МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Донецкий национальный технический университет»

Факультет ИСП

Кафедра ПИ им Л.П.Фельдмана

Лабораторная работа № 4

на тему: «Создание самодокументирующегося кода»

по курсу: «Профессиональная практика программной инженерии»

Проверил:

асс. каф. ПИ им. Л.П.Фельдмана Филипишин Д.А.

Выполнил:

ст. гр. ПИ-21б

Красозов А.С.

Донецк-2024

Цель работы – научиться добавлять в программный код специальным образом оформление докблок-комментарии, для последующей автоматической генерации API reference, а также познакомиться с форматом оформления документации DocBook.

Вариант 11. Видео игра симулятор (Аналогичная Tennis Manager 2024);

1. Регистрация на GitHub (рис. 1). Аккаунт: [ArtemKrasozov](https://github.com/ArtemKrasozov). Репозиторий: [TennisSimPro](https://github.com/ArtemKrasozov/TennisSimPro).

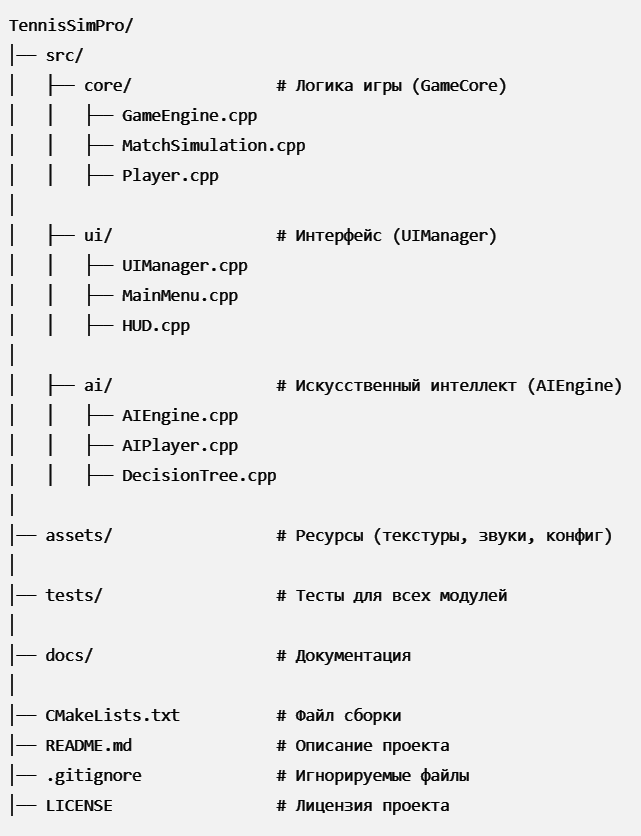


Рисунок 1 – Структура проекта

#include "AIEngine.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

/\*\*

 \* @brief Конструктор AIEngine. Инициализирует генератор случайных чисел.

 \*/

AIEngine::AIEngine() {

    std::srand(std::time(nullptr));  // Инициализация генератора случайных чисел

}

/\*\*

 \* @brief Рассчитывает следующий ход ИИ.

 \*

 \* @param aiPlayer Игрок, управляемый ИИ.

 \* @param opponent Противник.

 \*/

void AIEngine::calculateMove(Player& aiPlayer, Player& opponent) {

    std::cout << "ИИ анализирует ситуацию...\n";

    int riskFactor = evaluateRisk(aiPlayer, opponent);

    std::string decision = makeStrategicDecision(riskFactor);

    std::cout << "ИИ принимает решение: " << decision << std::endl;

}

/\*\*

 \* @brief Оценивает уровень риска для текущего состояния игры.

 \*

 \* @param aiPlayer Игрок, управляемый ИИ.

 \* @param opponent Противник.

 \* @return int Уровень риска (чем выше, тем опаснее ситуация).

 \*/

int AIEngine::evaluateRisk(const Player& aiPlayer, const Player& opponent) {

    int scoreDifference = opponent.getScore() - aiPlayer.getScore();

    int randomFactor = std::rand() % 10;  // Дополнительный случайный фактор для разнообразия поведения

    int riskLevel = scoreDifference + randomFactor;

    if (riskLevel < 5) {

        std::cout << "Риск низкий. Можно атаковать!\n";

    } else if (riskLevel < 10) {

        std::cout << "Риск средний. Требуется сбалансированная стратегия.\n";

    } else {

        std::cout << "Риск высокий. Лучше играть на удержание.\n";

    }

    return riskLevel;

}

/\*\*

 \* @brief Принимает стратегическое решение на основе уровня риска.

 \*

 \* @param riskFactor Уровень риска.

 \* @return std::string Выбранная стратегия.

 \*/

std::string AIEngine::makeStrategicDecision(int riskFactor) {

    if (riskFactor < 5) {

        return aggressivePlay();

    } else if (riskFactor < 10) {

        return balancedPlay();

    } else {

        return defensivePlay();

    }

}

/\*\*

 \* @brief Выполняет агрессивную стратегию.

 \*

 \* @return std::string Описание выбранного агрессивного действия.

 \*/

std::string AIEngine::aggressivePlay() {

    int move = std::rand() % 3;

    switch (move) {

        case 0: return "Мощный удар по линии";

        case 1: return "Подход к сетке и атака";

        case 2: return "Удар с вращением, чтобы сбить противника с толку";

    }

    return "Агрессивный удар";

}

/\*\*

 \* @brief Выполняет сбалансированную стратегию.

 \*

 \* @return std::string Описание выбранного сбалансированного действия.

 \*/

std::string AIEngine::balancedPlay() {

    int move = std::rand() % 3;

    switch (move) {

        case 0: return "Средняя подача и контроль темпа игры";

        case 1: return "Чередование длинных и коротких ударов";

        case 2: return "Выжидательная игра с быстрым контрударом";

    }

    return "Сбалансированная стратегия";

}

/\*\*

 \* @brief Выполняет оборонительную стратегию.

 \*

 \* @return std::string Описание выбранного оборонительного действия.

 \*/

std::string AIEngine::defensivePlay() {

    int move = std::rand() % 3;

    switch (move) {

        case 0: return "Глубокий оборонительный удар в дальний угол";

        case 1: return "Минимальный риск, удержание мяча в игре";

        case 2: return "Применение резаных ударов для снижения скорости игры";

    }

    return "Оборонительная тактика";

}

/\*\*

 \* @brief Прогнозирует исход игры на основе текущего состояния игроков.

 \*

 \* @param aiPlayer Игрок, управляемый ИИ.

 \* @param opponent Противник.

 \* @return int 1 - победа, 0 - поражение.

 \*/

int AIEngine::predictOutcome(const Player& aiPlayer, const Player& opponent) {

    int aiSkill = std::rand() % 100 + aiPlayer.getScore();

    int opponentSkill = std::rand() % 100 + opponent.getScore();

    std::cout << "ИИ прогнозирует вероятность победы...\n";

    std::cout << "Уровень ИИ: " << aiSkill << " | Уровень соперника: " << opponentSkill << std::endl;

    return aiSkill > opponentSkill ? 1 : 0; // 1 - победа, 0 - проигрыш

}

Рисунок 2 – Пример файл с ДокБлоками AIEngine.cpp

Sphinx

Sphinx — это популярный инструмент для генерации документации, который изначально был разработан для Python, но теперь используется для создания документации на различных языках программирования, включая C++.

Sphinx работает с текстовыми файлами в формате reStructuredText (.rst) и генерирует документацию в различных форматах, таких как HTML, LaTeX, PDF и других. Для работы с кодом на C++ часто используется расширение Breathe, которое позволяет интегрировать Sphinx с Doxygen, обеспечивая более удобное извлечение комментариев из исходных файлов C++.

Doxygen

Doxygen — это инструмент для автоматической генерации документации из исходного кода. Он поддерживает множество языков программирования, включая C++, Java, Python и другие. Doxygen анализирует код, извлекает комментарии, содержащие специальные метки (например, @brief, @param, @return), и генерирует документацию в виде HTML, LaTeX, PDF и других форматов.

Взаимодействие Sphinx и Doxygen

Sphinx и Doxygen часто используются вместе для создания качественной и подробной документации для C++ проектов.

* Doxygen используется для извлечения информации о коде (классы, методы, функции и т. д.) и генерации документации в формате, который можно обработать другими инструментами (например, в формате XML).
* Sphinx с расширением Breathe использует эти XML файлы от Doxygen для интеграции с остальной частью документации, создавая более гибкие и расширяемые системы документации.

Таким образом, Doxygen извлекает информацию о коде и генерирует XML-вывод, а Sphinx (с помощью Breathe) использует эти данные для создания более структурированной документации, добавляя дополнительные элементы, такие как подробные объяснения, графики и другие части, которые могут быть удобно организованы в документации.

Процесс

1. Запуск Doxygen для генерации XML-файлов из исходного кода.
2. Настройка Sphinx для использования с Breathe, чтобы подключить сгенерированные Doxygen файлы.
3. Использование Sphinx для создания окончательной документации в нужном формате (HTML, PDF и т. д.).

Таким образом, Doxygen извлекает информацию о коде, а Sphinx помогает интегрировать и улучшить эту информацию в более полную и понятную документацию для разработчиков.

Структура полученных файлов представлена на рисунке 3.

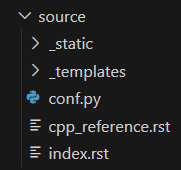


Рисунок 3 – Структура папки source

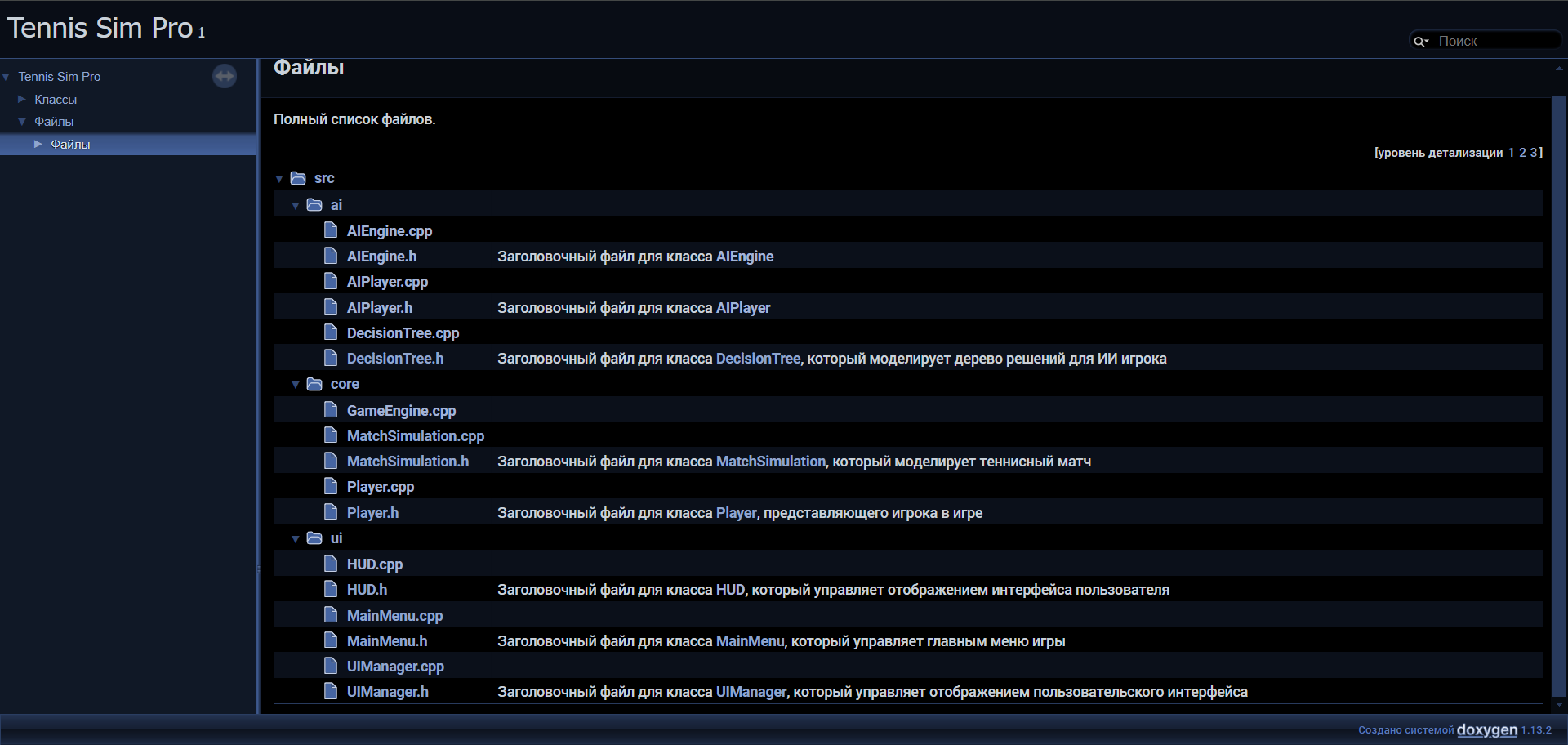


Рисунок 4 – Список файлов с полученной документации

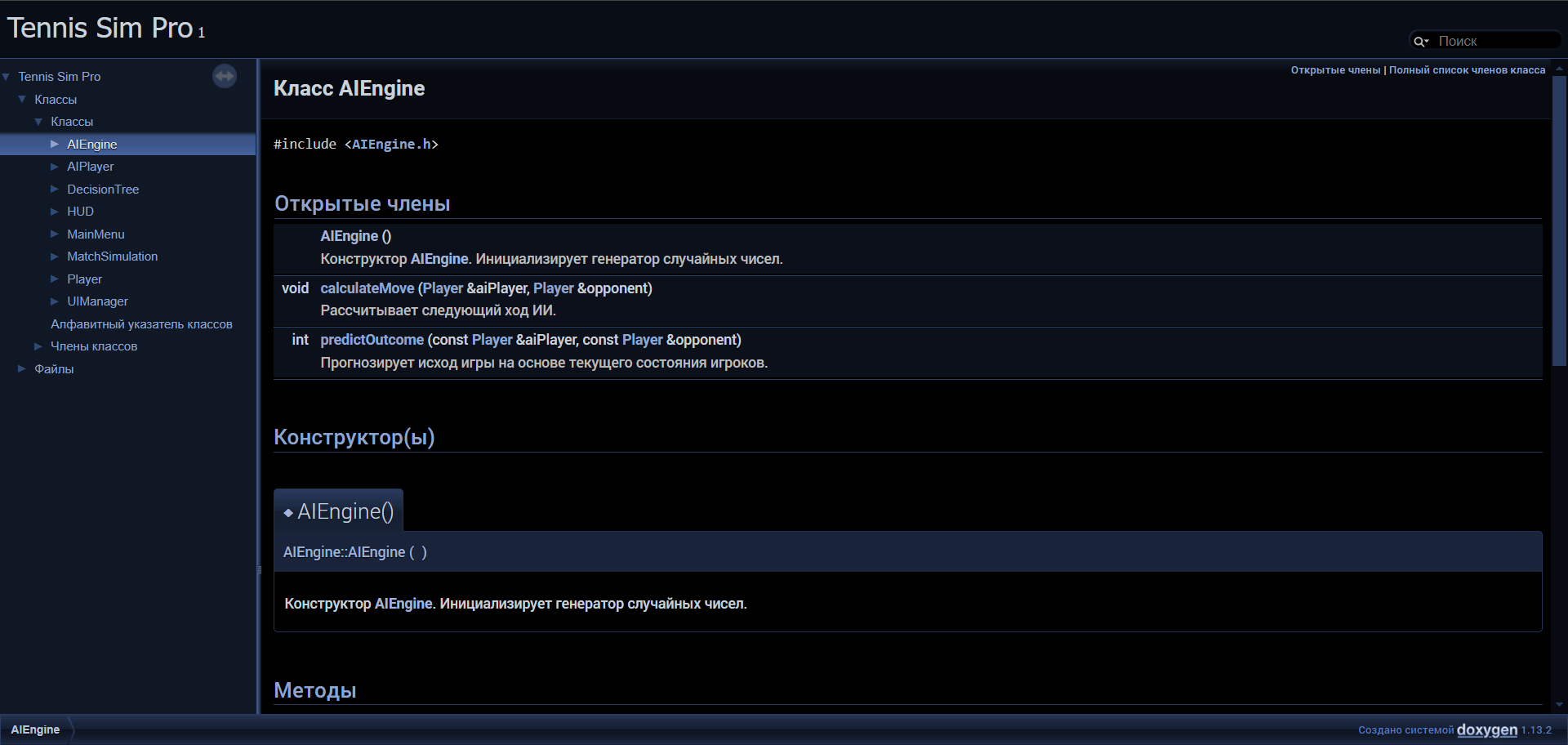


Рисунок 5 – Пример документации файла AIEngine.cpp