МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

Работа допущена к защите

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Руководитель

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Выпуск и сопровождение программных продуктов»

на тему: «Разработка системы контроля версий для графического редактора»

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Аллянов М. Д.

Шифр 191006

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лукьянов П.В.

Оценка: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_» Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Орел 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

УТВЕРЖДАЮ:

\_\_\_\_\_\_\_\_\_\_\_\_И. о. зав. кафедрой

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**ЗАДАНИЕ**

**на курсовой проект**

по дисциплине «Выпуск и сопровождение программных продуктов»

Студент Аллянов М.Д. Шифр 191006

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

1 Тема курсового проекта

«Разработка системы контроля версий для графического редактора»

2 Срок сдачи студентом законченной работы «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 202\_\_

3 Исходные данные

Разработать систему контроля версий для графического редактора. Проанализировать и описать предметную область, сформулировать требования к ПО. Разработать модель системы контроля версий. Произвести проектирование архитектуры и структуры приложения и системы контроля версий. Разработать основные алгоритмы приложения и системы контроля версий. Реализовать и протестировать ПО.

4 Содержание курсового проекта

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

5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта, приложение.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лукьянов П.В.

Задание принял к исполнению: «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 202\_\_

Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc115724677)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ» 6](#_Toc115724678)

[1.1 Анализ предметной области: игра «Сортировка цветов» 6](#_Toc115724679)

[1.2 Анализ предметной области: система контроля версий 7](#_Toc115724680)

[1.3 Формулирование требований к разрабатываемому программному обеспечению 8](#_Toc115724681)

[2 РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ» 9](#_Toc115724682)

[3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ» 11](#_Toc115724683)

[3.1 Проектирование игры «Сортировка цветов» 11](#_Toc115724684)

[3.2 Проектирование системы контроля версий 13](#_Toc115724685)

[3.3 Проектирование алгоритмов приложения 14](#_Toc115724686)

[3 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ» 18](#_Toc115724687)

[4.1 Особенности программной реализации 18](#_Toc115724688)

[4.2 Особенности реализации пользовательского интерфейса 19](#_Toc115724689)

[ЗАКЛЮЧЕНИЕ 23](#_Toc115724690)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24](#_Toc115724691)

[ПРИЛОЖЕНИЕ А (обязательное) РЕАЛИЗАЦИЯ ПРОГРАММЫ 25](#_Toc115724698)

# 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 консольного и 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 суть игры заключается в перемещении цветов из одной колбы в другую, подходящим решением будет являться реализация сохранения некоторой разницы между состояниями игры.

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 дочерние фиксации (те фиксации, для которых текущая фиксация является родителем).

Кроме того, будем сохранять следующие элементы, хранящие разницу между состояниями игры:

1. Выбранная колба (или ее отсутствие);
2. Целевая колба (или ее отсутствие);
3. Уровень игры.

Сама структура данных, хранящая все фиксации, на физическом уровне будет представлять собой список. 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, обозначающую ту или иную колбу, или при нажатии на кнопку перехода между уровнями. Для хранения данных за пределами программы будут использоваться два файла: для хранения полного состояния системы контроля версий (необходим для автосохранения/автозагрузки и перехода между уровнями), а также файл для хранения указанных выше флагов, необходимых для оптимизации операций перехода к любому другому состоянию.

Схема системы контроля версий представлена на рисунке 2.

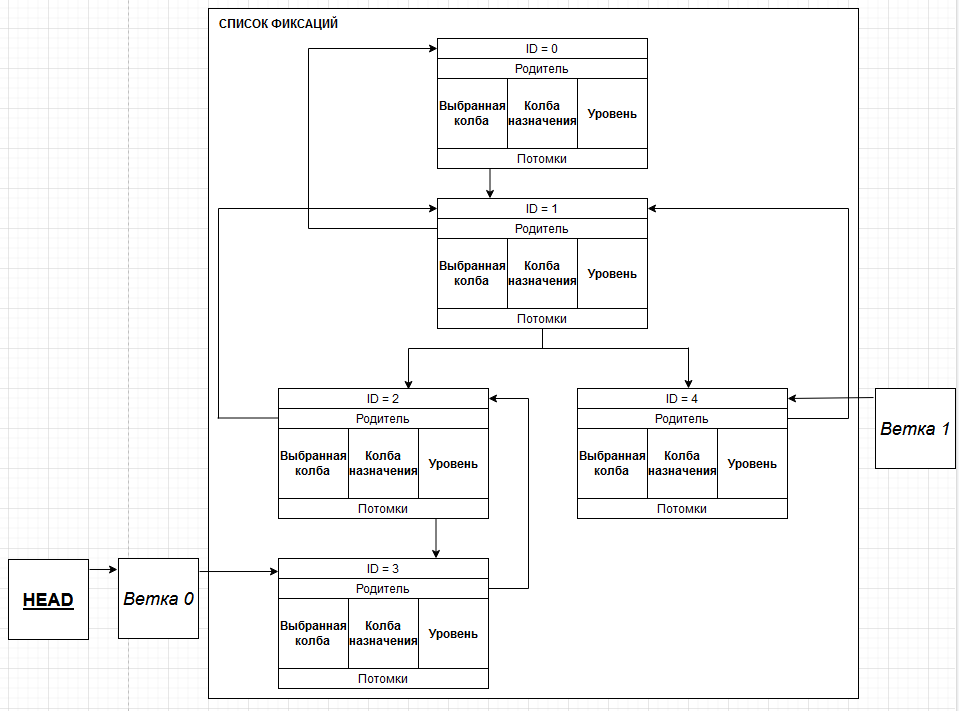


Рисунок 2 – Схема разрабатываемой системы контроля версий

# 3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ»

**3.1 Проектирование игры «Сортировка цветов»**

Основной программой является наше игровое приложение "Сортировка цветов". Так как мы разрабатываем небольшое игровое приложение, отделим логику от интерфейса при помощи реализации паттерна MVP (Model-View-Presenter) [8].

Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем. Presenter должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе.

Логика представления должна иметь ссылку на экземпляр Presenter. Все события представления передаются для обработки в Presenter и практически никогда не обрабатываются логикой представления.

Наше игровое приложено содержит пять классов: один класс для работы с графическим интерфейсом (View), два класса, содержащие логику игры (класс Colba для обработки логики взаимодействия с колбой и класс Game для реализации самого игрового процесса), класс для связи логики и интерфейса (Presenter) и класс для работы с параметрами игры (State). Эти параметры будут необходимы для реализации системы контроля версий.

Диаграмма классов нашего игрового приложения приведена на рисунке 3.

Класс View содержит в себе ссылку на необходимый Presenter, а также на графическое окно (Canvas). Кроме того, в классе реализованы методы для отображения и обновления графического интерфейса, метод, отображающий окно помощи, а также методы обработки перемещений между состояниями.

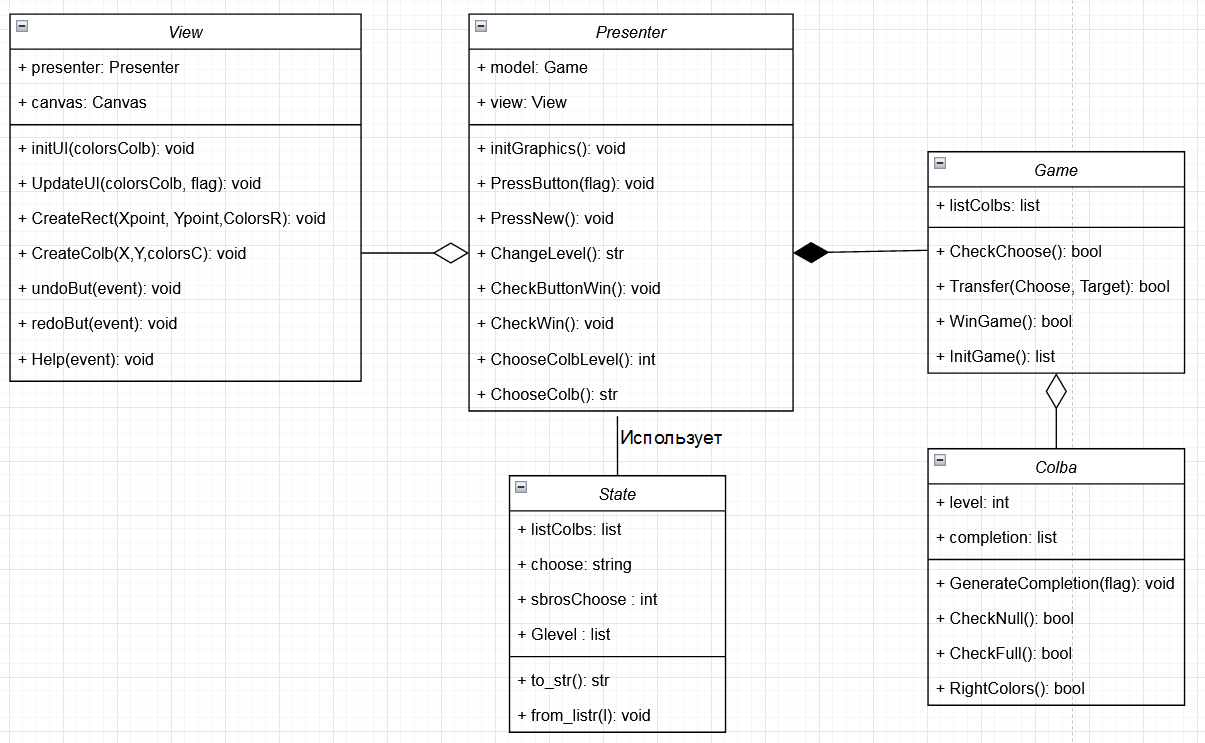


Рисунок 3 – Диаграмма классов для игры "Сортировка цветов"

Класс Colba содержит поля, отвечающие за размер колбы (для реализации нашего ПО установим его равным трем) и ее содержание. Также класс содержит методы, необходимые для заполнения колбы, проверки ее на пустоту и на наполненность, а также для проверки на правильность ее содержания.

Класс Game содержит список колб, с которыми происходит взаимодействие во время игры, а также методы для проверки текущего состояния игры (выбрана колба или нет) и ее завершения. Кроме того, есть метод, отвечающий за логику перемещения цвета между колбами.

Класс State хранит весь список колб, выбранную колбу и флаг выбора, а также уровень. Его методы направлены на сохранение и получение этой информации.

**3.2 Проектирование системы контроля версий**

Кроме игровой логики, в нашем программном обеспечении необходимо реализовать систему контроля версий. Она будет реализована двумя классами:

1. Класс Commit, необходимый для создания фиксации в системе контроля версий;
2. Класс Git, реализующий основную логику работы с системой контроля версий.

Диаграмма классов нашей системы контроля версий приведена на рисунке 4.

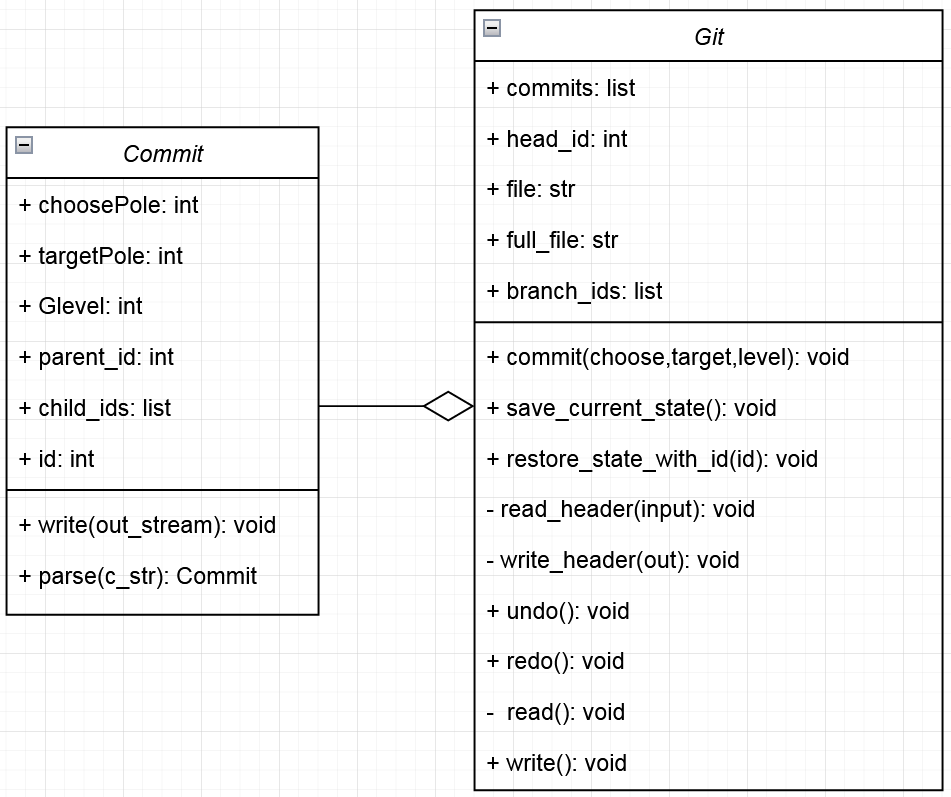


Рисунок 4 – Диаграмма классов для игры "Сортировка цветов"

Класс Commit включает в себя поля, хранящие выбранную колбу, колбу назначения, текущий уровень, а также идентификаторы родительской, всех дочерних фиксаций и идентификатор самой фиксации. Его методы считывают данные и формируют из них информацию, необходимую для игры, а также записывают данные в файл.

Класс Git включает в себя список всех фиксаций, идентификатор текущей фиксации, имена файлов, хранящие полные копии состояния игры или информацию о фиксациях, а также список идентификаторов (концов веток).

Его методы необходимы для осуществления фиксации изменений, откатывания состояния игры к предыдущему или дочернему состоянию (если дочерних будет несколько, будет предложено выбрать фиксацию, в которую надо перейти). Кроме этого присутствуют методы для работы с информацией в момент старта или во время закрытия приложения, осуществления полного копирования состояния или загрузки (на основе полной копии), а также методы для работы со специальной информацией (заголовки файлов).

**3.3 Проектирование алгоритмов приложения**

Ключевым алгоритмом в нашем игровом приложении является алгоритм непосредственно игрового цикла (на конкретном уровне). Рассмотрим его с момента, следующего за начальной инициализации интерфейса.

Игровой процесс будет продолжаться до тех пор, пока пользователь его не закроет. При этом, все это время ожидается нажатие пользователя на кнопку с номером колбы (кроме случая перехода на другой уровень).

Если пользователь нажал на кнопку и при этом не была установлена первая колба (колба выбора), то соответствующий флаг будет установлен о информации о выборе сохранится при помощи системы контроля версий.

Если первая колба была установлена, то устанавливается флаг с колбой назначения. Если обе колбы выбраны корректно, то по специальному алгоритму в цикле происходит обработку двух колб.

Если пользователь попытался неправильно смешать цвета, то состояние не изменится, а система контроля версий сохранит информацию о попытке неудачного обмена.

После обмена, в зависимости от его исхода, обновится интерфейс и программа снова будет ожидать действие пользователя.

Схема алгоритма, описывающего основной игровой цикл, приведена на рисунке 5.

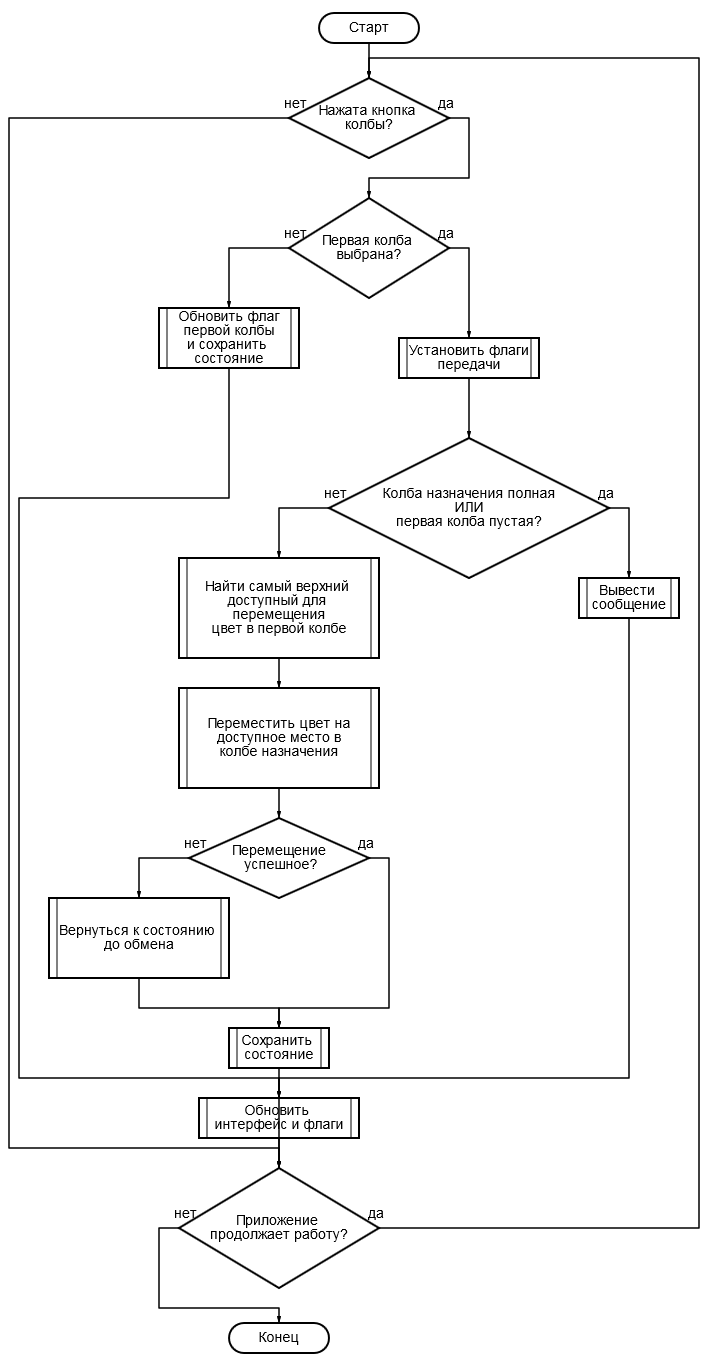


Рисунок 5 – Основной игровой цикл приложения

Далее рассмотрим алгоритм, который отвечает за сохранение в систему контроля версий - алгоритм фиксации изменений. Для его описания будем использовать язык псевдокода (Рисунок 6).

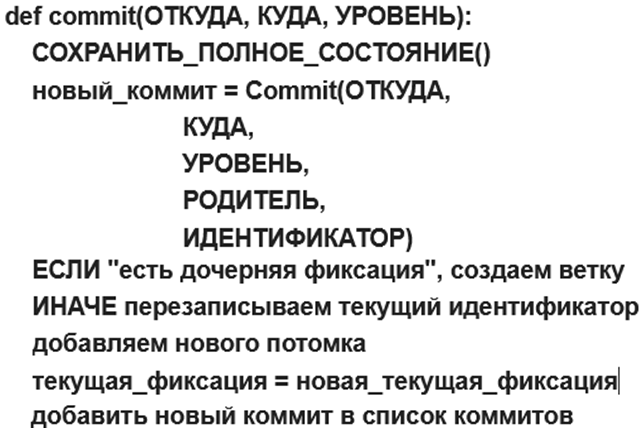


Рисунок 6 – Алгоритм фиксации изменений

Алгоритм создает новую фиксацию при помощи переданных параметров, а также связывает фиксацию с текущим деревом.

Если у текущей фиксации есть дочерняя, то мы создаем ветку. Иначе произойдет обычная перезапись текущего идентификатора в списке веток.

Далее рассмотрим два алгоритма: откат к предыдущей фиксации и применение изменений, хранящихся в следующей фиксации.

Алгоритм отката изменяет каждое значение текущего состояния на то, что хранится в необходимой нам фиксации.

Но, так как в нашем игровом приложении есть бесконечное количество генерирующихся уровней, мы можем откатываться с использованием флагов только в пределах одного уровня. Если уровень изменился, мы должны не просто заменить флаги, а полностью изменить все игровое состояние.

В любом случае после изменения состояния выведется информация.

Если идентификатор родительской фиксации равен -1, то мы достигли корня дерева истории изменений.

Алгоритм применения изменений, хранящихся в фиксации сначала проверяет, есть ли доступные нам для перехода дочерние фиксации. Если их нет, выводится соответствующее сообщение и алгоритм заканчивает работу.

В противном случае мы перемещаемся к следующей фиксации (если их было несколько, пользователю предлагается выбрать нужную). Если уровень следующей фиксации отличается от уровня текущей, то мы полностью загружаем это состояние. Иначе мы применяем те изменения, которая хранит выбранная нами фиксация.

Схемы этих двух алгоритмов приведены на рисунке 7.

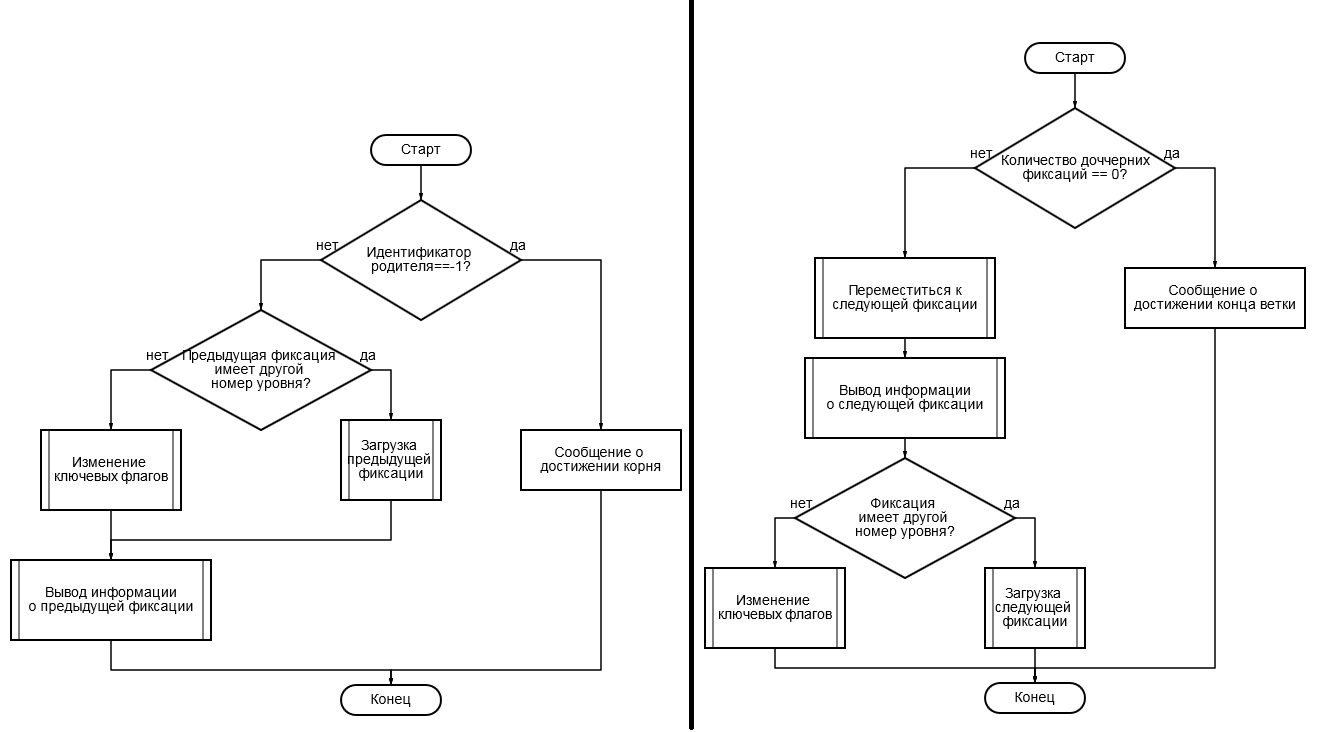


Рисунок 6 – Алгоритм отката изменений (слева) и алгоритм применения изменений, хранящихся в фиксации (справа)

# 3 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ»

**4.1 Особенности программной реализации**

Программная реализация выполнена на языке Python 3.7 с использованием библиотек Tkinter и Canvas, предназначенных для реализации простого графического интерфейса.

Tkinter является событийно-ориентированной библиотекой. В приложениях такого типа имеется главный цикл обработки событий [4]. В Tkinter такой цикл запускается методом mainloop.

Для простоты вне описанных выше классов заведем функцию для начальной инициализации программы.

Ниже приведен фрагмент кода инициализации приложения.

def main():

root = Tk()

view = View()

root.mainloop()

git.write()

if \_\_name\_\_ == '\_\_main\_\_':

main()

В нашем случае выход из цикла произойдет только при завершении работы с приложением. Как видно из приведенного выше фрагмента кода, мы инициируем только класс представления (View), все остальные классы инициализируются через него.

При завершении работы с приложением происходит запись всех фиксаций.

Так как мы указали, что наше приложение содержит консольный интерфейс, будем в него выводить информацию о текущих состояниях. Эта информация будет отображаться при нажатии на кнопки "Z" (обрабатывающую переход к предыдущему состоянию) и кнопку "X" обрабатывающую переход к дочернему состоянию). Также через консоль будем спрашивать у пользователя, к какому конкретно дочернему состоянию он хочет перейти (при наличии нескольких). Фрагменты кода, отвечающие за реализацию этих фрагментов приведены ниже.

print("ID-------------",cc.id)

print("CHOOSE---------",cc.choosePole)

print("TARGET---------",cc.targetPole)

print("LEVEL----------",cc.Glevel)

print("PARENT---------",cc.parent\_id)

print("CHILDS---------",\*cc.child\_ids)

def safe\_read\_int(msg):

while True:

try:

return int(input(msg))

except ValueError:

print('[ERR-] Provide integer.')

if len(commit.child\_ids) > 1:

cid = safe\_read\_int(f'[INFO] Select branch index from 0 to ({len(commit.child\_ids) - 1}): ')

Ключевая часть реализации приложения приведена в приложении А.

**4.2 Особенности реализации пользовательского интерфейса**

Так как наше приложение является сочетанием оконного (графического) и консольного интерфейса, приведем примеры внешнего вида нашей программы.

На рисунке 7 можно увидеть главное окно нашего приложения.

На нем видны основные элементы интерфейса игры.

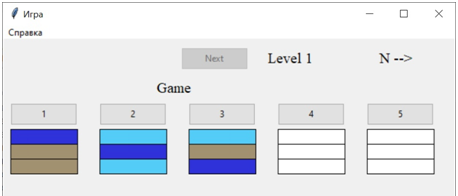


Рисунок 7 – Главное окно приложения

В центре расположены колбы с перемешанными в них цветами. Над ними расположены кнопки, через взаимодействие с которыми осуществляется игровое взаимодействие.

Сверху справа расположена метка, указывающая на текущую выбранную колбу. Если колба не выбрана или только что был перемещен цвет, метка принимает значение "N". В противном случае отображается номер выбранной колбы (Рисунок 8).

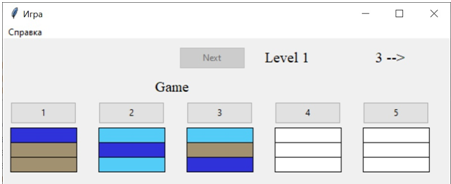


Рисунок 8 – Пример выбранной колбы

Сверху по центру расположена кнопка перехода на следующий уровень. Она станет доступна только при прохождении текущего уровня (Рисунок 9).

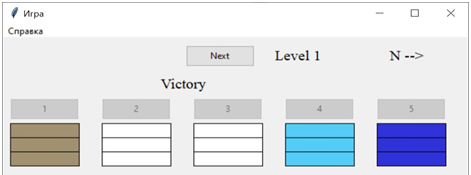


Рисунок 9 – Пример прохождения уровня в приложении

Если метка выбора колбы была установлена и нажат номер какой-либо другой колбы, внешний вид программы обновится: самый верхний цвет из выбранной колбы исчезнет и добавится в самый низ колбы назначения (при условии возможности осуществления всех описанных выше действий). Пример изменения интерфейса наглядно представлен на рисунке 10. В данном случае было произведено перемещение цвета из колбы 3 в колбу 4.

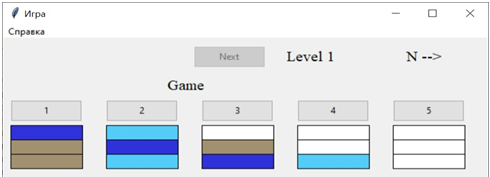


Рисунок 10 – Пример перераспределения цветов

При достижении какого-либо состояния мы можем вернуться к предыдущему при помощи кнопки "Z".Ее нажатие либо уберет выбранную колбу (если последним действием колба была выбрана), либо отменит последнее перемещение. На рисунке 11 показан пример такого отката. Последним действием был перемещен цвет из колбы 2 в колбу 4. При нажатии на кнопку "Z" состояние вернулось к состоянию только с выбранной колбой 2.

Если теперь, находясь в полученном состоянии нажать на кнопку "X", то состояние вернется к состоянию до отката (вызов дочернего состояния).

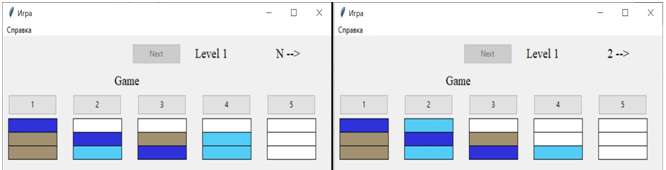


Рисунок 11 – Пример перераспределения цветов

В конце рассмотрим консольный интерфейс. На экране консоли отображаются все состояния, в которых мы находимся. Информация о состоянии отображается при каждом откате (вперед или назад) игрового процесса, а также при каждом создании нового коммита (при игровом действии). Пример внешнего вида окна консоли приведен на рисунке 12.

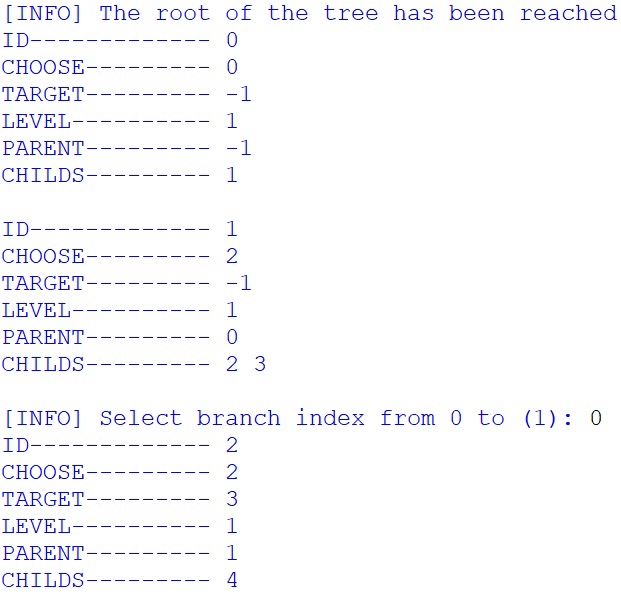


Рисунок 12 – Пример внешнего вида консольного окна

На нем видны состояния с их свойствами (параметры, которые мы сохраняем). Если мы попробуем переместиться в состояние до самого первого коммита (или в состояние после самого позднего), выведется сообщение-предупреждение.

Если параметр CHILDS содержит больше одного значения, выведется сообщение, запрашивающее ввод необходимой пользователю ветки (из предлагаемого диапазона).

# 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