МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Донецкий национальный технический университет»

ОТЧЁТ

по лабораторной работе №1

на тему: «Работа с ветками»

по курсу ««Профессиональная практика программной инженерии»

Выполнил: студент группы ПИ-20а

Синяев А. В.

(подпись) «\_\_\_» \_\_\_\_\_\_\_2024г.

Приняла:

Незамова Л. В.

(подпись) «\_\_\_» \_\_\_\_\_\_\_2024г.

2024 г.

Ветка lab2 – содержит модули GPU, APU, MMU, Input Mapper, ROM Loader, Sound Blaster Emulator, Save State Manager, Debugging Tools.

Ветка Lib2 – содержит модули Video Renderer, Input Recorder, Cheat Code Manager, BIOS Emulator, Game Genie Emulator, Network Play, Frame Limiter, Language Localization.

Ветка Laba2 – содержит модули Configuration Manager, GUI Controller, Auto-Update Manager, Save File Converter, Netplay Lobby, Rewind Feature, Game Database, Screen Capture.

Ветка lab2 содержит файлы:

GPU.cpp – в котором находяться модуль GPU:

a. Модуль, эмулирующий работу центрального процессора японской консоли Sega Genesis.

b. Компонент, эмулирующий работу графического процессора Sega Genesis, отвечающий за вывод изображения на экран.

APU.cpp – в котором находяться модуль APU: Модуль, эмулирующий аудио процессор Sega Genesis, отвечающий за генерацию звука.

MMU.cpp – в котором находяться модуль Memory Management Unit (MMU): Компонент, отвечающий за управление доступом к памяти и эмуляцию работы памяти Sega Genesis.

Input Mapper.cpp – в котором находяться модуль Input Mapper: Модуль, отвечающий за эмуляцию взаимодействия с игровым контроллером Sega Genesis.

ROM.cpp – в котором находяться модуль ROM Loader: Компонент, обеспечивающий загрузку ROM-файлов игр и их эмуляцию.

Sound Blaster.cpp – в котором находяться модуль Sound Blaster Emulator: Модуль, эмулирующий звуковую карту Sound Blaster для улучшения качества звука.

Save.cpp – в котором находяться модуль Save State Manager: Компонент, отвечающий за сохранение и восстановление текущего состояния игры.

Debugging Tools.cpp – в котором находяться модуль Debugging Tools: Набор инструментов для отладки и тестирования эмулятора.

Ветка Lib2 содержит файлы:

Video Renderer.cpp – в котором находяться модуль Video Renderer: Модуль, отвечающий за отображение графики Sega Genesis на экране компьютера.

Input Recorder.cpp – в котором находяться модуль Input Recorder: Компонент, позволяющий записывать и воспроизводить ввод с контроллера для повторного тестирования игр.

Cheat Code.cpp – в котором находяться модуль Cheat Code Manager: Модуль для управления чит-кодами и их эмуляции.

BIOS.cpp – в котором находяться модуль BIOS Emulator: Компонент, эмулирующий работу BIOS Sega Genesis.

Game Genie Emulator.cpp – в котором находяться модуль Game Genie Emulator: Модуль для эмуляции расширений Game Genie и обеспечения поддержки чит-кодов.

Network Play.cpp – в котором находяться модуль Network Play: Компонент, позволяющий играть по сети с другими пользователями.

Frame Limiter.cpp – в котором находяться модуль Frame Limiter: Модуль, ограничивающий частоту обновления кадров для эмуляции реального времени.

Language Localization.cpp – в котором находяться модуль Language Localization: Компонент, позволяющий переводить интерфейс эмулятора на разные языки.

Ветка Laba2 содержит файлы:

Configuration Manager.cpp – в котором находяться модуль Configuration Manager: Модуль, позволяющий настраивать параметры эмулятора и сохранять их для последующего использования.

GUI.cpp – в котором находяться модуль GUI Controller: Компонент, отвечающий за создание графического интерфейса пользователя для удобного управления эмулятором.

Auto-Update Manager.cpp – в котором находяться модуль Auto-Update Manager: Модуль для автоматического обновления эмулятора до последней версии.

Save File Converter.cpp – в котором находяться модуль Save File Converter: Компонент, позволяющий конвертировать сохраненные файлы игр в различные форматы.

Netplay Lobby.cpp – в котором находяться модуль Netplay Lobby: Модуль для создания и управления комнатами для сетевой игры.

Rewind Feature.cpp – в котором находяться модуль Rewind Feature: Компонент, позволяющий откатывать время в игре для повторного прохождения уровней или исправления ошибок.

Game Database.cpp – в котором находяться модуль Game Database: Модуль, содержащий информацию о всех совместимых с эмулятором играх Sega Genesis.

Screen Capture.cpp – в котором находяться модуль Screen Capture: Компонент, позволяющий делать снимки экрана во время игры и сохранять их в различных форматах.

Программный код:

1. GPU.cpp (рис. 1)

struct GPUCentral{

void executelnstruction(int opcode) {

// Здесь должна быть реализация выполнения инструкции

}

};

struct GPUGraphic{

void renderFrame() {

// Эмуляция вывода изображения на экран

}

};

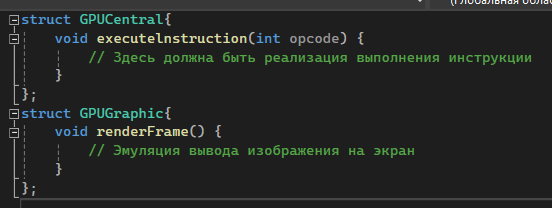


Рисунок 1 – GPU.cpp

1. APU.cpp (рис. 2)

#include <vector>

struct AudioProcessor {

std::vector<int> soundBuffer;

AudioProcessor() {

// Инициализация звукового буфера

}

void generateSound() {

// Генерация звука

}

void playSound() {

// Воспроизведение звука

}

};

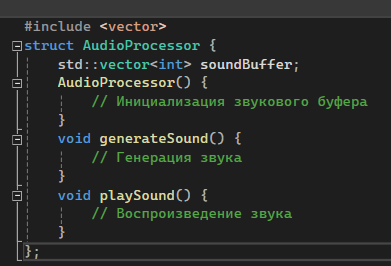


Рисунок 2 – APU.cpp

1. MMU.cpp (рис. 3)

#include <iostream>

#include <vector>

struct MemoryBlock {

std::vector<uint8\_t> data;

bool readOnly;

};

class MMU {

private:

std::vector<MemoryBlock> memory;

public:

MMU() {

// Инициализация памяти Sega Genesis

memory.resize(65536); // Общий объем памяти

// Заполнение памяти данными

// ...

// Назначение регионов памяти как доступных только для чтения

// ...

}

uint8\_t readByte(uint32\_t address) {

// Чтение байта из памяти по адресу address

}

void writeByte(uint32\_t address, uint8\_t value) {

// Запись байта value в память по адресу address

}

};

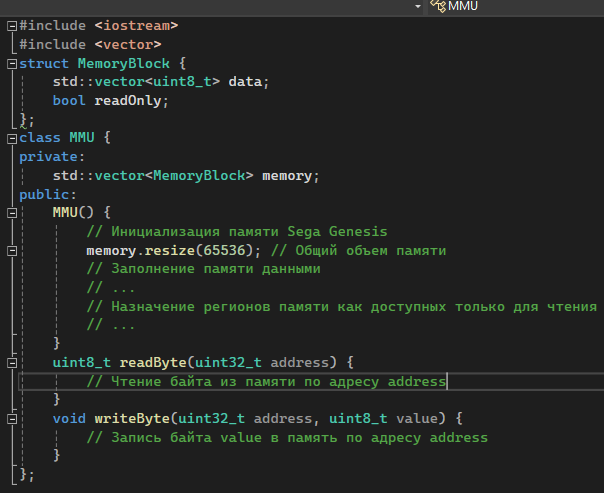


Рисунок 3 – MMU.cpp

1. Input Mapper.cpp (рис. 4)

#include <string>

#include <iostream>

struct InputMapper {

void ReadInputFromFile(std::string fileName) {

// Функция для чтения ввода из файла

}

void DisplayInfo() {

// Функция для отображения необходимой информации

}

void PlayMusic() {

// Функция для эмуляции музыкального сопровождения

}

void HandleTiming() {

// Функция для обработки тайминга титров

}

};

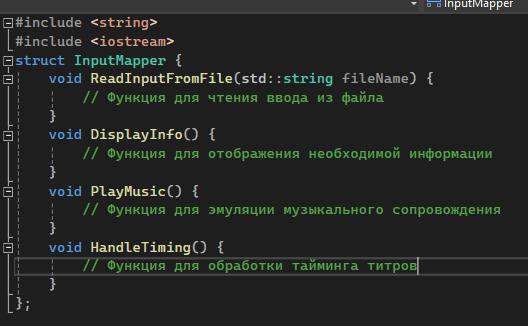


Рисунок 4 – Input Mapper.cpp

1. ROM.cpp (рис. 5)

#include <iostream>

#include <fstream>

struct RomLoader {

std::string romFilePath;

void loadRomFile() {

//Загрузка файла

}

void emulateGame() {

//Эмуляция игры

}

};

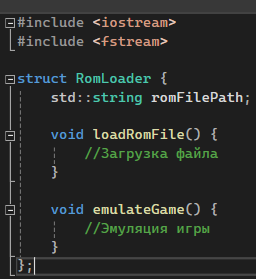


Рисунок 5 – ROM.cpp

1. Sound Blaster.cpp (рис. 6)

#include <iostream>

struct SoundBlasterEmulator {

void playSound(const std::string& sound) {

// Здесь должен быть код для воспроизведения звука real Sound Blaster

}

void processTiming(const std::string& title) {

// Здесь должен быть код обработки тайминга титров

}

};

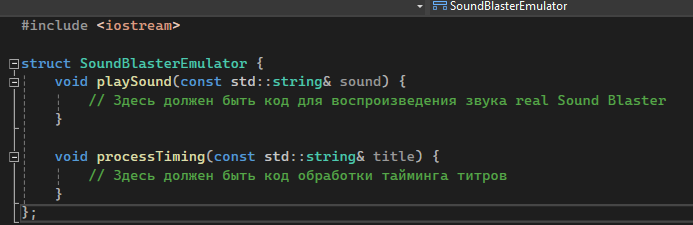


Рисунок 6 – Sound Blaster.cpp

1. Save.cpp (рис. 7)

#include <iostream>

#include <fstream>

// Структура, представляющая состояние игры

struct GameState {

int level;

int score;

bool isPaused;

// Другие переменные, определяющие состояние игры

};

// Класс, отвечающий за сохранение и восстановление состояния игры

class SaveStateManager {

public:

static void saveState(const GameState& gameState, const std::string& fileName) {

// Сохранение текущего состояния игры в файл

}

static GameState loadState(const std::string& fileName) {

// Восстановление состояния игры из файла

}

};

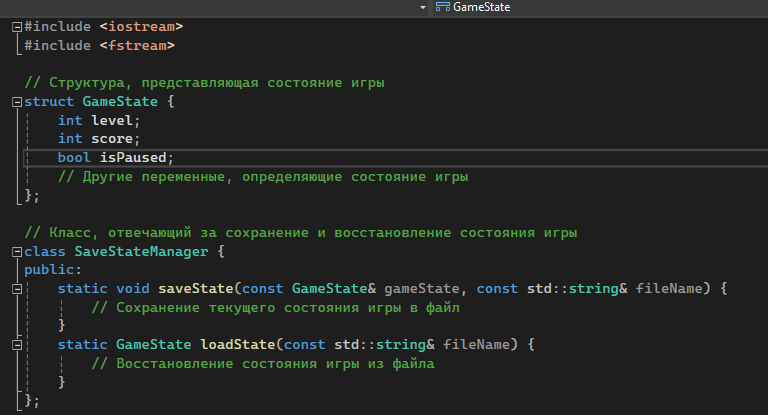


Рисунок 7 – Save.cpp

1. Debugging Tools.cpp (рис. 8)

#include <iostream>

#include <fstream>

void writeLog(const std::string& message) {

// Функция для записи отладочной информации в файл

}

struct plange {

void readFromFile(const std::string& filename) {

// чтение из файла и отображение необходимой информации

}

void playMusic() {

// музыкальное сопровождение

}

void processTiming() {

// обработка тайминга титров

}

};

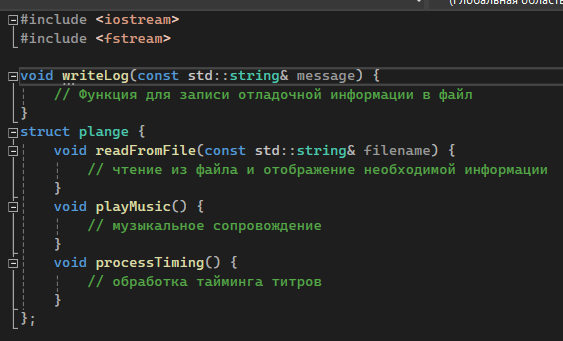


Рисунок 8 – Debugging Tools.cpp

1. Video Renderer.cpp (рис. 9)

#include <iostream>

struct VideoRenderer {

// Объявление переменных и функций для отображения графики Sega Genesis

void render() {

// Отображение графики Sega Genesis на экране компьютера

}

};

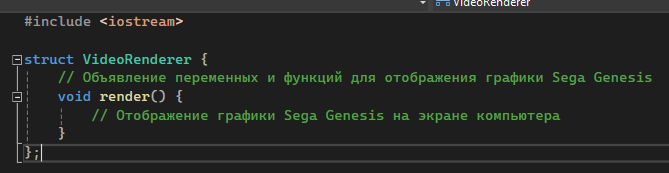


Рисунок 9 – Video Renderer.cpp

1. Input Recorder.cpp (рис. 10)

#include <iostream>

#include <vector>

class InputRecorder {

private:

std::vector<int> inputSequence;

public:

void recordInput(int input) {

//записывает вводные данные в последовательность

}

void replayInput() {

//воспроизводит записанную последовательность ввода

}

};

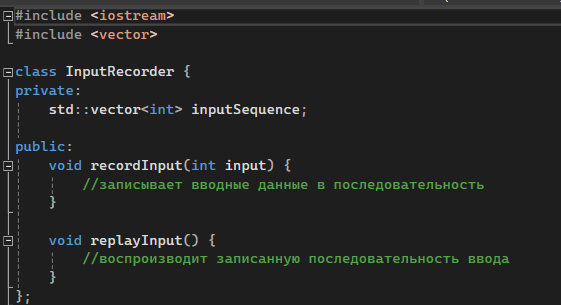


Рисунок 10 – Input Recorder.cpp

1. Cheat Code.cpp (рис. 11)

#include <iostream>

#include <string>

#include <map>

// Класс для управления чит-кодами и их эмуляции

class CheatCodeManager {

private:

std::map<std::string, bool> cheatCodes; // Мапа для хранения активных чит-кодов

public:

void addCheatCode(const std::string& code) {

// Метод для добавления чит-кода

}

void removeCheatCode(const std::string& code) {

// Метод для удаления чит-кода

}

bool isCheatCodeActivated(const std::string& code) {

// Метод для проверки активации чит-кода

}

void activateCheatCode(const std::string& code) {

// Метод для активации чит-кода

}

void deactivateCheatCode(const std::string& code) {

// Метод для деактивации чит-кода

}

};

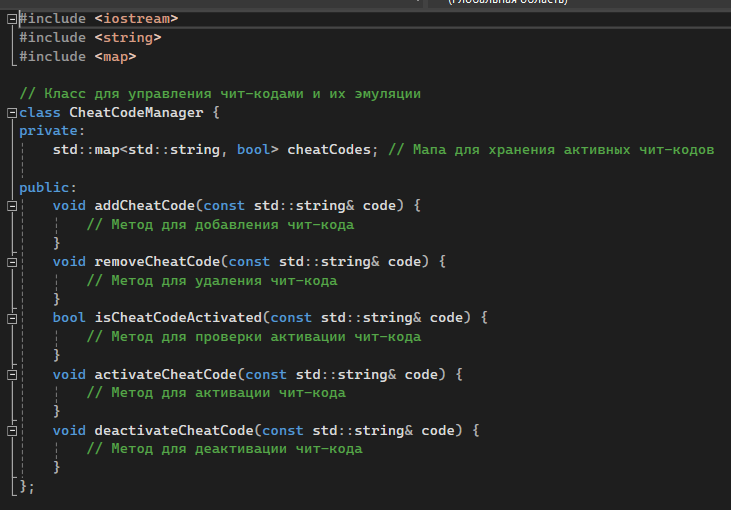


Рисунок 11 – Cheat Code.cpp

1. BIOS.cpp (рис. 12)

class BiosEmulator {

public:

void emulateSegaGenesisBios() {

//Здесь код для запуска Sega Genesis BIOS

}

};

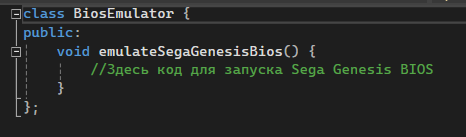


Рисунок 12 – BIOS.cpp

1. Game Genie Emulator.cpp (рис. 13)

class GameGenieEmulator {

public:

void emulateGameGenieExtension() {

//Код для эмулятора Game Genie extension

}

void enableCheatCodes() {

//код для включения чит-кодов

}

};

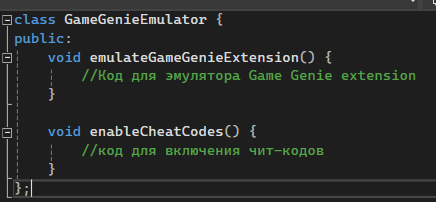


Рисунок 13 – Game Genie Emulator.cpp

1. Network Play.cpp (рис. 14)

class NetworkPlay {

public:

void playWithOtherUsers() {

//код для игры в игры с другими пользователями по сети

}

};

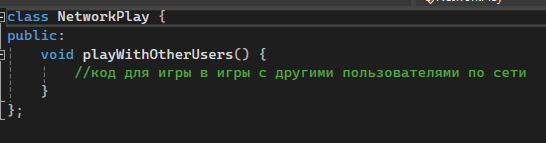


Рисунок 14 – Network Play.cpp

1. Frame Limiter.cpp (рис. 15)

class FrameLimiter {

public:

void limitFrameRate() {

//код для ограничения частоты кадров для эмуляции в реальном времени

}

};

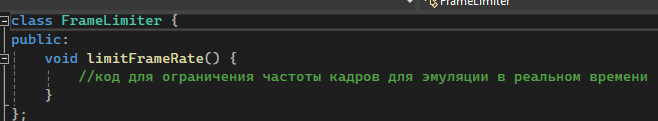


Рисунок 15 – Frame Limiter.cpp

1. Language Localization.cpp (рис. 16)

#include <string>

class LanguageLocalization {

public:

void translateInterfaceToLanguage(std::string language) {

//код для перевода интерфейса эмулятора на определенный язык

}

};

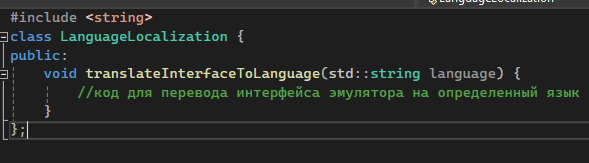


Рисунок 16 – Language Localization.cpp

1. Configuration Manager.cpp (рис. 17)

class ConfigurationManager {

public:

void configureEmulator() {

//код для настройки параметров эмулятора

}

void saveConfiguration() {

//код для сохранения конфигурации эмулятора для последующего использования

}

};

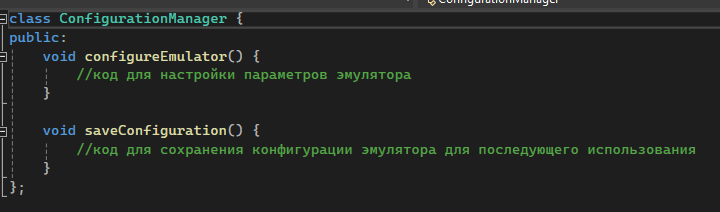


Рисунок 17 – Configuration Manager.cpp

1. GUI.cpp (рис. 18)

class GuiController {

public:

void createGraphicalUserInterface() {

//код для создания графического пользовательского интерфейса для управления эмулятором

}

};

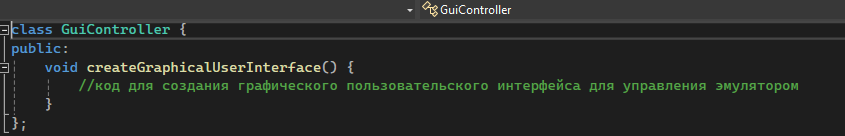


Рисунок 18 – GUI.cpp

1. Auto-Update Manager.cpp (рис. 19)

class AutoUpdateManager {

public:

void updateToLatestVersion() {

//код для автоматического обновления эмулятора до последней версии

}

};

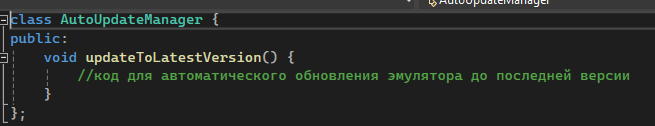


Рисунок 19 – Auto-Update Manager.cpp

1. Save File Converter.cpp (рис. 20)

#include <string>

class SaveFileConverter {

public:

void convertSavedFiles(std::string inputFile, std::string outputFile) {

//код для преобразования сохраненных игровых файлов в различные форматы

}

};

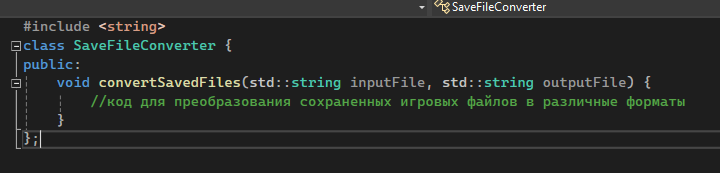


Рисунок 20 – Save File Converter.cpp

1. Netplay Lobby.cpp (рис. 21)

class NetplayLobby {

public:

void createLobbies() {

//код для создания комнат для сетевой игры и управления ими

}

};

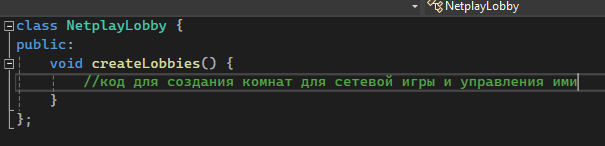


Рисунок 21 – Netplay Lobby.cpp

1. Rewind Feature.cpp (рис. 22)

class RewindFeature {

public:

void rewindGame() {

//код для перемотки времени в игре назад для воспроизведения или исправления ошибок

}

};

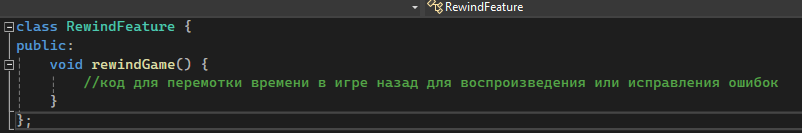


Рисунок 22 – Rewind Feature.cpp

1. Game Database.cpp (рис. 23)

#include <string>

class GameDatabase {

public:

void retrieveGameInfo(std::string gameTitle) {

//код для получения информации о совместимых играх Sega Genesis

}

};

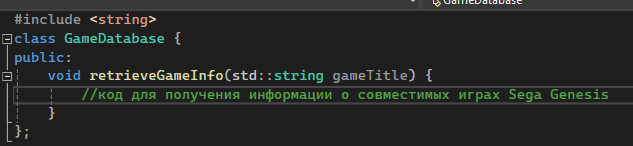


Рисунок 23 – Game Database.cpp

1. Screen Capture.cpp (рис. 24)

#include <string>

class ScreenCapture {

public:

void captureScreen(std::string capturePath, std::string format) {

//код для захвата и сохранения скриншотов во время игрового процесса

}

};

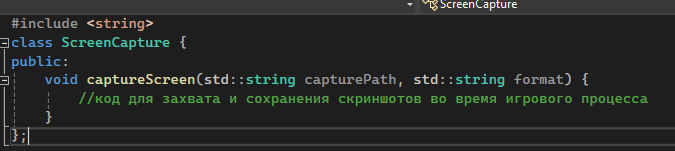


Рисунок 24 – Screen Capture.cpp

Создаем ветку lab2 и переходим в неё (рис. 25)

Команда “git checkout -b <навзание ветки>” создает новую ветку и переключается на нее. То есть, эта команда одновременно создает новую ветку и переключает Вас на нее, чтобы Вы могли начать работу в этой ветке.

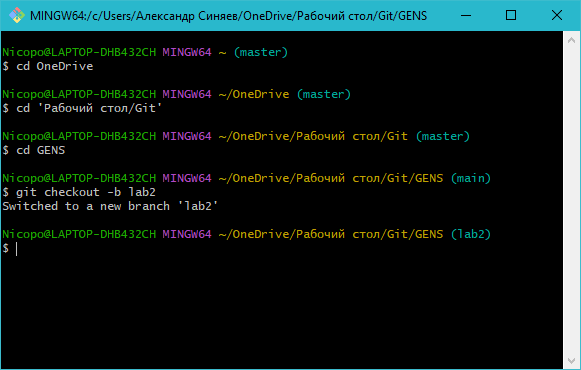


Рисунок 25 – Создание ветки lab2

Создаем в ветке файлы и ставим коммит, передаем файлы (рис. 26)

Команда `git add` используется для добавления изменений файлов в индекс (staging area) перед коммитом. Команда `git status` показывает текущее состояние репозитория, включая измененные файлы, файлы в индексе и другую информацию о состоянии репозитория.

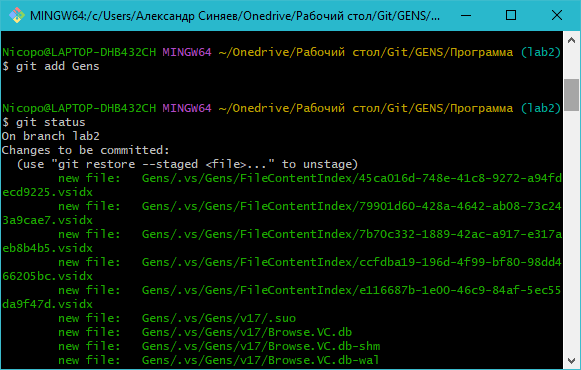


Рисунок 25 – Создаем в ветке файлы и ставим коммит

Вывод лог коммитов (рис. 26)

Команда git log --pretty=format:\"%h %ad | %s %d [%an]\" --graph --date=short выводит историю коммитов в Git с определенным форматированием.

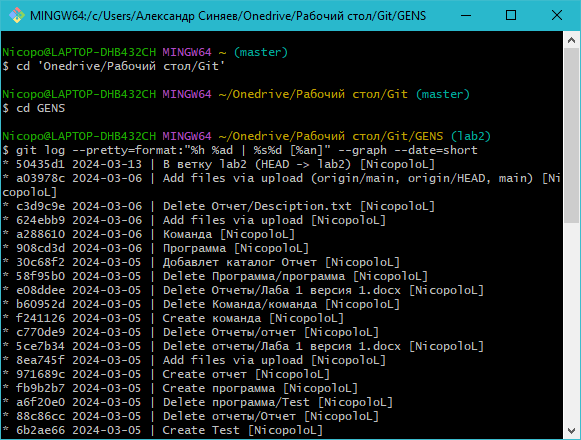
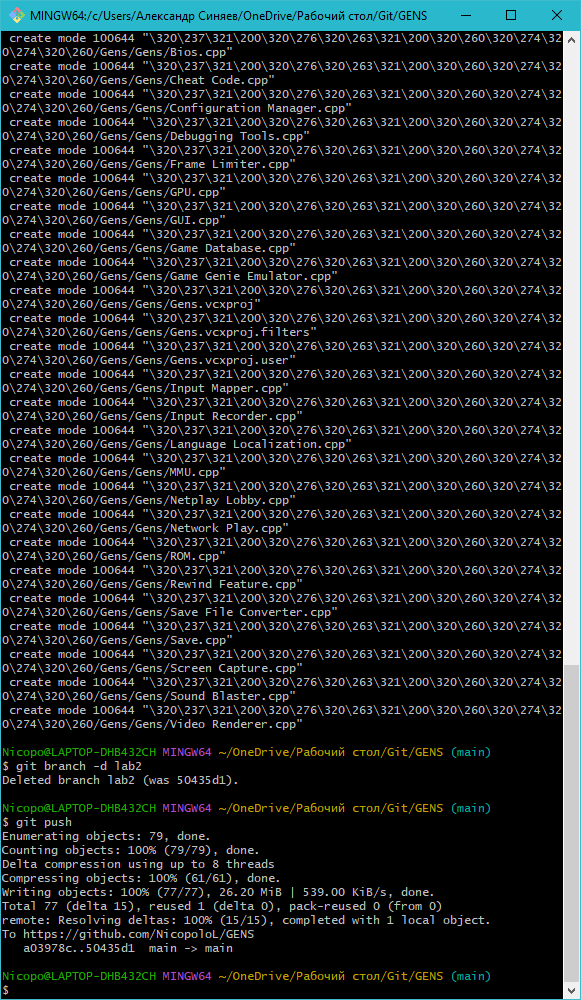
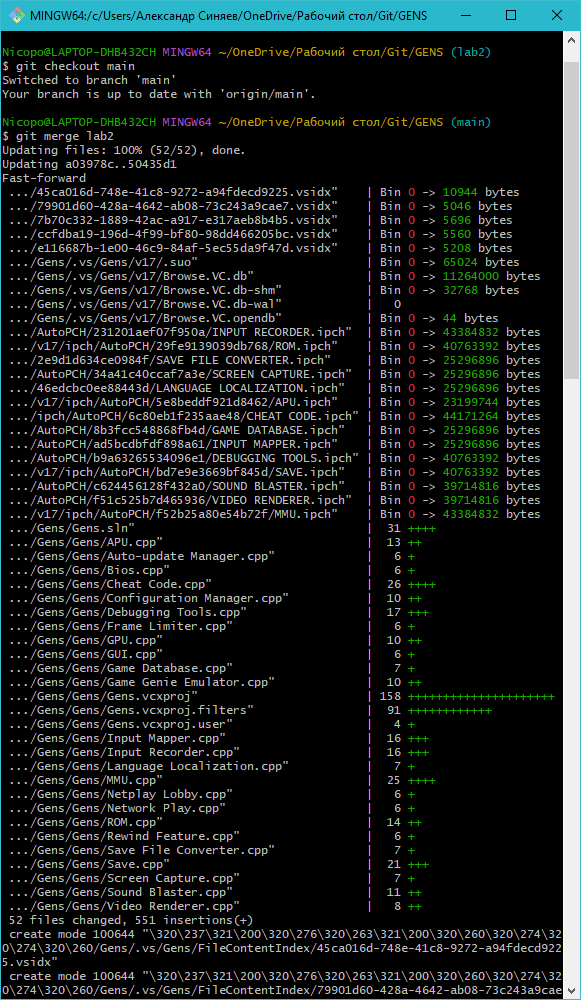


Рисунок 26 – Вывод лог коммитов

Переключаем рабочую копию на main и передаем изменения с ветки, а далее удаляем ветку

Команда git checkout в Git используется для переключения между ветками или восстановления файлов из репозитория. Команда git merge используется для слияния изменений из одной ветки в другую. Она объединяет изменения из указанной ветки в текущую ветку. Команда git branch возвращает список всех веток проекта (локальных и удаленных) и помечает текущую активную ветку.



Ссылка [NicopoloL/GENS (github.com)](https://github.com/NicopoloL/GENS)