

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

Л.В.В. Руководитель

« 24 » декабря 2021 г.

КУРСОВОЙ ПРОЕКТ

по дисциплине «Выпуск и сопровождение программных продуктов»

на тему: «Разработка системы контроля версий для игры “Морской бой”»

Студент Беликов П.Г. Беликов П.Г.

Шифр 180818

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Направленность (профиль) Промышленная разработка программного обеспечения

Группа 81ПГ

Руководитель Л.В.В. Лукьянов П.В.

Оценка: « отлично »

Дата 24.12.2021

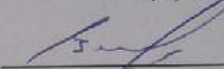
Орел 2021

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:

 Зав. кафедрой  
«22» сентября 2021 г.

**ЗАДАНИЕ**  
на курсовой проект

по дисциплине «Выпуск и сопровождение программных продуктов»

Студент Беликов П.Г.

Шифр 180818

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Направленность (профиль) Промышленная разработка программного  
обеспечения

Группа 81ПГ

1 Тема курсового проекта

«Разработка системы контроля версий для игры “Морской бой”»

2 Срок сдачи студентом законченной работы «24» 12 2021 г.

### 3 Исходные данные

Принципы работы системы контроля версий.

### 4 Содержание курсового проекта

Исследование предметной области для системы контроля версий для игры "Морской бой"

Разработка модели системы контроля версий для игры "Морской бой"

Проектирование основных алгоритмов работы системы контроля версий для игры "Морской бой"

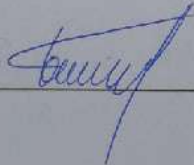
Реализация системы контроля версий для игры "Морской бой"

### 5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта; приложение.

Руководитель  Лукьянов П.В.

Задание принял к исполнению: « 22 » 09 202 1

Подпись студента 

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ДЛЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ” .....	5
1.1 Анализ предметной области.....	5
1.2 Формирование требований к программному обеспечению.....	6
2 РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ” .....	8
3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ” .....	10
3.1 Проектирование игры “Морской бой” .....	10
3.2 Проектирование системы контроля версий .....	10
3.3 Проектирование алгоритмов системы контроля версий.....	12
4 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ” .....	18
4.1 Особенности программной реализации.....	18
4.2 Пример работы программы.....	18
ЗАКЛЮЧЕНИЕ .....	26
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	27
Приложение 1 .....	28
Приложение 2 .....	39

## ВВЕДЕНИЕ

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

В последнее время файлы являются конечным результатом для многих профессий (для примера, писательскую деятельность, научные работы и, конечно, разработку программного обеспечения). Тратится много времени и сил на разработку и поддержку этих файлов, и никто не хочет, чтобы пришлось тратить еще больше времени и сил на восстановление данных потерянных в результате каких-либо изменений.

Целью курсовой работы является проектирование и разработка системы контроля версий для игры “Морской бой”.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Исследовать предметную область.
2. Определить требования к системе контроля версий.
3. Спроектировать алгоритмы для игры.
4. Спроектировать алгоритмы для системы контроля версий.
5. Реализовать игру.
6. Реализовать систему контроля версий.

# 1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ДЛЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ”

## 1.1 Анализ предметной области.

Предметной областью работы является система контроля версий для игры “Морской бой”.

Морской бой - игра для двух участников, в которой игроки по очереди называют координаты на неизвестной им карте соперника. Если у соперника по этим координатам имеется корабль (координаты заняты), то корабль или его часть «топится», а попавший получает право сделать ещё один ход. Цель игрока — первым потопить все корабли противника.

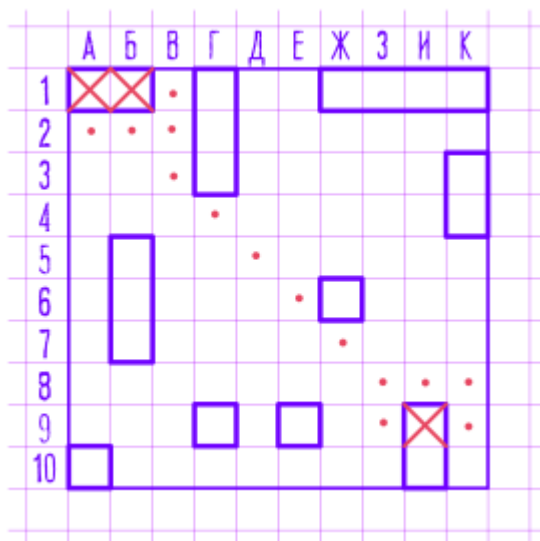


Рисунок 1 – Пример одного поля игры.

Игровое поле — обычно квадрат 10×10 у каждого игрока, на котором размещается флот кораблей. Вертикали обычно нумеруются сверху вниз, а горизонтали помечаются буквами слева направо. При этом используются буквы русского алфавита от «а» до «к» (буквы «ё» и «й» обычно пропускаются) либо от «а» до «и» (с использованием буквы «ё»), либо буквы латинского алфавита от «а» до «j». Иногда используется слово «республика» или «снегурочка», так как в этих 10-буквенных словах ни одна буква не повторяется. Поскольку существуют различные варианты задания системы координат, то об этом лучше заранее договориться.

Размещаются:

- 1 корабль — ряд из 4 клеток («четырёхпалубный»; линкор)
- 2 корабля — ряд из 3 клеток («трёхпалубные»; крейсера)
- 3 корабля — ряд из 2 клеток («двухпалубные»; эсминцы)
- 4 корабля — 1 клетка («однопалубные»; торпедные катера)

При размещении корабли не могут касаться друг друга сторонами и углами.

Система контроля версий – это система, хранящая какое-то количество версий какого-либо объекта. В дальнейшем при помощи СКВ можно вернуться к определенной версии.

Все сохраненные версии хранятся в репозитории. Репозиторий содержит определенное количество веток (минимум одна – мастер ветка – создается вместе с репозиторием). Сохраненные версии хранятся в коммитах, расположенных в определенной последовательности внутри веток.

Разрабатываемое ПО является консольным приложением, которое позволяет играть в игру “Морской бой” с компьютером, а также СКВ, которая позволяет управлять состоянием игры.

## **1.2 Формирование требований к программному обеспечению.**

Поскольку основной разрабатываемой частью является именно система контроля версий, то и требования будут относиться непосредственно к ней.

Функциональные требования:

- система позволяет как сохранять состояния игры с помощью коммитов, так и удалять их;
- система позволяет создавать и удалять новые ветки;
- система позволяет перемещаться между ветками;
- система имеет консольный интерфейс;
- система доступна в любой момент игры.

Нефункциональные требования:

- репозиторий представляет собой один единственный файл;
- каждая ветка представляет собой отдельный файл;
- в репозитории всегда должна быть хотя бы одна ветка (мастер по умолчанию).



## 2 РАЗРАБОТКА МОДЕЛИ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ”

Исходя из анализа предметной области и установленных требований, необходимо разработать модель системы контроля версий для игры “Морской бой”.

Для установленной предметной области реализовать классическую систему контроля версий не получится. По изначальной задумке, каждая последующая версия должна хранить изменения относительно предыдущей версии (в данном случае речь идет о последовательности ходов). Однако в морском бое, помимо ходов, сделанных игроком, существуют ходы, которая делает сама система (игра).

Речь идет о ситуации, при которой корабль одной из сторон оказывается полностью уничтожен. В таком случае игра самостоятельно заполняет поля вокруг уничтоженного корабля “выстрелами мимо”. Проблема в том, что в это заполнение может попасть и выстрел, сделанный самим игроком: в таком случае, при сохранении изменений относительно предыдущей версии может произойти случай, при котором одно и то же поле было атаковано дважды.

Таким образом, на Рисунке 2 представлена схема структуры хранения данных.

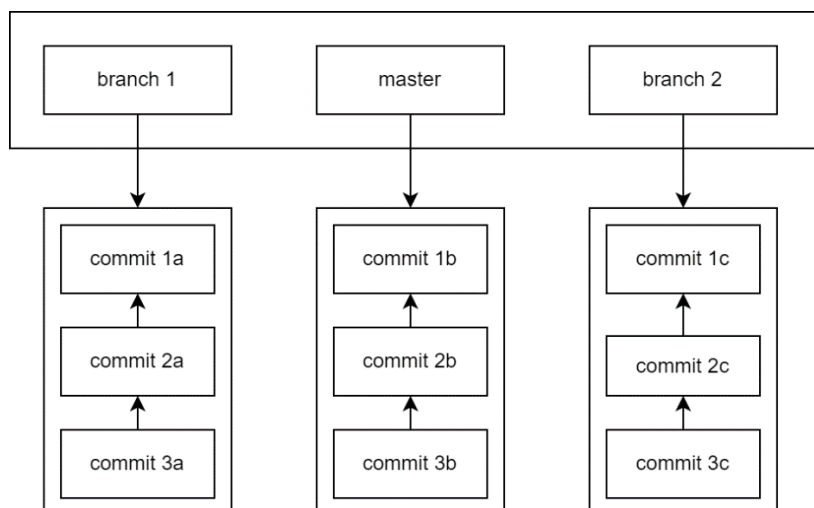


Рисунок 2 – Схема структуры хранения данных.

Каждый коммит будет храниться внутри ветки ссылаясь на предыдущий коммит данной ветки, т.е. одна ветка будет содержать список связанных между собой коммитов, в каждом из которых будут храниться данные о состоянии игры на момент его создания и список новых ходов, относительно предыдущего коммита.

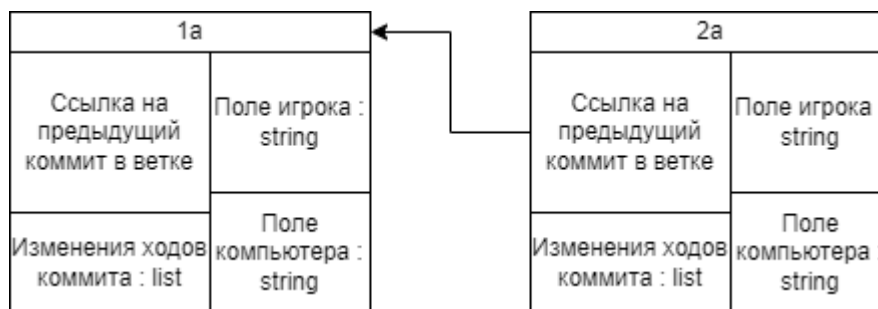


Рисунок 3 – Схема хранения коммита.

На Рисунке 3 изображена схема данных, которые хранятся в коммите.

Каждый коммит содержит информацию о полях игрока и компьютера (записанную в виде строки), изменения ходов относительно предыдущего коммита и ссылку на предыдущих коммит данной ветки. Таким образом, при загрузке коммита будут данные будут загружаться последовательно, начиная с первого коммита и до выбранного пользователем.

Исходя из всего вышеперечисленного, была предложена такая организация данных:

- репозиторий представляет собой отдельную папку на диске;
- ветка представляет собой файл внутри репозитория;
- каждый коммит представляет собой строку данных внутри файла ветки (каждый новый коммит – новая строка).

### 3 ПРОЕКТИРОВАНИЕ ОСНОВНЫХ АЛГОРИТМОВ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ”

#### 3.1 Проектирование игры “Морской бой”

Основной программой является приложение игры “Морской бой”. На Рисунке 4 представлена диаграмма классов данного приложения.

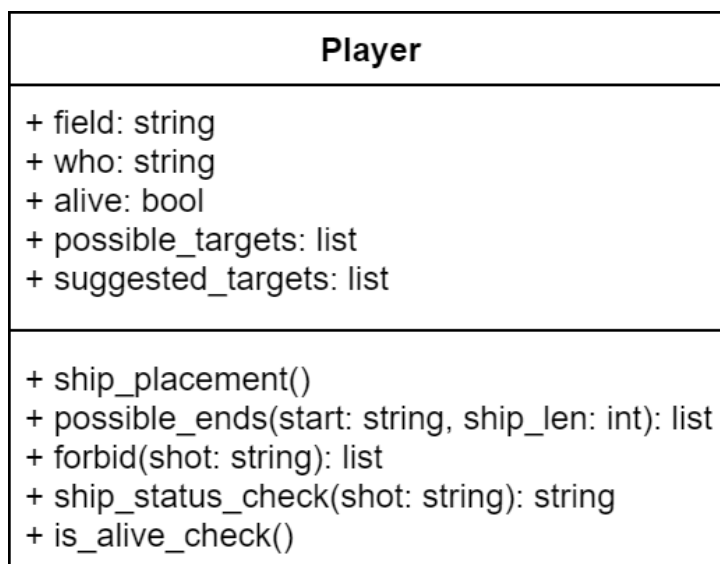


Рисунок 4 – Диаграмма классов приложения “Морской бой”.

Основным и единственным классом является класс Player. Он хранит в себе состояние игрового поля игрока, его имя, статус, возможные цели и предложенные цели (для игры компьютера).

Методы класса позволяют автоматически или вручную расставить корабли, заполнить поля вокруг уничтоженного корабля, проверить статус корабля (жив или уничтожен), а также проверить состояние игрока.

Сама игра будет производиться посредством создания двух экземпляров класса Player и отображением их полей в консольном интерфейсе.

#### 3.2 Проектирование системы контроля версий

Система контроля версий позволяет сохранять и загружать определенные состояния игры.

На Рисунке 5 представлена диаграмма классов системы контроля версий.

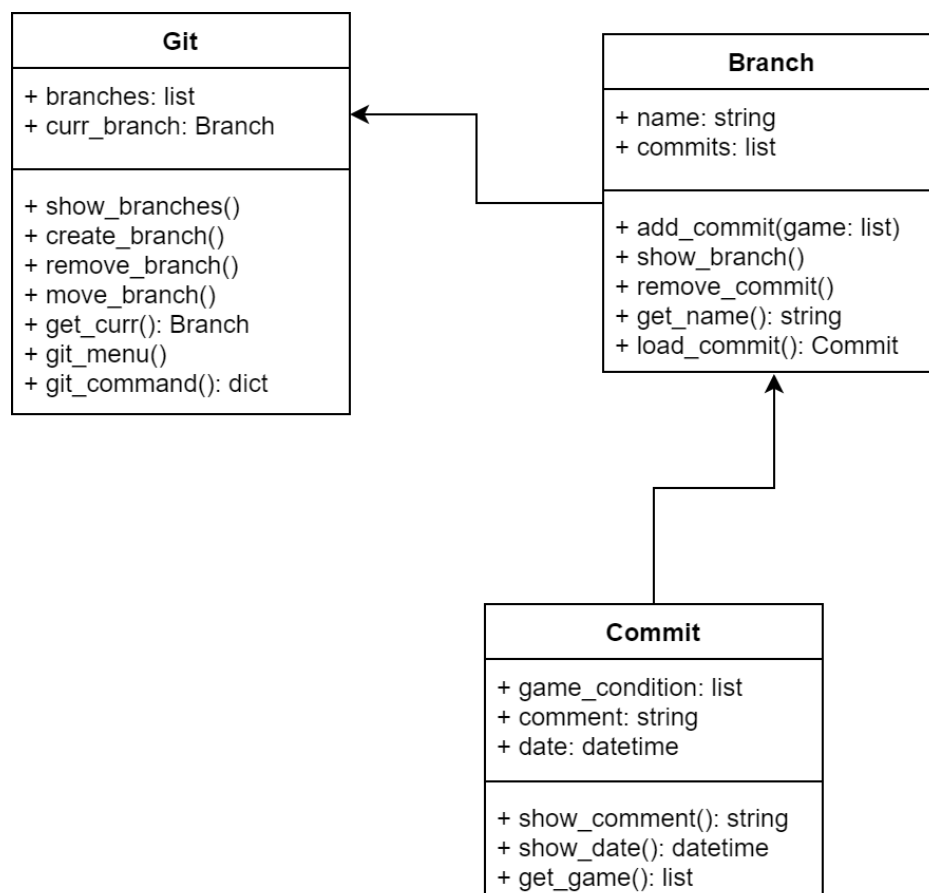


Рисунок 5 – Диаграмма классов системы контроля версий.

Основным и собирательным классом является класс **Git**. Он содержит в себе информацию обо всех существующих ветках (в том числе о текущей). Именно в нем осуществляется управление ветками: создание, переход, удаление, просмотр.

Класс **Branch** хранит информацию обо всех коммитах, сохраненных на нем, а также свое собственное имя. В данном классе осуществляется добавление, просмотр, удаление и загрузка коммитов.

Класс **Commit** хранит в себе определенное состояние игры, время сохранения и комментарий игрока к нему. В системе контроля версий это самый базовый класс.

### 3.3 Проектирование алгоритмов системы контроля версий.

Основными алгоритмами для системы контроля версий являются алгоритмы добавления и удаления коммита, создания и удаления ветки.

Алгоритм добавления коммита представлен на рисунке 6.

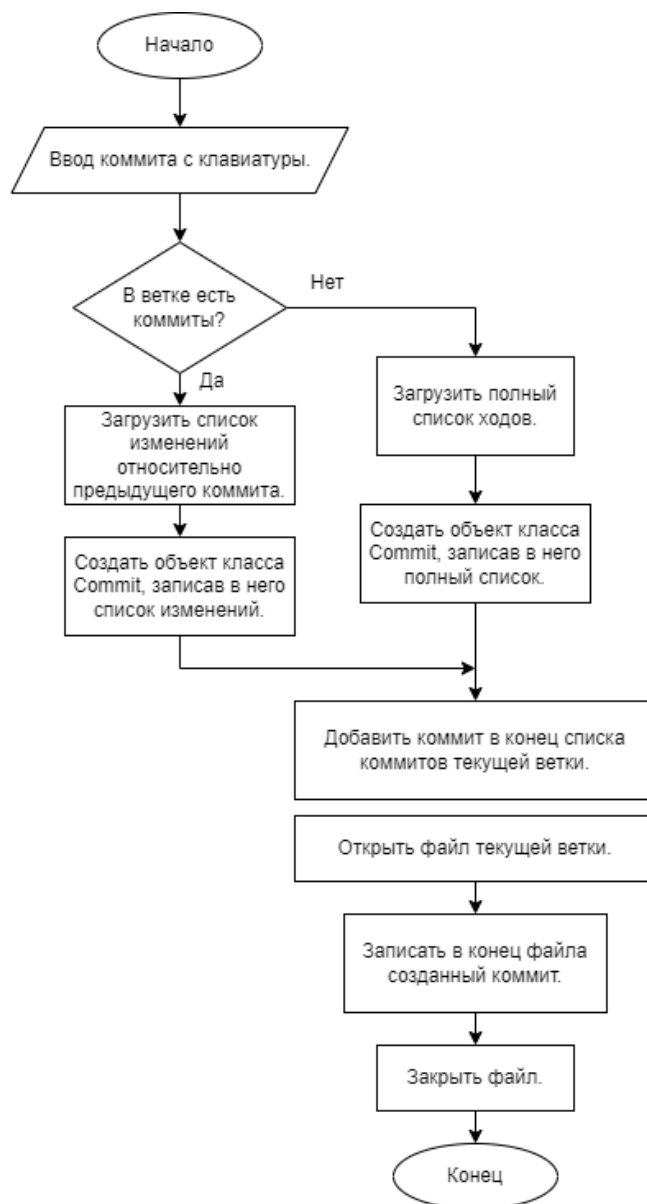


Рисунок 6 – Алгоритм добавления коммита.

В самом начале система просит пользователя ввести комментарий, после чего собирает текущие данные времени и состояние игры, после чего создает коммит. Стоит отметить, что в состояние игры входит состояние двух игроков и порядок ходов партии. Затем программа открывает файл текущей ветки и записывает данные созданного коммита в виде одной строчки, после чего закрывает файл.

Алгоритм удаления коммита представлен на Рисунке 7.

При удалении коммита программа сначала показывает все существующие коммиты на текущей ветке и просит выбрать нужный. После этого выбранный коммит удаляется из списка коммитов текущей ветки. Новый список перезаписывается в файл ветки.

На Рисунке 8 представлен алгоритм создания ветки.

При создании новой ветки программа сначала просит пользователя ввести имя новой ветки. Затем создается объект класса Branch и добавляется в общий список веток. В конце создается файл для хранения коммитов созданной ветки.



Рисунок 7 – Алгоритм удаления коммита.

На Рисунке 9 представлен алгоритм удаления ветки.

Как и в предыдущем алгоритме, сначала программа показывает список всех существующих веток и просит выбрать нужную. Если выбранная ветка

является текущей, то пользователь получит сообщение об ошибке и предложение сменить ветку. Если выбранная ветка не является текущей, то она будет удалена из общего списка веток. Файл для хранения ее коммитов будет также удален с компьютера.

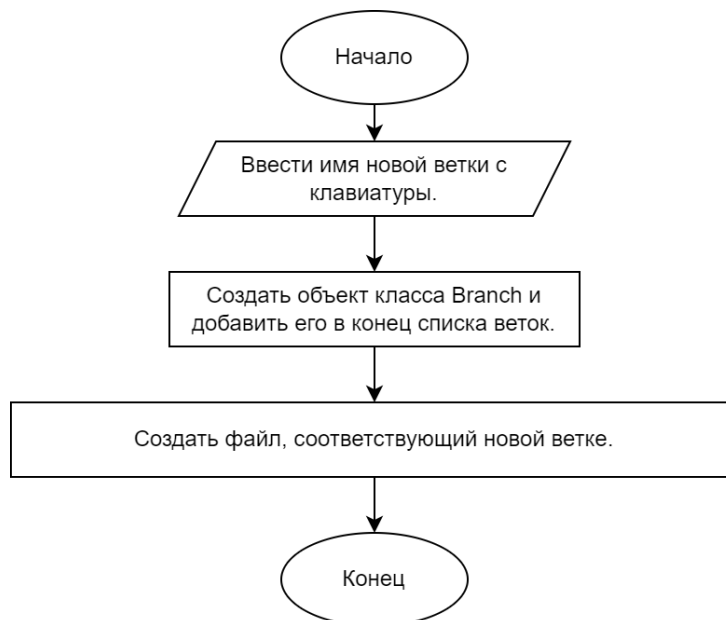


Рисунок 8 - Алгоритм создания ветки.

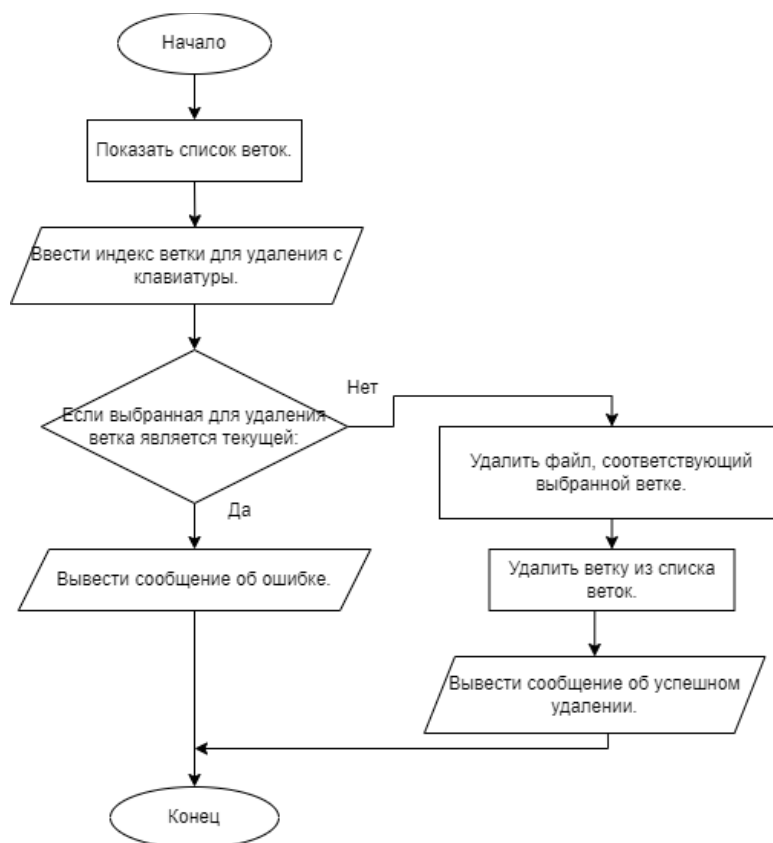


Рисунок 9 – Алгоритм удаления ветки.

На Рисунке 10 представлен алгоритм загрузки коммита.

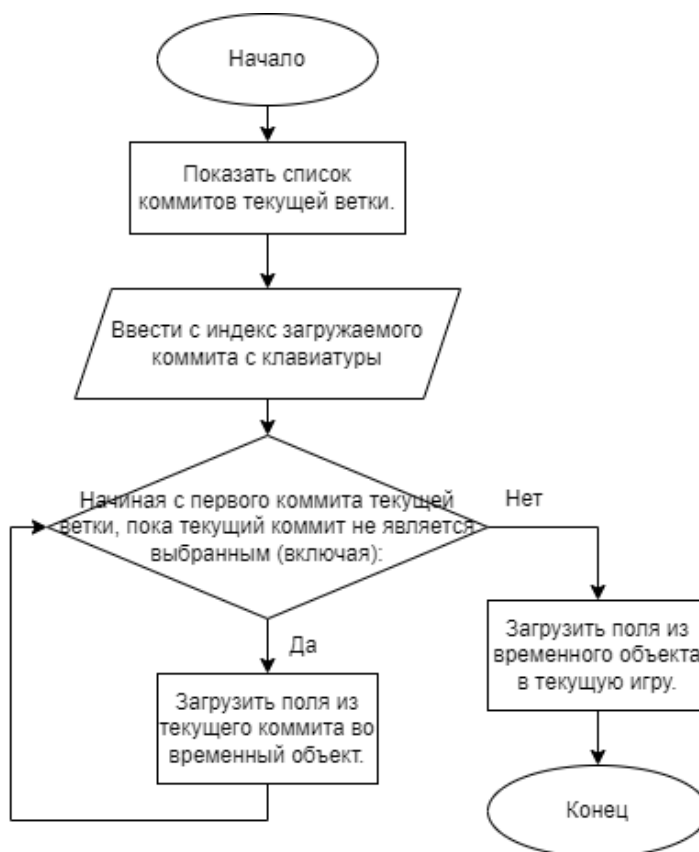


Рисунок 10 – Алгоритм загрузки коммита.

Чтобы загрузить коммит, система сначала предлагает пользователю выбрать один из коммитов на текущей ветке. После выбора, из коммитов, начиная с первого коммита текущей ветки и до выбранного (включительно), извлекаются поля и последовательно записываются во временный объект. После считывания последнего (выбранного) коммита, данные из временного объекта переносятся в текущую игру.

На Рисунке 11 представлен алгоритм перемещения по ветке в направлении вперед.

Перемещение по коммитам вперед осуществляется только если какой-то коммит был загружен в игру. Если был загружен последний коммит в ветке, то система предупредит, что дальше загружать нечего. Если был загружен не последний коммит, то система перейдет и загрузит следующий по списку коммит (загружаемый коммит содержит ссылку на изначальный коммит).



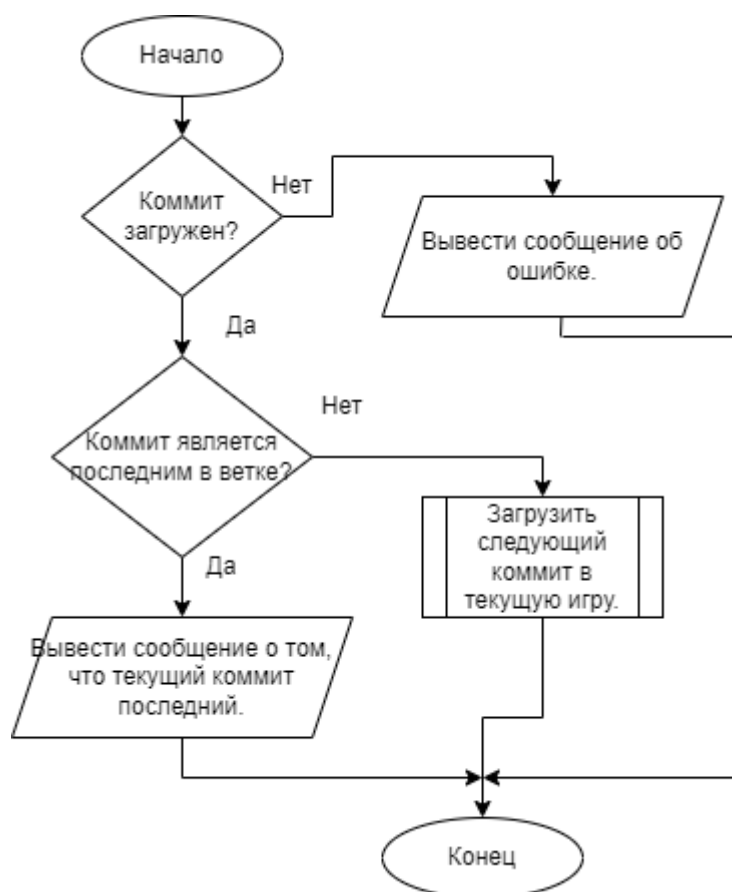


Рисунок 11 – Алгоритм перехода к следующему коммиту.

На Рисунке 12 изображен алгоритм перемещения по ветке в направлении назад.

Перемещение по коммитам назад осуществляется только если какой-то коммит был загружен в игру. Если был загружен первый коммит в ветке, то система предупредит, что дальше загружать нечего. Если был загружен не первый коммит, то система перейдет и загрузит предыдущий по списку коммит (текущий коммит содержит ссылку на загружаемый коммит).

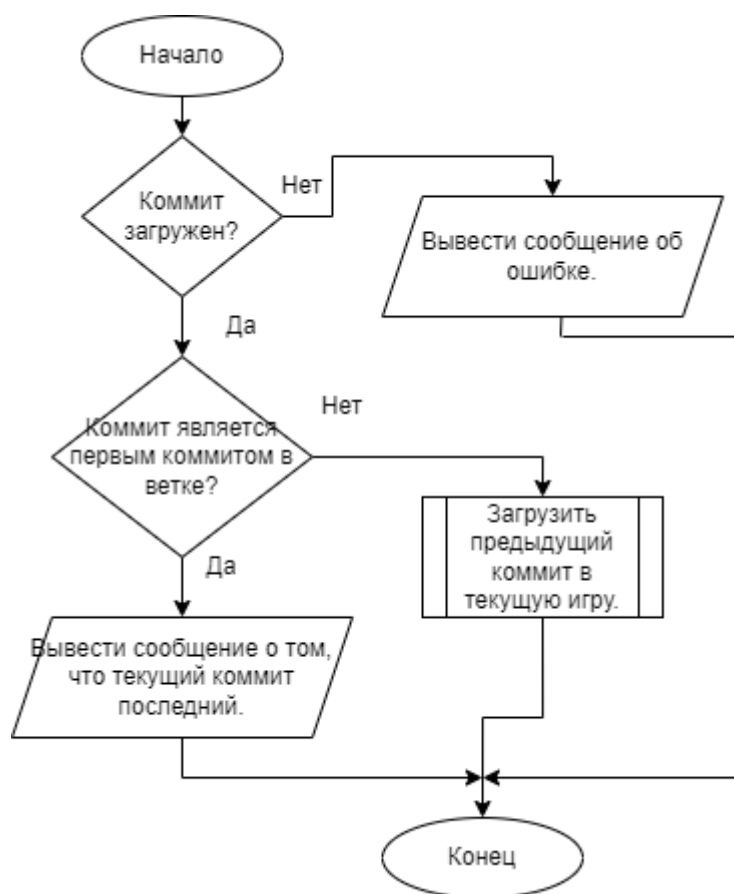


Рисунок 12 – Алгоритм перехода к предыдущему коммиту.

## 4 РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ИГРЫ “МОРСКОЙ БОЙ”

### 4.1 Особенности программной реализации.

При запуске программы, все сохраненные ветки и их коммиты загружаются в локальные объекты соответствующих классов. Т.е. для каждого файла создается объект класса Branch, а для каждой строчки внутри файла создается объект класса Commit. Стоит также отметить, что ветка master создается по умолчанию при запуске программы (в случае ее отсутствия). Веткой по умолчанию при запуске программы всегда является master.

```
self.branches = []
self.branches.append(Branch("master"))
for file in os.listdir(str(pathlib.Path().resolve())+"/git/"):
    if file != "master.pickle":
        self.branches.append(Branch(file.replace(".pickle", "")))
self.curr_branch = self.branches[0]
```

Загрузка коммита происходит путем извлечения данных из сохраненного объекта и загрузкой их в текущую сессию. При это обновляются состояния полей обоих текущих игроков и последовательность сделанных ходов.

```
com_game = git.curr_branch.load_commit()
human = Player(who="human", mas=com_game[0])
comp = Player(mas=com_game[1])
game_log = com_game[2]["log"]
```

### 4.2 Пример работы программы.

Программа выполнена с использованием консольного интерфейса. После запуска программы перед пользователем предстает окно, изображенное

на Рисунке 13. Стоит обратить внимание на сообщение, которое сигнализирует о том, что была создана ветка master.



Рисунок 13 – Начальное окно программы.

Мы видим два пустых поля: поле игрока и его противника. Пользователю предлагается расставить корабли вручную или автоматически.

Выбрав автоматическую расстановку кораблей, они будут расставлены случайным образом.

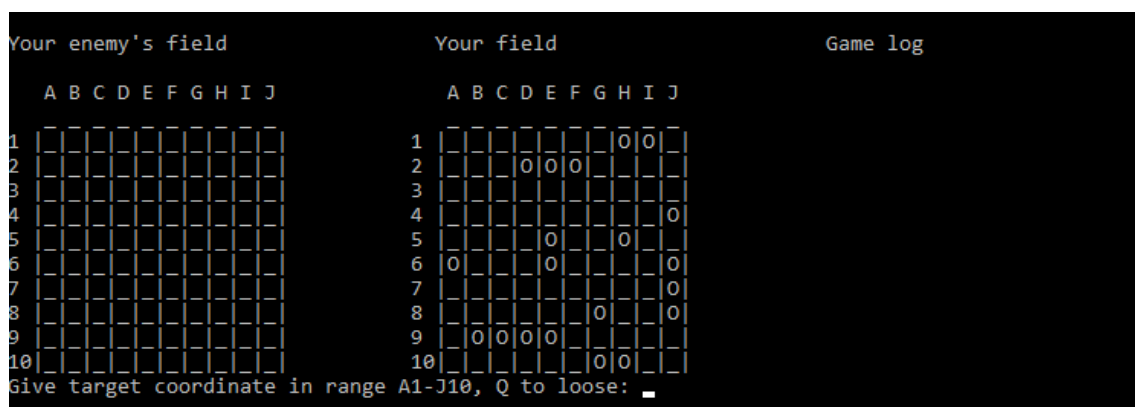


Рисунок 14 – Автоматическая расстановка кораблей.

Если пользователь решит сделать расстановку вручную, то программа предложит сначала выбрать стартовую клетку корабля, а затем, в зависимости от выбранной клетки и длины корабля, все возможные варианты расстановки данного корабля. Такой выбор повториться для каждого корабля в игре.

Чтобы сделать ход, пользователь должен ввести координату клетки, в которую он хочет сделать выстрел. Выстрелом можно или промазать, или попасть. В любом случае ход будет добавлен в общий пул сделанных ходов. Компьютер сделает свой ход сразу после пользователя.

```

enter M if you would like to place your ships manually or any other for automatic placement: m
Give first coordinate of [0000] : B1
Possible ends ['E1', 'B4']
Give last coordinate of [0000] ship: E1

Your enemy's field          Your field          Game log
  A B C D E F G H I J      A B C D E F G H I J
1 | | | | | | | | | |      1 | | 0 | 0 | 0 | | | | | |
2 | | | | | | | | | |      2 | | | | | | | | | |
3 | | | | | | | | | |      3 | | | | | | | | | |
4 | | | | | | | | | |      4 | | | | | | | | | |
5 | | | | | | | | | |      5 | | | | | | | | | |
6 | | | | | | | | | |      6 | | | | | | | | | |
7 | | | | | | | | | |      7 | | | | | | | | | |
8 | | | | | | | | | |      8 | | | | | | | | | |
9 | | | | | | | | | |      9 | | | | | | | | | |
10| | | | | | | | | |     10| | | | | | | | | |
Give first coordinate of [000] : 

```

Рисунок 15 – Ручная расстановка кораблей.

```

Your enemy's field          Your field          Game log
  A B C D E F G H I J      A B C D E F G H I J      4: comp  D9
1 | | | | | | | | | |      1 | | | | | | | 0 | 0 | | |      3: you   G4
2 | | | | | | | | | |      2 | | | | | | | | | |      2: comp  E2
3 | | | | | | | | | |      3 | | | 0 | 0 | 0 | | | | |      1: you   G3
4 | | | | | | | | | |      4 | 0 | | | | | | | 0 | 0 | | |
5 | | | | | | | | | |      5 | | | | | | | | | | | 0 |
6 | | | | | | | | | |      6 | 0 | | 0 | | | | | | 0 | 0 |
7 | | | | | | | | | |      7 | | | | | | | | | | | 0 |
8 | | | | | | | | | |      8 | | | | | | | | | | | 0 |
9 | | | | | | | | | |      9 | 0 | 0 | 0 | | 0 | | | | 0 |
10| | | | | | | | | |     10| | | | | | | | | | |
Give target coordinate in range A1-J10, Q to loose: 

```

Рисунок 16 – Ход пользователя.

Для взаимодействия с интерфейсом системы контроля версий, пользователь должен во время игра ввести команду git. После этого откроется список команд для взаимодействия с системой контроля версий.

```

Give target coordinate in range A1-J10, Q to loose: git
-----
| 1. Show all branches.
| 2. Show all commits on current branch.
| 3. Change current branch.
| 4. Create new branch.
| 5. Remove branch.
| 6. Add new commit.
| 7. Remove commit.
| 8. Load commit.
| 9. Continue playing.
| 0. Exit.
-----

```

Рисунок 17 – Список команд системы контроля версий.

Команда 1 показывает все существующие ветки и указывает на текущую.

```

-----
1. master (current)
2. pavel
-----

-----
|1. Show all branches.
|2. Show all commits on current branch.
|3. Change current branch.
|4. Create new branch.
|5. Remove branch.
|6. Add new commit.
|7. Remove commit.
|8. Load commit.
|9. Continue playing.
|0. Exit.
-----

Choose command:

```

Рисунок 18 – Все ветки.

Команда 2 показывает все коммиты на текущей ветке. Каждый коммит имеет порядковый номер, комментарий пользователя и дату добавления.

```

master
1      | first commit on master      | 2021-12-09 17:48:42.268720
-----

-----
|1. Show all branches.
|2. Show all commits on current branch.
|3. Change current branch.
|4. Create new branch.
|5. Remove branch.
|6. Add new commit.
|7. Remove commit.
|8. Load commit.
|9. Continue playing.
|0. Exit.
-----

Choose command:

```

Рисунок 19 – Все коммиты на ветке.

Команда 3 позволяет сменить текущую ветку. После смены ветки будут отображаться только те коммиты, которые были сохранены на ней.

<pre> 1. master (current) 2. pavel ----- Choose the branch: 2_ </pre>	<pre> ----- 1. master 2. pavel (current) ----- </pre>
---	---

Рисунок 20 – Смена текущей ветки.

Команда 4 позволяет создать новую ветку. Вместе с веткой создается и новый файл для хранения коммитов.

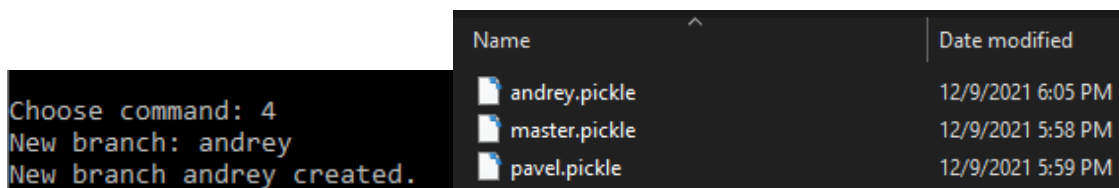


Рисунок 21 – Создание новой ветки.

Команда 5 позволяет удалить одну из существующих веток. Вместе с веткой удаляется и файл, в котором хранятся ее коммиты.

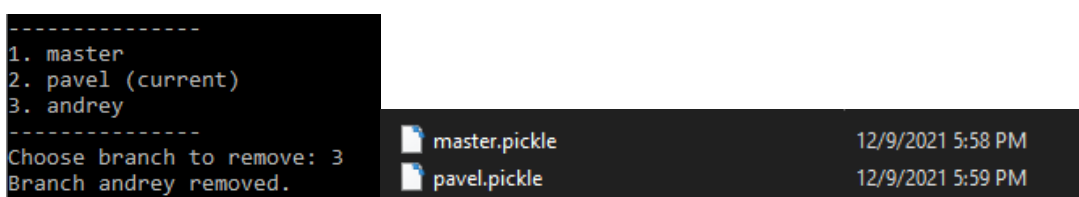


Рисунок 22 – Удаление ветки.

Команда 6 позволяет создать новый коммит – сохранить текущую версию игры.

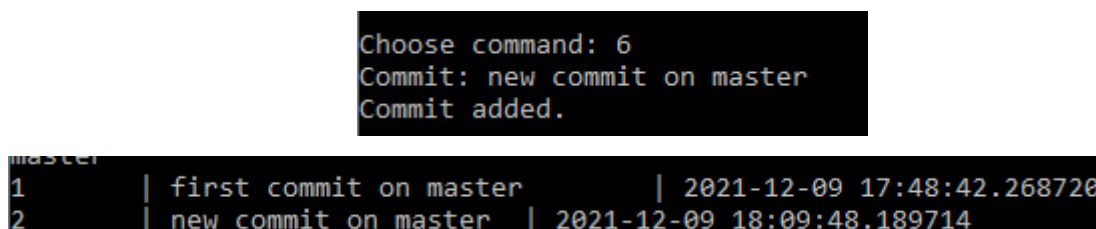


Рисунок 23 – Создание нового коммита.

Команда 7 позволяет удалить один из существующих коммитов. Для удаления будут доступны только те коммиты, которые находятся на текущей ветке.

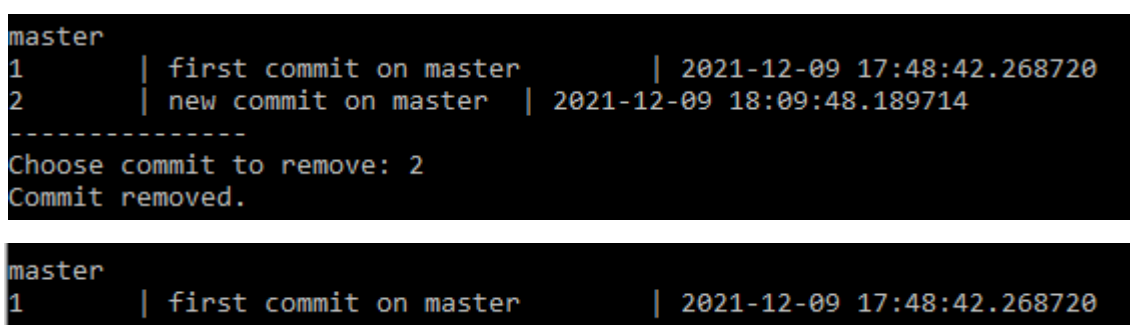


Рисунок 24 – Удаление коммита.

Команда 8 позволяет загрузить один из сохраненных ранее комитов. Для загрузки будут предложены только те коммиты, которые находятся на текущей ветке.

```

master
1      | first commit on master      | 2021-12-09 17:48:42.268720
-----
Choose commit to load: 1

Your enemy's field      Your field      Game log
  A B C D E F G H I J      A B C D E F G H I J      79: comp  J2
1 |•|•|•|•|_|_|_|_|_|_|      1 |•|_|_|_|_|_|•|0|X|•|_|      78: you   D10
2 |•|X|•|•|_|_|_|_|_|_|      2 |_|•|•|_|_|•|•|_|_|•|•|      77: comp  C8
3 |•|X|•|•|_|_|•|_|_|_|_|      3 |_|_|0|0|0|•|_|_|_|_|•|      76: you   D9
4 |•|X|•|•|_|_|•|_|_|_|_|      4 |0|_|•|_|_|_|_|0|X|_|_|•|      75: comp  D5
5 |•|•|•|•|_|_|_|_|_|_|      5 |_|_|•|•|_|_|_|_|•|•|X|      74: you   D8
6 |•|•|•|•|_|_|_|_|_|_|      6 |0|_|X|_|_|•|_|_|0|_|0|      73: comp  B10
7 |X|•|X|•|_|_|_|_|_|_|      7 |_|•|•|_|_|_|_|0|_|_|      72: comp  H1
8 |•|•|X|•|_|_|_|_|_|_|      8 |•|•|_|_|_|_|0|•|_|_|      71: you   D7
9 |X|•|X|•|_|_|_|_|_|_|      9 |X|0|0|•|0|_|_|0|_|_|      70: comp  C5
10|X|•|X|•|_|_|_|_|_|_|      10|•|•|_|_|_|_|_|_|•|•|      69: you   D6
Give target coordinate in range A1-J10, Q to loose: _      68: comp  I5

```

Рисунок 25 – Загрузка коммита.

Команда 9 закрывает меню системы контроля версий, после чего продолжается либо текущая, либо загруженная с помощью коммита игра.

Команда 0 завершает работу программы.

```

Give target coordinate in range A1-J10, Q to loose: git
-----
| 1. Show all branches.
| 2. Show all commits on current branch.
| 3. Change current branch.
| 4. Create new branch.
| 5. Remove branch.
| 6. Add new commit.
| 7. Remove commit.
| 8. Load commit.
| 9. Continue playing.
| 0. Exit.
-----
Choose command: 0
C:\Users\Павел\Desktop\Curse>_

```

Рисунок 26 – Завершение работы программы.

В процессе самой игры, помимо ходов, поддерживаются 3 команды.

При нажатии “s” в процессе игры произойдет быстрое сохранение текущей партии.



```

Give target coordinate in range A1-J10, Q to lose: s
Commit added.
Saved!

Your enemy's field      Your field      Game log
  A B C D E F G H I J    A B C D E F G H I J
1 | | | | | | | | | |    1 | | 0 | 0 | 0 | | | | | |
2 | | X | | | | | | |    2 | | 0 | | | | | | | 0 | 0 |
3 | | | | | | | | | |    3 | | 0 | | | | | | | | | |
4 | | | | | | | | | |    4 | | | | | 0 | 0 | 0 | | | |
5 | | | | | | | | | |    5 | | | | | | | | | | 0 |
6 | | | | | | | | | |    6 | | 0 | 0 | X | 0 | | | | |
7 | | | | | | | | | |    7 | | | | | | | | | | 0 |
8 | | | | | | | | | |    8 | | | | | | | | | | | |
9 | | | | | | | | | |    9 | | | | | 0 | | | | | | |
10| | | | | | | | | |    10| 0 | | | 0 | 0 | | | | |
Give target coordinate in range A1-J10, Q to lose:

```

8: comp G1  
7: you B5  
6: comp D4  
5: comp D6  
4: you B4  
3: comp A7  
2: you B3  
1: you B2 hit. Ship status is hurted.

Рисунок 27 – Быстрое сохранение.

При нажатии “f” происходит быстрая загрузка следующего коммита текущей ветки. Если такого коммита нет, то будет выведено сообщение об ошибке.

```

Your enemy's field      Your field      Game log
  A B C D E F G H I J    A B C D E F G H I J
1 | | | | | | | | | |    1 | | 0 | 0 | 0 | | | | | |
2 | | X | | | | | | |    2 | | 0 | | | | | | | 0 | 0 |
3 | | | | | | | | | |    3 | | 0 | | | | | | | | | |
4 | | | | | | | | | |    4 | | | | | X | 0 | 0 | | | |
5 | | | | | | | | | |    5 | | | | | | | | | | 0 |
6 | | | | | | | | | |    6 | | 0 | 0 | X | 0 | | | | |
7 | | | | | | | | | |    7 | | | | | | | | | | 0 |
8 | | | | | | | | | |    8 | | | | | | | | | | | |
9 | | | | | | | | | |    9 | | | | | 0 | | | | | | |
10| | | | | | | | | |    10| 0 | | | 0 | 0 | | | | |
Give target coordinate in range A1-J10, Q to lose: f

Last commit on Branch reached!

Your enemy's field      Your field      Game log
  A B C D E F G H I J    A B C D E F G H I J
1 | | | | | | | | | |    1 | | 0 | | 0 | 0 | | | | | |
2 | | X | | | | | X |    2 | | 0 | | | | | | | 0 | 0 |
3 | | | | | | | | | |    3 | | 0 | | | | | | | | | |
4 | | | | | | | | | |    4 | | | | | X | X | 0 | | | |
5 | | | | | | | | | |    5 | | | | | | | | | | 0 |
6 | | | | | | | | | |    6 | | 0 | 0 | X | 0 | | | | |
7 | | | | | | | | | |    7 | | | | | | | | | | 0 |
8 | | | | | | | | | |    8 | | | | | | | | | | | |
9 | | | | | | | | | |    9 | | | | | 0 | | | | | | |
10| | | | | | | | | |    10| 0 | | | 0 | 0 | | | | |
Give target coordinate in range A1-J10, Q to lose:

```

21: comp B7  
20: you D6  
19: comp D5  
18: you D5  
17: comp E8  
16: you D4  
15: comp I9  
14: comp E4  
13: you D3  
12: comp A6  
11: you D2  
10: comp C3

32: comp J6  
31: you H6  
30: comp I5  
29: comp F4  
28: you H5  
27: you H4 hit. Ship status is hurted.  
26: comp C1  
25: you H3  
24: you H2 hit. Ship status is dead.  
23: comp A8  
22: you H1  
21: comp B7

Рисунок 28 – Переход к следующему коммиту.

При нажатии “b” происходит быстрая загрузка предыдущего коммита текущей ветки. Если такого коммита нет, то будет выведено сообщение об ошибке.

```

Your enemy's field          Your field          Game log
  A B C D E F G H I J      A B C D E F G H I J
1 | | | | | | | | | |      1 | | 0 | | 0 | 0 | | | | |
2 | | X | | | | | | |      2 | | 0 | | | | | | | 0 | 0 |
3 | | | | | | | | | |      3 | | 0 | | | | | | | |
4 | | | | | | | | | |      4 | | | | | X | X | 0 | |
5 | | | | | | | | | |      5 | | | | | | | | | | 0 |
6 | | | | | | | | | |      6 | | 0 | 0 | X | 0 | | | |
7 | | | | | | | | | |      7 | | | | | | | | | | 0 |
8 | | | | | | | | | |      8 | | | | | | | | | | |
9 | | | | | | | | | |      9 | | | | | 0 | | | | |
10| | | | | | | | | |      10| 0 | | | 0 | 0 | | | |
Give target coordinate in range A1-J10, Q to lose: b

Your enemy's field          Your field          Game log
  A B C D E F G H I J      A B C D E F G H I J
1 | | | | | | | | | |      1 | | 0 | | 0 | 0 | | | | |
2 | | X | | | | | | |      2 | | 0 | | | | | | | 0 | 0 |
3 | | | | | | | | | |      3 | | 0 | | | | | | | |
4 | | | | | | | | | |      4 | | | | | X | 0 | 0 | |
5 | | | | | | | | | |      5 | | | | | | | | | | 0 |
6 | | | | | | | | | |      6 | | 0 | 0 | X | 0 | | | |
7 | | | | | | | | | |      7 | | | | | | | | | | 0 |
8 | | | | | | | | | |      8 | | | | | | | | | | |
9 | | | | | | | | | |      9 | | | | | 0 | | | | |
10| | | | | | | | | |      10| 0 | | | 0 | 0 | | | |
Give target coordinate in range A1-J10, Q to lose:

```

32: comp J6  
31: you H6  
30: comp I5  
29: comp F4  
28: you H5  
27: you H4 hit. Ship status is hurted.  
26: comp C1  
25: you H3  
24: you H2 hit. Ship status is dead.  
23: comp A8  
22: you H1  
21: comp B7

21: comp B7  
20: you D6  
19: comp D5  
18: you D5  
17: comp E8  
16: you D4  
15: comp I9  
14: comp E4  
13: you D3  
12: comp A6  
11: you D2  
10: comp C3

Рисунок 29 – Переход к предыдущему коммиту.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была проанализирована предметная область, спроектированы и реализованы игра “Морской бой” и система контроля версий для нее. Таким образом, мы достигли поставленной цели.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Git: Git book [Электронный ресурс]. – Режим доступа: <https://git-scm.com/book/ru/v2/Введение-О-системе-контроля-версий> (дата обращения 05.11.21).
2. Atlassian: Узнать о Git [Электронный ресурс]. – Режим доступа: <https://www.atlassian.com/ru/git/tutorials/what-is-version-control> (дата обращения 10.11.2021)
3. Learnapidoc: Система контроля версий [Электронный ресурс]. – Режим доступа: <https://starkovden.github.io/Version-control-system.html> (дата обращения 15.11.2021)

**Приложение 1****Листинг кода системы контроля версий для игры “Морской бой”****Файл git.py**

```
#import game_war
import os
import datetime
import time
import pathlib
import pickle
import random
#import re

def ch(l):
    r = []
    for xx in l:
        x = xx.split(":")[1]
        if "you" in x:
            r.append(x.replace(" you ", "").replace(" comp ", "")[0] +
x.replace(" you ", "").replace(" comp ", "")[1])
    return r

class Commit:

    def __init__(self, comment, game, l_c):
        self.player = game[0]
        self.ai = game[1]
        self.logs = game[2]
        self.comment = comment
        self.date = datetime.datetime.now()
```

```
self.l_c = l_c
```

```
def show_comment(self):
```

```
    return self.comment
```

```
def show_date(self):
```

```
    return self.date
```

```
def get_game(self):
```

```
    return [self.player, self.ai, self.logs]
```

```
class Branch:
```

```
    def __init__(self, name, c=-100):
```

```
        self.name = name
```

```
        self.commits = []
```

```
        self.curr_commit = c
```

```
        f = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle",
'ab+')

```

```
        f.close()
```

```
        if
```

```
os.path.getsize(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle") > 0:
```

```
        fl = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle",
'rb')

```

```
        self.commits = pickle.load(fl)
```

```
        fl.close()
```

```
        print("New branch { } created.".format(name))
```

```
    def load_next(self):
```

```

if self.curr_commit == -100:
    print()
    print("No commit loaded!")
    return "NONE"
else:
    if self.curr_commit+1 == len(self.commits): # было + 1
        print("No more commits.")
        return "NONE"
    else:
        self.curr_commit += 1
        #print(self.curr_commit)
        if self.curr_commit == len(self.commits) - 1:
            print()
            print("Last commit on Branch reached!")
            return self.load_commit(op=self.curr_commit)

def load_past(self):
    if self.curr_commit == -100:
        print()
        print("No commit loaded!")
        return "NONE"
    else:
        if self.curr_commit-1 < 0:
            print("No more commits.")
            return "NONE"
        else:
            self.curr_commit -= 1
            #print(self.curr_commit)
            if self.curr_commit == 0:
                print()

```

```

        print("First commit on Branch reached!")
        return self.load_commit(op=self.curr_commit)

def add_commit(self, game, op=-1, git=-1):
    if op == -1:
        commit = input("Commit: ")
    else:
        commit = "Commit " + str(random.randint(1, 1000))

    if len(self.commits) == 0:
        self.commits.append(Commit(comment=commit,      game=game,
l_c="first"))
        self.curr_commit = 0
    elif self.curr_commit + 1 == len(self.commits):
        xx = self.commits[-1]
        ll = []
        ll = self.commits[0].get_game()
        for x in range(1, len(self.commits)):
            print("lol")
            ll[0] = self.commits[x].player
            ll[1] = self.commits[x].ai
            ll[2] = ll[2] + self.commits[x].logs
        xxx = []
        xxx.append(game[0])
        xxx.append(game[1])
        xxx.append(game[2][len(ll[2]):]) #["log"]
        self.commits.append(Commit(comment=commit,      game=xxx,
l_c=xxx))
        self.curr_commit += 1
    else:

```



```

if self.curr_commit == -100:
    name = self.name + "_" + str(random.randint(1000, 9999))
    #new = Branch(name[:18])
    git.branches.append(Branch(name[:20]))
    #git.branches.append(Branch(name))
    #git.branches[-1].commits = self.commits[:self.curr_commit+1]
    git.branches[-1].commits.append(Commit(comment=commit,
game=game, l_c="first"))
    print("Done!")
    git.curr_branch = git.branches[-1]
    git.curr_branch.curr_commit = 0
    f = open(str(pathlib.Path().resolve())+"/git/"+name+".pickle",
'wb')

    pickle.dump(git.curr_branch.commits, f)
    f.close()
    fl = open(str(pathlib.Path().resolve())+"/git/"+name+".pickle",
'rb')

    git.curr_branch.commits = pickle.load(fl)
    fl.close()
else:
    ll = []
    ll = self.commits[0].get_game()
    for x in range(1, self.curr_commit+1):
        ll[0] = self.commits[x].player
        ll[1] = self.commits[x].ai
        ll[2] = ll[2] + self.commits[x].logs
    xxx = []
    xxx.append(game[0])
    xxx.append(game[1])
    xxx.append(game[2][len(ll[2]):])

```

```

        if xxx[1] != self.commits[self.curr_commit+1].ai and xxx[0] !=
self.commits[self.curr_commit+1].player:
            name = self.name + "_" + str(random.randint(1000, 9999))
            #new = Branch(name[:18])
            git.branches.append(Branch(name[:20]))
            #git.branches.append(Branch(name))
            git.branches[-1].commits = self.commits[:self.curr_commit+1]
            git.branches[-1].commits.append(Commit(comment=commit,
game=xxx, l_c=git.branches[-1].commits[-1]))
            print("Done!")
            git.curr_branch = git.branches[-1]
            git.curr_branch.curr_commit = self.curr_commit + 1
            #git.branches.append(new)

self.curr_commit = len(self.commits) - 1
f = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'wb')
pickle.dump(self.commits, f)
f.close()
fl = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'rb')
self.commits = pickle.load(fl)
fl.close()
print("Commit added.")

def show_branch(self):
    os.system("cls")
    print("-" * 15)
    print(self.name)
    for i in range(0, len(self.commits)):

```

```

        e = self.commits[i].l_c
        print("{0} \t| {1} \t| {2} \t| {3} \t| {4}".format(i+1,
self.commits[i].show_comment(),
self.commits[i].show_date(),
ch(self.commits[i].logs), e if e == "first" else e.comment))
        print("-" * 15)

```

```

def remove_commit(self):
    self.show_branch()
    idx = int(input("Choose commit to remove: "))
    #self.commits.pop(idx-1)
    self.commits = self.commits[:idx-1]
    f = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'wb')
    pickle.dump(self.commits, f)
    f.close()
    fl = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'rb')
    self.commits = pickle.load(fl)
    fl.close()
    print("Commit removed.")

```

```

def get_name(self):
    return self.name

```

```

def load_commit(self, op=-1):
    if op == -1:
        self.show_branch()
        idx = int(input("Choose commit to load: "))
    else:
        idx = op + 1 # по счету
    self.curr_commit = idx - 1
    #return self.commits[idx-1].get_game()

```

```

ll = []
ll = self.commits[0].get_game()
for x in range(1, idx):
    ll[0] = self.commits[x].player
    ll[1] = self.commits[x].ai
    ll[2] = ll[2] + self.commits[x].logs
#self.commits = self.commits[:idx]
f = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'wb')
pickle.dump(self.commits, f)
f.close()
fl = open(str(pathlib.Path().resolve())+"/git/"+self.name+".pickle", 'rb')
self.commits = pickle.load(fl)
fl.close()
return ll

```

```
class Git:
```

```

    def __init__(self):
        self.branches = []
        self.branches.append(Branch("master"))
        for file in os.listdir(str(pathlib.Path().resolve())+"/git/"):
            if file != "master.pickle":
                self.branches.append(Branch(file.replace(".pickle", "")))
        self.curr_branch = self.branches[0]

    def create_branch(self):
        name = input("New branch: ")
        self.branches.append(Branch(name))

```

```

def show_branches(self):
    os.system("cls")
    print("-" * 15)
    for i in range(0, len(self.branches)):
        if self.branches[i] == self.curr_branch:
            print("{0}. {1} (current)".format(i+1,
self.branches[i].get_name()))
        else:
            print("{0}. {1}".format(i+1, self.branches[i].get_name()))
    print("-" * 15)

def remove_branch(self):
    self.show_branches()
    idx = int(input("Choose branch to remove: "))
    r_name = self.branches[idx-1].get_name()
    if self.branches[idx-1] == self.curr_branch:
        print("You can't remove current branch! Change the branch and then
remove.")
    else:
        os.remove(str(pathlib.Path().resolve())+"/git/"+self.branches[idx-
1].name + ".pickle")
        self.branches.pop(idx-1)
        print("Branch {0} removed.".format(r_name))

def move_branch(self):
    self.show_branches()
    i = int(input("Choose the branch: "))
    self.curr_branch = self.branches[i-1]
    self.curr_branch.curr_commit = 0

```

```

def get_curr(self):
    return self.curr_branch

def git_menu(self):
    print("\t\t\t" + "-" * 15)
    print("\t\t\t1. Show all branches.")
    print("\t\t\t2. Show all commits on current branch.")
    print("\t\t\t3. Change current branch.")
    print("\t\t\t4. Create new branch.")
    print("\t\t\t5. Remove branch.")
    print("\t\t\t6. Add new commit.")
    print("\t\t\t7. Remove commit.")
    print("\t\t\t8. Load commit.")
    print("\t\t\t9. Continue playing.")
    print("\t\t\t0. Exit.")
    print("\t\t\t" + "-" * 15)

def git_command(self, i):
    return {1: self.show_branches,
            2: self.curr_branch.show_branch,
            3: self.move_branch,
            4: self.create_branch,
            5: self.remove_branch,
            6: self.curr_branch.add_commit,
            7: self.curr_branch.remove_commit,
            8: self.curr_branch.load_commit,
            10: self.curr_branch.load_next,
            11: self.curr_branch.load_past,
            #9: exit,
            0: exit}.get(i, "Error!")

```

```
def cycle(self):
    while True:
        #os.system("cls")
        self.git_menu()
        i = int(input("Choose command: "))
        if i == 6:
            game = "game"
            players = "pl"
            self.git_command(i)(game, players)
        else:
            self.git_command(i)()
```

## Файл gg.py

[illegible]



```

        self.who = who
        self.alive = True
        self.possible_targets = ALL_TARGETS[:]
        self.suggested_targets = []          # list of coordinates if enemy's
ship was hit but not sank

    def ship_placement(self, how = "randomly"):
        for ship in SHIPS:
            ship_length = len(ship)

            while True:
                if how == "randomly":
                    end1 = random.choice(ALL_TARGETS)
                else:
                    end1 = input("Give first coordinate of [" + ship + "] : ").upper()
                if end1 in ALL_TARGETS and
self.possible_ends(end1,ship_length):
                    if ship_length==1:
                        end2 = end1          # for [0] ships
                        break
                    elif how == "manualy":
                        print("Not possible to put this ship beginning from point ",
end1)
                    while True and ship_length>1:
                        if how == "randomly":
                            end2 = random.choice(self.possible_ends(end1,ship_length))
                        else:
                            print("Possible ends ", self.possible_ends(end1,ship_length))
                            end2 = input("Give last coordinate of [" + ship + "] ship:
").upper()

```

```

        if (end2 in ALL_TARGETS) and (end2 in
self.possible_ends(end1,ship_length)):
            break

            # actual ship placement after all checks are
done

    edge1 = [int(end1[1:])+1, ABCJ.find(end1[0])*2 + 2]
    edge2 = [int(end2[1:])+1, ABCJ.find(end2[0])*2 + 2]
    if edge1[1]==edge2[1]:
        for x in range(0,abs(edge1[0]-edge2[0])+1):
            self.field[min(edge1[0],edge2[0])+x][edge1[1]] = "O"
    if edge1[0]==edge2[0]:
        for x in range(0,abs(edge1[1]-edge2[1])+2,2):
            self.field[edge1[0]][min(edge1[1],edge2[1])+x] = "O"

    self.forbid(end1,"O","z","dead")    # mark forbidden neibours cells
with 'z' to avoid next ship placement there

    if how == "manualy":
        display_fields()

def possible_ends(self, start, ship_len):
    """checks whether inputed start point of ship (i.e.[000]) has possible
ends on field, returns list of such ends (0-1-2-3 or 4 values like A5)"""
    result = []
    col_number = ABCJ.find(start[0])*2 + 2
    row_number = int(start[1:])+1

    if self.field[row_number][col_number]!="_":
        return []
    elif ship_len == 1 and self.field[row_number][col_number]=="_":
        return start

```

```

for arrow in [[0,2],[1,0],[0,-2],[-1,0]]:    #'right','down','left','up'
    col_num = col_number
    row_num = row_number
    for x in range(1,ship_len):
        col_num += arrow[1]
        row_num += arrow[0]
        if row_num<2 or row_num>11 or col_num<2 or col_num>20 or
self.field[row_num][col_num]!="_":
            break
        elif x==ship_len-1:
            result.append(ABCJ[int((col_num-2)/2)]+str(row_num-1))

return result

```

```

def forbid(self, shot, target = "X", marker = "z", ship_is = "dead"):
    """gives list of forbidden coordinates near dead or hurted ship, while
mark this cells with 'z' sign """
    col_number = ABCJ.find(shot[0])*2 + 2
    row_number = int(shot[1:])+1
    result = []
    my_queue = []
    history = []
    my_queue.append((row_number,col_number))
    history.append((row_number,col_number))

    while my_queue:
        for arrow in [[0,2],[1,0],[0,-2],[-1,0]]:    # 'right','down','left','up'
            row_num = my_queue[0][0] + arrow[0]
            col_num = my_queue[0][1] + arrow[1]

```

```

        if row_num<2 or row_num>11 or col_num<2 or col_num>20:
            continue

        if self.field[row_num][col_num] == target and
tuple((row_num,col_num)) not in history:

            history.append((row_num,col_num))
            my_queue.append((row_num,col_num))

        if ship_is == "dead" and self.field[row_num][col_num] == "_":
            self.field[row_num][col_num] = marker
            result.append(ABCJ[int((col_num-2)/2)]+str(row_num-1)) #
CHECK!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        for arrow in [[-1,2],[1,2],[1,-2],[-1,-2]]: # 'right-up','right-
down','left-down','left-up'
            row_num = my_queue[0][0] + arrow[0]
            col_num = my_queue[0][1] + arrow[1]

            if row_num<2 or row_num>11 or col_num<2 or col_num>20:
                continue
            elif self.field[row_num][col_num] == "_":
                self.field[row_num][col_num] = marker
                result.append(ABCJ[int((col_num-2)/2)]+str(row_num-1)) #
CHECK!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

            my_queue.pop(0)

    return result

```

```

def ship_status_check(self, shot):
    result = "dead."
    col_number = ABCJ.find(shot[0])*2 + 2
    row_number = int(shot[1:])+1
    my_queue = []
    history = []
    my_queue.append((row_number,col_number))
    history.append((row_number,col_number))

    while my_queue:
        for arrow in [[0,2],[1,0],[0,-2],[-1,0]]:    # 'right','down','left','up'
            row_num = my_queue[0][0] + arrow[0]
            col_num = my_queue[0][1] + arrow[1]

            if row_num<2 or row_num>11 or col_num<2 or col_num>20:
                continue
            if (self.field[row_num][col_num] == "O" or
self.field[row_num][col_num] == "X") and tuple((row_num,col_num)) not in
history:

                history.append((row_num,col_num))
                my_queue.append((row_num,col_num))
                if self.field[row_num][col_num] == "O":
                    result = "hurtet."
                my_queue.pop(0)

        return " hit. Ship status is " + result

def is_alive_check(self):

```

```

        """checks whether player still has "O" ships and is alive (do nothing) or
        dead (self.alive = False)"""

```

```

        for i in range(len(self.field)):
            if "O" in self.field[i]:
                break
            if i == len(comp.field) - 1:
                self.alive = False

```

```

def display_fields(hide_z = True):

```

```

    print()
    print("Your enemy's field", " " * 15, "Your field", " "*20, "Game log")
    print()

```

```

    for i in range(len(human.field)):

```

```

        str2print1 = ""

```

```

        str2print2 = ""

```

```

        for j in range(len(human.field[0])):

```

```

            str2print1 += human.field[i][j].replace("z","_") if hide_z else
            human.field[i][j]

```

```

            str2print2 += comp.field[i][j].replace("O","_").replace("z","_") if
            hide_z else comp.field[i][j].replace("O","_")

```

```

            if len(game_log)>i:

```

```

                str2print1 += " "*7 + game_log[-i-1]

```

```

            print(str2print2 + " " * 10 + str2print1)

```

```

if __name__ == '__main__':

```

```

    print("Welcome to SeaBattle game!")

```

```

human = Player(who="human")
comp = Player()

git = Git()

game_log = []
display_fields()

if input("enter M if you would like to place your ships manually or any
other for automatic placement: ").upper() == "M":
    human.ship_placement("manualy")
else:
    human.ship_placement("randomly")
    comp.ship_placement("randomly")
#-----Game-----
---#

while True:
    while human.alive and comp.alive:          # human player turn

        display_fields()
        while True:                            # waiting for valid target
            shot = input("Give target coordinate in range A1-J10, Q to lose:
").upper()

            if shot == "Q":
                human.alive = False
                break

            if shot in ALL_TARGETS:
                break

            if shot == "git" or shot == "GIT" or shot == "F" or shot == "B" or
shot == "S":
                break

```

```

if shot != "Q" and shot != "GIT" and shot != "F" and shot != "B" and
shot != "S":

```

```

    game_log += [str(len(game_log)+1)+ ": you  " + shot]

```

```

elif shot == "git" or shot == "GIT":

```

```

    while True:

```

```

        git.git_menu()

```

```

        i = int(input("Choose command: "))

```

```

        if i == 6:

```

```

            list_player = {"field": human.field,
                           "who": human.who,
                           "alive": human.alive,
                           "p_t": human.possible_targets,
                           "s_t": human.suggested_targets}

```

```

            list_ai = {"field": comp.field,
                       "who": comp.who,
                       "alive": comp.alive,
                       "p_t": comp.possible_targets,
                       "s_t": comp.suggested_targets}

```

```

            logg = game_log

```

```

            game = [list_player, list_ai, logg]

```

```

            git.git_command(i)(game, git=git)

```

```

        elif i == 9:

```

```

            break

```

```

        elif i == 8:

```

```

            "load commit"

```

```

            com_game = git.curr_branch.load_commit()

```

```

            human = Player(who="human", mas=com_game[0])

```

```

            comp = Player(mas=com_game[1])

```

```

            game_log = com_game[2]#[ "log"]

```

```

        else:

```



```

        git.git_command(i)()
elif shot == "F" or shot == "f": # вперед по ветке
    com_game = git.git_command(10)()
    if com_game != "NONE":
        human = Player(who="human", mas=com_game[0])
        comp = Player(mas=com_game[1])
        game_log = com_game[2]
    continue

elif shot == "B": # назад по ветке
    com_game = git.git_command(11)()
    if com_game != "NONE":
        human = Player(who="human", mas=com_game[0])
        comp = Player(mas=com_game[1])
        game_log = com_game[2]
    continue

elif shot == "S":
    list_player = {"field": human.field,
                  "who": human.who,
                  "alive": human.alive,
                  "p_t": human.possible_targets,
                  "s_t": human.suggested_targets}
    list_ai = {"field": comp.field,
              "who": comp.who,
              "alive": comp.alive,
              "p_t": comp.possible_targets,
              "s_t": comp.suggested_targets}
    logg = game_log
    game = [list_player, list_ai, logg]
    git.git_command(6)(game, op=1, git=git)

```

```

        print("Saved!")
        continue
    else:
        shot = "A1"
    if shot == "git" or shot == "GIT" or shot == "B" or shot == "F" or
shot == "S":
        continue                # for correct closure
    col_num = ABCJ.find(shot[0])*2 + 2
    row_num = int(shot[1:])+1

    if comp.field[row_num][col_num]=="O":
        comp.field[row_num][col_num]="X"
        game_log[-1] += comp.ship_status_check(shot)
        comp.is_alive_check()
    elif comp.field[row_num][col_num]=="X":
        break
    else:
        comp.field[row_num][col_num]="•"
        break

while human.alive and comp.alive:                # comp player turn

    if comp.suggested_targets:
        shot = comp.suggested_targets.pop()
    else:
        shot = random.choice(comp.possible_targets)
        comp.possible_targets.remove(shot)

                                # здесь удалять из Возможных ходов
те, что возвращает Forbid (дописать ему возвращение и входной параметр-
флаг, если не нужна модификация поля)

```

```

game_log += [str(len(game_log)+1)+ ": comp " + shot]

col_num = ABCJ.find(shot[0])*2 + 2
row_num = int(shot[1:])+1

if human.field[row_num][col_num]=="O":
    human.field[row_num][col_num]="X"
    human.is_alive_check()
else:
    human.field[row_num][col_num]="•"
    break

if not (comp.alive and human.alive):
    display_fields()
    if not comp.alive: print("You win!")
    else: print("You lose")
    if input("Whould you like to play again? y/n [y] : ") not in
['n','N','no','No','NO']:
        human = Player(who="human")
        comp = Player()
        game_log = []
        display_fields()
        if input("enter M if you would like to place your ships manualy or
any other for automatic placement: ").upper() == "M":
            human.ship_placement("manualy")
        else:
            human.ship_placement("randomly")
            comp.ship_placement("randomly")
    else:
        break

```