Toc122596724)

[3.3 Проектирование алгоритмов приложения 13](#_Toc122596725)

[4 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА 17](#_Toc122596726)

[4.1 Особенности программной реализации 17](#_Toc122596727)

[4.2 Особенности реализации пользовательского интерфейса 19](#_Toc122596728)

[ЗАКЛЮЧЕНИЕ 20](#_Toc122596729)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 21](#_Toc122596730)

[ПРИЛОЖЕНИЕ А (обязательное) РЕАЛИЗАЦИЯ ПРОГРАММЫ 22](#_Toc122596735)

# ВВЕДЕНИЕ

Информационные системы очень сильно упрощают деятельность человека. В современном мире невозможно представить себе жизнь без помощи технических средств. Графический редактор предназначен для работы с цифровыми изображениями. Файлы могут видоизменяться, а также подвержены повреждениям. Для решения этой проблемы существуют системы контроля версий.

Целью курсового проекта является изучение и разработка системы контроля версий для графического редактора.

Чтобы достичь поставленной цели, выполним следующие задачи:

— анализ предметной области системы контроля версий для графического редактора;

— разработка модели системы контроля версий для графического редактора;

— проектирование основных алгоритмов системы контроля версий для графического редактора;

— реализация системы контроля версий для графического редактора.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА**

**1.1 Описание предметной области**

Графические редакторы – это специальные программы для обработки графических изображений (создания, редактирования, просмотра) [1].

Растровые графические редакторы являются наилучшим средством обработки фотографий и рисунков, так как обеспечивают высокую точность передачи градаций цветов и полутонов. Примеры растровых графических редакторов: Paint.NET, Adobe Photoshop, Corel Photo-Paint [1].

Как уже было сказано, иногда файлы приходится дополнять, изменять и так далее; в процессе файл может быть поврежден. Для этого были созданы системы контроля версий.

Сегодня в мире, где существует огромное количество сложных систем, существует необходимость видоизменения электронных документов на различных стадиях их разработки. За время своего существования электронный документ может быть подвержен большому количеству изменений. Однако часто так бывает, что для дальнейшей работы необходима не только последняя версия документа, но и различные предыдущие варианты [2].

Безусловно, можно хранить несколько различных вариантов необходимого документа, но данный способ неэффективен. Нам приходится тратить кучу времени и сил, необходимо особое внимание и велика вероятность ошибки. Кроме того нам приходится хранить огромное количество практически идентичных документов [2].

Вследствие этого были разработаны программные средства, которые упрощают данный механизм. Данные средства именуются системами контроля версий. Существует несколько систем такого рода, каждая из которых актуальна при определенных условиях их использования [2].

Основная идея систем контроля версий - запоминать все внесенные изменения, а так же комментарии пользователей, их вносивших. После этого становится понятно кто, когда и что менял, зачем и почему. И, что немаловажно, все эти изменения можно потом откатить на любой момент времени [2].

**1.2 Анализ предметной области: система контроля версий**

Система контроля версий - система, записывающая изменения в файл или набор файлов в течение времени работы над какими-либо данными. Одним из основных преимуществ такой системы является возможность вернуться к определенной версии данных. Это позволяет отменить те или иные изменения, посмотреть на предыдущие версии и что-то изменить.

Системы контроля версий преимущественно используются для хранения кода, но, в целом, они могут применяться для хранения файлов любого типа.

К основным типам систем контроля версий относятся:

1. **Децентрализованные системы контроля версий.** В настоящее время ярким представителем является Git. В таких системах клиенты не просто извлекают последний снимок всех файлов (состояние файлов на определённый момент времени) - они полностью копируют репозиторий, включая всю его историю. Такая операция называется *клонированием*. В этом случае, если какой-нибудь сервер, с которым взаимодействуют разработчики, выйдет из строя, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждый клон (копия) репозитория является полной резервной копией всех данных [3].
2. **Централизованные системы контроля версий.** Такие системы контроля версий предполагают сохранение версий проектов на  общий сервер, с которого потом получают нужные версии клиенты.
3. **Локальные системы контроля версий.** Отличаются от централизованных систем тем, что нет возможности вести совместную работу с другими участниками проекта [4].

Рассмотрим систему контроля версий Git, как наиболее популярную систему.

Подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда пользователь делает коммит, то есть сохраняет состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён [4]. Git представляет свои данные как, скажем, поток снимков, где каждый снимок содержит уникальный идентификатор.

Так же Git позволяет создавать ветки – это по своей сути различные варианты изменения проекта исходя из текущего состояния.

**1.3 Формулирование требований к разрабатываемому программному обеспечению**

Произведя анализ предметной области, были сформулированы основные функциональные и нефункциональные требования. Функциональные требования описывают то, как некая система должна работать. Нефункциональные требования являются менее строгими и описывают качество итоговой системы, выражают ее характеристики [5].

К функциональным требованиям относится:

1. Использования различных инструментов для создания изображений на холсте;
2. Сохранение изменений состояния холста;
3. Перемещение между сохраненными состояниями: предыдущее (при наличии) или последующее (при наличии);
4. Наличие ветвления;
5. Наличие графического интерфейса.

К группе нефункциональных требований можно отнести следующие:

1. Требование к надежности: программа должна иметь защиту от некорректных действий пользователей и ошибочных исходных данных.
2. Требования к масштабируемости: приложение должно иметь возможность добавления отдельных частей к уже существующим компонентам программы.

# 2 РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА

Так как мы разрабатываем систему контроля версий для графического редактора, точный аналог популярной системы Git будет неподходящим для нашей задачи. Следовательно, необходимо разработать свою систему контроля версий.

Учитывая, что большая часть действий пользователей не будет изменяться ими вдальнейшем, подходящим решением будет являться реализация сохранения некоторой разницы между состояниями холста.

Так как в функциональных требованиях было указано наличие ветвления, реализуем некоторые идеи, которые были заложены в Git. Одной из таких идей будет являться сохранение ссылки на предыдущую фиксацию (родителя). Также для простоты перехода между состояниями будем хранить все фиксации на ветви.

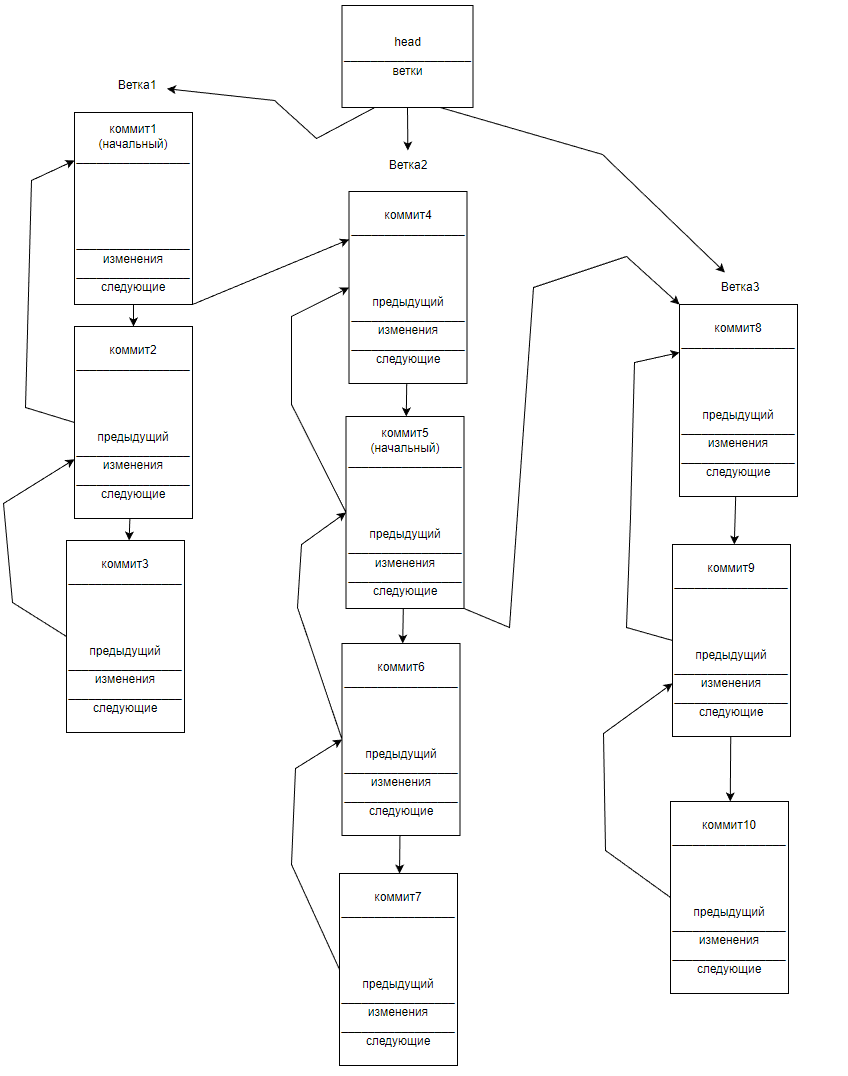
Будем сохранять следующие элементы, хранящие разницу между состояниями:

1. Координаты фигуры;
2. Цвет;
3. Размер кисти;
4. Тип фигуры,

Сама структура данных, хранящая все фиксации, на физическом уровне будет представлять собой массив. На логическом уровне история изменений будет представлять собой дерево: всегда есть одна стартовая фиксация, от которой будут идти ссылки на другие фиксации.

Фиксации будут сохраняться при нажатии на кнопку. Для хранения данных за пределами программы будут использоваться файлы: для хранения полного состояния системы контроля версий (что представляет собой начальных коммит, относительно которого будут производитсья последующие сохранения), файлы для хранения последующих сохранений и файл для хранения текущего состояния систему контроля версий: какая ветка в данный момент используется, какой коммит является текущим, список всех доступных веток.

Схема системы контроля версий представлена на рисунке 1.

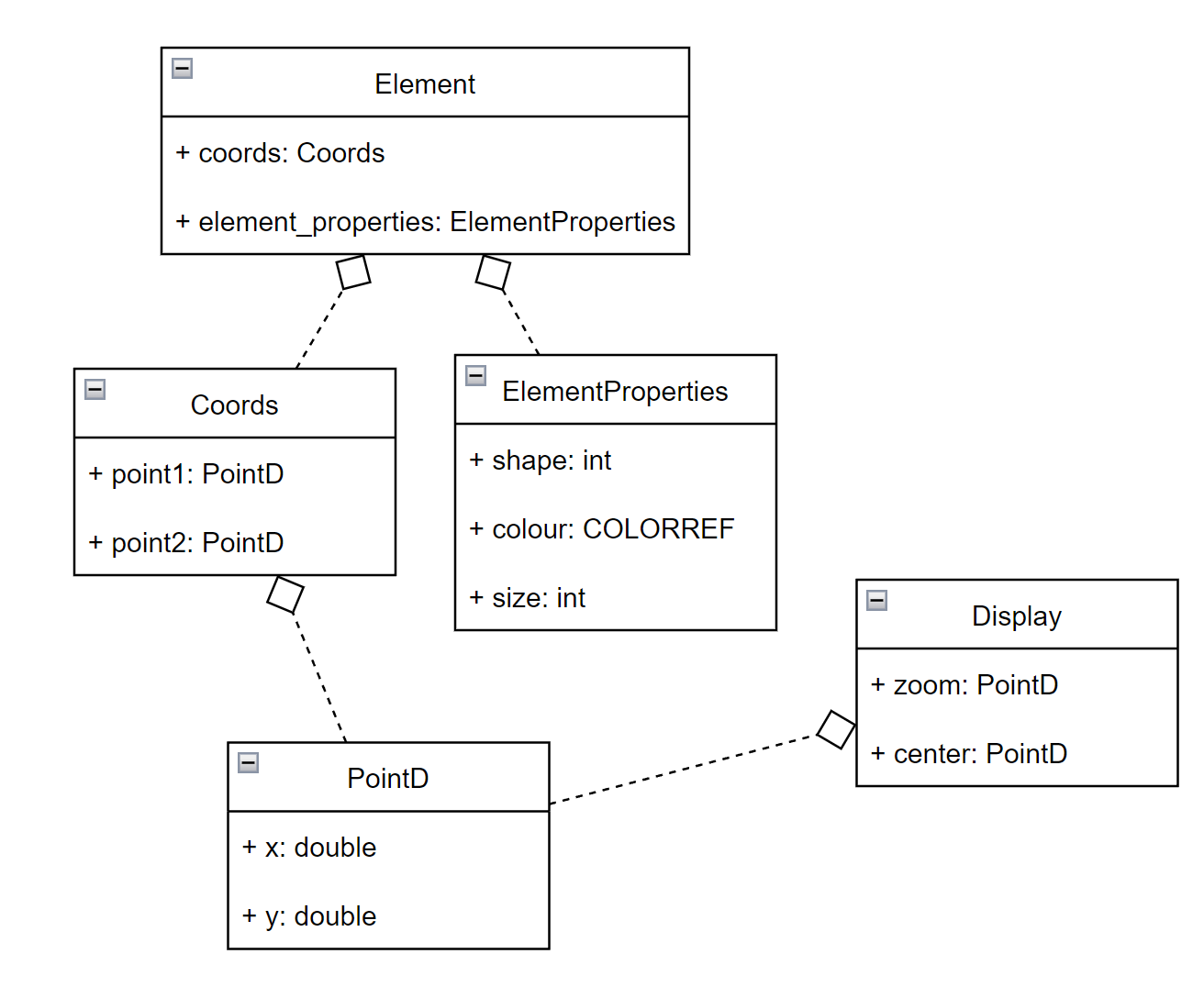
Рисунок 1 – Схема разрабатываемой системы контроля версий

# 3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА

**3.1 Проектирование графического редактора**

Основной программой является наш графический редактор.

Наше приложение содержит 5 структур. Диаграмма классов приведена на рисунке 2.

Рисунок 2 – Диаграмма классов для текстового редактора

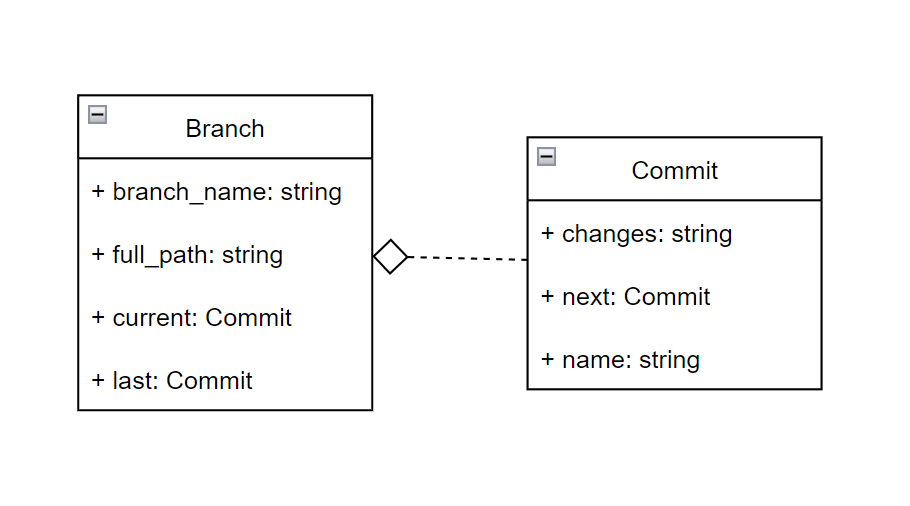
Структура Element содержит поля coords типа другой нашей структуры Coords и element\_properties типа другой нашей структуры ElementProperties. Данная структура позволяет хранить элементы, которые появляются на холсте в результате рисования. Каждый элемент имеет координаты и свойства. Координаты начала и конца элемента, представленные структурой Coords, хранят два поля: точку 1 и точку 2. Эти точки также являются структурами, хранящими координаты X и Y точки. Помимо координат элемент хранит свойства: размер кисти, которой он был нарисован, цвет, форму.

Кроме структуры Element важную роль играет и структура Display. Она хранит состояния зума и расположение центра экрана относительно центра координат (который изначально совпадает с центром экрана при нулевом смещении).

**3.2 Проектирование системы контроля версий**

Кроме логики графического редактора, в нашем программном обеспечении необходимо реализовать систему контроля версий.

Диаграмма классов системы контроля версий представлена на рисунке 3.

Рисунок 3 – Диаграмма классов для системы контроля версий

Класс Branch представляет собой ветку и имеет следующие поля: имя ветки (branch\_name), путь по коммитам от первого до последнего коммита на ветке (full\_path), текущий коммит (current) и последний коммит (last).

Класс Commit представляет собой единицу сохранения аналогично коммиту в Git. Данный класс имеет следующие атрибуты: список изменений в виде стоки (changes), следующий коммит (next) и название коммита (name).

Эти классы позволяют осуществить весь необходимый функционал нашей системы контроля версий: сохранение изменений, откат до предыдущей версии и ветвления, позволяющие сделать откат до прежней версии и создать сохранение оттуда - при этом последнее сохранение на первоначальной ветке не потеряется.

**3.3 Проектирование алгоритмов приложения**

Основными алгоритмами нашего графического редактора являются алгоритмы рисования. Рассмотрим алгоритм обработки рисования, изображенный на рисунке 4.

1) Сначала производится выбор инструментов для рисования: тип фигуры, цвет, толщина кисти.

2) Далее при нажатии левой кнопки мыши по холсту включается режим отрисовки: движения мышью теперь изменяю появившуюся в режиме предпросмотра фигуру.

3) Если кнопка всё ещё нажата, то (4), иначе (7)

4) Движение мышью

5) Отрисовка изменений

6) (3)

7) Сохранение последний параметров фигуры

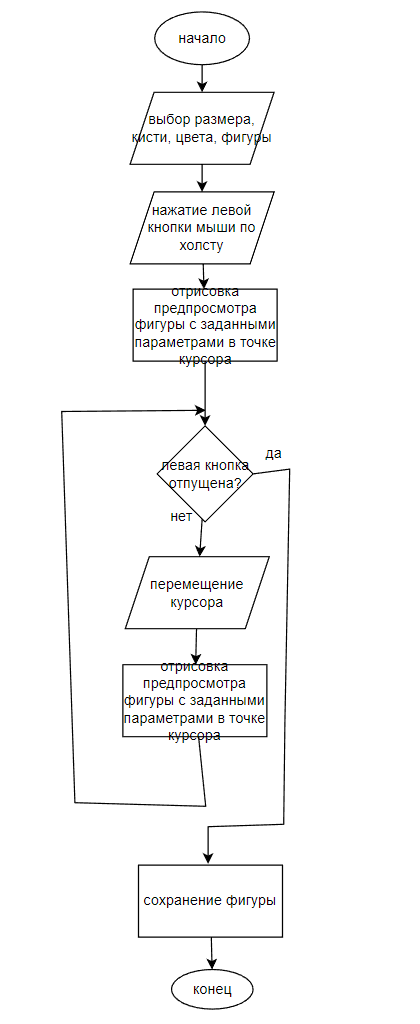


Рисунок 4 – Алгоритм рисования

Рассмотрим алгоритм сохранения (создания коммита) на той же ветке. Схема данного алгоритма изображена на рисунке 5.

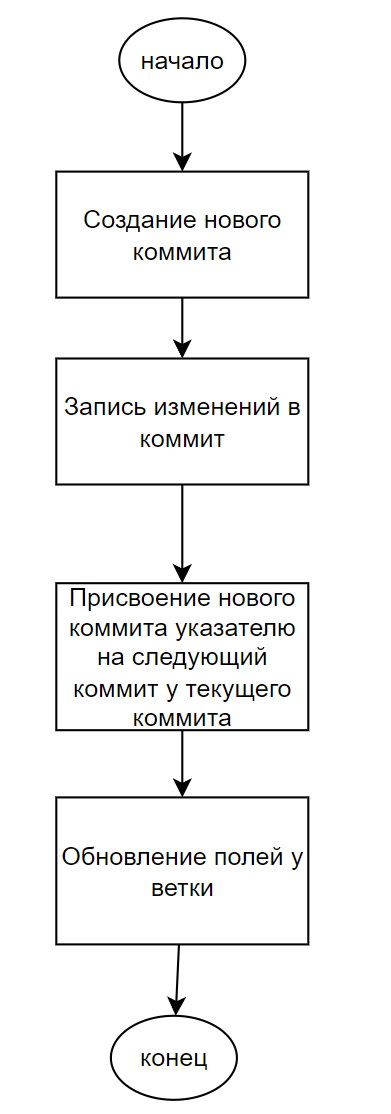
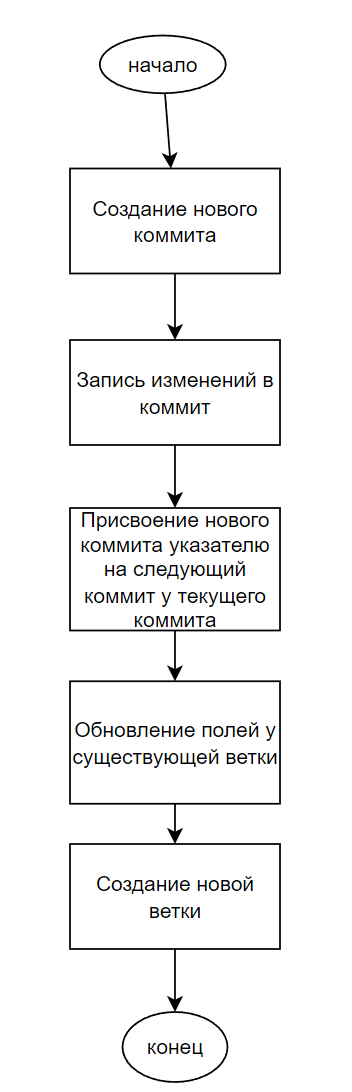


Рисунок 5 – Алгоритм сохранения на существующей ветке

Рассмотрим алгоритм сохранения на старом коммите (с созданием новой ветки). Алгоритм изображен на рисунке 6.

Рисунок 6 – Алгоритм сохранения с созданием новой ветки

# 4 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ГРАФИЧЕСКОГО РЕДАКТОРА

**4.1 Особенности программной реализации**

Программная реализация выполнена на языке Си.

Создадим функцию для инициализации и подготовке файлов, а также создающую первый коммит. Создание первого коммита было вынесено в эту функцию, т.к. первый коммит особенный - он хранит не изменения относительно других коммитов, а полное состояние программы.

Ниже приведен фрагмент кода инициализации приложения.

void initialize()

{

CreateDirectoryA(vc\_path, NULL);

...

FILE \*fptr = fopen(branches\_csv, "wb");

fprintf(fptr, "%s,%s,%s,%s\n", "branch\_name", "full\_path", "current", "last");

...

fprintf(fptr, "%s,%s,%s,%s\n", rand\_string\_alloc(branchNameSize), commit\_name, commit\_name, commit\_name);

fclose(fptr);

...

CreateDirectoryA(full\_path, NULL);

...

FILE \*fptr2 = fopen(commit\_csv, "wb");

// содержимое файла с начальным коммитом

writeInit(fptr2);

fclose(fptr2);

}

В результате работы данной функции появляется главная папка системы контроля версий. В ней лежит главный файл системы контроля версий, названный BRANCHES.csv. В нем записана вся информация по веткам, аналогично файлу .git в Git репозитории. Помимо данного файла в главной папке лежит папка с первым коммитом. Он содержит полное сохранение состояния холста графического редактора. Последующие коммиты будут создаваться внутри этой папки с первым коммитом.

void commit()

{

...

FILE \*branches = fopen(branches\_csv, "r+b");

...

int c;

while ((c = getc(branches)) != EOF)

{

char buff[9999];

buff[0]=c;

long posBefore = ftell(branches);

fgets(&buff[1], 9998, (FILE\*)branches);

long posAfter = ftell(branches);

...

if (!current) continue;

...

CreateDirectoryA(dir\_path, NULL);

...

FILE \*newF = fopen(commitFile, "wb");

writeDiff(newF);

fclose(newF);

...

fprintf(branches, "%s,%s,%s,%s\n", branch\_name, new\_path, commit\_name, commit\_name);

...

break;

}

fclose(branches);

}

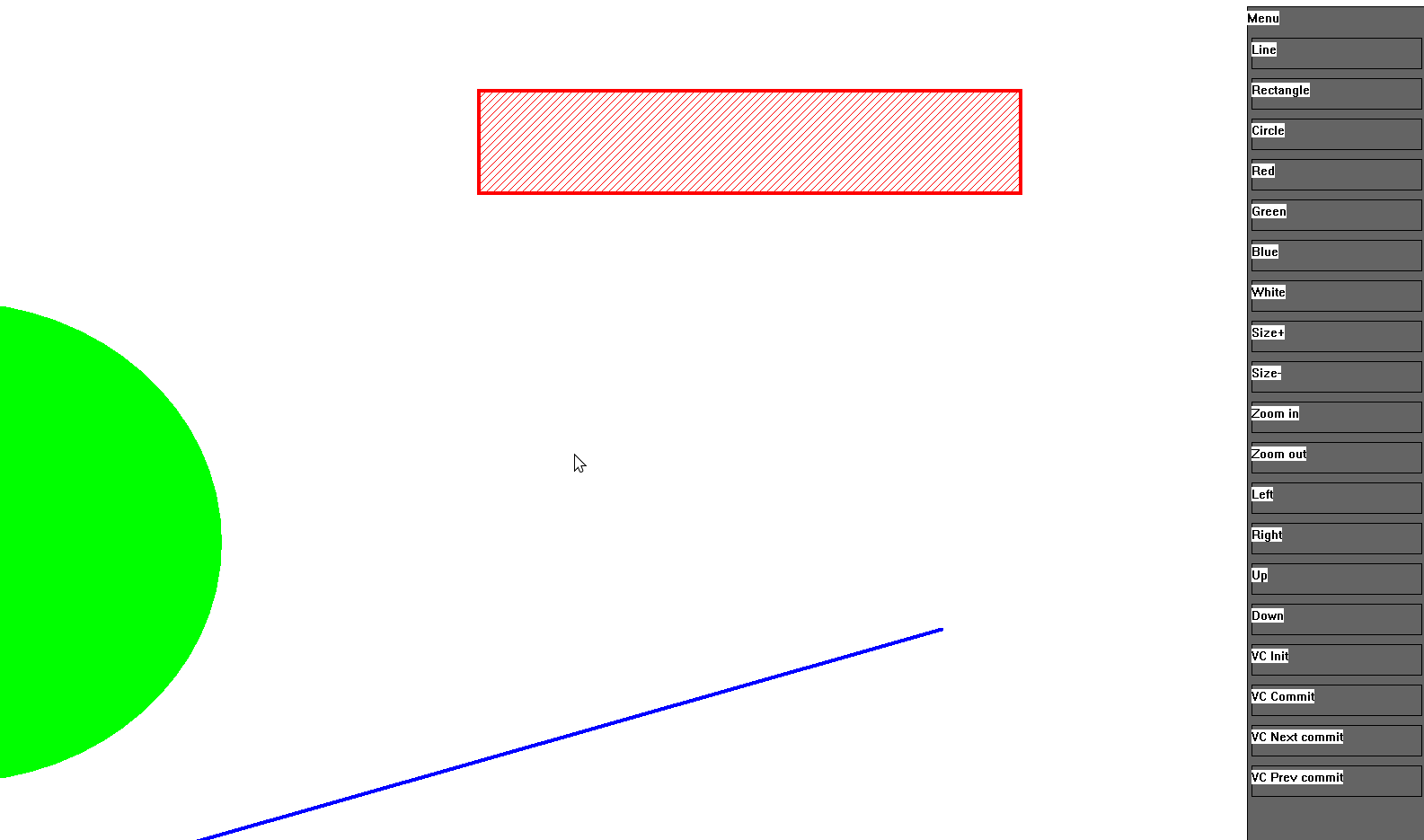
При совершении коммита происходит создание новой папки с коммитом. В ней создается файл с изменениями коммита. В конце обновляются данные в файле, хранящем информацию о всех ветках.

**4.2 Особенности реализации пользовательского интерфейса**

Для нашего приложения был разработан графический интерфейс. Приведем примеры внешнего вида нашей программы.

На рисунке 7 можно увидеть главное окно нашего приложения.

На нем виден холст, занимающий большую часть окна, и боковое меню, содержащее весь набор необходимых функций для осуществления рисования и управления системой контроля версий.

Рисунок 7 – Главное окно приложения

В боковом меню есть такие функции, как выбор параметров для рисования: тип фигуры, цвет, размер кисти, зум, перемещение по холсту, кнопки управления системой контроля версий: инициализация, создание коммита, переключение по коммитам.

# ЗАКЛЮЧЕНИЕ

В настоящей работе была описана и реализована система контроля версий для графического редактора.

Мы проанализировали предметную область, спроектировали и разработали приложение, целью которого является редактирование изображений. Также была спроектирована и разработана система контроля версий, хранящая изменения состояния холста графического редактора.

Таким образом, можно сказать, что все поставленные перед нами задачи были выполнены и цель курсового проекта была достигнута.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Компьютерная графика [Электронный ресурс] – Режим доступа: https://interneturok.ru/lesson/informatika/7-klass/narezka-dsh/kompyuternaya-grafika-chast-3-graficheskie-redaktory (дата обращения: 12.12.2022)

2. Системы контроля версий [Электронный ресурс] – Режим доступа: https://revolution.allbest.ru/programming/00521312\_0.html (дата обращения: 12.12.2022)

3. Git - О системе контроля версий [Электронный ресурс] – Режим доступа: https://ru.hexlet.io/courses/git\_base/lessons/vcs\_intro/theory\_unit (дата обращения 25.09.22).

4. Progit [Электронный ресурс] – Режим доступа: https://losst.ru/wp-content/uploads/2016/08/progit-ru.1027.pdf (дата обращения 25.09.22).

5. Функциональные и нефункциональные требования (FunctionalandNon-functionalRequirements) [Электронный ресурс]. - Режим доступа: https://studfile.net/preview/2152457/page:4/ (дата обращения: 25.09.22).

# ПРИЛОЖЕНИЕ А (обязательное) РЕАЛИЗАЦИЯ ПРОГРАММЫ

Файл Elements.h

#ifndef ELEMENTS\_H

#define ELEMENTS\_H

#include <windows.h>

#include "g\_var.h"

typedef struct POINTD

{

double x;

double y;

}PointD;

typedef struct DISPLAY

{

PointD zoom;

PointD center;

}Display;

typedef struct COORDINATESS

{

PointD point1;

PointD point2;

}Coords;

typedef struct ELEMENTPROPERTIES

{

int shape;

COLORREF colour;

int size;

}ElementProperties;

typedef struct ELEMENT

{

Coords coords;

ElementProperties properties;

}Element;

#endif

Файл renameMe.c

#include "renameMe.h"

extern BOOL drawing;

extern ElementProperties currentElement;

extern Element elem[elemBufferSize];

extern int currentIndex;

extern int elemCount;

extern Display display;

extern BOOL zooming;

extern POINT p1;

extern struct menu menu;

extern char debugLine[500];

void NextElem()

{

currentIndex++;

elemCount++;

if (elemCount > elemBufferSize) elemCount = elemBufferSize;

if (currentIndex > elemCount - 1) currentIndex = 0;

}

BOOL Line(HDC hdc, int x1, int y1, int x2, int y2)

{

MoveToEx(hdc, x1, y1, NULL);

return LineTo(hdc, x2, y2);

}

PointD Zoom(double x,double y,RECT window)

{

PointD center;

center.x = (window.right-window.left - (menu.area.botRight.x - menu.area.topLeft.x))/2.0;

center.y = (window.bottom-window.top)/2.0;

PointD inZoom;

inZoom.x=center.x + (center.x + display.center.x - x)\*display.zoom.x;

inZoom.y=center.y + (center.y + display.center.y - y)\*display.zoom.y;

return inZoom;

}

PointD ZoomReverce(double x,double y,RECT window)

{

PointD center;

center.x = (window.right-window.left - (menu.area.botRight.x - menu.area.topLeft.x))/2.0;

center.y = (window.bottom-window.top)/2.0;

PointD inZoom;

inZoom.x=center.x + display.center.x + (center.x - x)/(display.zoom.x);

inZoom.y=center.y + display.center.y + (center.y - y)/(display.zoom.y);

return inZoom;

}

VOID UpdateWin(HWND hwnd)

{

RECT window;

GetClientRect(hwnd, &window);

window.left = 0;

InvalidateRect(hwnd, &window, 1);

}

VOID UpdateWin1(HWND hwnd,RECT window)

{

InvalidateRect(hwnd, &window, 1);

}

VOID ZoomRectangle(RECT window,int x1, int y1, int x2, int y2)

{

if(fabs(x1-x2)!=0&&fabs(y1-y2)!=0)

{

PointD f1 = ZoomReverce(x1,y1,window);

PointD f2 = ZoomReverce(x2,y2,window);

display.zoom.x = fabs(display.zoom.x)\*(fabs((window.right-window.left-(menu.area.botRight.x - menu.area.topLeft.x))/fabs(x1-x2)));

display.zoom.y = fabs(display.zoom.y)\*(fabs((window.bottom-window.top)/fabs(y1-y2)));

double min = min(f1.x,f2.x);

display.center.x=((min-(window.right-window.left - (menu.area.botRight.x - menu.area.topLeft.x))/2.0)+fabs(f1.x-f2.x)/2);

min = min(f1.y,f2.y);

display.center.y=(min-window.bottom/2.0)+fabs(f1.y-f2.y)/2;

}

}

VOID DrawAxes(HDC memDc,RECT window)

{

double w = window.right-window.left - (menu.area.botRight.x - menu.area.topLeft.x);

double h = window.bottom-window.top;

PointD point = Zoom(w/2.0,0,window);

HPEN hPen = CreatePen(PS\_DASH,2,RGB(0, 0, 0));

SelectObject(memDc, hPen);

Line(memDc, point.x, 0, point.x, h);

point = Zoom(0,h/2.0,window);

Line(memDc,0, point.y, w, point.y);

DeleteObject(hPen);

}

LRESULT CALLBACK FrameWndProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam) {

switch (Message) {

case WM\_CREATE:

{

ShowWindow(hwnd, SW\_SHOWMAXIMIZED);

RECT window;

GetClientRect(hwnd,&window);

currentElement.shape = shapeLine;

currentElement.size = 1;

display.zoom.x=1;

display.zoom.y=1;

display.center.x = 0;

display.center.y = 0;

currentIndex = 0;

elemCount = 1;

UpdateWin1(hwnd,window);

break;

}

case WM\_LBUTTONDOWN:

{

if (MenuButtonPressed(hwnd, FALSE)) break;

RECT window;

GetClientRect(hwnd,&window);

PointD firstPoint = ZoomReverce(LOWORD(lParam), HIWORD(lParam), window);

elem[currentIndex].coords.point1 = firstPoint;

elem[currentIndex].properties.shape = currentElement.shape;

elem[currentIndex].properties.colour = currentElement.colour;

elem[currentIndex].properties.size = currentElement.size;

drawing = TRUE;

break;

}

case WM\_MOUSEMOVE:

{

if (!drawing) break;

RECT window;

GetClientRect(hwnd,&window);

PointD secondPoint = ZoomReverce(LOWORD(lParam), HIWORD(lParam), window);

elem[currentIndex].coords.point2 = secondPoint;

UpdateWin1(hwnd,window);

break;

}

case WM\_LBUTTONUP:

{

if (MenuButtonPressed(hwnd, TRUE)) break;

drawing = FALSE;

NextElem();

RECT window;

GetClientRect(hwnd,&window);

UpdateWin1(hwnd,window);

break;

}

case WM\_RBUTTONDOWN:

{

zooming = TRUE;

p1.x = LOWORD(lParam);

p1.y = HIWORD(lParam);

break;

}

case WM\_RBUTTONUP:

{

if (!zooming) break;

zooming = FALSE;

RECT window;

GetClientRect(hwnd, &window);

ZoomRectangle(window, p1.x, p1.y, LOWORD(lParam), HIWORD(lParam));

UpdateWin(hwnd);

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HDC memDc = CreateCompatibleDC(hdc);

RECT window;

GetClientRect(hwnd, &window);

HBITMAP bmp = CreateCompatibleBitmap(hdc, window.right, window.bottom);

SelectObject(memDc, bmp);

FillRect(memDc, &window, (HBRUSH)(RGB(255,255,255)));

//DrawAxes(memDc, window);

for (int i = 0; i < elemCount; i++)

{

double x1 = elem[i].coords.point1.x;

double y1 = elem[i].coords.point1.y;

double x2 = elem[i].coords.point2.x;

double y2 = elem[i].coords.point2.y;

PointD f1 = Zoom(x1,y1,window);

PointD f2 = Zoom(x2,y2,window);

if (elem[i].properties.shape == shapeLine)

{

HPEN hPen = CreatePen(PS\_SOLID, elem[i].properties.size, elem[i].properties.colour);

SelectObject(memDc, hPen);

Line(memDc, f1.x, f1.y, f2.x, f2.y);

DeleteObject(hPen);

}

else if (elem[i].properties.shape == shapeRectangle)

{

HPEN hPen = CreatePen(PS\_DASH, elem[i].properties.size, elem[i].properties.colour);

HBRUSH hBrush = CreateHatchBrush(HS\_BDIAGONAL, elem[i].properties.colour);

SelectObject(memDc, hPen);

SelectObject(memDc, hBrush);

Rectangle(memDc, f1.x, f1.y, f2.x, f2.y);

DeleteObject(hBrush);

DeleteObject(hPen);

}

else if (elem[i].properties.shape == shapeEllipse)

{

HPEN hPen = CreatePen(PS\_SOLID, elem[i].properties.size, elem[i].properties.colour);

HBRUSH hBrush = CreateSolidBrush(elem[i].properties.colour);

SelectObject(memDc, hBrush);

SelectObject(memDc, hPen);

Ellipse(memDc, f1.x, f1.y, f2.x, f2.y);

DeleteObject(hBrush);

DeleteObject(hPen);

}

}

if(zooming)

{

double x1 = elem[currentIndex].coords.point2.x;

double y1 = elem[currentIndex].coords.point2.y;

PointD f1 = Zoom(x1,y1,window);

HPEN hPen = CreatePen(PS\_DASH,3,RGB(255, 0, 0));

HBRUSH hBrush = GetStockObject(HOLLOW\_BRUSH);

SelectObject(memDc, hPen);

SelectObject(memDc, hBrush);

Rectangle(memDc, p1.x, p1.y, f1.x, f1.y);

DeleteObject(hPen);

DeleteObject(hBrush);

}

DrawUI(memDc, window);

char str[4];

//sprintf(str,"Zx-%lf; Zy-%lf; Cx-%lf; Cy-%lf;clientWin; X-%lu;Y-%lu;",display.zoom.x,display.zoom.y,display.center.x,display.center.y,window.right,window.bottom);

//sprintf(str,"sizeof(Element)=%d, sizeof(elem)=%d (%d)",sizeof(Element), sizeof(elem), sizeof(elem)/1024/1024);

//SetWindowText(hwnd, str);

BitBlt(hdc, 0, 0, window.right, window.bottom, memDc, 0, 0, SRCCOPY);

DeleteDC(memDc);

DeleteObject(bmp);

EndPaint(hwnd, &ps);

break;

}

case WM\_ERASEBKGND:

{

return 0;

}

case WM\_KEYDOWN:

{

switch (wParam)

{

case VK\_OEM\_PLUS:

{

ButtonZoomIn(hwnd);

break;

}

case VK\_OEM\_MINUS:

{

ButtonZoomOut(hwnd);

break;

}

case VK\_LEFT:

{

ButtonLeft(hwnd);

break;

}

case VK\_RIGHT:

{

ButtonRight(hwnd);

break;

}

case VK\_UP:

{

ButtonUp(hwnd);

break;

}

case VK\_DOWN:

{

ButtonDown(hwnd);

break;

}

case VK\_DELETE:

{

display.zoom.x\*=1.1;

UpdateWin(hwnd);

break;

}

case VK\_INSERT:

{

display.zoom.x/=1.1;;

UpdateWin(hwnd);

break;

}

}

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

break;

}

default:

return DefWindowProc(hwnd, Message, wParam, lParam);

}

return 0;

}

int MenuButtonPressed(HWND hwnd, BOOL skip\_processing)

{

POINT cursorCoords;

GetCursorPos(&cursorCoords);

ScreenToClient(hwnd, &cursorCoords);

if (CursorInsideArea(menu.buttons[0].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonLine();

return 1;

}

else if (CursorInsideArea(menu.buttons[1].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonRect();

return 1;

}

else if (CursorInsideArea(menu.buttons[2].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonEllipse();

return 1;

}

else if (CursorInsideArea(menu.buttons[3].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonRed();

return 1;

}

else if (CursorInsideArea(menu.buttons[4].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonGreen();

return 1;

}

else if (CursorInsideArea(menu.buttons[5].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonBlue();

return 1;

}

else if (CursorInsideArea(menu.buttons[6].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonWhite();

return 1;

}

else if (CursorInsideArea(menu.buttons[7].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonSizePlus();

return 1;

}

else if (CursorInsideArea(menu.buttons[8].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonSizeMinus();

return 1;

}

else if (CursorInsideArea(menu.buttons[9].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonZoomIn(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[10].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonZoomOut(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[11].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonLeft(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[12].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonRight(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[13].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonUp(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[14].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonDown(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[15].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonInitVC(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[16].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonVCCommit(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[17].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonVCNextCommit(hwnd);

return 1;

}

else if (CursorInsideArea(menu.buttons[18].area, cursorCoords))

{

if (skip\_processing) return 2;

ButtonVCPrevCommit(hwnd);

return 1;

}

else

{

return 0;

}

}

Файл versionControl.c

#include "versionControl.h"

extern BOOL drawing;

extern ElementProperties currentElement;

extern Element elem[elemBufferSize];

extern int currentIndex;

extern int elemCount;

extern Display display;

extern BOOL zooming;

extern POINT p1;

const char\* vc\_path = "./VersionControl";

void initialize()

{

CreateDirectoryA(vc\_path, NULL);

char branches\_csv[9999] = "";

str\_concat(branches\_csv, 2, vc\_path, "/BRANCHES.csv");

FILE \*fptr = fopen(branches\_csv, "wb");

fprintf(fptr, "%s,%s,%s,%s\n", "branch\_name", "full\_path", "current", "last");

char \*commit\_name = rand\_string\_alloc(commitNameSize);

fprintf(fptr, "%s,%s,%s,%s\n", rand\_string\_alloc(branchNameSize), commit\_name, commit\_name, commit\_name);

fclose(fptr);

char full\_path[9999] = "";

str\_concat(full\_path, 3, vc\_path, "/", commit\_name);

CreateDirectoryA(full\_path, NULL);

char commit\_csv[9999] = "";

str\_concat(commit\_csv, 4, full\_path, "/", commit\_name, ".csv");

FILE \*fptr2 = fopen(commit\_csv, "wb");

// содержимое файла с начальным коммитом

writeInit(fptr2);

fclose(fptr2);

}

void commit()

{

char branches\_csv[9999] = "";

str\_concat(branches\_csv, 2, vc\_path, "/BRANCHES.csv");

FILE \*branches = fopen(branches\_csv, "r+b");

char buff\_skip[9999];

fgets(buff\_skip, 9999, (FILE\*)branches);

int c;

while ((c = getc(branches)) != EOF)

{

char buff[9999];

buff[0]=c;

long posBefore = ftell(branches);

fgets(&buff[1], 9998, (FILE\*)branches);

long posAfter = ftell(branches);

char \*branch\_name = strtok(buff, ",");

char \*full\_path = strtok(NULL, ",");

char \*current = strtok(NULL, ",");

char \*last = strtok(NULL, "\n");

if (!current) continue;

char new\_path[9999] = "";

char \*currentPos = strstr(full\_path, current);

if (currentPos) printf("-=-=-=");

memcpy(new\_path, full\_path, currentPos - full\_path);

new\_path[currentPos - full\_path] = '\0';

char \*commit\_name = rand\_string\_alloc(commitNameSize);

str\_concat(new\_path, 3, current, "/", commit\_name);

char dir\_path[9999] = "";

str\_concat(dir\_path, 3, vc\_path, "/", new\_path);

CreateDirectoryA(dir\_path, NULL);

char commitFile[9999] = "";

str\_concat(commitFile, 4, dir\_path, "/", commit\_name, ".csv");

FILE \*newF = fopen(commitFile, "wb");

writeDiff(newF);

fclose(newF);

if (strcmp(current, last) == 0)

{

fseek(branches, posBefore-posAfter - 1, SEEK\_CUR);

fprintf(branches, "%s,%s,%s,%s\n", branch\_name, new\_path, commit\_name, commit\_name);

}

else

{

fseek(branches, posBefore-posAfter - 1, SEEK\_CUR);

fprintf(branches, "%s,%s,,%s\n", branch\_name, full\_path, last);

fprintf(branches, "%s,%s,%s,%s\n", rand\_string\_alloc(branchNameSize), new\_path, commit\_name, commit\_name);

}

break;

}

fclose(branches);

}

void nextCommit()

{

char branches\_csv[9999] = "";

str\_concat(branches\_csv, 2, vc\_path, "/BRANCHES.csv");

FILE \*branches = fopen(branches\_csv, "r+b");

char buff\_skip[9999];

fgets(buff\_skip, 9999, (FILE\*)branches);

int c;

while ((c = getc(branches)) != EOF)

{

char buff[9999];

buff[0]=c;

long posBefore = ftell(branches);

fgets(&buff[1], 9998, (FILE\*)branches);

long posAfter = ftell(branches);

char \*branch\_name = strtok(buff, ",");

char \*full\_path = strtok(NULL, ",");

char \*current = strtok(NULL, ",");

char \*last = strtok(NULL, "\n");

if (!current) continue;

char \*currentPos = strstr(full\_path, current);

char nextCurrent[999];

memcpy(nextCurrent, currentPos + commitNameSize, commitNameSize-1);

nextCurrent[commitNameSize-1] = '\0';

fseek(branches, posBefore-posAfter - 1, SEEK\_CUR);

fprintf(branches, "%s,%s,%s,%s\n", branch\_name, full\_path, nextCurrent, last);

break;

}

fclose(branches);

}

void prevCommit()

{

char branches\_csv[9999] = "";

str\_concat(branches\_csv, 2, vc\_path, "/BRANCHES.csv");

FILE \*branches = fopen(branches\_csv, "r+b");

char buff\_skip[9999];

fgets(buff\_skip, 9999, (FILE\*)branches);

int c;

while ((c = getc(branches)) != EOF)

{

char buff[9999];

buff[0]=c;

long posBefore = ftell(branches);

fgets(&buff[1], 9998, (FILE\*)branches);

long posAfter = ftell(branches);

char \*branch\_name = strtok(buff, ",");

char \*full\_path = strtok(NULL, ",");

char \*current = strtok(NULL, ",");

char \*last = strtok(NULL, "\n");

if (!current) continue;

char \*currentPos = strstr(full\_path, current);

char nextCurrent[999];

memcpy(nextCurrent, currentPos - commitNameSize, commitNameSize-1);

nextCurrent[commitNameSize-1] = '\0';

fseek(branches, posBefore-posAfter - 1, SEEK\_CUR);

fprintf(branches, "%s,%s,%s,%s\n", branch\_name, full\_path, nextCurrent, last);

break;

}

fclose(branches);

}

void writeInit(FILE \*f)

{

fprintf(f, "%s,%s,%s,%s,%s,%s,%s,%s\n", "#","x1", "y1", "x2", "y2", "shape", "size", "colour");

for (int i = 0; i < elemCount - 1; i++)

{

fprintf(f, "%d,%lf,%lf,%lf,%lf,%d,%d,%lu\n", i, elem[i].coords.point1.x, elem[i].coords.point1.y, elem[i].coords.point2.x, elem[i].coords.point2.y, elem[i].properties.shape, elem[i].properties.size, elem[i].properties.colour);

}

}

void writeDiff(FILE \*f)

{

/\*fprintf(f, "%s,%s,%s,%s,%s,%s,%s,%s\n", "#","x1", "y1", "x2", "y2", "shape", "size", "colour");

for (int i = 0; i < elemCount - 1; i++)

{

fprintf(f, "%d,%lf,%lf,%lf,%lf,%d,%d,%lu\n", i, elem[i].coords.point1.x, elem[i].coords.point1.y, elem[i].coords.point2.x, elem[i].coords.point2.y, elem[i].properties.shape, elem[i].properties.size, elem[i].properties.colour);

}\*/

}

char\* rand\_string\_alloc(size\_t size)

{

char \*s = malloc(size + 1);

if (s) {

rand\_string(s, size);

}

return s;

}

char \*rand\_string(char \*str, size\_t size)

{

const char charset[] = "abcdefghijklmnopqrstuvwxyz0123456789";

if (size)

{

--size;

for (size\_t n = 0; n < size; n++) {

int key = rand() % (int) (sizeof charset - 1);

str[n] = charset[key];

}

str[size] = '\0';

}

return str;

}

BOOL startsWith(const char \*str, const char \*starts\_with)

{

size\_t len = strlen(starts\_with);

return (strncmp(str, starts\_with, len) == 0);

}

char\* escape(char\* buffer)

{

int i,j;

int l = strlen(buffer) + 1;

char esc\_char[]= { '\a','\b','\f','\n','\r','\t','\v','\\'};

char essc\_str[]= { 'a', 'b', 'f', 'n', 'r', 't', 'v','\\'};

char\* dest = (char\*)calloc( l\*2,sizeof(char));

char\* ptr=dest;

for(i=0;i<l;i++){

for(j=0; j< 8 ;j++){

if( buffer[i]==esc\_char[j] ){

\*ptr++ = '\\';

\*ptr++ = essc\_str[j];

break;

}

}

if(j == 8 )

\*ptr++ = buffer[i];

}

\*ptr='\0';

return dest;

}

void str\_concat(char \*result, int count, ...)

{

va\_list args;

va\_start(args, count);

for(int i = 0; i < count; i++)

{

strcat(result, va\_arg(args, char\*));

}

va\_end(args);

}

void str\_split(char\* str, char\* delimiter, int delimiter\_len, int count, ...)

{

va\_list args;

va\_start(args, count);

for(int i = 0; i < count; i++)

{

char\* delimiterPos = strstr(str, delimiter);

if (delimiterPos != NULL)

{

char \*arg = va\_arg(args, char\*);

memcpy(arg, str, delimiterPos - str);

arg[delimiterPos - str] = '\0';

str = delimiterPos + delimiter\_len;

}

else

{

char \*arg = va\_arg(args, char\*);

memcpy(arg, str, delimiterPos - str);

}

}

va\_end(args);

}