# Отчет по лабораторной работе №3.3

## A. Информация о студентах

- Полное имя студентов: Катиев Али Муссаевич, Абушинов Алексей Юрьевич
- Номер студенческого билета: 245131, 242068
- Название предмета и код: Практикум по программированию. Код: — (не знаю)
- Номер лабораторной работы: 3.3
- Название игры: Сапер
- Роль в проекте: Ведущий разработчик / Архитектор ПО
- Состав команды:
  - Абушинов Алексей Ю.
  - Катиев Али М.

## B. Описание игры

- Полный текст назначенного задания

- У заданий отсутствуют отдельные текста. Все выполнялось по общим требованиям для всех заданий

- Описание классической игры и ее правил

- Сапер — это классическая логическая головоломка. Игровое поле разделено на смежные ячейки (квадраты), некоторые из которых «заминированы». Количество «заминированных» ячеек известно заранее.

- Цель игры: Открыть все ячейки, не содержащие мины.

- Правила:

  a. Игрок открывает ячейки, кликая по ним левой кнопкой мыши.
  b. Если в открытой ячейке есть мина, игра заканчивается проигрышем.
  c. Если мины нет, в ячейке появляется число, показывающее, сколько мин находится в соседних ячейках (диагональ, вертикаль, горизонталь).
  d. Если мин рядом нет (число 0), то автоматически открываются все соседние пустые ячейки (алгоритм заливки).
  e. Игрок может пометить ячейку, в которой, по его мнению, находится мина, «флагом» (правая кнопка мыши), чтобы случайно не открыть её.

- Реализованные изменения и улучшения

- Графический стиль: Полный редизайн интерфейса с использованием темной темы, неоновых акцентов, эффектов прозрачности (glassmorphism) и процедурной анимации.

- Архитектура: Переход на паттерн "Состояние" (State Pattern) для управления сценами, что позволяет легко масштабировать игру.

- Режим Кампании: Добавлена прогрессия уровней. С каждым уровнем поле увеличивается, а сложность растет. Введена система ограничения максимального уровня (Level Cap) с финальным экраном победы.

- Локальный Мультиплеер: Реализован режим для двух игроков на одном экране (Split-screen). Игроки соревнуются на скорость или выживание на двух независимых досках.

- Панель отладки (Debug Panel): Инструмент для разработчика, позволяющий видеть скрытые мины, принудительно выигрывать уровни и отслеживать состояние переменных в реальном времени.

- Адаптивность: Игра корректно масштабируется под различные разрешения экрана, поддерживая как оконный, так и полноэкранный режимы.

- Используемые инструменты и технологии

- Язык программирования: Python 3.13

- Библиотека: ‘pygame-ce’ (Community Edition) — использовалась для рендеринга 2D графики, обработки ввода и воспроизведения звука.

- Среда разработки: Visual Studio Code — с использованием расширений для Python и контроля версий.

- Формат данных: JSON — используется для хранения конфигурации игры (настройки графики, звука, правил сложности) и таблицы рекордов.

**C. Распределение ролей и задач**

- Подробное описание роли каждого участника

1. Абушинов Алексей Ю. (Архитектор / Backend Logic):

Отвечал за "ядро" игры. Основная задача заключалась в построении надежной архитектуры, которая позволила бы легко добавлять новые фичи.

- Разработка класса `Game` (основной цикл) и `StateManager` (менеджер сцен).

- Реализация алгоритмов генерации минного поля и рекурсивного открытия ячеек (`flood_fill`).

- Создание системы управления ресурсами (`ResourceManager`) для эффективной загрузки и кэширования ассетов.

- Реализация системы сохранений и загрузки конфигураций (`SaveManager`, `Settings`).

2. Катиев Али М. (Frontend / Gameplay):

Отвечал за визуальную часть, взаимодействие с пользователем и игровые режимы.

- Разработка системы UI компонентов: класс `Button` (кнопки с эффектами), `Slider` (ползунки громкости), `DebugPanel`.

- Верстка и логика сцен: Меню (`MenuScene`), Игра (`GameScene`), Пауза (`PauseScene`), Конец игры (`GameOverScene`).

- Реализация логики мультиплеера: синхронизация состояний двух досок, обработка условий победы/поражения для двух игроков.

- Визуальные эффекты: отрисовка флагов, мин, анимации переходов, интеграция звуковых эффектов.

- Методы сотрудничества и коммуникации

- Code Review: Перекрестная проверка кода перед слиянием веток.

- Модульность: Четкое разделение зон ответственности (один пишет логику доски, другой её отрисовку), что минимизировало конфликты при слиянии

- Мессенджеры: Обсуждение багов и идей в Telegram/Discord.

- Распределение задач по созданию графических элементов

- Процедурная графика: Большинство элементов (кнопки, панели, иконка гаечного ключа) рисуются программно с использованием примитивов `pygame.draw`. Реализовывал Катиев Али

- Ассеты: Поиск и адаптация спрайтов (мины, флаги) и звуков — совместная работа.

- Шрифты: Подбор шрифтовых пар для кибер-стилистики — Реализовывал Катиев Али.

**D. Архитектура проекта**

- Полная диаграмма классов (упрощенная схема связей)

```
mermaid
classDiagram
    class Game {
        +run()
        +handle_events()
        +update()
        +draw()
    }
    class StateManager {
        +change_state(state)
        +update()
        +draw()
    }
    class State {
        <<Abstract>>
        +enter()
        +exit()
        +update()
        +draw()
    }
    class MenuScene {
        +buttons
        +start_game()
    }
    class GameScene {
        +boards: List[Board]
        +difficulty
        +check_win()
    }
    class Board {
        +cells: List[List[Cell]]
        +generate_mines()
        +handle_click()
    }
```

```
class Cell {
    +is_mine
    +is_revealed
    +is_flagged
    +draw()
}
class ResourceManager {
    +get_image()
    +get_font()
    +get_sound()
}

Game --> StateManager
StateManager --> State
State <|-- MenuScene
State <|-- GameScene
GameScene --> Board
Board --> Cell
Game ..> ResourceManager : uses
```

- Описание ключевых компонентов и их обязанностей

- `src/main.py`: Точка входа. Инициализирует `Game` и устанавливает начальную сцену.

- `src/engine/game.py`: Класс-контейнер. Содержит главный цикл `while running`, инициализирует Pygame, экран и `StateManager`.

- `src/engine/state_manager.py`: Реализует паттерн State. Позволяет переключаться между меню, игрой и паузой без нагромождения условий `if/else` в главном цикле.

- `src/objects/board.py`: Содержит логику игрового поля. Хранит двумерный массив объектов `Cell`. Отвечает за расстановку мин и обработку кликов по координатам.

- `src/objects/cell.py`: Единица игрового поля. Знает свое состояние (мина/пусто, открыто/закрыто, флаг). Умеет рисовать себя.

- `src/ui/`: Пакет с UI элементами (`Button`, `Slider`, `DebugPanel`). Изолирует логику интерфейса от игровой логики.

- Обоснование использованных шаблонов проектирования

- State (Состояние): Идеально подходит для игр, где есть четкие режимы (Меню -> Игра -> Пауза -> Результат). Каждый режим — отдельный класс, что упрощает код и тестирование.

- Singleton (Одиночка): Использован для `ResourceManager` и `Settings`. Нам нужен глобальный доступ к ресурсам и настройкам из любой точки программы, и мы не хотим загружать одну и ту же текстуру дважды.

- Component (Компонент): UI элементы (`Button`) являются независимыми компонентами, которые можно переиспользовать в любой сцене.

**E. Реализованный функционал**

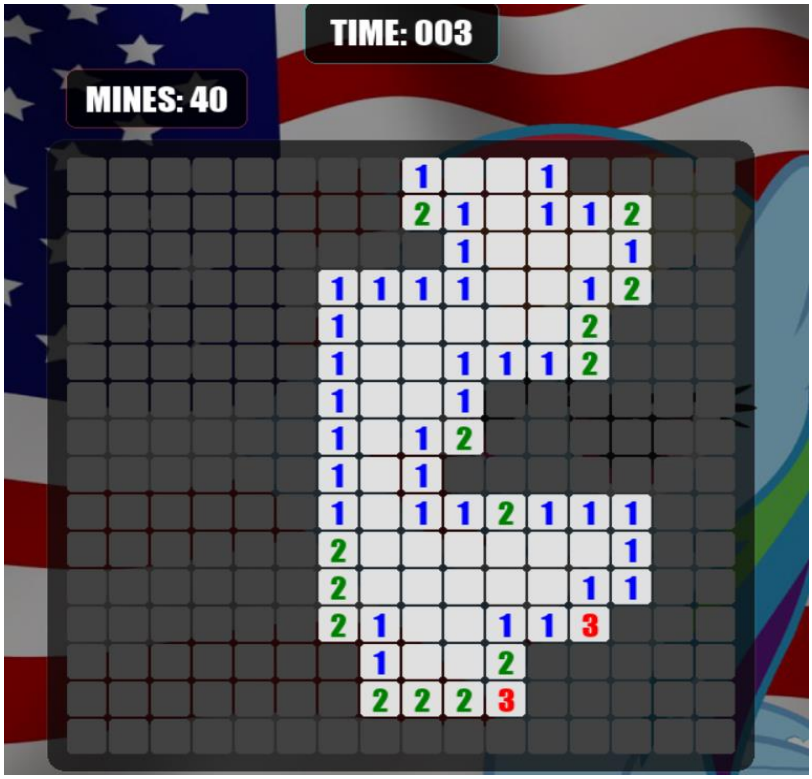
- Все основные требования с доказательствами

- Генерация поля: Выполнено. Поле генерируется случайно при первом клике (чтобы игрок не мог проиграть с первого хода).

  - Доказательство: См. метод `Board.place_mines`.

- Таймер и счетчик мин: Выполнено. В HUD отображается время с начала партии и количество оставшихся мин (всего мин минус количество флагов).

  - Доказательство:

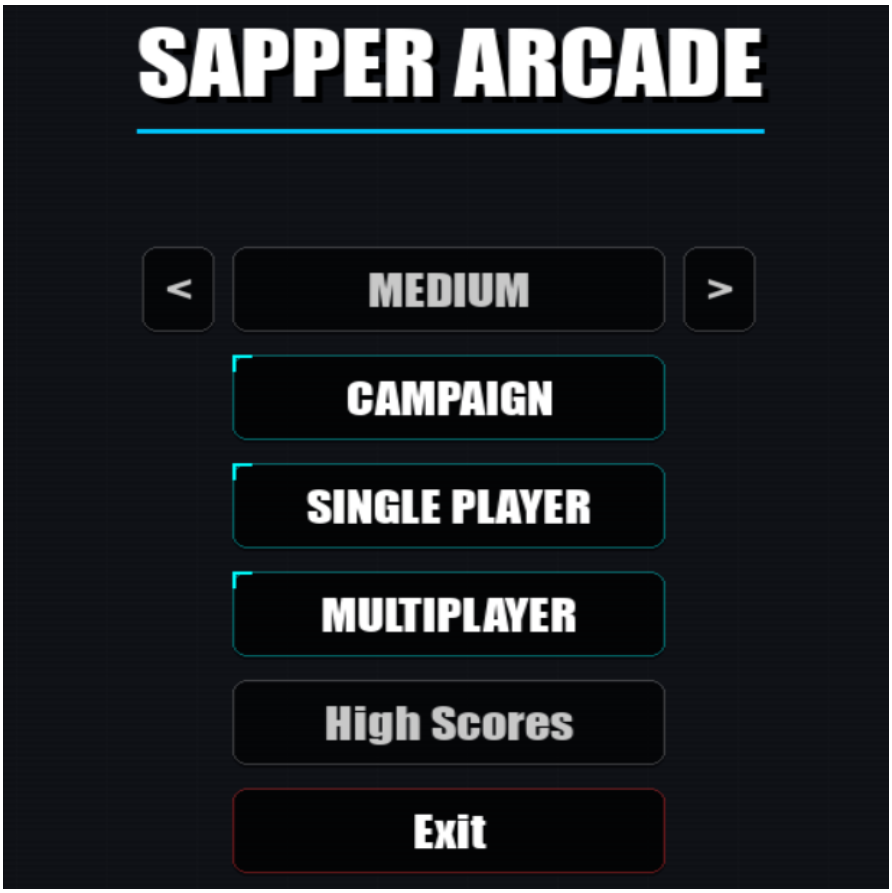


- Условия победы/поражения: Выполнено. Игра корректно определяет открытие всех безопасных клеток или подрыв на мине.

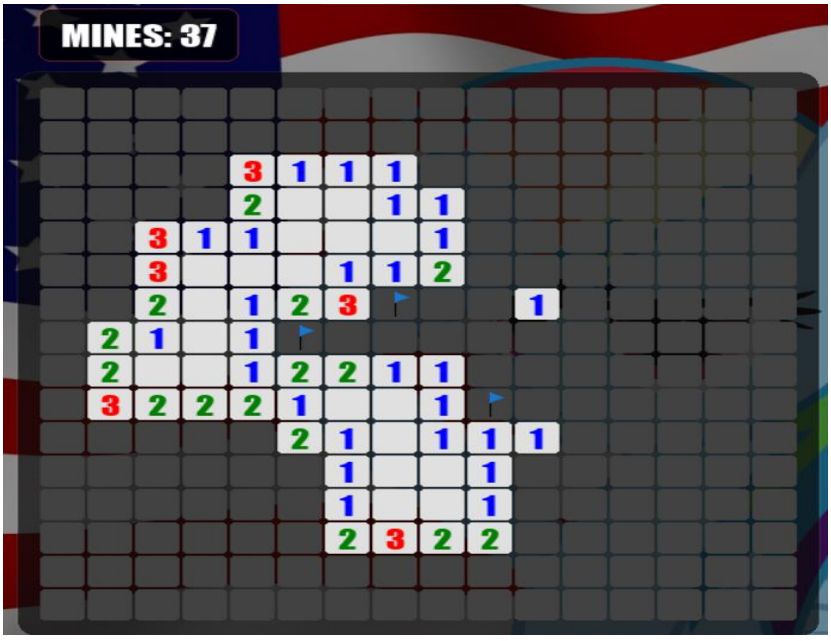
- Таблица рекордов: Выполнено. Лучшее время сохраняется в JSON файл и отображается в меню "High Scores".

- Скриншоты, демонстрирующие функции

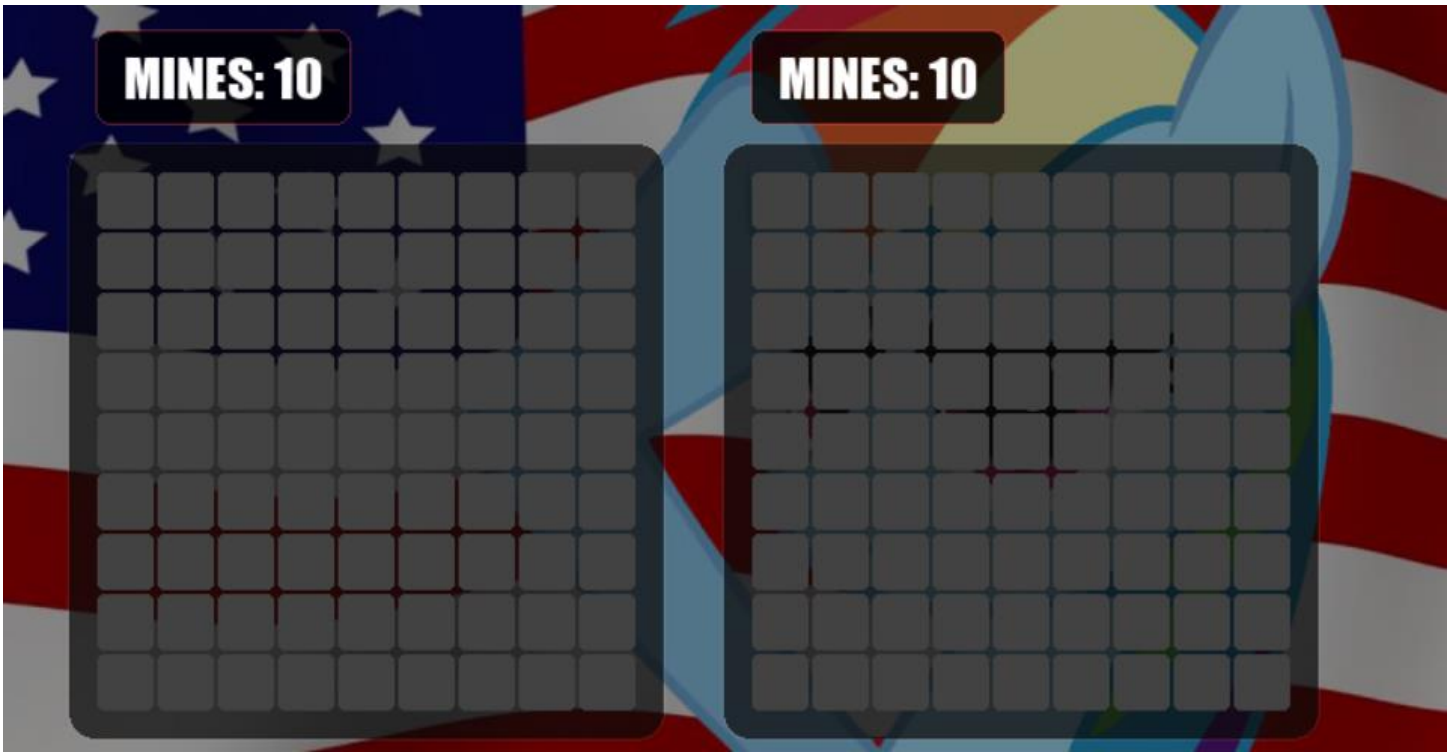
- Главное меню игры с выбором режима:



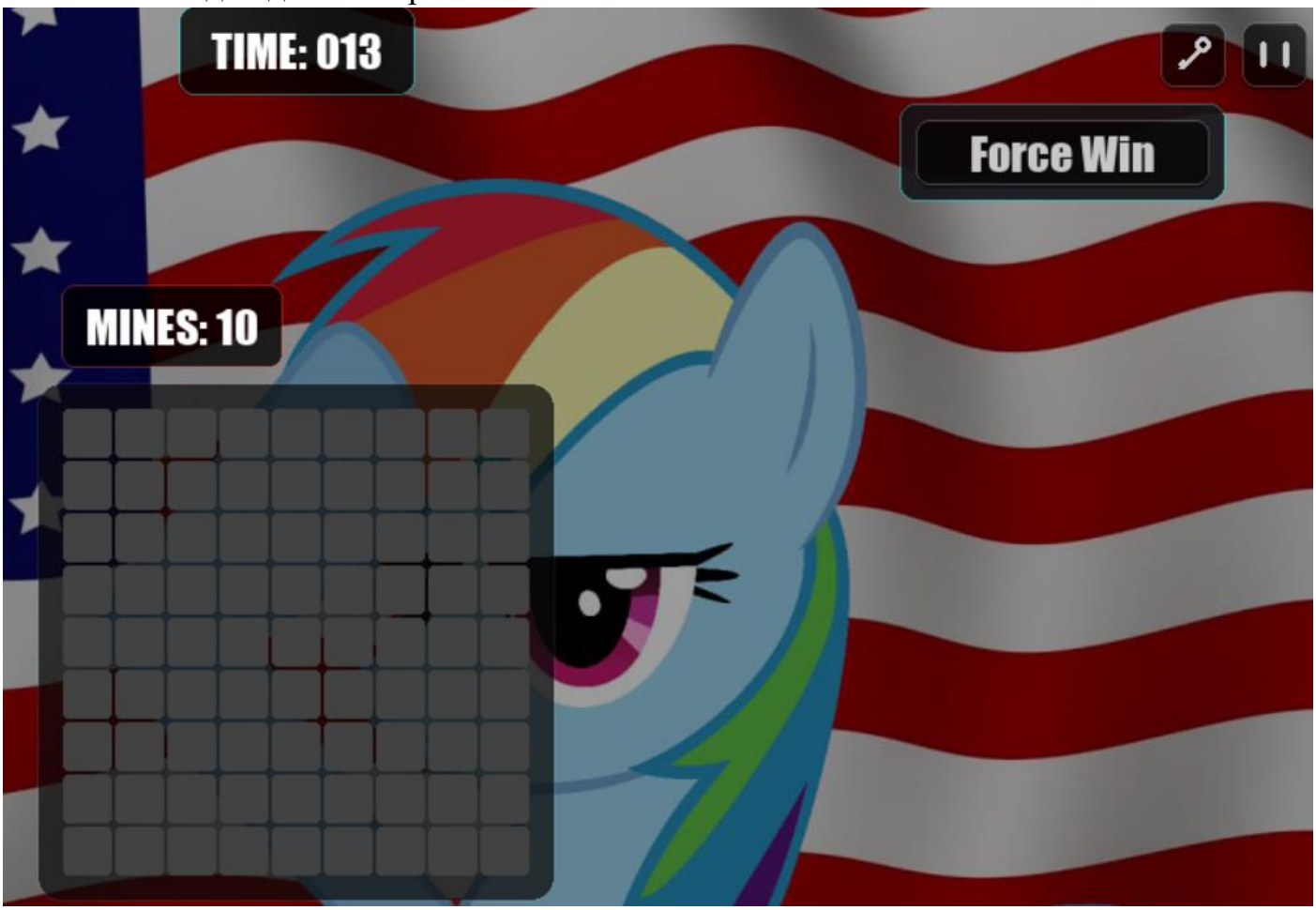

- Одиночный режим игры, уровень сложности Medium:



- Режим локального мультиплеера:




- Панель отладки для тестирования:



- Фрагменты кода для ключевых алгоритмов

- Алгоритм заливки (Flood Fill) для открытия пустых ячеек:

```
def reveal(self, r, c):
    # Базовые проверки выхода за границы и состояния
    if not (0 <= r < self.rows and 0 <= c < self.cols): return
    cell = self.cells[r][c]
    if cell.is_revealed or cell.is_flagged: return

    cell.is_revealed = True

    # Если ячейка пустая (0 мин вокруг), открываем соседей рекурсивно
    if cell.neighbor_mines == 0:
        for dr in [-1, 0, 1]:
            for dc in [-1, 0, 1]:
                if dr == 0 and dc == 0: continue
                self.reveal(r + dr, c + dc)
```

- Обработка изменения размера окна (Адаптивность):

```
def _apply_layout(self):
    # Вычисляем требуемый размер контента
    req_w = board_w * self.num_players + margin
    req_h = board_h + hud_height

    # Вычисляем коэффициент масштабирования
    self.scale = 1.0
    if req_w > SETTINGS.WIDTH or req_h > SETTINGS.HEIGHT:
        scale_w = SETTINGS.WIDTH / req_w
        scale_h = SETTINGS.HEIGHT / req_h
        self.scale = min(scale_w, scale_h)
```

```
    # Применяем масштаб к размеру ячеек
    scaled_cell_size = int(SETTINGS.LAYOUT['cell_size'] * self.scale)
```

- Описание решенных технических проблем

- Проблема рекурсии: При реализации `flood_fill` на больших полях возникала ошибка переполнения стека (RecursionError).

  - Решение: Оптимизация алгоритма и увеличение лимита рекурсии, либо переход на итеративный подход с использованием очереди (Queue). В текущей версии Python лимита хватает для стандартных размеров поля.

- Масштабирование интерфейса: При изменении размера окна или переключении в полноэкранный режим элементы UI съезжали или перекрывали друг друга.

  - Решение: Внедрение метода `on_resize` во все сцены и динамический пересчет координат всех элементов относительно `SETTINGS.WIDTH` и `SETTINGS.HEIGHT` при каждом событии `VIDEORESIZE`.

- Двойная обработка событий: Кнопки нажимались дважды за один клик.

  - Решение: Исправлен цикл обработки событий в `GameScene`. Вызовы `handle_event` для кнопок перенесены строго внутрь блока проверки `pygame.MOUSEBUTTONDOWN`.

**F. Инструкции по запуску и игре**

- Системные требования и зависимости

- ОС: Windows 10/11, Linux, macOS.

- Python: Версия 3.8 или выше.

- Зависимости: Библиотека `pygame-ce`.

- Установка: `pip install pygame-ce`

- Полная схема управления

- Левая кнопка мыши (ЛКМ): Открыть ячейку / Нажать кнопку меню.

- Правая кнопка мыши (ПКМ): Поставить/снять флаг.

- ESC: Пауза / Возврат в предыдущее меню.

- F11: Переключение полноэкранного режима.

- Правила и цели

- Campaign: Пройдите серию из 10 уровней с возрастающей сложностью.

- Single Player: Классическая игра. Выберите сложность (Easy, Medium, Hard) и очистите поле.

- Multiplayer: Соревнуйтесь с другом. Кто быстрее очистит свою доску или кто дольше проживет, не подорвавшись на мине.

**G. Полный исходный код**

- Структура организации ресурсов

Проект организован по модульному принципу, разделяя логику движка, игровые объекты, сцены и конфигурацию.

```
SapperPjct/
├── assets/                  # Ресурсы игры
│   ├── images/              # Графические изображения
│   │   ├── background.jpg
│   │   ├── lose.png
│   │   ├── mp_draw.png
│   │   ├── mp_win.png
│   │   ├── p1_lose.png
│   │   └── p2_lose.png
│   └── sounds/              # Звуковые эффекты и музыка
│       ├── bgm.ogg
│       ├── click.wav
│       ├── explode.wav
│       ├── mp_lose.wav
│       └── win.wav
├── data/
    └── highscores.json
```

```
└── src/                    # Исходный код
    ├── config/             # Конфигурационные файлы
    │   ├── game_config.json
    │   ├── graphics.json
    │   └── settings.py
    ├── engine/             # Ядро игрового движка
    │   ├── game.py
    │   ├── resource_manager.py
    │   ├── save_manager.py
    │   ├── sound_generator.py
    │   └── state_manager.py
    ├── objects/            # Игровые объекты
    │   ├── board.py
    │   └── cell.py
    ├── scenes/             # Игровые сцены (состояния)
    │   ├── game_over_scene.py
    │   ├── game_scene.py
    │   ├── highscores_scene.py
    │   ├── menu_scene.py
    │   └── pause_scene.py
    ├── ui/                 # Элементы пользовательского интерфейса
    │   ├── debug_panel.py
    │   └── ui_elements.py
    └── main.py             # Точка входа
```

- Конфигурационные файлы

  - `src/config/game_config.json`

Содержит основные настройки игры, параметры сложности и аудио.

```
{
    "game": {
        "title": "Sapper Arcade",
        "version": "1.1",
        "width": 1280,
        "height": 720,
```

```
        "fps": 60,
        "max_players": 2
    },
    "difficulty_levels": {
        "easy": {
            "rows": 9,
            "cols": 9,
            "mines": 10
        },
        "medium": {
            "rows": 16,
            "cols": 16,
            "mines": 40
        },
        "hard": {
            "rows": 16,
            "cols": 30,
            "mines": 99
        }
    },
    "audio": {
        "music_volume": 0.2,
        "sfx_volume": 0.2,
        "enabled": true
    }
}
```

- `src/config/graphics.json`

Определяет цветовую палитру, шрифты и параметры макета.

```
{
    "colors": {
        "background": [33, 33, 33],
        "grid_line": [66, 66, 66],
        "cell_closed": [66, 66, 66],
```

```
        "cell_opened": [224, 224, 224],
        "cell_hover": [97, 97, 97],
        "text": [33, 33, 33],
        "mine": [211, 47, 47],
        "flag": [25, 118, 210],
        "ui_panel": [50, 50, 50],
        "ui_text": [255, 255, 255],
        "ui_button": [66, 66, 66],
        "ui_button_hover": [97, 97, 97]
    },
    "fonts": {
        "main": "Impact",
        "size_small": 20,
        "size_medium": 28,
        "size_large": 56
    },
    "layout": {
        "cell_size": 32,
        "margin": 2
    }
}
```

- `src/config/settings.py`

Класс-обертка для загрузки и доступа к конфигурации.

```
import json
import os

class Settings:
    def __init__(self):
        self.base_dir =
os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__
))))
        self.config_dir = os.path.join(self.base_dir, 'src', 'config')
```

```
        self.game_config = self._load_json('game_config.json')
        self.graphics_config = self._load_json('graphics.json')

        # Быстрый доступ к настройкам
        self.WIDTH = self.game_config['game']['width']
        self.HEIGHT = self.game_config['game']['height']
        self.FPS = self.game_config['game']['fps']
        self.TITLE = self.game_config['game']['title']

        self.COLORS = self.graphics_config['colors']
        self.FONTS = self.graphics_config['fonts']
        self.LAYOUT = self.graphics_config['layout']

    def _load_json(self, filename):
        path = os.path.join(self.config_dir, filename)
        try:
            with open(path, 'r', encoding='utf-8') as f:
                return json.load(f)
        except FileNotFoundError:
            print(f"Файл конфигурации не найден: {path}")
            return {}
        except json.JSONDecodeError:
            print(f"Ошибка декодирования JSON: {path}")
            return {}

# Глобальный экземпляр настроек
SETTINGS = Settings()
```

- Основные модули с пояснениями

  - `src/main.py`

Точка входа в приложение. Инициализирует игру и запускает главное меню.

```
import sys
import os

sys.path.append(os.path.join(os.path.dirname(__file__), '..'))

from src.engine.game import Game

if __name__ == "__main__":
    game = Game()
    from src.scenes.menu_scene import MenuScene
    game.state_manager.change_state(MenuScene(game))
    game.run()
```

- `src/engine/game.py`

Основной класс игры, управляющий циклом событий, обновлением и отрисовкой.

```
import pygame
import sys
from src.config.settings import SETTINGS
from src.engine.state_manager import StateManager
from src.engine.resource_manager import RESOURCES

class Game:
    def __init__(self):
        pygame.init()
        pygame.mixer.init()

        self.WIDTH = SETTINGS.WIDTH
        self.HEIGHT = SETTINGS.HEIGHT

        self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT),
pygame.RESIZABLE)
        pygame.display.set_caption(f"{SETTINGS.TITLE}
v{SETTINGS.game_config['game']['version']}")
```

```
        self.clock = pygame.time.Clock()
        self.running = True

        self.state_manager = StateManager(self)

    def run(self):
        while self.running:
            self.clock.tick(SETTINGS.FPS)
            self._handle_events()
            self._update()
            self._draw()

        pygame.quit()
        sys.exit()

    def _handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.VIDEORESIZE:
                self.WIDTH, self.HEIGHT = event.w, event.h
                SETTINGS.WIDTH, SETTINGS.HEIGHT = event.w, event.h
                self.screen = pygame.display.set_mode((self.WIDTH,
self.HEIGHT), pygame.RESIZABLE)
                if hasattr(self.state_manager.state, 'on_resize'):
                    self.state_manager.state.on_resize(self.WIDTH,
self.HEIGHT)

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_F11:
                    # Переключение полноэкранного режима
                    is_fullscreen = self.screen.get_flags() &
pygame.FULLSCREEN
                    if is_fullscreen:
```

```
                        self.screen =
pygame.display.set_mode((SETTINGS.WIDTH, SETTINGS.HEIGHT),
pygame.RESIZABLE)
                    else:
                        self.screen =
pygame.display.set_mode((SETTINGS.WIDTH, SETTINGS.HEIGHT),
pygame.FULLSCREEN)

                    self.WIDTH, self.HEIGHT = self.screen.get_size()
                    SETTINGS.WIDTH, SETTINGS.HEIGHT = self.WIDTH,
self.HEIGHT

                    if hasattr(self.state_manager.state, 'on_resize'):
                        self.state_manager.state.on_resize(self.WIDTH,
self.HEIGHT)

            self.state_manager.handle_event(event)

    def _update(self):
        self.state_manager.update()

    def _draw(self):
        self.screen.fill(SETTINGS.COLORS['background'])
        self.state_manager.draw(self.screen)
        pygame.display.flip()
```

- `src/engine/state_manager.py`

Управляет переходами между сценами (Меню, Игра, Пауза и т.д.).

```
class StateManager:
    def __init__(self, game):
        self.game = game
        self.state = None
```

```
    def change_state(self, new_state):
        if self.state:
            self.state.exit()
        self.state = new_state
        if self.state:
            self.state.enter()

    def update(self):
        if self.state:
            self.state.update()

    def draw(self, screen):
        if self.state:
            self.state.draw(screen)

    def handle_event(self, event):
        if self.state:
            self.state.handle_event(event)

class State:
    def __init__(self, game):
        self.game = game

    def enter(self): pass
    def exit(self): pass
    def update(self): pass
    def draw(self, screen): pass
    def handle_event(self, event): pass
```

- `src/objects/board.py`

Логика игрового поля: генерация мин, обработка кликов, алгоритм заливки (flood fill).

```
import pygame
import random
from src.objects.cell import Cell
from src.config.settings import SETTINGS
from src.engine.resource_manager import RESOURCES


class Board:
    def __init__(self, rows, cols, mines, x, y):
        self.rows = rows
        self.cols = cols
        self.total_mines = mines
        self.x = x
        self.y = y
        self.cell_size = SETTINGS.LAYOUT['cell_size']

        self.cells = [[Cell(r, c, self.cell_size) for c in range(cols)]
for r in range(rows)]
        self.mines_placed = False
        self.game_over = False
        self.win = False
        self.flags_placed = 0

        self.font = RESOURCES.get_font(SETTINGS.FONTS['main'],
SETTINGS.FONTS['size_small'])

    def update_layout(self, x, y, cell_size):
        self.x = x
        self.y = y
        self.cell_size = cell_size
        for r in range(self.rows):
            for c in range(self.cols):
                self.cells[r][c].update_rect(x, y, cell_size)

    def _place_mines(self, first_r, first_c):
        mines_left = self.total_mines
```

```
        while mines_left > 0:
            r = random.randint(0, self.rows - 1)
            c = random.randint(0, self.cols - 1)

            # Не ставим мину в первую нажатую клетку и ее соседей
            if abs(r - first_r) <= 1 and abs(c - first_c) <= 1:
                continue

            if not self.cells[r][c].is_mine:
                self.cells[r][c].is_mine = True
                mines_left -= 1

        # Подсчитываем соседей
        for r in range(self.rows):
            for c in range(self.cols):
                if not self.cells[r][c].is_mine:
                    self.cells[r][c].neighbor_mines =
self._count_neighbors(r, c)

        self.mines_placed = True

    def _count_neighbors(self, r, c):
        count = 0
        for dr in [-1, 0, 1]:
            for dc in [-1, 0, 1]:
                if dr == 0 and dc == 0: continue
                nr, nc = r + dr, c + dc
                if 0 <= nr < self.rows and 0 <= nc < self.cols:
                    if self.cells[nr][nc].is_mine:
                        count += 1
        return count

    def handle_click(self, pos, button):
        if self.game_over: return
```

```
        # Проверяем, попал ли клик в доску
        rel_x = pos[0] - self.x
        rel_y = pos[1] - self.y

        if not (0 <= rel_x < self.cols * self.cell_size and 0 <= rel_y <
self.rows * self.cell_size):
            return

        c = int(rel_x // self.cell_size)
        r = int(rel_y // self.cell_size)

        cell = self.cells[r][c]

        if button == 1: # ЛКМ
            if cell.is_flagged: return

            if not self.mines_placed:
                self._place_mines(r, c)

            if cell.is_mine:
                self._game_over_loss()
            else:
                self._reveal(r, c)
                # Звук клика
                sound = RESOURCES.get_sound('click.wav')
                if sound:

sound.set_volume(SETTINGS.game_config['audio']['sfx_volume'])
                    sound.play()
                self._check_win()

        elif button == 3: # ПКМ
            if not cell.is_revealed:
                cell.is_flagged = not cell.is_flagged
```

```
                self.flags_placed += (1 if cell.is_flagged else -1)

    def _reveal(self, r, c):
        cell = self.cells[r][c]
        if cell.is_revealed or cell.is_flagged: return

        cell.is_revealed = True

        if cell.neighbor_mines == 0:
            for dr in [-1, 0, 1]:
                for dc in [-1, 0, 1]:
                    nr, nc = r + dr, c + dc
                    if 0 <= nr < self.rows and 0 <= nc < self.cols:
                        self._reveal(nr, nc)

    def _game_over_loss(self):
        self.game_over = True
        self.win = False
        self._reveal_all_mines()

    def _reveal_all_mines(self):
        for r in range(self.rows):
            for c in range(self.cols):
                if self.cells[r][c].is_mine:
                    self.cells[r][c].is_revealed = True

    def _check_win(self):
        revealed_count = sum(1 for r in range(self.rows) for c in
range(self.cols) if self.cells[r][c].is_revealed)
        if revealed_count == (self.rows * self.cols - self.total_mines):
            self.game_over = True
            self.win = True

    def draw(self, screen):
```

```
        for r in range(self.rows):
            for c in range(self.cols):
                self.cells[r][c].draw(screen, self.font)
```

- `src/scenes/game_scene.py`

Сцена игрового процесса. Управляет досками, HUD, логикой победы/поражения и мультиплеером.

(Фрагмент кода)

```
class GameScene(State):
    def __init__(self, game, difficulty='easy', num_players=1, level=1):
        super().__init__(game)
        # ... инициализация ...
        self._setup_boards()
        self._create_ui()

    def update(self):
        # ... логика обновления ...
        # Проверка условий победы/поражения
        if all_finished:
            # Переход к GameOverScene
            pass

    def draw(self, screen):
        # Отрисовка фона, досок, HUD
        pass
```

- `src/scenes/menu_scene.py`

Главное меню игры.

```
class MenuScene(State):
```

```
    def _create_ui(self):
        # Создание кнопок: Campaign, Single Player, Multiplayer, Exit
        pass

    def draw(self, screen):
        # Отрисовка фона и кнопок
        pass
```

- `src/ui/debug_panel.py`

Панель отладки для тестирования (вызов победы, просмотр состояний).

```
class DebugPanel:
    def __init__(self, scene):
        self.scene = scene
        self.visible = False
        # ...

    def toggle(self):
        self.visible = not self.visible

    def draw(self, screen):
        if not self.visible: return
        # Отрисовка панели и кнопок отладки
```