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
«12» декабря 2022 г.

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Выпуск и сопровождение программных продуктов»

на тему: «Разработка системы контроля версий для игры "Сортировка цветов"»

Студент _____ Бессонов М.П.

Шифр 191009

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

Руководитель _____ Лукьянов П.В.

Оценка: «отлично» Дата 12.12.2022.

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» декабря 2022г.

# ЗАДАНИЕ
## на курсовой проект

по дисциплине «Выпуск и сопровождение программных продуктов»

Студент Бессонов М.П. Шифр 191009

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

1 Тема курсового проекта

«Разработка системы контроля версий для игры "Сортировка цветов"»

2 Срок сдачи студентом законченной работы «22» декабря 2022

3 Исходные данные

Разработать систему контроля версий для игры "Сортировка цветов". Проанализировать и описать предметную область, сформулировать требования к ПО. Разработать модель системы контроля версий. Произвести проектирование архитектуры и структуры игрового приложения и системы контроля версий. Разработать основные алгоритмы игрового приложения и системы контроля версий. Реализовать и протестировать ПО.

4 Содержание курсового проекта

Анализ предметной области системы контроля версий для игры «Сортировка цветов»

Разработка модели системы контроля версий для игры «Сортировка цветов»

Проектирование основных алгоритмов системы контроля версий для игры «Сортировка цветов»

Реализация системы контроля версий для игры «Сортировка цветов»

5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта, приложение.

Руководитель _____ Лукьянов П.В.

Задание принял к исполнению: «27» сентября 2022

Подпись студента _____

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Современный мир предъявляет высокие требования к выпускникам вуза, которые по завершении обучения должны обладать рядом компетенций. Одной из важнейших является компетенция техническая. Современный специалист в области IT должен уметь работать в разных программах и с разными ресурсами. Кроме того, он должен уметь работать с различными изменяющимися данными.

Одним из примеров таких данных является любой программный продукт, созданием которого занимается, например, программный инженер. Так как любая достаточно большая программа не может быть написана за один день, а также в такой программе неизбежно будут выявляться различные ошибки, любому разработчику хотелось бы сохранять различные версии своей программы.

Одним из самых простых способов является полное копирование необходимых файлов с присваиванием им специального имени. Существенным недостатком этого метода может являться быстрый рост содержимого файлов и, как итог, рост шанса какой-либо ошибки.

Решением проблемы могут выступить системы контроля версий, которые позволяют вести историю изменений, а также управлять ими.

В некоторых случаях алгоритмы и подходы систем контроля версий можно использовать в игровых приложениях как интересную механику игрового процесса.

Объектом нашей работы является создание системы контроля версий для игры "Сортировка цветов". Предметом нашей работы является разработка модели системы контроля версий для игры "Сортировка цветов".

Цель нашей работы: приобрести навыки разработки системы контроля версий на примере реализации системы контроля версий для игры "Сортировка цветов".

Для достижения цели нам необходимо решить соответствующие задачи:

1) Анализ предметной области игрового приложения и системы контроля версий;

2) Разработка модели системы контроля версий для игры "Сортировка цветов";

3) Проектирование основных алгоритмов игры и алгоритмов системы контроля версий;

4) Реализация игрового приложения и системы контроля версий.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ «СОРТИРОВКА ЦВЕТОВ»

## 1.1 Анализ предметной области: игра «Сортировка цветов»

Игры, основной целью которых является распределение различных разноцветных объектов, являются довольно популярными логическими играми-головоломками. За время своего существования игры такого типа получили широкое распространение на многих устройствах, к числу которых относятся персональные компьютеры и мобильные устройства. Такие игры могут содержать достаточно большое количество неповторяющихся уровней (возможно, различной сложности). Они подходят человеку любого возраста, позволяя ему тренировать свой мозг, решая головоломки.

Основной целью таких игр является разделение цветов таким образом, чтобы все колбы (или другие емкости) были заполнены каким-либо своим цветом (чтобы цвета не были смешаны) [3]. Основным ограничением является то, что выбранный цвет можно перелить либо в пустую колбу, либо в колбу, на вершине которой расположен тот же цвет, который мы хотим перелить (при условии, что в колбе "назначения" есть место).

Пример игрового приложения (начальное и финальное состояние прохождения уровня) представлен на рисунке 1.

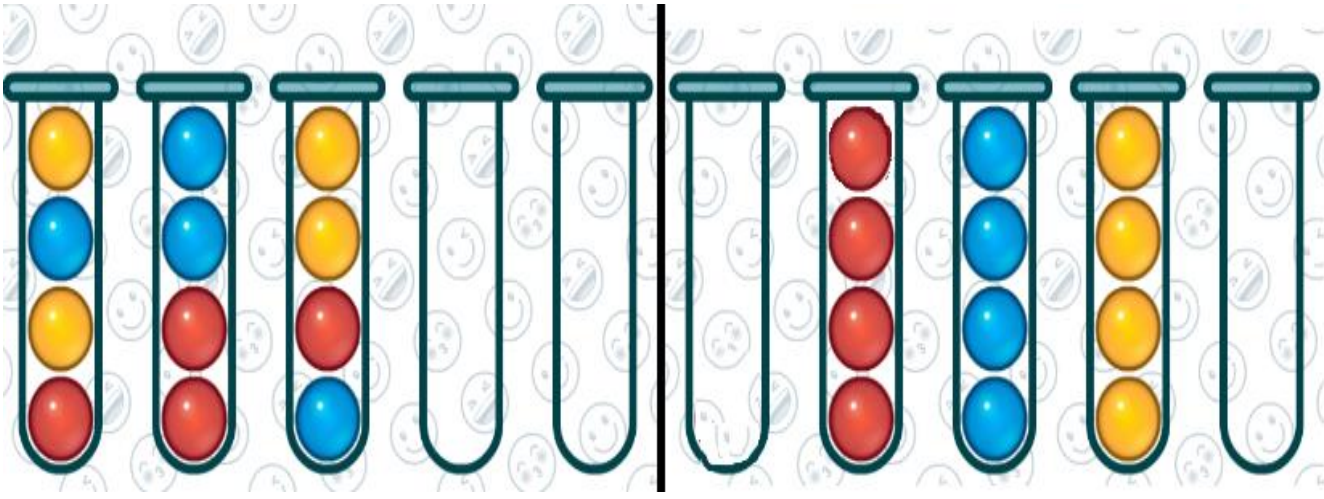

Рисунок 1 – Пример внешнего вида игрового приложения

## 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 [1].

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 [2].

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 [2]. 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) Использование механик для взаимодействия с игрой;
2) Сохранение изменений состояния игрового процесса;
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 ИГРЫ «СОРТИРОВКА ЦВЕТОВ»

Так как мы разрабатываем систему контроля версий для игрового приложения, 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.

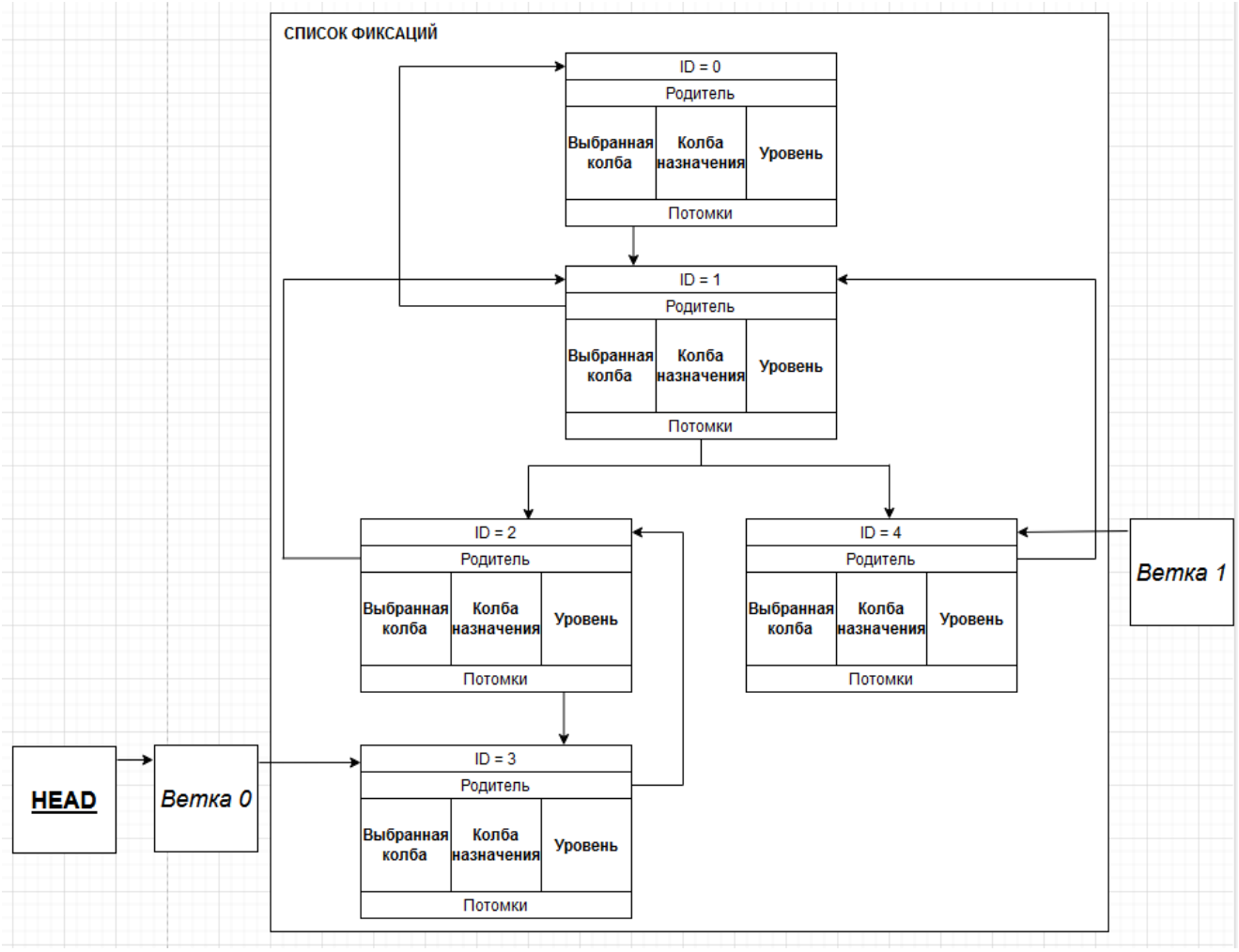


Рисунок 2 – Схема разрабатываемой системы контроля версий

# 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) [6].

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.

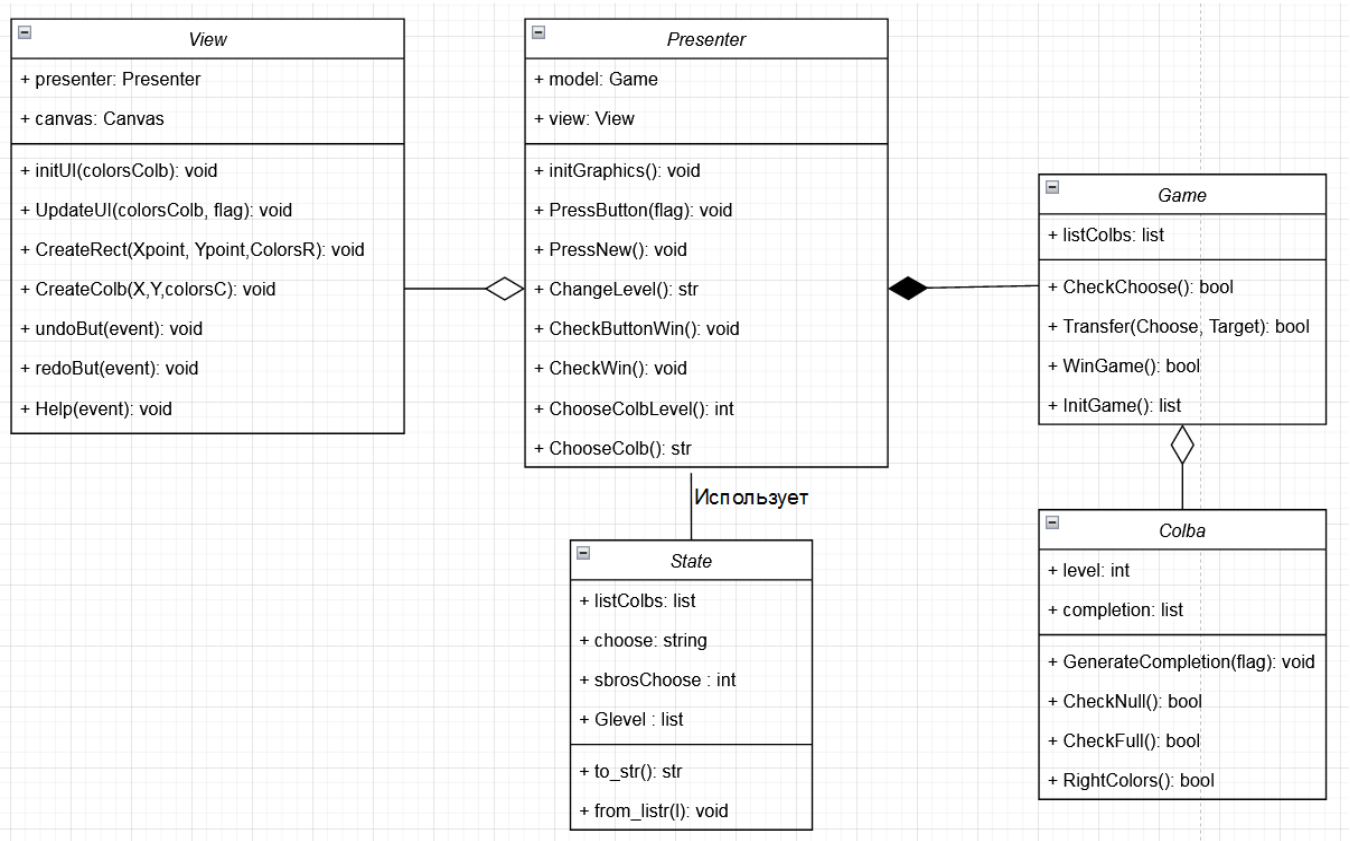


Рисунок 3 – Диаграмма классов для игры "Сортировка цветов"

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.

Класс Presenter обеспечивает связь логики и интерфейса, обрабатывая нажатия кнопок, оперируя полями модели и интерфейса, а также выполняя различные проверки, необходимые для осуществления игрового процесса.

### 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.

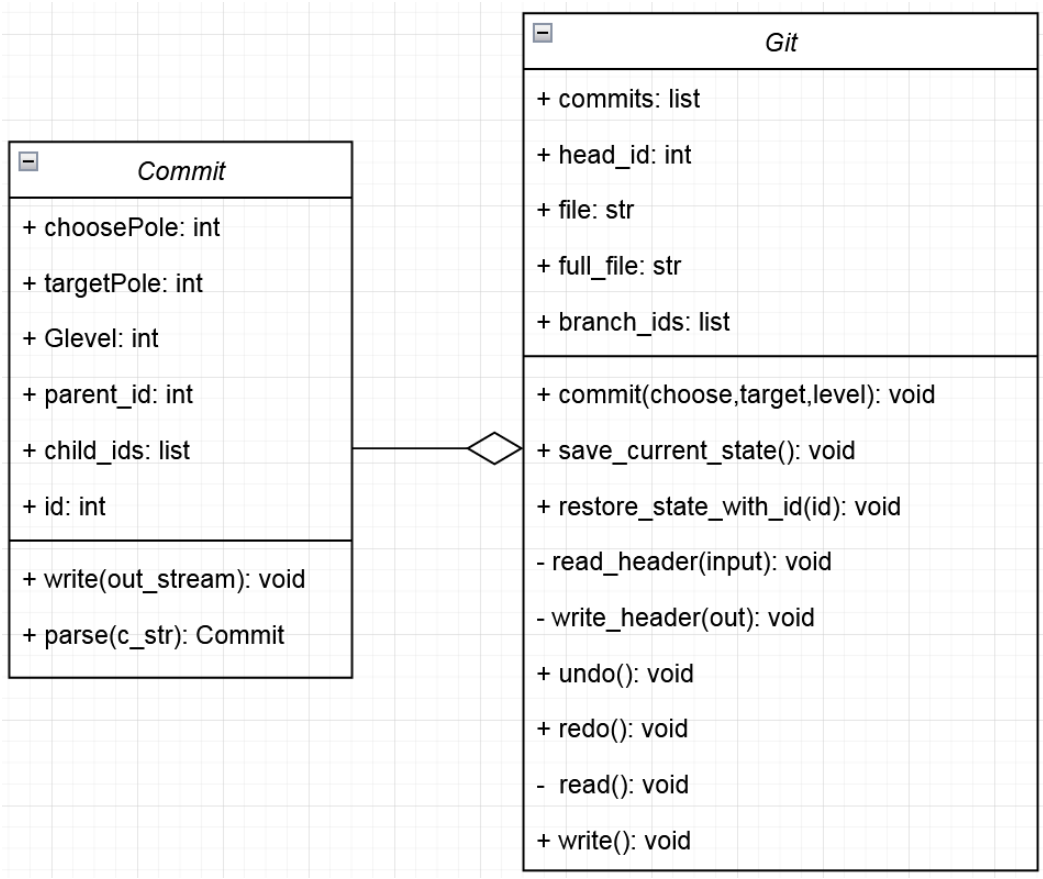


Рисунок 4 – Диаграмма классов для системы контроля версий

Класс Commit включает в себя поля, хранящие выбранную колбу, колбу назначения, текущий уровень, а также идентификаторы родительской, всех дочерних фиксаций и 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 (родительскому) 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, а информация 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 обработка 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.

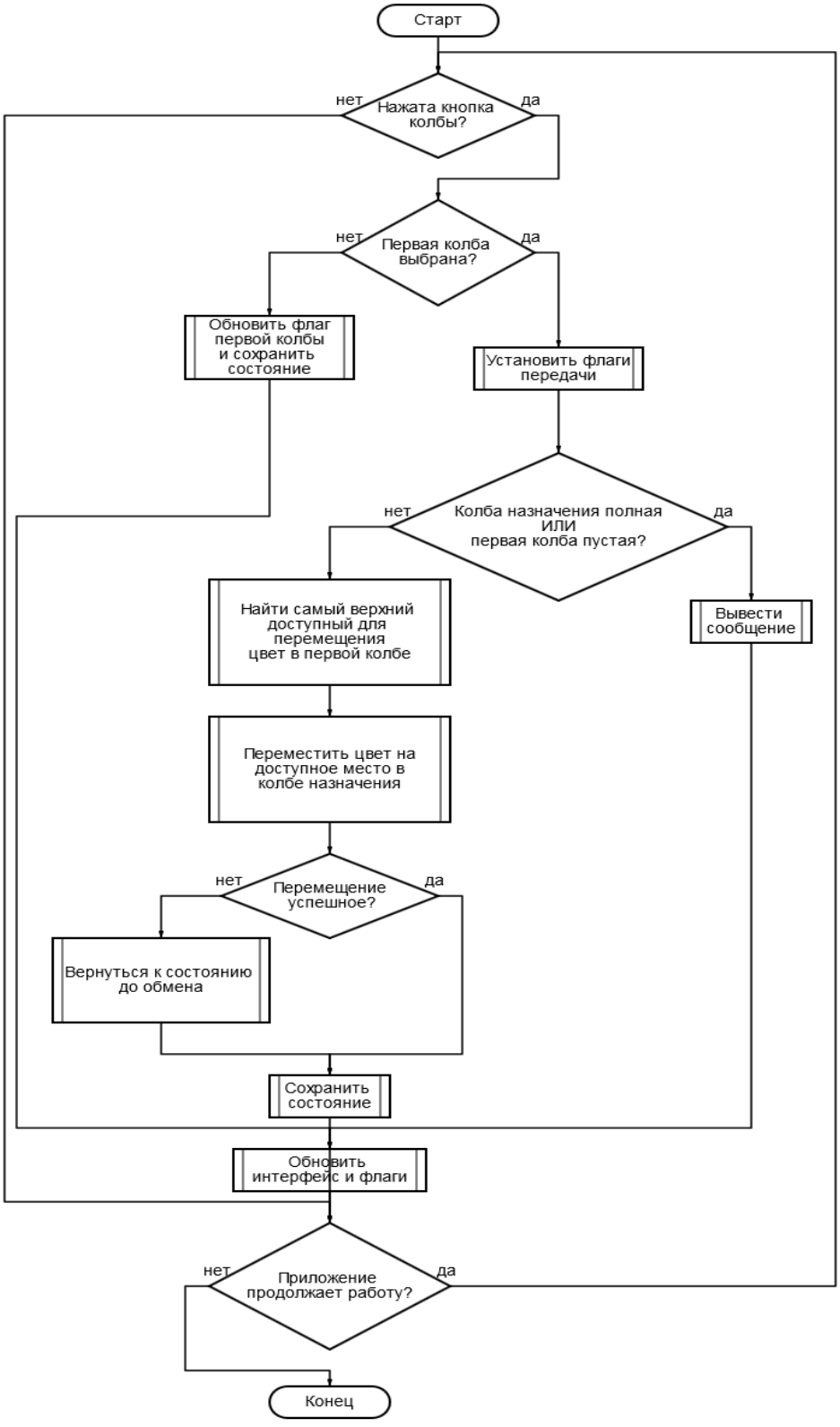


Рисунок 5 – Основной игровой цикл приложения

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

```
def commit(ОТКУДА, КУДА, УРОВЕНЬ):
    СОХРАНИТЬ_ПОЛНОЕ_СОСТОЯНИЕ()
    новый_коммит = Commit(ОТКУДА,КУДА,
                          УРОВЕНЬ, РОДИТЕЛЬ,
                          ИДЕНТИФИКАТОР)
    ЕСЛИ "есть дочерняя фиксация"
      проверить совпадение с текущей пользовательской
      ЕСЛИ "совпадение есть", изменить идентификатор
      ИНАЧЕ создать ветку
    ИНАЧЕ
      добавить потомка в ветку
      текущая_фиксация = новая_текущая_фиксация
      добавить новый коммит в список коммитов
```

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 проверяем, есть ли среди этих фиксаций такая, которая соответствует текущему выбору пользователя. Если есть, то, чтобы не множить одинаковые состояния, происходит смещение идентификатора. Иначе будет создана новая ветка.

Если дочерних фиксаций нет, алгоритм просто добавит информацию о созданном коммите и перезапишет идентификатор текущей фиксации.

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

Алгоритм отката изменяет каждое значение текущего состояния, используя при этом текущую фиксацию и смещая идентификатор к предыдущей 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 необходимые изменения, используя значения, хранящиеся в выбранной нами дочерней фиксации.

Схемы этих двух алгоритмов приведены на рисунке 7.

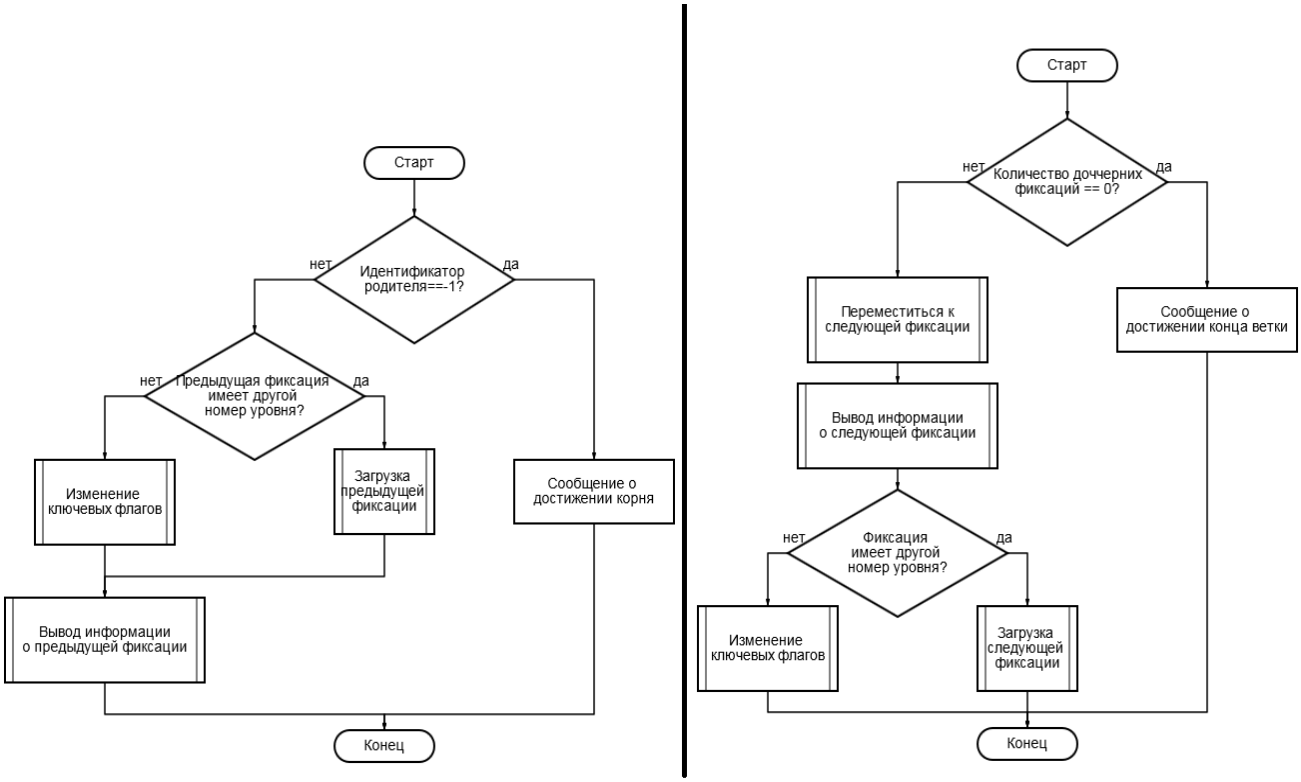


Рисунок 7 – Алгоритм отката изменений (слева) и алгоритм применения изменений, хранящихся в фиксации (справа)

# 4 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-] Ожидается целое число.')
        if len(commit.child_ids) > 1:
                    cid = safe_read_int(f'[INFO] Выберите индекс ветки от 0 до ({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 8 можно увидеть главное окно нашего приложения.

На нем видны основные элементы интерфейса игры.


Рисунок 8 – 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 9).


Рисунок 9 – Пример выбранной колбы

Сверху по центру расположена кнопка перехода на следующий уровень. Она станет доступна только при прохождении текущего уровня (Рисунок 10).

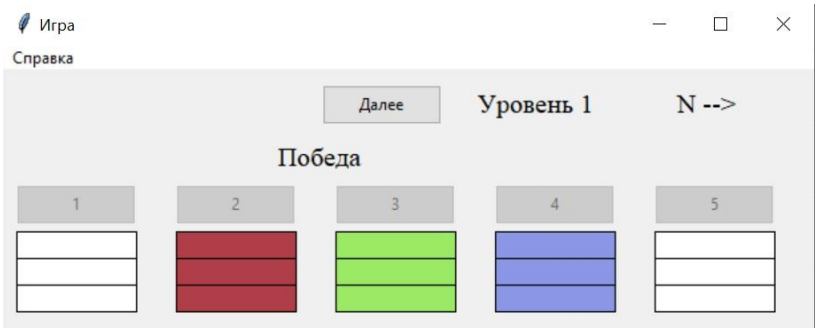


Рисунок 10 – Пример прохождения уровня в приложении

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 на пустое место снизу колбы назначения (при условии возможности осуществления всех описанных выше действий). Пример изменения интерфейса наглядно представлен на рисунке 11. В данном случае было произведено перемещение цвета из колбы 3 в колбу 4.


Рисунок 11 – 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 12 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).

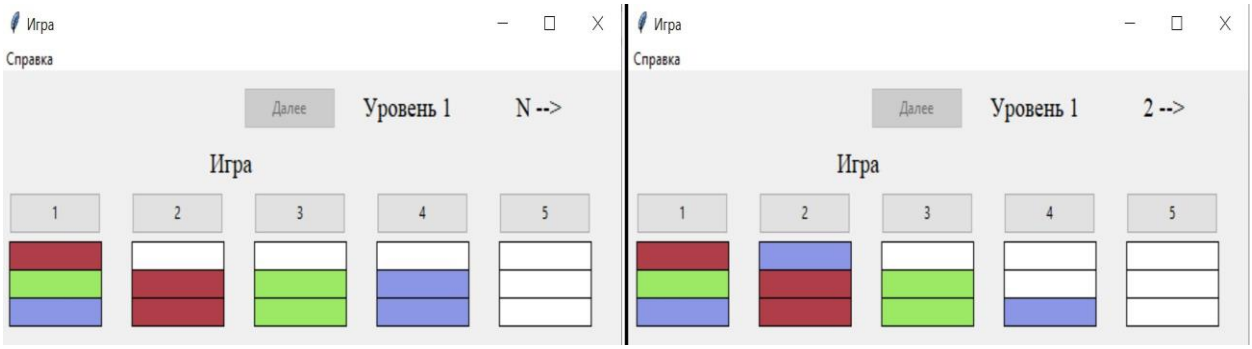


Рисунок 12 – Пример изменения состояния

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 13.

```
[INFO] Корень дерева был достигнут
ID---------- 0
CHOOSE------- 0
TARGET------- -1
LEVEL-------- 1
PARENT------- -1
CHILDS------- 1

ID---------- 1
CHOOSE------- 2
TARGET------- -1
LEVEL-------- 1
PARENT------- 0
CHILDS------- 2 3

[INFO] Выберите индекс ветки от 0 до (1): 0
ID---------- 2
CHOOSE------- 2
TARGET------- 3
LEVEL-------- 1
PARENT------- 1
CHILDS-------

[INFO] Конец ветви был достигнут
```

Рисунок 13 – 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 игры "Сортировка цветов".

Мы проанализировали предметную область, спроектировали и разработали игровое приложение, целью которого является разделение цветов по колбам. Также была спроектирована и разработана система контроля версий, хранящая полные снимки состояния игры, а также определенные флаги, упрощающие переходы между состояниями игры.

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

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. 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).

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

3. SortPuz КАК ИГРАТЬ [Электронный ресурс]. - Режим доступа: https://igry-gid.ru/voprosy/sortpuz-kak-igrat.html?ysclid=l8h59yj7r0227754666 (дата обращения: 25.09.2022).

4. Модуль Tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python [Электронный ресурс]. - Режим доступа: http://kabinet-vplaksina.narod.ru/olderfiles/5/Modul_tkinter.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).

6. Шаблон проектирования MVP. Описание и пример программы [Электронный ресурс]. - Режим доступа: http://vector-sol.ru/Blog/8?ysclid=l8h5osojfd743383961 (дата обращения: 25.09.22).

# ПРИЛОЖЕНИЕ А

## (обязательное)

## РЕАЛИЗАЦИЯ ПРОГРАММЫ

```
class View(Frame):
  def __init__(self):
    super().__init__()
    self.presenter = Presenter(self)
    self.canvas = Canvas(self)
    self.presenter.InitGraphics()

  def initUI(self,colorsColb):
    self.master.title("Игра")
    self.master.bind('z', self.undoBut)
    self.master.bind('x', self.redoBut)
    geom=str(10+120*self.presenter.ChooseColbLevel())+"x"+"300"
    self.master.geometry(geom)
    mainmenu = Menu(self.master)
    self.master.config(menu=mainmenu)
    mainmenu.add_command(label='Справка',command=lambda:self.Help())
    self.pack(fill=BOTH, expand=1)
    for i in range(self.presenter.ChooseColbLevel()):
      self.CretateColb(10+120*i,120,colorsColb[i].completion)
    self.canvas.pack(fill=BOTH, expand=1)
    for i in range(self.presenter.ChooseColbLevel()):
      Button1 = Button(self, text=str(i+1),command=lambda
i=i:self.presenter.PressButton(i+1))
Button1.place(x=55+120*i,y=100,anchor="c",height=30,width=90,bordermode=OUTSIDE)
    ButtonN = Button(self, text="Next",command=lambda:self.presenter.PressNew(3))
ButtonN.place(x=285,y=25,anchor="c",height=30,width=90,bordermode=OUTSIDE)
    ButtonN['state'] = 'disabled'
    labelV = Label(text="Game", font='Times 15')
  labelV.place(x=250,y=65,anchor="c",height=30,width=90,bordermode=OUTSIDE)
    text = "Level "+self.presenter.ChangeLevel()
    labelL = Label(text=text, font='Times 15')
labelL.place(x=400,y=25,anchor="c",height=30,width=90,bordermode=OUTSIDE)
    text = self.presenter.ChooseColb()+" --> "
    labelP = Label(text=text, font='Times 15')
  labelP.place(x=550,y=25,anchor="c",height=30,width=90,bordermode=OUTSIDE)
  def UpdateUI(self,colorsColb,flagV):
    buttons=[]
    self.master.title("Игра")
    geom=str(10+120*self.presenter.ChooseColbLevel())+"x"+"300"
    self.master.geometry(geom)
    self.canvas.destroy()
    self.canvas = Canvas(self)
    for i in range(self.presenter.ChooseColbLevel()):
      self.CretateColb(10+120*i,120,colorsColb[i].completion)
    self.canvas.pack(fill=BOTH, expand=1)
    for i in range(self.presenter.ChooseColbLevel()):
      Button1 = Button(self, text=str(i+1),command=lambda
i=i:self.presenter.PressButton(i+1))
      Button1.place(x=55+120*i, y=100, anchor="c", height=30, width=90,
bordermode=OUTSIDE)
```

```
            buttons.append(Button1)
        ButtonN = Button(self, text="Next",command=lambda:self.presenter.PressNew())
    ButtonN.place(x=285,y=25,anchor="c",height=30,width=90, bordermode=OUTSIDE)
      labelV = Label(text="Game", font='Times 15')
      labelV.place(x=250,y=65,anchor="c",height=30,width=90,bordermode=OUTSIDE)
      text = "Level "+self.presenter.ChangeLevel()
      labelL = Label(text=text, font='Times 15')
      labelL.place(x=400,y=25,anchor="c",height=30,width=90,bordermode= OUTSIDE)
      text = self.presenter.ChooseColb()+" --> "
      labelP = Label(text=text, font='Times 15')
      labelP.place(x=550,y=25,anchor="c",height=30,width=90,bordermode= OUTSIDE)
       if(not flagV):
          ButtonN['state'] = 'disabled'
          for x in buttons: x['state'] = 'enable'
       else:
          labelV['text'] = 'Victory'
          ButtonN['state'] = 'enable'
          for x in buttons:
             x['state'] = 'disabled'
    def CreateRect(self, Xpoint, Ypoint, colorsR):
      self.canvas.create_rectangle(
         Xpoint, Ypoint, Xpoint+90, Ypoint+20,
         outline="Black", fill=colorsR)
    def CretateColb(self,X,Y,colorsC):
      for i in range(self.presenter.ChooseColbLevel()-2):
         if(colorsC[self.presenter.ChooseColbLevel()-3-i]=='0' or
           colorsC[self.presenter.ChooseColbLevel()-3-i]=='0\n'                    or
colorsC[self.presenter.ChooseColbLevel()-3-i]==0): self.CreateRect(X,Y+20*i,"WHITE")
         else: self.CreateRect(X,Y+20*i,colorsC[self.presenter.ChooseColbLevel()-3-i])

    def undoBut(self,event):
      temp1=int(State.Glevel)
      git.undo()
      if(int(State.Glevel)==temp1): self.UpdateUI(State.listColbs,0)
      else: self.presenter.model = Game()
         if(int(State.sbrosChoose)==1): State.choose="N"
         self.UpdateUI(State.listColbs,0)
      self.presenter.CheckButtonWin()

    def redoBut(self,event):
      temp1=int(State.Glevel)
      git.redo()
      if(int(State.Glevel)==temp1): self.UpdateUI(State.listColbs,0)
      else:
         self.presenter.model = Game()
         if(int(State.sbrosChoose)==1):
            State.choose="N"
         self.UpdateUI(State.listColbs,0)
      self.presenter.CheckButtonWin()

  class Colba:
    def __init__(self):
      self.level = 3
      self.completion = ['0']*int(3)
    def GenerateCompletion(self, flag1):
      temp=0
      flag1 = int(flag1)
```

```
        for i in range(flag1,flag1+3):
            self.completion[temp]=COLORS[i]
            temp+=1

    def CheckNull(self):
        if  (all(x == 0 for x in self.completion) or
            all(x == '0' for x in self.completion)): return True
        else: return False

    def CheckFull(self):
        if ((0 in self.completion) or ("0" in self.completion) or ('0\n' in self.completion)):
            return False
        else: return True

    def RightColors(self):
        return True if all(x == self.completion[0] for x in self.completion) else False

class Game:
    def __init__(self):
        State.listColbs = self.InitGame()

    def CheckChoose(self):
        return True if self.CheckChoose != -1 else False

    def Transfer(self, ColbaChoose, ColbaTarget):
        succes = 0
        position = -1
        choose = 0
        if ColbaTarget.CheckFull():
            print("Колба назначения полная")
            return 0
        if ColbaChoose.CheckNull():
            print("Выбранная колба пустая")
            return 0
        for i in range(len(ColbaChoose.completion)-1,-1,-1):
            if(ColbaChoose.completion[i]!=0 and ColbaChoose.completion[i]!='0'):
                choose, ColbaChoose.completion[i] = ColbaChoose.completion[i], '0'
                position = i
                break
        if ColbaTarget.CheckNull():
            ColbaTarget.completion[0], succes = choose, 1
        else:
            for i in range(1, len(ColbaTarget.completion)):
                if((ColbaTarget.completion[i]==0 or (ColbaTarget.completion[i]=='0')) and
ColbaTarget.completion[i-1]==choose):
                    ColbaTarget.completion[i], succes = choose, 1
                    break
        if(succes==0):
            print("Неправильный выбор колбы")
            ColbaChoose.completion[position] = choose
        else: return succes

    def WinGame(self):
        return True if all(x.RightColors() for x in State.listColbs) else False

    def InitGame(self):
        global COLORS
```

```
    colbs = []
    if (not State.listColbs):
        for i in range(int(len(COLORS)/int(3))):
            colba = Colba()
            colba.GenerateCompletion(int(3)*int(i))
            colbs.append(colba)
        colbs.append(Colba())
        colbs.append(Colba())
    else:
        for i in range(int(len(State.listColbs))):
            colba = Colba()
            for z in range(3):
                colba.completion[z]=State.listColbs[i][z]
            colbs.append(colba)
        colbs[-1].completion[-1]=colbs[-1].completion[-1][:-1]
    return colbs
```

**class Presenter:**

```
    def __init__(self,view):
        self.model = Game()
        self.view = view

    def InitGraphics(self):
        if(int(State.sbrosChoose)==1):
            State.choose="N"
        self.view.initUI(State.listColbs)
        if (self.model.WinGame()==True):
            State.sbrosChoose = 1
            self.view.UpdateUI(State.listColbs,1)

    def PressButton(self,flag):
        if(State.choose!="N"):
            State.sbrosChoose = 1
            res = self.model.Transfer(State.listColbs[int(State.choose)-1],State.listColbs[flag-1])
            if(res==1): git.commit(int(State.choose)-1,flag-1,State.Glevel)
            else: git.commit(int(State.choose)-1,-2,State.Glevel)
            State.choose = "N"
        else:
            State.choose = flag
            State.sbrosChoose = 0
            git.commit(int(State.choose)-1,-1,State.Glevel)
        self.CheckWin()

    def PressNew(self):
        global COLORS
        temp = int(State.Glevel)
        COLORS=[]
        COLORS = GenerateRandomColors()
        State.listColbs = []
        self.model = Game()
        State.Glevel = str(temp+1)
        self.view.UpdateUI(State.listColbs,0)
        git.commit( 0,-1,State.Glevel)


    def ChangeLevel(self):
        return str(State.Glevel)
```

```
    def CheckButtonWin(self):
        if (self.model.WinGame()==True):
            State.sbrosChoose = 1
            self.view.UpdateUI(State.listColbs,1)
    def CheckWin(self):
        if (self.model.WinGame()==True):
            self.view.UpdateUI(State.listColbs,1)
        else: self.view.UpdateUI(State.listColbs,0)
    def ChooseColbLevel(self):
        return int(State.listColbs[0].level)+2
    def ChooseColb(self):
        return str(State.choose)

class State:
    listColbs = []
    choose = "N"
    sbrosChoose = 0
    Glevel = 1

    @staticmethod
    def to_str():
        SaveList = []
        for b in State.listColbs:
            for z in b.completion: SaveList.append(z)
        l = [State.choose,State.sbrosChoose,State.Glevel,*SaveList]
        return ','.join(list(map(lambda x: str(x), l)))

    @staticmethod
    def from_list(l):
        State.choose = l[0]
        State.sbrosChoose = l[1]
        State.Glevel = l[2]
        body = l[3:]
        State.listColbs.clear()
        for i in range(0, len(body), int(3)):
            colba = []
            for z in range(int(3)):
                colba.append(body[i+z])
            State.listColbs.append(colba)
        if(len(State.listColbs[-1][-1])==2):
            State.listColbs[-1][-1]=='0'

class Commit:
    def __init__(self, choosePole=0, targetPole=-1, Glevel=1, parent_id=-1, id=0):
        self.choosePole = choosePole
        self.targetPole = targetPole
        self.Glevel = Glevel
        self.parent_id = parent_id
        self.child_ids = []
        self.id = id

    def write(self, out_stream):
        l = [self.choosePole,
            self.targetPole,
            self.Glevel,
            self.parent_id,
            self.id,
```

```
            *self.child_ids]
        out_stream.write(','.join(list(map(lambda x: str(x), l))))

    @staticmethod
    def parse(c_str: str):
        c = Commit()
        data = list(map(lambda x: str(x), c_str.rstrip('\n\r').split(',')))
        c.choosePole = data[0]
        c.targetPole = data[1]
        c.Glevel = data[2]
        c.parent_id = data[3]
        c.id = data[4]
        c.child_ids = data[5:]
        return c

class Git:
    def __init__(self):
        self.commits = [Commit()]
        self.head_id = 0
        self.git = 'git.dt'
        self.git_ls = 'git_ls.dt'
        self.branch_ids = [0]
        self.__read()
        if len(self.commits) > 1:
            self.restore_state_with_id(int(self.head_id))

    def commit(self, choosePole, targetPole, Glevel):
        self.save_current_state()
        new_commit = Commit(choosePole=choosePole,
                            targetPole=targetPole,
                            Glevel=Glevel,
                            parent_id=self.head_id,
                            id=len(self.commits))
        if len(self.commits[int(self.head_id)].child_ids) > 0:
            self.branch_ids.append(new_commit.id)
        else:
            self.branch_ids.remove(int(self.head_id))
            self.branch_ids.append(new_commit.id)
        self.commits[int(self.head_id)].child_ids.append(new_commit.id)
        self.head_id = new_commit.id
        self.commits.append(new_commit)
    def save_current_state(self):
        with open(self.git_ls, 'a') as f:
            f.write(f'{self.head_id},{State.to_str()}\n')
    def restore_state_with_id(self, s_id):
        try:
            print(f'[INFO] Try to restore {s_id}.')
            with open(self.git_ls, 'r') as f:
                for line in f:
                    data = list(map(lambda x: x, line.split(',')))
                    if (int(data[0]) == int(int(s_id) - 1)):
                        print('[INFO] restoring...')
                        if(int(int(s_id)) == 0): data[1]="N"
                        State.from_list(data[1:])
                        return
                print('[INFO] restoring...')
                if(int(int(s_id)) == 0): data[1]="N"
```

```
                State.from_list(data[1:])
            except FileNotFoundError: pass
        def __read_header(self, input_stream):
            self.head_id = int(input_stream.readline())
            self.branch_ids = list(map(lambda x: int(x), input_stream.readline().split(',')))
        def __write_header(self, out_stream):
            out_stream.write(str(self.head_id) + '\n')
            out_stream.write(','.join(map(lambda x: str(x), self.branch_ids)) + '\n')
        def undo(self):
            if int(self.commits[int(self.head_id)].parent_id) == -1:
                print('[INFO] The root of the tree has been reached')
            else:
                self.head_id = int(self.head_id)
                commit = self.commits[self.head_id]
                commit1 = self.commits[int(commit.parent_id)]
                if(int(State.Glevel)!=int(commit1.Glevel)):
                    self.restore_state_with_id(int(commit.parent_id))
                    self.head_id = int(commit.parent_id)
                else:
                    self.head_id = commit.parent_id
                    State.Glevel = commit.Glevel
                    if(int(commit.targetPole)!=-1 and int(commit.targetPole)!=-2):
                        State.choose = int(commit.choosePole)+1
                        State.sbrosChoose=1
                        temp1 = -1
                        temp2 = -1
                        if(int(commit.targetPole)!=int(commit.choosePole)):
                            for i in range(len(State.listColbs[int(commit.choosePole)].completion)-1,-1,-1):
                                if(State.listColbs[int(commit.choosePole)].completion[i]=='0'):
                                    temp1 = i
                            for i in range(len(State.listColbs[int(commit.targetPole)].completion)-1):
                                if(State.listColbs[int(commit.targetPole)].completion[i+1]=='0'                    and
State.listColbs[int(commit.targetPole)].completion[i]!='0'):
                                    temp2 = i
            State.listColbs[int(commit.choosePole)].completion[int(temp1)],State.listColbs[int(commit.targ
etPole)].completion[int(temp2)]\
=State.listColbs[int(commit.targetPole)].completion[int(temp2)],State.listColbs[int(commit.choosePole)].
completion[int(temp1)]
                    else:
                        if(int(commit.targetPole)==-2): State.choose = int(commit.choosePole)+1
                        else: State.choose = "N"
            cc = self.commits[int(self.head_id)]
            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)
            print()
        def redo(self):
            self.head_id = int(self.head_id)
            if len(self.commits[self.head_id].child_ids) == 0:
                print('[INFO] The end of the branch has been reached')
            else:
                commit = self.commits[self.head_id]
                cid = 0
                if len(commit.child_ids) > 1:
```

```
                    cid = safe_read_int(f'[INFO] Select branch index from 0 to ({len(commit.child_ids) -
1}): ')
                self.head_id = commit.child_ids[cid]
                cc = self.commits[int(self.head_id)]
                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)
                print()
                if(int(State.Glevel)!=int(cc.Glevel)):
                    self.restore_state_with_id(int(self.head_id))
                else:
                    State.Glevel = cc.Glevel
                    if(int(cc.targetPole)!=-1):
                        State.choose = "N"
                        State.sbrosChoose=1
                        temp1 = -1
                        temp2 = -1
                        if(int(cc.targetPole)!=int(cc.choosePole) and int(cc.targetPole)!=-2):
                            for i in range(len(State.listColbs[int(cc.choosePole)].completion)):
                                if(State.listColbs[int(cc.choosePole)].completion[i]!='0'):
                                    temp1 = i
                            for i in range(len(State.listColbs[int(cc.targetPole)].completion)-1,-1,1):
                                if(State.listColbs[int(cc.targetPole)].completion[i]=='0'):
                                    temp2 = i
                            if(temp2==-1):
                                temp2 = 0
            State.listColbs[int(cc.choosePole)].completion[int(temp1)],State.listColbs[int(cc.targetPole)].co
mpletion[int(temp2)]\
=State.listColbs[int(cc.targetPole)].completion[int(temp2)],State.listColbs[int(cc.choosePole)].completion
[int(temp1)]
                    else:
                        if(cc.choosePole!="N"): State.choose = int(cc.choosePole)+1
                        else: State.choose = "N"
    def __read(self):
        try:
            with open(self.git, 'r') as f:
                self.__read_header(f)
                self.commits.clear()
                for line in f:
                    self.commits.append(Commit.parse(line))
        except FileNotFoundError:
            pass
    def write(self):
        with open(self.git, 'w') as f:
            f.write('')
            self.__write_header(f)
            for c in self.commits:
                c.write(f)
                f.write('\n')
        f.close()
```